Description of STM32H7 HAL and low-layer drivers

Introduction

STM32Cube is an STMicroelectronics original initiative to significantly improve developer productivity by reducing development effort, time and cost. STM32Cube covers the STM32 portfolio.

STM32Cube includes:

- STM32CubeMX, a graphical software configuration tool that allows the generation of C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per Series (such as STM32CubeH7 for STM32H7 Series)
  - The STM32Cube HAL, STM32 abstraction layer embedded software ensuring maximized portability across the STM32 portfolio. HAL APIs are available for all peripherals.
  - Low-layer APIs (LL) offering a fast light-weight expert-oriented layer which is closer to the hardware than the HAL. LL APIs are available only for a set of peripherals.
  - A consistent set of middleware components such as RTOS, USB, TCP/IP and Graphics.
  - All embedded software utilities, delivered with a full set of examples.

The HAL driver layer provides a simple, generic multi-instance set of APIs (application programming interfaces) to interact with the upper layer (application, libraries and stacks). The HAL driver APIs are split into two categories: generic APIs, which provide common and generic functions for all the STM32 series and extension APIs, which include specific and customized functions for a given line or part number. The HAL drivers include a complete set of ready-to-use APIs that simplify the user application implementation. For example, the communication peripherals contain APIs to initialize and configure the peripheral, manage data transfers in polling mode, handle interrupts or DMA, and manage communication errors. The HAL drivers are feature-oriented instead of IP-oriented. For example, the timer APIs are split into several categories following the IP functions, such as basic timer, capture and pulse width modulation (PWM). The HAL driver layer implements run-time failure detection by checking the input values of all functions. Such dynamic checking enhances the firmware robustness. Run-time detection is also suitable for user application development and debugging.

The LL drivers offer hardware services based on the available features of the STM32 peripherals. These services reflect exactly the hardware capabilities, and provide atomic operations that must be called by following the programming model described in the product line reference manual. As a result, the LL services are not based on standalone processes and do not require any additional memory resources to save their states, counter or data pointers. All operations are performed by changing the content of the associated peripheral registers. Unlike the HAL, LL APIs are not provided for peripherals for which optimized access is not a key feature, or for those requiring heavy software configuration and/or a complex upper-level stack (such as USB).

The HAL and LL are complementary and cover a wide range of application requirements:

- The HAL offers high-level and feature-oriented APIs with a high-portability level. These hide the MCU and peripheral complexity from the end-user.
- The LL offers low-level APIs at register level, with better optimization but less portability. These require deep knowledge of the MCU and peripheral specifications.
- All STM32H7 single- and dual-core lines are supported by the same STM32CubeH7 HAL and LL drivers.

The HAL- and LL-driver source code is developed in Strict ANSI-C, which makes it independent of the development tools. It is checked with the CodeSonar® static analysis tool. It is fully documented.

It is compliant with MISRA C®:2012 standard.

This user manual is structured as follows:

- Supporting dual-core architectures
- Overview of HAL drivers
- Overview of low-layer drivers
- Cohabiting of HAL and LL drivers
- Detailed description of each peripheral driver: configuration structures, functions, and how to use the given API to build your application



1 General information

The STM32CubeH7 MCU Package runs on STM32H7 32-bit microcontrollers based on the Arm® Cortex®-M processor.

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

2 Acronyms and definitions

Table 1. **Acronyms and definitions**

Acronym	Definition
ADC	Analog-to-digital converter
AES	Advanced encryption standard
ANSI	American national standards institute
API	Application programming interface
BSP	Board support package
CAN	Controller area network
CEC	Consumer electronic controller
CMSIS	Cortex microcontroller software interface standard
COMP	Comparator
CORDIC	Trigonometric calculation unit
CPU	Central processing unit
CRC	CRC calculation unit
CRYP	Cryptographic processor
CSS	Clock security system
DAC	Digital to analog converter
DLYB	Delay block
DCMI	Digital camera interface
DFSDM	Digital filter sigma delta modulator
DMA	Direct memory access
DMAMUX	Direct memory access request multiplexer
DSI	Display serial interface
DTS	Digital temperature sensor
ESE	Security enable Flash user option bit
ETH	Ethernet controller
EXTI	External interrupt/event controller
FDCAN	Flexible data-rate controller area network unit
FLASH	Flash memory
FMAC	Filtering mathematical calculation unit
FMC	Flexible memory controller
FW	Firewall
GFXMMU	Chrom-GRC
GPIO	General purpose I/Os
HAL	Hardware abstraction layer
HASH	Hash processor
HCD	USB host controller driver
HRTIM	High-resolution timer
I2C	Inter-integrated circuit

Acronym	Definition
I2S	Inter-integrated sound
ICACHE	Instruction cache
IRDA	Infrared data association
IWDG	Independent watchdog
JPEG	Joint photographic experts group
LCD	Liquid crystal display controller
LTDC	LCD TFT display controller
LPTIM	Low-power timer
LPUART	Low-power universal asynchronous receiver/transmitter
MCO	Microcontroller clock output
MDIOS	Management data input/output (MDIO) slave
MDMA	Master direct memory access
MMC	MultiMediaCard
MPU	Memory protection unit
MSP	MCU specific package
NAND	NAND Flash memory
NOR	NOR Flash memory
NVIC	Nested vectored interrupt controller
OCTOSPI	Octo-SPI interface
OPAMP	Operational amplifier
OTFDEC	On-the-fly decryption engine
OTG-FS	USB on-the-go full-speed
PKA	Public key accelerator
PCD	USB peripheral controller driver
PPP	STM32 peripheral or block
PSSI	Parallel synchronous slave interface
PWR	Power controller
QSPI (QUADSPI)	Quad-SPI Flash memory
RAMECC	RAM ECC monitoring
RCC	Reset and clock controller
RNG	Random number generator
RTC	Real-time clock
SAI	Serial audio interface
SD	Secure digital
SDMMC	SD/SDIO/MultiMediaCard card host interface
SMARTCARD	Smartcard IC
SMBUS	System management bus
SPI	Serial peripheral interface
SPDIFRX	SPDIF-RX Receiver interface
SRAM	SRAM external memory
SWPMI	Serial wire protocol master interface

Acronym	Definition
SysTick	System tick timer
TIM	Advanced-control, general-purpose or basic timer
TSC	Touch sensing controller
UART	Universal asynchronous receiver/transmitter
UCPD	USB Type-C® and Power Delivery interface
USART	Universal synchronous receiver/transmitter
VREFBUF	Voltage reference buffer
WWDG	Window watchdog
USB	Universal serial bus

3 Support of dual-core architectures

The STM32H7 microcontrollers come in two product lines:

- Single-core lines, based on an Arm® Cortex®-M7 architecture.
- Dual-core lines based on an Arm® Cortex®-M7 and Arm® Cortex®-M4 architecture.

Supporting dual-core architectures with STM32H7 HAL and LL drivers

In STM32H7 dual-core lines, all peripherals are available for both Arm® Cortex®-M7 (CM7) and Arm® Cortex®-M4 (CM4) cores. There is no default peripheral allocation to a given core. There is consequently a great advantage to share the same peripheral drivers between the cores. This mainly targets standard peripherals which features must be available for both cores.

However for most system peripheral HAL drivers (such as RCC, GPIO, FLASH, PWR and HSEM), some features can be present in dual-core lines but not in the single-core lines.

Other features (such as boot sequence, extended interrupt and event handling and power management) may be configured differently in the two lines.

STM32H7 HAL and LL drivers have consequently been built to support both single- and dual-core architectures, with the following considerations:

- *DUAL_CORE*: this define statement is used to delimit the code (such as defines, functions and macros) that is available only in dual-core architectures. This define statement is automatically available through the STM32 CMSIS dual-core device files stm32h7XYxx.h.

Figure 1. Example of code portion used for dual-core architectures

```
void HAL_PWR_EnterSTOPMode(uint32_t Regulator, uint8_t STOPEntry)
{
  (...)
#if defined(DUAL_CORE)
  /* Keep DSTOP mode when D1 domain enters Deep sleep */
  CLEAR_BIT(PWR->CPUCR, PWR_CPUCR_PDDS_D1);
  CLEAR_BIT(PWR->CPU2CR, PWR_CPU2CR_PDDS_D1);
#else
  /* Keep DSTOP mode when D1 domain enters Deep sleep */
  CLEAR_BIT(PWR->CPUCR, PWR_CPUCR_PDDS_D1);

  /* Keep DSTOP mode when D2 domain enters Deep sleep */
  CLEAR_BIT(PWR->CPUCR, PWR_CPUCR_PDDS_D2);

  /* Keep DSTOP mode when D3 domain enters Deep sleep */
  CLEAR_BIT(PWR->CPUCR, PWR_CPUCR_PDDS_D3);
#endif /*DUAL_CORE*/
 (...)
}
```

- *CORE_CM4*: this define statement is used to delimit the code that contains a specific configuration/code portion applying to the Arm® Cortex®-M4 core. This define statement must be available in the compiler preprocessor define list at application level for the Arm® Cortex®-M4 target configuration.
- *CORE_CM7*: this define statement is used to delimit the code containing a specific configuration/code portion applying to the Arm® Cortex®-M7 core. This define statement must be available in the compiler preprocessor defines list at application level for the Arm® Cortex®-M7 target configuration.

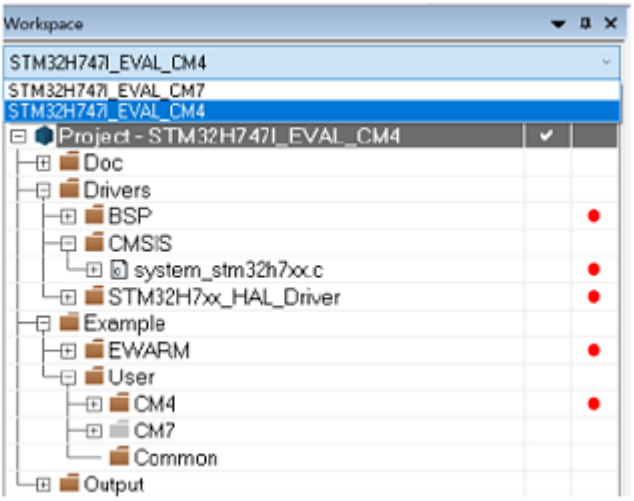
Dual-core architecture applications

The dual-core examples and applications available within the STM32CubeH7 MCU Package are structured as follows:

- One project (one workspace) per example/application, in order to be compatible with the single-core architecture.
- Two target project configurations per workspace (one per core), named STM32YYxx_CM7 and STM32YYxx_CM4.

The target configurations can be configured individually by setting the following options: target device, linker options, read-only (RO) and read/write (RW) zones, preprocessor symbols and defines (CORE_CM7 , CORE_CM4). This allows compiling user code linked and programmed separately for each core, and generating two binaries, CM7 and CM4.

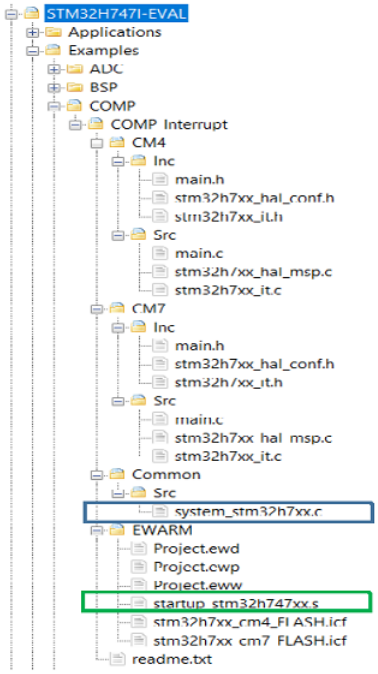
Figure 2. Dual-core project structure



The applicative layer source files of dual-core examples and applications is organized as follows:

- Two separate folders, *CM7* and *CM4* for Arm® Cortex®-M7 and Arm® Cortex®-M4, respectively.
- Each *CM7* and *CM4* folder includes in turns the following subfolders:
  - *Inc* that hosts all the header files for Arm® Cortex®-M7 and -M4.
  - *Src* that hosts the source code files for Arm® Cortex®-M7 and -M4.
- A *common* folder, with *Inc* and *Src* subfolders, that hosts the common header and source files for both cores.

Figure 3. Dual-core example structure



4 Overview of HAL drivers

The HAL drivers are designed to offer a rich set of APIs and to interact easily with the application upper layers.

Each driver consists of a set of functions covering the most common peripheral features. The development of each driver is driven by a common API which standardizes the driver structure, the functions and the parameter names.

The HAL drivers include a set of driver modules, each module being linked to a standalone peripheral. However, in some cases, the module is linked to a peripheral functional mode. As an example, several modules exist for the USART peripheral: UART driver module, USART driver module, SMARTCARD driver module and IRDA driver module.

The HAL main features are the following:

- Cross-family portable set of APIs covering the common peripheral features as well as extension APIs in case of specific peripheral features.
- Three API programming models: polling, interrupt and DMA.
- APIs are RTOS compliant:
  - Fully reentrant APIs
  - Systematic usage of timeouts in polling mode
- Support of peripheral multi-instance allowing concurrent API calls for multiple instances of a given peripheral (such as USART1 or USART2)
- All HAL APIs implement user-callback functions mechanism:
  - Peripheral Init/DeInit HAL APIs can call user-callback functions to perform peripheral system level Initialization/De-Initialization (clock, GPIOs, interrupt, DMA)
  - Peripherals interrupt events
  - Error events
- Object locking mechanism: safe hardware access to prevent multiple spurious accesses to shared resources.
- Timeout used for all blocking processes: the timeout can be a simple counter or a timebase.

4.1 HAL and user-application files

4.1.1 HAL driver files

HAL drivers are composed of the following set of files:

Table 2. HAL driver files

File	Description
stm32h7xx_hal_ppp.c	Main peripheral/module driver file It includes the APIs that are common to all STM32 devices. *Example: stm32h7xx_hal_adc.c, stm32h7xx_hal_irda.c.*
stm32h7xx_hal_ppp.h	Header file of the main driver C file It includes common data, handle and enumeration structures, define statements and macros, as well as the exported generic APIs. *Example:stm32h7xx_hal_adc.h,stm32h7xx_hal_irda.h.*
stm32h7xx_hal_ppp_ex.c	Extension file of a peripheral/module driver. It includes the specific APIs for a given part number or family, as well as the newly defined APIs that overwrite the default generic APIs if the internal process is implemented in different way. *Example:stm32h7xx_hal_adc_ex.c,stm32h7xx_hal_flash_ex.c.*
stm32h7xx_hal_ppp_ex.h	Header file of the extension C file It includes the specific data and enumeration structures, define statements and macros, as well as the exported device part number specific APIs *Example: stm32h7xx_hal_adc_ex.h,stm32h7xx_hal_flash_ex.h.*
stm32h7xx_hal.c	This file is used for HAL initialization and contains DBGMCU, Remap and Time Delay based on SysTick APIs.
stm32h7xx_hal.h	stm32h7xx_hal.c header file
stm32h7xx_hal_msp_template.c	Template file to be copied to the user application folder It contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
stm32h7xx_hal_conf_template.h	Template file allowing to customize the drivers for a given application
stm32h7xx_hal_def.h	Common HAL resources such as common define statements, enumerations, structures and macros

4.1.2 User-application files

The minimum files required to build an application using the HAL are listed in the table below:

Table 3. User-application files

File	Description
system_stm32h7xx.c	This file contains SystemInit() that is called at startup just after reset and before branching to the main program. It does not configure the system clock at startup (contrary to the standard library). This is to be done using the HAL APIs in the user files. It allows relocating the vector table in internal SRAM.
startup_stm32h7xx.s	Toolchain specific file that contains reset handler and exception vectors. For some toolchains, it allows adapting the stack/heap size to fit the application requirements.
stm32h7xx_flash.icf (optional)	Linker file for EWARM toolchain allowing mainly adapting the stack/heap size to fit the application requirements.
stm32h7xx_hal_msp.c	This file contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.

File	Description
stm32h7xx_hal_conf.h	This file allows the user to customize the HAL drivers for a specific application. It is not mandatory to modify this configuration. The application can use the default configuration without any modification.
stm32h7xx_it.c/.h	This file contains the exceptions handler and peripherals interrupt service routine, and calls HAL_IncTick() at regular time intervals to increment a local variable (declared in *stm32h7xx*_hal.c) used as HAL timebase. By default, this function is called each 1ms in Systick ISR. . The PPP_IRQHandler() routine must call HAL_PPP_IRQHandler() if an interrupt based process is used within the application.
main.c/.h	This file contains the main program routine, mainly: • Arm® Cortex®-M7 instruction and data cache enabling • Call to HAL_Init() • assert_failed() implementation • system clock configuration • peripheral HAL initialization and user application code.

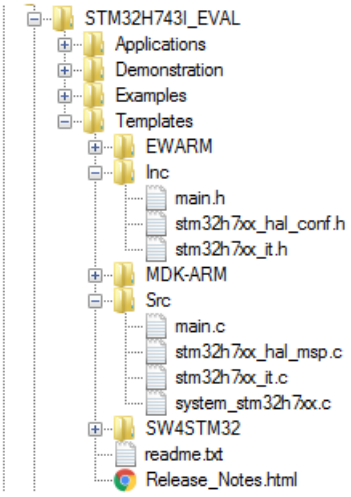
The STM32Cube package comes with ready-to-use project templates, one for each supported board. Each project contains the files listed above and a preconfigured project for the supported toolchains.

Each project template provides empty main loop function and can be used as a starting point to get familiar with project settings for STM32Cube. Its features are the following:

- It contains the sources of HAL, CMSIS and BSP drivers which are the minimal components to develop a code on a given board.
- It contains the include paths for all the firmware components.
- It defines the STM32 device supported, and allows configuring the CMSIS and HAL drivers accordingly.
- It provides ready to use user files preconfigured as defined below:
  - HAL is initialized
  - SysTick ISR implemented for HAL_Delay()
  - System clock configured with the maximum device frequency.

Note: If an existing project is copied to another location, then include paths must be updated.

Figure 4. Example of project template



4.2 HAL data structures

Each HAL driver can contain the following data structures:

- Peripheral handle structures
- Initialization and configuration structures
- Specific process structures.

4.2.1 Peripheral handle structures

The APIs have a modular generic multi-instance architecture that enables working with several IP instances simultaneously.

PPP_HandleTypeDef *handle is the main structure that is implemented in the HAL drivers. It handles the peripheral/module configuration and registers and embeds all the structures and variables needed to follow the peripheral device flow.

The peripheral handle is used for the following purposes:

- Multi-instance support: each peripheral/module instance has its own handle. As a result instance resources are independent.
- Peripheral process intercommunication: the handle is used to manage shared data resources between the process routines.

  Example: global pointers, DMA handles, state machine.
- Storage : this handle is used also to manage global variables within a given HAL driver.

An example of peripheral structure is shown below:

```
typedef struct
{
USART_TypeDef *Instance; /* USART registers base address */
USART_InitTypeDef Init; /* Usart communication parameters */
uint8_t *pTxBuffPtr;/* Pointer to Usart Tx transfer Buffer */
uint16_t TxXferSize; /* Usart Tx Transfer size */
__IO uint16_t TxXferCount;/* Usart Tx Transfer Counter */
uint8_t *pRxBuffPtr;/* Pointer to Usart Rx transfer Buffer */
uint16_t RxXferSize; /* Usart Rx Transfer size */
__IO uint16_t RxXferCount; /* Usart Rx Transfer Counter */
DMA_HandleTypeDef *hdmatx; /* Usart Tx DMA Handle parameters */
DMA_HandleTypeDef *hdmarx; /* Usart Rx DMA Handle parameters */
HAL_LockTypeDef Lock; /* Locking object */
__IO HAL_USART_StateTypeDef State; /* Usart communication state */
__IO HAL_USART_ErrorTypeDef ErrorCode;/* USART Error code */
}USART_HandleTypeDef;
```

Note:

1. *The multi-instance feature implies that all the APIs used in the application are reentrant and avoid using global variables because subroutines can fail to be reentrant if they rely on a global variable to remain unchanged but that variable is modified when the subroutine is recursively invoked. For this reason, the following rules are respected:*
   - *Reentrant code does not hold any static (or global) non-constant data: reentrant functions can work with global data. For example, a reentrant interrupt service routine can grab a piece of hardware status to work with (for example serial port read buffer) which is not only global, but volatile. Still, typical use of static variables and global data is not advised, in the sense that only atomic read-modify-write instructions should be used in these variables. It should not be possible for an interrupt or signal to occur during the execution of such an instruction.*
   - *Reentrant code does not modify its own code.*
2. *When a peripheral can manage several processes simultaneously using the DMA (full duplex case), the DMA interface handle for each process is added in the PPP_HandleTypeDef.*
3. *For the shared and system peripherals, no handle or instance object is used. The peripherals concerned by this exception are the following:*
   - *GPIO*
   - *SYSTICK*
   - *NVIC*
   - *PWR*
   - *RCC*
   - *FLASH*

4.2.2 Initialization and configuration structure

These structures are defined in the generic driver header file when it is common to all part numbers. When they can change from one part number to another, the structures are defined in the extension header file for each part number.

```
typedef struct
        {
        uint32_t BaudRate;  /*!< This member configures the UART communication baud rate
*/
        uint32_t WordLength;/*!< Specifies the number of data bits transmitted or received
in a frame */
        uint32_t StopBits;  /*!< Specifies the number of stop bits transmitted */
        uint32_t Parity;    /*!< Specifies the parity mode */
        uint32_t Mode;      /*!< Specifies whether the Receive or Transmit mode is enabled
or disabled */
        uint32_t HwFlowCtl; /*!< Specifies whether the hardware flow control mode is enabled
or disabled.*/
        uint32_t OverSampling;    /*!< Specifies whether the Over sampling 8 is enabled or
disabled, to achieve higher speed (up to f_PCLK/8) */
        uint32_t OneBitSampling;  /*!< Specifies whether a single sample or three samples'
majority vote is selected  */
        uint32_t Prescaler;       /*!< Specifies the prescaler value used to divide the UART
clock source */
        uint32_t FIFOMode;        /*!< Specifies if the FIFO mode will be used. This
parameter can be a value */
        uint32_t TXFIFOThreshold; /*!< Specifies the TXFIFO threshold level */
        uint32_t RXFIFOThreshold; /*!< Specifies the RXFIFO threshold level */
    }UART_InitTypeDef;
```

Note: The config structure is used to initialize the sub-modules or sub-instances. See below example:

```
HAL_ADC_ConfigChannel (ADC_HandleTypeDef* hadc, ADC_ChannelConfTypeDef* sConfig)
```

4.2.3 Specific process structures

The specific process structures are used for specific process (common APIs). They are defined in the generic driver header file.

Example:

```
HAL_PPP_Process (PPP_HandleTypeDef* hadc,PPP_ProcessConfig* sConfig)
```

4.3 API classification

The HAL APIs are classified into the following categories:

- **Generic APIs:** common generic APIs applying to all STM32 devices. These APIs are consequently present in the generic HAL driver files of all STM32 microcontrollers.

```
HAL_StatusTypeDef HAL_ADC_Init(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_DeInit(ADC_HandleTypeDef *hadc);
HAL_StatusTypeDef HAL_ADC_Start(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_Stop(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_Start_IT(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_Stop_IT(ADC_HandleTypeDef* hadc);
void HAL_ADC_IRQHandler(ADC_HandleTypeDef* hadc);
```

- **Extension APIs:**

  This set of APIs are **family specific APIs** that apply to a given family. They are located in the extension HAL driver file (see example below related to the ADC).

```
HAL_StatusTypeDef HAL_ADCEx_Calibration_Start(ADC_HandleTypeDef* hadc, uint32_t
SingleDiff); uint32_t HAL_ADCEx_Calibration_GetValue(ADC_HandleTypeDef* hadc, uint32_t
SingleDiff);
```

The following table summarizes the location of the different categories of HAL APIs in the driver files.

Table 4. API classification

	Generic file	Extension file
Common APIs	X	X
Family specific APIs	-	X

Note: Family specific APIs are only related to a given family. This means that if a specific API is implemented in another family, and the arguments of this latter family are different, additional structures and arguments might need to be added.

The IRQ handlers are used for common and family specific processes.

4.4 Devices supported by HAL drivers

Table 5. List of devices supported by HAL drivers

IP/module	STM32H742xx	STM32H743xx	STM32H753xx	STM32H750xx	STM32H753xx	STM32H745xx	STM32H755xx	STM32H747xx	STM32H757xx	STM32H7A3xx	STM32H7A3xxQ	STM32H7B0xx	STM32H7B0xxQ	STM32H7B3xx	STM32H7B3xxQ	STM32H723xx	STM32H733xx	STM32H725xx	STM32H735xx	STM32H730xx	STM32H730xxQ
stm32h7xx_hal.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_adc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_adc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_cec.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_comp.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_cortex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_crc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_crc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_cryp.c	No	No	Yes	Yes	Yes	No	Yes	No	Yes	No	No	Yes	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes
stm32h7xx_hal_cryp_ex.c	No	No	Yes	Yes	Yes	No	Yes	No	Yes	No	No	Yes	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes
stm32h7xx_hal_dac.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_dac_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_dcmi.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_dfsdm.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_dma.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_dma_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7x_hal_dma2d.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7x_hal_dsi.c	No	No	No	No	No	No	No	Yes	Yes	No	No	No	No	No	No	No	No	No	No	No	No
stm32h7xx_hal_dts.c	No	No	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_eth.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_eth_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_fdcan.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_flash.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_flash_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_gpio.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_gfxmmu.c	No	No	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No	No
stm32h7xx_hal_hash.c	No	No	Yes	Yes	Yes	No	Yes	No	Yes	No	No	Yes	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes
stm32h7xx_hal_hash_ex.c	No	No	Yes	Yes	Yes	No	Yes	No	Yes	No	No	Yes	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes
stm32h7xx_hal_hcd.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_hrtim.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No	No	No	No	No	No	No	No
stm32h7xx_hal_hsem.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

IP/module	STM32H742xx	STM32H743xx	STM32H753xx	STM32H750xx	STM32H753xx	STM32H745xx	STM32H755xx	STM32H747xx	STM32H757xx	STM32H7A3xx	STM32H7A3xxQ	STM32H7B0xx	STM32H7B0xxQ	STM32H7B3xx	STM32H7B3xxQ	STM32H723xx	STM32H733xx	STM32H725xx	STM32H735xx	STM32H730xx	STM32H730xxQ
stm32h7xx_hal_i2c.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_i2c_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_i2s.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_i2s_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_irda.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_iwdg.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_jpeg.c	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No	No
stm32h7xx_hal_lptim.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_ltdc.c	No	Yes	Yes	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_mdios.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_mdma.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_mmc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_mmc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_nand.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_nor.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_opamp.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_opamp_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_ospi.c	No	No	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_otfdec.c	No	No	No	No	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes
stm32h7xx_hal_pcd.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_pcd_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_pssi.c	No	No	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_pwr.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_pwr_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_qspi.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No	No	No	No	No	No	No	No
stm32h7xx_hal_rcc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_rcc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_rng.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_rtc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_rtc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_sai.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_sai_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

IP/module	STM32H742xx	STM32H743xx	STM32H753xx	STM32H750xx	STM32H753xx	STM32H745xx	STM32H755xx	STM32H747xx	STM32H757xx	STM32H7A3xx	STM32H7A3xxQ	STM32H7B0xx	STM32H7B0xxQ	STM32H7B3xx	STM32H7B3xxQ	STM32H723xx	STM32H733xx	STM32H725xx	STM32H735xx	STM32H730xx	STM32H730xxQ
stm32h7xx_hal_sd.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_sd_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_sdram.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_smartcard.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_smartcard_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_smbus.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_spdifrx.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_spi.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_spi_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_sram.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_swpmi.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_tim.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_tim_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_uart.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_uart_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_usart.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_hal_wwdg.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_ll_delayblock.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_ll_fmc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_ll_sdmmc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32h7xx_ll_usb.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

4.5 HAL driver rules

4.5.1 HAL API naming rules

The following naming rules are used in HAL drivers:

Table 6. HAL API naming rules

	Generic	Family specific APIs	Device specific APIs
File names	*stm32h7xx_hal_ppp (c/h)*	*stm32h7xx_hal_ppp_ex (c/h)*	*stm32h7xx_ hal_ppp_ex (c/h)*
Module name	*HAL_PPP_ MODULE*		
Function name	*HAL_PPP_Function HAL_PPP_FeatureFunction_MODE*	*HAL_PPPEx_Function HAL_PPPEx_FeatureFunction_MODE*	*HAL_PPPEx_Function HAL_PPPEx_FeatureFunction_MODE*
Handle name	*PPP_HandleTypedef*	*NA*	*NA*
Init structure name	*PPP_InitTypeDef*	*NA*	*PPP_InitTypeDef*
Enum name	*HAL_PPP_StructnameTypeDef*	*NA*	*NA*

- The **PPP** prefix refers to the peripheral functional mode and not to the peripheral itself. For example, if the USART, PPP can be USART, IRDA, UART or SMARTCARD depending on the peripheral mode.
- The constants used in one file are defined within this file. A constant used in several files is defined in a header file. All constants are written in uppercase, except for peripheral driver function parameters.
- typedef variable names should be suffixed with _TypeDef.
- Registers are considered as constants. In most cases, their name is in uppercase and uses the same acronyms as in the STM32H7 reference manuals.
- Peripheral registers are declared in the PPP_TypeDef structure (for example ADC_TypeDef) in stm32h7xxx.h header file:

  stm32h7xxx.h corresponds to stm32h742xx.h, stm32h743xx.h, stm32h745xx.h, stm32h747xx.h, stm32h750xx.h, stm32h753xx.h, stm32h755xx.h, stm32h757xx.h, stm32h7a3xx.h, stm32h7a3xxq.h, stm32h7b3xx.h, stm32h7b3xxq.h stm32h7b0xx.h, stm32h7B0xxq.h, stm32h725xx.h, stm32h723xx.h, stm32h730xx.h, stm32h730xxq.h, stm32h733xx.h and stm32h735xx.h.
- Peripheral function names are prefixed by HAL_, then the corresponding peripheral acronym in uppercase followed by an underscore. The first letter of each word is in uppercase (for example HAL_UART_Transmit()). Only one underscore is allowed in a function name to separate the peripheral acronym from the rest of the function name.
- The structure containing the PPP peripheral initialization parameters are named PPP_InitTypeDef (for example ADC_InitTypeDef).
- The structure containing the Specific configuration parameters for the PPP peripheral are named PPP_xxxxConfTypeDef (for example ADC_ChannelConfTypeDef).
- Peripheral handle structures are named PPP_HandleTypedef (e.g DMA_HandleTypeDef)
- The functions used to initialize the PPP peripheral according to parameters specified in PPP_InitTypeDef are named HAL_PPP_Init (for example HAL_TIM_Init()).
- The functions used to reset the PPP peripheral registers to their default values are named HAL_PPP_DeInit (for example HAL_TIM_DeInit()).
- The **MODE** suffix refers to the process mode, which can be polling, interrupt or DMA. As an example, when the DMA is used in addition to the native resources, the function should be called: *HAL_PPP_Function_DMA()*.

- The **Feature** prefix should refer to the new feature.
  Example: *HAL_ADC_Start*() refers to the injection mode.

4.5.2 HAL general naming rules

- For the shared and system peripherals, no handle or instance object is used. This rule applies to the following peripherals:
  - GPIO
  - SYSTICK
  - NVIC
  - RCC
  - FLASH

Example: The *HAL_GPIO_Init()* requires only the GPIO address and its configuration parameters.

```
HAL_StatusTypeDef HAL_GPIO_Init (GPIO_TypeDef* GPIOx, GPIO_InitTypeDef *Init)
{
/*GPIO Initialization body */
}
```

- The macros that handle interrupts and specific clock configurations are defined in each peripheral/module driver. These macros are exported in the peripheral driver header files so that they can be used by the extension file. The list of these macros is defined below:

Note: This list is not exhaustive and other macros related to peripheral features can be added, so that they can be used in the user application.

Table 7. Macros handling interrupts and specific clock configurations

Macros	Description
__HAL_PPP_ENABLE_IT(__HANDLE__, __INTERRUPT__)	Enables a specific peripheral interrupt
__HAL_PPP_DISABLE_IT(__HANDLE__, __INTERRUPT__)	Disables a specific peripheral interrupt
__HAL_PPP_GET_IT (__HANDLE__, __ INTERRUPT __)	Gets a specific peripheral interrupt status
__HAL_PPP_CLEAR_IT (__HANDLE__, __ INTERRUPT __)	Clears a specific peripheral interrupt status
__HAL_PPP_GET_FLAG (__HANDLE__, __FLAG__)	Gets a specific peripheral flag status
__HAL_PPP_CLEAR_FLAG (__HANDLE__, __FLAG__)	Clears a specific peripheral flag status
__HAL_PPP_ENABLE(__HANDLE__)	Enables a peripheral
__HAL_PPP_DISABLE(__HANDLE__)	Disables a peripheral
__HAL_PPP_XXXX (__HANDLE__, __PARAM__)	Specific PPP HAL driver macro
__HAL_PPP_GET_ IT_SOURCE (__HANDLE__, __ INTERRUPT __)	Checks the source of specified interrupt

- NVIC and SYSTICK are two Arm® Cortex® core features. The APIs related to these features are located in the stm32h7xx_hal_cortex.c file.
- When a status bit or a flag is read from registers, it is composed of shifted values depending on the number of read values and of their size. In this case, the returned status width is 32 bits. Example:
  STATUS = XX | (YY << 16) or STATUS = XX | (YY << 8) | (YY << 16) | (YY << 24).

- The PPP handles are valid before using the HAL_PPP_Init() API. The init function performs a check before modifying the handle fields.

```
 HAL_PPP_Init(PPP_HandleTypeDef)
if(hppp == NULL)
 {
 return HAL_ERROR;
 }
```

- The macros defined below are used:
  - Conditional macro:

```
#define ABS(x) (((x) > 0) ? (x) : -(x))
```

  - Pseudo-code macro (multiple instructions macro):

```
#define __HAL_LINKDMA(__HANDLE__, __PPP_DMA_FIELD_, __DMA_HANDLE_) \
do{ \
(__HANDLE__)->__PPP_DMA_FIELD_ = &(__DMA_HANDLE_); \
(__DMA_HANDLE_).Parent = (__HANDLE__); \
} while(0)
```

4.5.3 HAL interrupt handler and callback functions

Besides the APIs, HAL peripheral drivers include:

- HAL_PPP_IRQHandler() peripheral interrupt handler that should be called from stm32h7xx_it.c
- User callback functions.

The user callback functions are defined as empty functions with “weak” attribute. They have to be defined in the user code.

There are three types of user callbacks functions:

- Peripheral system level initialization/ de-Initialization callbacks: HAL_PPP_MspInit() and HAL_PPP_MspDeInit
- Process complete callbacks : HAL_PPP_ProcessCpltCallback
- Error callback: HAL_PPP_ErrorCallback.

Table 8. Callback functions

Callback functions	Example
HAL_PPP_MspInit() / _DeInit()	Example: HAL_USART_MspInit() Called from HAL_PPP_Init() API function to perform peripheral system level initialization (GPIOs, clock, DMA, interrupt)
HAL_PPP_ProcessCpltCallback	Example: HAL_USART_TxCpltCallback Called by peripheral or DMA interrupt handler when the process completes
HAL_PPP_ErrorCallback	Example: HAL_USART_ErrorCallback Called by peripheral or DMA interrupt handler when an error occurs

4.6 HAL generic APIs

The generic APIs provide common generic functions applying to all STM32 devices. They are composed of four APIs groups:

- **Initialization and de-initialization functions:**HAL_PPP_Init(), HAL_PPP_DeInit()
- **IO operation functions**: HAL_PPP_Read(), HAL_PPP_Write(),HAL_PPP_Transmit(), HAL_PPP_Receive()
- **Control functions**: HAL_PPP_Set (), HAL_PPP_Get ().
- **State and Errors functions**: HAL_PPP_GetState (), HAL_PPP_GetError ().

For some peripheral/module drivers, these groups are modified depending on the peripheral/module implementation.

Example: in the timer driver, the API grouping is based on timer features (such as PWM, OC and IC).

The initialization and de-initialization functions allow initializing a peripheral and configuring the low-level resources, mainly clocks, GPIO, alternate functions (AF) and possibly DMA and interrupts. The *HAL_DeInit()* function restores the peripheral default state, frees the low-level resources and removes any direct dependency with the hardware.

The IO operation functions perform a row access to the peripheral payload data in write and read modes.

The control functions are used to change dynamically the peripheral configuration and set another operating mode.

The peripheral state and errors functions allow retrieving in run time the peripheral and data flow states, and identifying the type of errors that occurred. The example below is based on the ADC peripheral. The list of generic APIs is not exhaustive. It is only given as an example.

Table 9. HAL generic APIs

Function group	Common API name	Description
Initialization group	*HAL_ADC_Init()*	This function initializes the peripheral and configures the low -level resources (clocks, GPIO, AF..)
	HAL_ADC_DeInit()	This function restores the peripheral default state, frees the low-level resources and removes any direct dependency with the hardware.
IO operation group	*HAL_ADC_Start ()*	This function starts ADC conversions when the polling method is used
	HAL_ADC_Stop ()	This function stops ADC conversions when the polling method is used
	HAL_ADC_PollForConversion()	This function allows waiting for the end of conversions when the polling method is used. In this case, a timout value is specified by the user according to the application.
	HAL_ADC_Start_IT()	This function starts ADC conversions when the interrupt method is used
	HAL_ADC_Stop_IT()	This function stops ADC conversions when the interrupt method is used
	HAL_ADC_IRQHandler()	This function handles ADC interrupt requests
	HAL_ADC_ConvCpltCallback()	Callback function called in the IT subroutine to indicate the end of the current process or when a DMA transfer has completed
	HAL_ADC_ErrorCallback()	Callback function called in the IT subroutine if a peripheral error or a DMA transfer error occurred
Control group	*HAL_ADC_ConfigChannel()*	This function configures the selected ADC regular channel, the corresponding rank in the sequencer and the sample time
	HAL_ADC_AnalogWDGConfig	This function configures the analog watchdog for the selected ADC
State and Errors group	*HAL_ADC_GetState()*	This function allows getting in run time the peripheral and the data flow states.
	HAL_ADC_GetError()	This fuction allows getting in run time the error that occurred during IT routine

4.7 HAL extension APIs

4.7.1 HAL extension model overview

The extension APIs provide specific functions or overwrite modified APIs for a specific family (series) or specific part number within the same family.

The extension model consists of an additional file, stm32h7xx_hal_ppp_ex.c, that includes all the specific functions and define statements (stm32h7xx_hal_ppp_ex.h) for a given part number.

Below an example based on the ADC peripheral:

Table 10. HAL extension APIs

Function group	Common API name
HAL_ADCEx_CalibrationStart()	This function is used to start the automatic ADC calibration

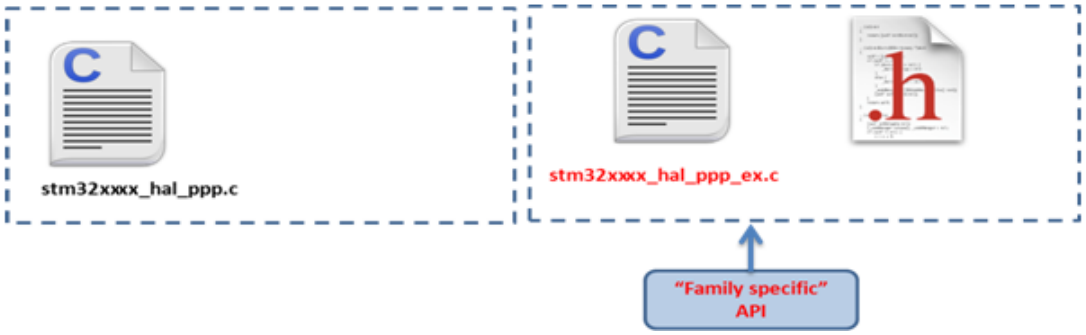
4.7.2 HAL extension model cases

The specific peripheral features can be handled by the HAL drivers in five different ways. They are described below.

Adding a family-specific function

In this case, the API is added in the extension driver C file and named HAL_PPPEx_Function ().

Figure 5. Adding family-specific functions

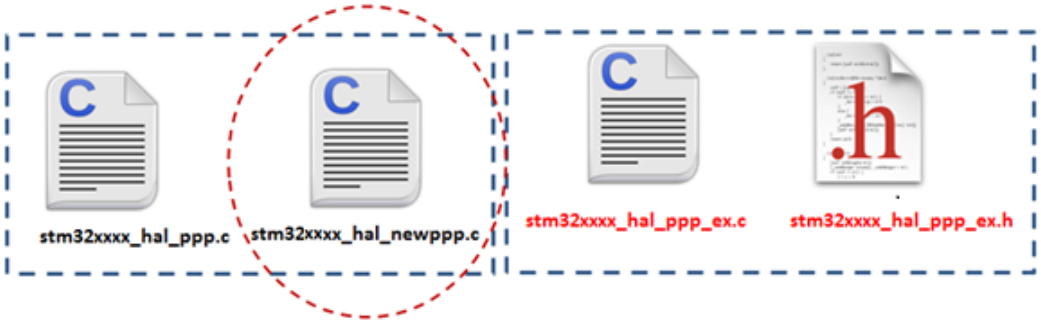


Adding a new peripheral (specific to a device belonging to a given family)

When a peripheral which is available only in a specific device is required, the APIs corresponding to this new peripheral/module (newPPP) are added in a new stm32h7xx_hal_newppp.c. However the inclusion of this file is selected in the stm32h7xx_hal_conf.h using the macro:

```
#define HAL_NEWPPP_MODULE_ENABLED
```

Figure 6. Adding new peripherals

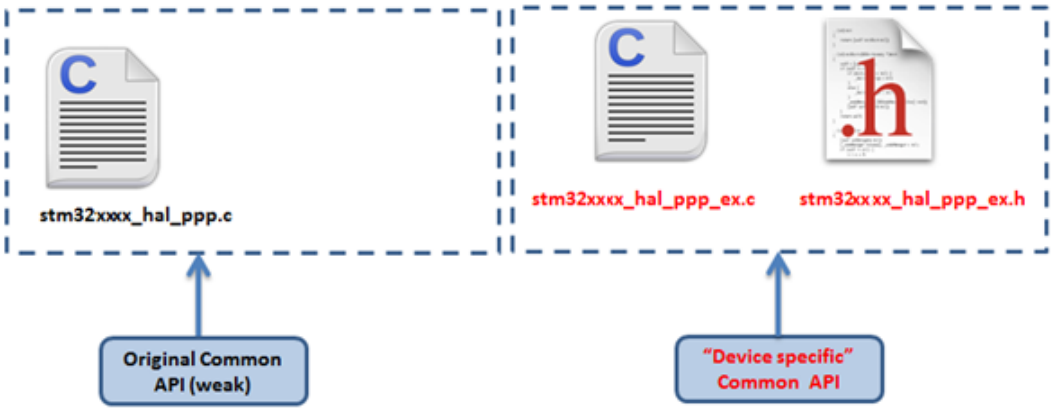


Example: stm32h7xx_hal_adc.c/h

Updating existing common APIs

In this case, the routines are defined with the same names in the stm32h7xx_hal_ppp_ex.c extension file, while the generic API is defined as *weak*, so that the compiler overwrites the original routine by the new defined function.

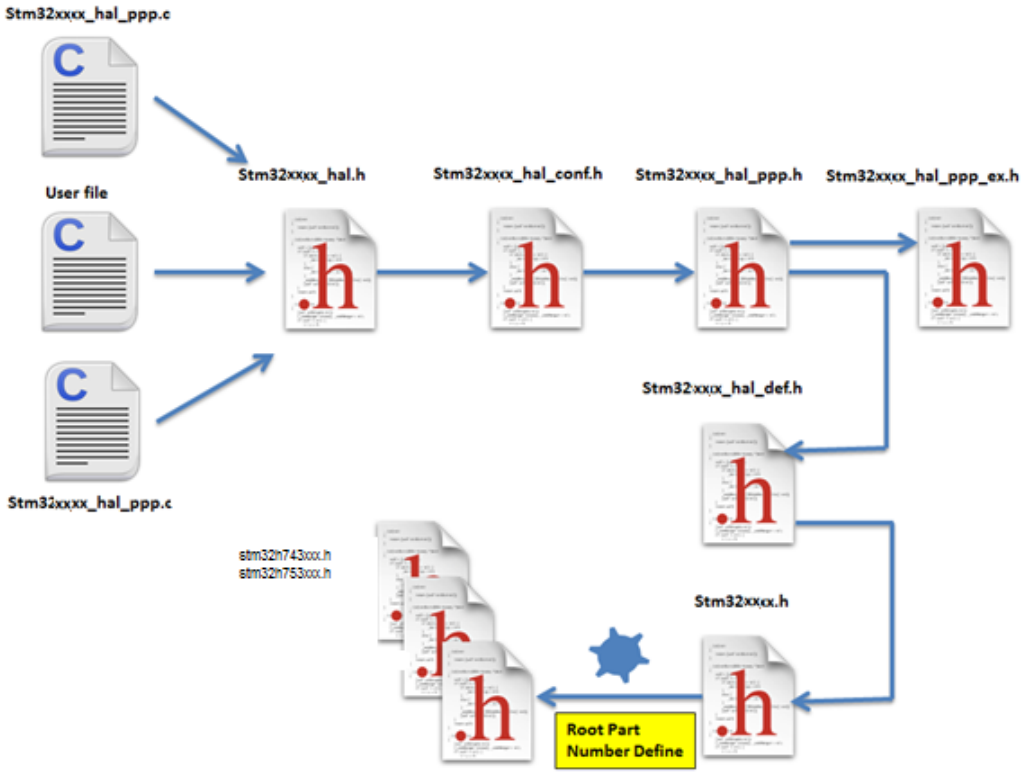
Figure 7. Updating existing APIs



4.8 File inclusion model

The header of the common HAL driver file (stm32h7xx_hal.h) includes the common configurations for the whole HAL library. It is the only header file that is included in the user sources and the HAL C sources files to be able to use the HAL resources.

Figure 8. File inclusion model



A PPP driver is a standalone module which is used in a project. The user must enable the corresponding USE_HAL_PPP_MODULE define statement in the configuration file.

```
/**********************************************************************
* @file stm32h7xx_hal_conf.h
* @author MCD Application Team
* @version VX.Y.Z * @date dd-mm-yyyy
* @brief This file contains the modules to be used
**********************************************************************
(…)
#define USE_HAL_USART_MODULE
#define USE_HAL_IRDA_MODULE
#define USE_HAL_DMA_MODULE
#define USE_HAL_RCC_MODULE
(…)
```

4.9 HAL common resources

The common HAL resources, such as common define enumerations, structures and macros, are defined in *stm32h7xx_hal_def.h.*The main common define enumeration is *HAL_StatusTypeDef*.

- **HAL Status**

  The HAL status is used by almost all HAL APIs, except for boolean functions and IRQ handler. It returns the status of the current API operations. It has four possible values as described below:

```
Typedef enum
{
HAL_OK = 0x00,
HAL_ERROR = 0x01,
HAL_BUSY = 0x02,
HAL_TIMEOUT = 0x03
} HAL_StatusTypeDef;
```

- **HAL Locked**

  The HAL lock is used by all HAL APIs to prevent accessing by accident shared resources.

```
typedef enum
{
HAL_UNLOCKED = 0x00, /*!<Resources unlocked */
HAL_LOCKED = 0x01 /*!< Resources locked */
} HAL_LockTypeDef;
```

  In addition to common resources, the stm32h7xx_hal_def.h file calls the stm32h7xx.h file in CMSIS library to get the data structures and the address mapping for all peripherals:

  – Declarations of peripheral registers and bits definition.

  – Macros to access peripheral registers hardware (such as Write register or Read register).

- **Common macros**

  – Macro defining HAL_MAX_DELAY

```
#define HAL_MAX_DELAY 0xFFFFFFFF
```

  – Macro linking a PPP peripheral to a DMA structure pointer:

```
#define __HAL_LINKDMA(__HANDLE__, __PPP_DMA_FIELD_, __DMA_HANDLE_) \
do{ \
  (__HANDLE__)->__PPP_DMA_FIELD_ = &(__DMA_HANDLE_); \
  (__DMA_HANDLE_).Parent = (__HANDLE__); \
} while(0)
```

4.10 HAL configuration

The configuration file, stm32h7xx_hal_conf.h, allows customizing the drivers for the user application. Modifying this configuration is not mandatory: the application can use the default configuration without any modification.

To configure these parameters, the user should enable, disable or modify some options by uncommenting, commenting or modifying the values of the related define statements as described in the table below:

Table 11. Define statements used for HAL configuration

Configuration item	Description	Default Value
HSE_VALUE	Defines the value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	25 000 000 Hz
HSE_STARTUP_TIMEOUT	Timeout for HSE start-up, expressed in ms	5000
HSI_VALUE	Defines the value of the internal oscillator (HSI) expressed in Hz.	64 000 000 Hz

Configuration item	Description	Default Value
LSE_VALUE	Defines the value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	32768 Hz
LSE_STARTUP_TIMEOUT	Timeout for LSE start-up, expressed in ms	5000
CSI_VALUE	Defines the value of the internal oscillator (CSI) expressed in Hz	4 000 000 Hz
LSI_VALUE	Defines the value of the internal oscillator (LSI) expressed in Hz	32 000 Hz
VDD_VALUE	VDD value	3300 (mV)
USE_RTOS	Enables the use of RTOS	FALSE (reserved for future use)

Note: The stm32h7xx_hal_conf_template.h file is located in the HAL drivers Inc folder. It should be copied to the user folder, renamed and modified as described above.

By default, the values defined in the stm32h7xx_hal_conf_template.h file are the same as the ones used for the examples and demonstrations. All HAL include files are enabled so that they can be used in the user code without modifications.

4.11 HAL system peripheral handling

This section gives an overview of how the system peripherals are handled by the HAL drivers. The full API list is provided within each peripheral driver description section.

4.11.1 Clocks

Two main functions can be used to configure the system clock:

- HAL_RCC_OscConfig (RCC_OscInitTypeDef *RCC_OscInitStruct). This function configures/enables multiple clock sources (HSE, HSI, LSE, LSI, PLL).
- HAL_RCC_ClockConfig (RCC_ClkInitTypeDef *RCC_ClkInitStruct, uint32_t FLatency). This function
  - selects the system clock source
  - configures AHB, APB1, APB2, APB3 and APB4 clock dividers
  - configures the number of Flash memory wait states
  - updates the SysTick configuration when HCLK clock changes.

Some peripheral clocks are not derived from the system clock (such as RTC, USB). In this case, the clock configuration is performed by an extended API defined in stm32h7xx_hal_rcc_ex.c: *HAL_RCCEx_PeriphCLKConfig(RCC_PeriphCLKInitTypeDef *PeriphClkInit)*.

Additional RCC HAL driver functions are available:

- HAL_RCC_DeInit() Clock de-initialization function that returns clock configuration to reset state
- Get clock functions that allow retreiving various clock configurations (such as system clock, HCLK, PCLK1 or PCLK2)
- MCO and CSS configuration functions

A set of macros are defined in stm32h7xx_hal_rcc.h and stm32h7xx_hal_rcc_ex.h. They allow executing elementary operations on RCC block registers, such as peripherals clock gating/reset control:

- __HAL_RCC_PPP_CLK_ENABLE/__HAL_RCC_PPP_CLK_DISABLE to enable/disable the peripheral clock
- __HAL_RCC_PPP_FORCE_RESET/__HAL_RCC_PPP_RELEASE_RESET to force/release peripheral reset
- __HAL_RCC_PPP_CLK_SLEEP_ENABLE/__HAL_RCC_PPP_CLK_SLEEP_DISABLE to enable/disable the peripheral clock during Sleep mode

4.11.2 GPIOs

GPIO HAL APIs are the following:

- HAL_GPIO_Init() / HAL_GPIO_DeInit()
- HAL_GPIO_ReadPin() / HAL_GPIO_WritePin()

- HAL_GPIO_TogglePin ().

In addition to standard GPIO modes (input, output, analog), the pin mode can be configured as EXTI with interrupt or event generation.

When selecting EXTI mode with interrupt generation, the user must call HAL_GPIO_EXTI_IRQHandler() from stm32h7xx_it.c and implement HAL_GPIO_EXTI_Callback()

The table below describes the GPIO_InitTypeDef structure field.

Table 12. Description of GPIO_InitTypeDef structure

Structure field	Description
Pin	Specifies the GPIO pins to be configured. Possible values: GPIO_PIN_x or GPIO_PIN_All, where x[0..15]
Mode	Specifies the operating mode for the selected pins: GPIO mode or EXTI mode. Possible values are: • GPIO mode – GPIO_MODE_INPUT : Input floating – GPIO_MODE_OUTPUT_PP : Output push-pull – GPIO_MODE_OUTPUT_OD : Output open drain – GPIO_MODE_AF_PP : Alternate function push-pull – GPIO_MODE_AF_OD : Alternate function open drain – GPIO_MODE_ANALOG : Analog mode • External Interrupt mode – GPIO_MODE_IT_RISING : Rising edge trigger detection – GPIO_MODE_IT_FALLING : Falling edge trigger detection – GPIO_MODE_IT_RISING_FALLING : Rising/Falling edge trigger detection • External Event mode – GPIO_MODE_EVT_RISING : Rising edge trigger detection – GPIO_MODE_EVT_FALLING : Falling edge trigger detection – GPIO_MODE_EVT_RISING_FALLING: Rising/Falling edge trigger detection
Pull	Specifies the Pull-up or Pull-down activation for the selected pins. Possible values are: GPIO_NOPULL GPIO_PULLUP GPIO_PULLDOWN
Speed	Specifies the speed for the selected pins Possible values are: GPIO_SPEED_FREQ_LOW GPIO_SPEED_FREQ_MEDIUM GPIO_SPEED_FREQ_HIGH GPIO_SPEED_FREQ_VERY_HIGH

Please find below typical GPIO configuration examples:

- Configuring GPIOs as output push-pull to drive external LEDs:

```
GPIO_InitStruct.Pin = GPIO_PIN_12 | GPIO_PIN_13 | GPIO_PIN_14 | GPIO_PIN_15;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_PULLUP;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_MEDIUM;
HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
```

- Configuring PA0 as external interrupt with falling edge sensitivity:

```
GPIO_InitStructure.Mode = GPIO_MODE_IT_FALLING;
GPIO_InitStructure.Pull = GPIO_NOPULL;
GPIO_InitStructure.Pin = GPIO_PIN_0;
HAL_GPIO_Init(GPIOA, &GPIO_InitStructure);
```

4.11.3 Cortex® NVIC and SysTick timer

The Cortex® HAL driver, stm32h7xx_hal_cortex.c, provides APIs to handle NVIC and SysTick. The supported APIs include:

- HAL_NVIC_SetPriority()/ HAL_NVIC_SetPriorityGrouping()
- HAL_NVIC_GetPriority() / HAL_NVIC_GetPriorityGrouping()
- HAL_NVIC_EnableIRQ()/HAL_NVIC_DisableIRQ()
- HAL_NVIC_SystemReset()
- HAL_SYSTICK_IRQHandler()
- HAL_NVIC_GetPendingIRQ() / HAL_NVIC_SetPendingIRQ () / HAL_NVIC_ClearPendingIRQ()
- HAL_NVIC_GetActive(IRQn)
- HAL_SYSTICK_Config()
- HAL_SYSTICK_CLKSourceConfig()
- HAL_SYSTICK_Callback()

4.11.4 PWR

The PWR HAL driver handles power management. The features shared between all STM32 Series are listed below:

- PVD configuration, enabling/disabling and interrupt handling
  - HAL_PWR_ConfigPVD()
  - HAL_PWR_EnablePVD() / HAL_PWR_DisablePVD()
  - HAL_PWR_PVD_IRQHandler()
  - HAL_PWR_PVDCallback()
- Wakeup pin configuration
  - HAL_PWR_EnableWakeUpPin() / HAL_PWR_DisableWakeUpPin()
- Low-power mode entry
  - HAL_PWR_EnterSLEEPMode()
  - HAL_PWR_EnterSTOPMode()
  - HAL_PWR_EnterSTANDBYMode()

4.11.5 EXTI

The EXTI is not considered as a standalone peripheral but rather as a service used by other peripheral, that are handled through EXTI HAL APIs. In addition, each peripheral HAL driver implements the associated EXTI configuration and function as macros in its header file.

The first 16 EXTI lines connected to the GPIOs, are managed within the GPIO driver. The GPIO_InitTypeDef structure allows configuring an I/O as external interrupt or external event.

The EXTI lines connected internally to the PVD, RTC, USB, and Ethernet, are configured within the HAL drivers of these peripheral through the macros given in the table below.

The EXTI internal connections depend on the targeted STM32 microcontroller (refer to the product datasheet for more details):

Table 13. Description of EXTI configuration macros

Macros	Description
__HAL_PPP_{SUBLOCK}__EXTI_ENABLE_IT()	Enables a given EXTI line interrupt Example:

Macros	Description
	__HAL_PWR_PVD_EXTI_ENABLE_IT()
__HAL_PPP_{SUBLOCK}__EXTI_DISABLE_IT()	Disables a given EXTI line. Example: __HAL_PWR_PVD_EXTI_DISABLE_IT()
__HAL_ PPP_{SUBLOCK}__EXTI_GET_FLAG()	Gets a given EXTI line interrupt flag pending bit status. Example: __HAL_PWR_PVD_EXTI_GET_FLAG()
__HAL_ PPP_{SUBLOCK}_EXTI_CLEAR_FLAG()	Clears a given EXTI line interrupt flag pending bit. Example; __HAL_PWR_PVD_EXTI_CLEAR_FLAG()
__HAL_ PPP_{SUBLOCK}_EXTI_GENERATE_SWIT()	Generates a software interrupt for a given EXTI line. Example: __HAL_PWR_PVD_EXTI_ GENERATE_SWIT ()
__HAL_PPP_SUBBLOCK_EXTI_ENABLE_EVENT()	Enable a given EXTI line event Example: __HAL_RTC_WAKEUP_EXTI_ENABLE_EVENT()
__HAL_PPP_SUBBLOCK_EXTI_DISABLE_EVENT()	Disable a given EXTI line event Example: __HAL_RTC_WAKEUP_EXTI_DISABLE_EVENT()
__HAL_ PPP_SUBBLOCK_EXTI_ENABLE_RISING_EDGE()	Configure an EXTI Interrupt or Event on rising edge
__HAL_ PPP_SUBBLOCK_EXTI_ENABLE_FALLING_EDGE()	Enable an EXTI Interrupt or Event on Falling edge
__HAL_ PPP_SUBBLOCK_EXTI_DISABLE_RISING_EDGE()	Disable an EXTI Interrupt or Event on rising edge
__HAL_ PPP_SUBBLOCK_EXTI_DISABLE_FALLING_EDGE()	Disable an EXTI Interrupt or Event on Falling edge
__HAL_ PPP_SUBBLOCK_EXTI_ENABLE_RISING_FALLING_EDGE()	Enable an EXTI Interrupt or Event on Rising/Falling edge
__HAL_ PPP_SUBBLOCK_EXTI_DISABLE_RISING_FALLING_EDGE()	Disable an EXTI Interrupt or Event on Rising/Falling edge

If the EXTI interrupt mode is selected, the user application must call HAL_PPP_FUNCTION_IRQHandler() (for example HAL_PWR_PVD_IRQHandler()), from stm32h7xx_it.c file, and implement HAL_PPP_FUNCTIONCallback() callback function (for example HAL_PWR_PVDCallback().

4.11.6 DMA

The DMA HAL driver allows enabling and configuring the peripheral to be connected to the DMA Channels (except for internal SRAM/FLASH memory which do not require any initialization). Refer to the product reference manual for details on the DMA request corresponding to each peripheral.

For a given channel, HAL_DMA_Init() API allows programming the required configuration through the following parameters:

- Transfer direction
- Source and destination data formats
- Circular, Normal control mode
- Channel priority level
- Source and destination Increment mode

Two operating modes are available:

- Polling mode I/O operation
  1. Use HAL_DMA_Start() to start DMA transfer when the source and destination addresses and the Length of data to be transferred have been configured.
  2. Use HAL_DMA_PollForTransfer() to poll for the end of current transfer. In this case a fixed timeout can be configured depending on the user application.

- Interrupt mode I/O operation
  1. Configure the DMA interrupt priority using HAL_NVIC_SetPriority().
  2. Enable the DMA IRQ handler using HAL_NVIC_EnableIRQ().
  3. Use HAL_DMA_Start_IT() to start DMA transfer when the source and destination addresses and the length of data to be transferred have been configured. In this case the DMA interrupt is configured.
  4. Use HAL_DMA_IRQHandler() called under DMA_IRQHandler() Interrupt subroutine.
  5. When data transfer is complete, HAL_DMA_IRQHandler() function is executed and a user function can be called by customizing XferCpltCallback and XferErrorCallback function pointer (that is a member of DMA handle structure).

Additional functions and macros are available to ensure efficient DMA management:

- Use HAL_DMA_GetState() function to return the DMA state and HAL_DMA_GetError() in case of error detection.
- Use HAL_DMA_Abort() function to abort the current transfer.

The most used DMA HAL driver macros are the following:

- __HAL_DMA_ENABLE: enables the specified DMA channel
- __HAL_DMA_DISABLE: disables the specified DMA channel
- __HAL_DMA_GET_FLAG: gets the DMA channel pending flags
- __HAL_DMA_CLEAR_FLAG: clears the DMA channel pending flags
- __HAL_DMA_ENABLE_IT: enables the specified DMA channel interrupts
- __HAL_DMA_DISABLE_IT: disables the specified DMA channel interrupts
- __HAL_DMA_GET_IT_SOURCE: checks whether the specified DMA channel interrupt has been enabled or not

Note: When a peripheral is used in DMA mode, the DMA initialization must be done in the HAL_PPP_MspInit() callback. In addition, the user application must associate the DMA handle to the PPP handle (refer to section "HAL IO operation functions").

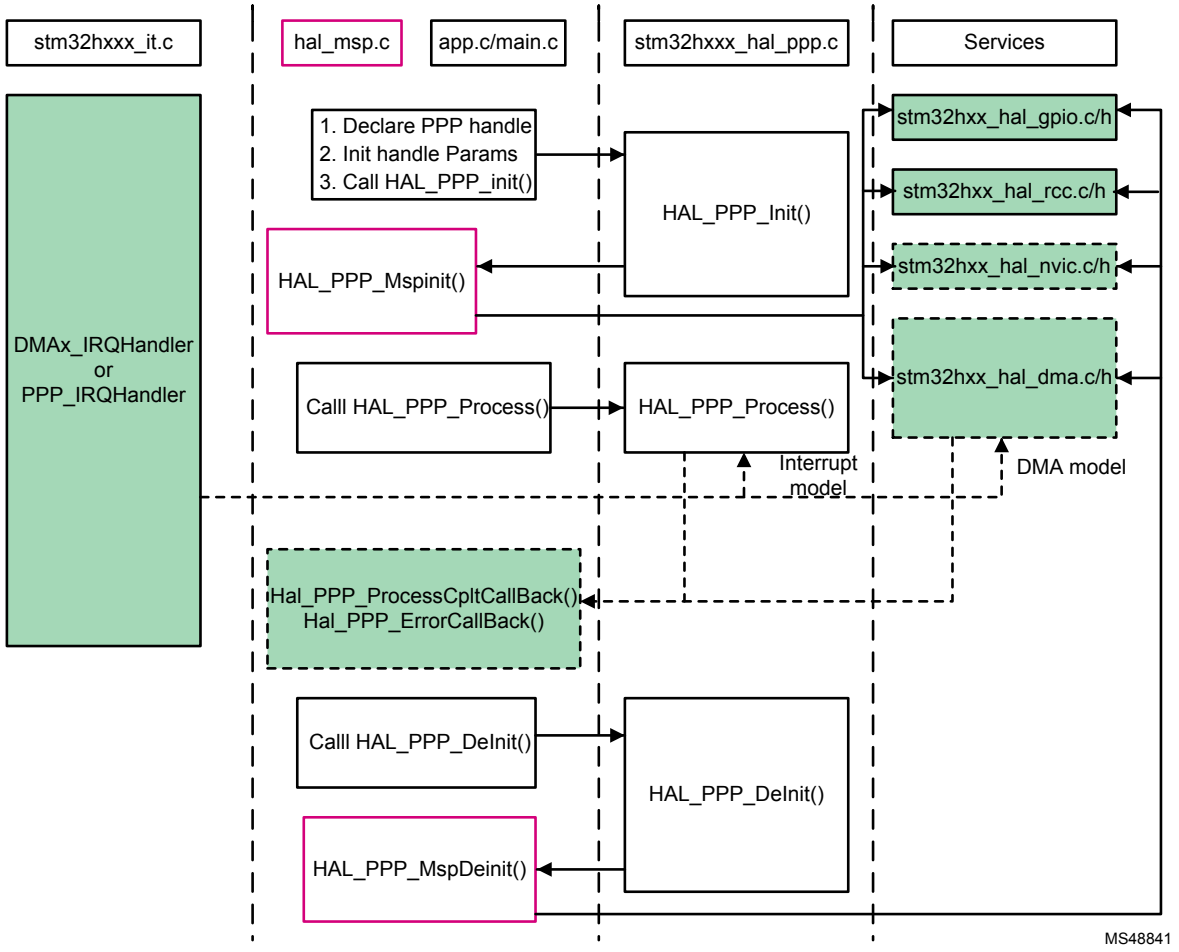
Note: DMA channel callbacks need to be initialized by the user application only in case of memory-to-memory transfer. However when peripheral-to-memory transfers are used, these callbacks are automatically initialized by calling a process API function that uses the DMA.

4.12 How to use HAL drivers

4.12.1 HAL usage models

The following figure shows the typical use of the HAL driver and the interaction between the application user, the HAL driver and the interrupts.

Figure 9. **HAL driver model**



Note: The functions implemented in the HAL driver are shown in green, the functions called from interrupt handlers in dotted lines, and the msp functions implemented in the user application in red. Non-dotted lines represent the interactions between the user application functions.

Basically, the HAL driver APIs are called from user files and optionally from interrupt handlers file when the APIs based on the DMA or the PPP peripheral dedicated interrupts are used.

When DMA or PPP peripheral interrupts are used, the PPP process complete callbacks are called to inform the user about the process completion in real-time event mode (interrupts). Note that the same process completion callbacks are used for DMA in interrupt mode.

4.12.2 HAL initialization

4.12.2.1 HAL global initialization

In addition to the peripheral initialization and de-initialization functions, a set of APIs are provided to initialize the HAL core implemented in file stm32h7xx_hal.c.

- HAL_Init(): this function must be called at application startup to
  - initialize data/instruction cache and pre-fetch queue
  - set SysTick timer to generate an interrupt each 1ms (based on HSI clock) with the lowest priority
  - call HAL_MspInit() user callback function to perform system level initializations (Clock, GPIOs, DMA, interrupts). HAL_MspInit() is defined as “weak” empty function in the HAL drivers.
- HAL_DeInit()
  - resets all peripherals
  - calls function HAL_MspDeInit() which is a user callback function to do system level De-Initalizations.

- HAL_GetTick(): this function gets current SysTick counter value (incremented in SysTick interrupt) used by peripherals drivers to handle timeouts.
- HAL_Delay(). this function implements a delay (expressed in milliseconds) using the SysTick timer.

  Care must be taken when using HAL_Delay() since this function provides an accurate delay (expressed in milliseconds) based on a variable incremented in SysTick ISR. This means that if HAL_Delay() is called from a peripheral ISR, then the SysTick interrupt must have highest priority (numerically lower) than the peripheral interrupt, otherwise the caller ISR is blocked.

4.12.2.2 *System clock initialization*

The clock configuration is done at the beginning of the user code. However the user can change the configuration of the clock in his own code.

Below the typical clock configuration sequence.

```
static void SystemClock_Config(void)
{
  RCC_ClkInitTypeDef RCC_ClkInitStruct;
  RCC_OscInitTypeDef RCC_OscInitStruct;
  HAL_StatusTypeDef ret = HAL_OK;

  /*!< Supply configuration update enable */
  HAL_PWREx_ConfigSupply(PWR_LDO_SUPPLY);

  /* The voltage scaling allows optimizing the power consumption when the device is
     clocked below the maximum system frequency, to update the voltage scaling value
     regarding system frequency refer to product datasheet.  */
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  while(!__HAL_PWR_GET_FLAG(PWR_FLAG_VOSRDY)) {}

  /* Enable HSE Oscillator and activate PLL with HSE as source */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.HSIState = RCC_HSI_OFF;
  RCC_OscInitStruct.CSIState = RCC_CSI_OFF;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;

  RCC_OscInitStruct.PLL.PLLM = 5;
  RCC_OscInitStruct.PLL.PLLN = 160;
  RCC_OscInitStruct.PLL.PLLFRACN = 0;
  RCC_OscInitStruct.PLL.PLLP = 2;
  RCC_OscInitStruct.PLL.PLLR = 2;
  RCC_OscInitStruct.PLL.PLLQ = 4;

  RCC_OscInitStruct.PLL.PLLVCOSEL = RCC_PLL1VCOWIDE;
  RCC_OscInitStruct.PLL.PLLRGE = RCC_PLL1VCIRANGE_2;
  ret = HAL_RCC_OscConfig(&RCC_OscInitStruct);
  if(ret != HAL_OK)
  {
    Error_Handler();
  }

/* Select PLL as system clock source and configure  bus clock dividers */
  RCC_ClkInitStruct.ClockType = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_HCLK |
RCC_CLOCKTYPE_D1PCLK1 | RCC_CLOCKTYPE_PCLK1 | \
                                 RCC_CLOCKTYPE_PCLK2  | RCC_CLOCKTYPE_D3PCLK1);

  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.SYSCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB3CLKDivider = RCC_APB3_DIV2;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_APB1_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_APB2_DIV2;
  RCC_ClkInitStruct.APB4CLKDivider = RCC_APB4_DIV2;
  ret = HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_4);
  if(ret != HAL_OK)
  {
    Error_Handler();
  }

  /*
  Note : The activation of the I/O Compensation Cell is recommended with communication
interfaces
          (GPIO, SPI, FMC, QSPI ...)  when  operating at  high frequencies(please refer to
product datasheet)
          The I/O Compensation Cell activation  procedure requires :
        - The activation of the CSI clock
        - The activation of the SYSCFG clock
        - Enabling the I/O Compensation Cell : setting bit[0] of register SYSCFG_CCCSR

          To do this please uncomment the following code
*/
```

```
  /*
  __HAL_RCC_CSI_ENABLE() ;

  __HAL_RCC_SYSCFG_CLK_ENABLE() ;

  HAL_EnableCompensationCell();
  */
}
```

Caution: **Consideration for firmware power configuration versus hardware board configuration**

STM32H7 boards are configured either in direct SMPS or in LDO power regulator mode.

Therefore the firmware running on a given board must match its hardware configuration, otherwise a deadlock might occur since ST-LINK will not be able to connect the target after reset. Refer to the corresponding board user manual to check which power configuration is implemented.

The firmware power configuration is performed in SystemClock_Config through the following lines:

- In case of “Direct SMPS” hardware configuration:
  HAL_PWREx_ConfigSupply(PWR_DIRECT_SMPS_SUPPLY);
- In case of “LDO” hardware configuration:
  HAL_PWREx_ConfigSupply(PWR_LDO_SUPPLY);

4.12.2.3 *HAL MSP initialization process*

The peripheral initialization is done through *HAL_PPP_Init()* while the hardware resources initialization used by a peripheral (PPP) is performed during this initialization by calling MSP callback function *HAL_PPP_MspInit().*

The MspInit callback performs the low level initialization related to the different additional hardware resources: RCC, GPIO, NVIC and DMA.

All the HAL drivers with handles include two MSP callbacks for initialization and de-initialization:

```
/**
* @brief Initializes the PPP MSP.
* @param hppp: PPP handle
* @retval None */
void __weak HAL_PPP_MspInit(PPP_HandleTypeDef *hppp) {
/* NOTE : This function Should not be modified, when the callback is needed,
the HAL_PPP_MspInit could be implemented in the user file */
}
/**
* @brief DeInitializes PPP MSP.
* @param hppp: PPP handle
* @retval None */
void __weak HAL_PPP_MspDeInit(PPP_HandleTypeDef *hppp) {
/* NOTE : This function Should not be modified, when the callback is needed,
the HAL_PPP_MspDeInit could be implemented in the user file */
}
```

The MSP callbacks are declared empty as weak functions in each peripheral driver. The user can use them to set the low level initialization code or omit them and use his own initialization routine.

The HAL MSP callback is implemented inside the *stm32h7xx_hal_msp.c* file in the user folders. An *stm32h7xx_hal_msp.c* file template is located in the HAL folder and should be copied to the user folder. It can be generated automatically by STM32CubeMX tool and further modified. Note that all the routines are declared as weak functions and could be overwritten or removed to use user low level initialization code.

stm32h7xx_hal_msp.c file contains the following functions:

Table 14. MSP functions

Routine	Description
void HAL_MspInit()	Global MSP initialization routine
void HAL_MspDeInit()	Global MSP de-initialization routine
void HAL_PPP_MspInit()	PPP MSP initialization routine

Routine	Description
void HAL_PPP_MspDeInit()	PPP MSP de-initialization routine

By default, if no peripheral needs to be de-initialized during the program execution, the whole MSP initialization is done in *Hal_MspInit()* and MSP De-Initialization in the *Hal_MspDeInit()*. In this case the *HAL_PPP_MspInit()* and *HAL_PPP_MspDeInit()* are not implemented.

When one or more peripherals needs to be de-initialized in run time and the low level resources of a given peripheral need to be released and used by another peripheral, HAL_PPP_MspDeInit() and *HAL_PPP_MspInit()* are implemented for the concerned peripheral and other peripherals initialization and de-Initialization are kept in the global *HAL_MspInit()* and the *HAL_MspDeInit()*.

If there is nothing to be initialized by the global *HAL_MspInit()* and *HAL_MspDeInit()*, the two routines can simply be omitted.

4.12.3 HAL I/O operation process

The HAL functions with internal data processing like transmit, receive, write and read are generally provided with three data processing modes as follows:

- Polling mode
- Interrupt mode
- DMA mode

4.12.3.1 Polling mode

In Polling mode, the HAL functions return the process status when the data processing in blocking mode is complete. The operation is considered complete when the function returns the HAL_OK status, otherwise an error status is returned. The user can get more information through the *HAL_PPP_GetState()* function. The data processing is handled internally in a loop. A timeout (expressed in ms) is used to prevent process hanging.

The example below shows the typical Polling mode processing sequence :

```
HAL_StatusTypeDef HAL_PPP_Transmit ( PPP_HandleTypeDef * phandle, uint8_t pData,
int16_tSize,uint32_tTimeout)
{
if((pData == NULL ) || (Size == 0))
{
return HAL_ERROR;
}
(…) while (data processing is running)
{
if( timeout reached )
{
return HAL_TIMEOUT;
}
}
(…)
return HAL_OK; }
```

4.12.3.2 Interrupt mode

In Interrupt mode, the HAL function returns the process status after starting the data processing and enabling the appropriate interruption. The end of the operation is indicated by a callback declared as a weak function. It can be customized by the user to be informed in real-time about the process completion. The user can also get the process status through the *HAL_PPP_GetState()* function.

In Interrupt mode, four functions are declared in the driver:

- *HAL_PPP_Process_IT()*: launches the process
- *HAL_PPP_IRQHandler()*: global PPP peripheral interruption
- *__weak HAL_PPP_ProcessCpltCallback ()*: callback relative to the process completion.
- *__weak HAL_PPP_ProcessErrorCallback()*: callback relative to the process Error.

To use a process in Interrupt mode, *HAL_PPP_Process_IT()* is called in the user file and HAL_PPP_IRQHandler in *stm32h7xx_it.c.*

The *HAL_PPP_ProcessCpltCallback()* function is declared as weak function in the driver. This means that the user can declare it again in the application. The function in the driver is not modified.

An example of use is illustrated below:

main.c file:

```
UART_HandleTypeDef UartHandle;
int main(void)
{
/* Set User Parameters */
UartHandle.Init.BaudRate = 9600;
UartHandle.Init.WordLength = UART_DATABITS_8;
UartHandle.Init.StopBits = UART_STOPBITS_1;
UartHandle.Init.Parity = UART_PARITY_NONE;
UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
UartHandle.Init.Mode = UART_MODE_TX_RX;
UartHandle.Init.Instance = USART1;
HAL_UART_Init(&UartHandle);
HAL_UART_SendIT(&UartHandle, TxBuffer, sizeof(TxBuffer));
while (1);
}
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
{
}
void HAL_UART_ErrorCallback(UART_HandleTypeDef *huart)
{
}
```

*stm32h7xx_it.c*file:

```
extern UART_HandleTypeDef UartHandle;
void USART1_IRQHandler(void)
{
HAL_UART_IRQHandler(&UartHandle);
}
```

4.12.3.3 DMA mode

In DMA mode, the HAL function returns the process status after starting the data processing through the DMA and after enabling the appropriate DMA interruption. The end of the operation is indicated by a callback declared as a weak function and can be customized by the user to be informed in real-time about the process completion. The user can also get the process status through the *HAL_PPP_GetState()* function. For the DMA mode, three functions are declared in the driver:

- *HAL_PPP_Process_DMA()*: launch the process
- *HAL_PPP_DMA_IRQHandler()*: the DMA interruption used by the PPP peripheral
- *__weak HAL_PPP_ProcessCpltCallback()*: the callback relative to the process completion.
- *__weak HAL_PPP_ErrorCpltCallback()*: the callback relative to the process Error.

To use a process in DMA mode, *HAL_PPP_Process_DMA()* is called in the user file and the *HAL_PPP_DMA_IRQHandler()* is placed in the *stm32h7xx_it.c*. When DMA mode is used, the DMA initialization is done in the *HAL_PPP_MspInit()* callback. The user should also associate the DMA handle to the PPP handle. For this purpose, the handles of all the peripheral drivers that use the DMA must be declared as follows:

```
typedef struct
{
PPP_TypeDef *Instance; /* Register base address */
PPP_InitTypeDef Init; /* PPP communication parameters */
HAL_StateTypeDef State; /* PPP communication state */
(…)
DMA_HandleTypeDef *hdma; /* associated DMA handle */
} PPP_HandleTypeDef;
```

The initialization is done as follows (UART example):

```
int main(void)
{
/* Set User Parameters */
UartHandle.Init.BaudRate = 9600;
UartHandle.Init.WordLength = UART_DATABITS_8;
UartHandle.Init.StopBits = UART_STOPBITS_1;
UartHandle.Init.Parity = UART_PARITY_NONE;
UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
UartHandle.Init.Mode = UART_MODE_TX_RX;
UartHandle.Init.Instance = UART1;
HAL_UART_Init(&UartHandle);
(..)
}
void HAL_USART_MspInit (UART_HandleTypeDef * huart)
{
static DMA_HandleTypeDef hdma_tx;
static DMA_HandleTypeDef hdma_rx;
(…)
__HAL_LINKDMA(UartHandle, DMA_Handle_tx, hdma_tx);
__HAL_LINKDMA(UartHandle, DMA_Handle_rx, hdma_rx);
(…)
}
```

The *HAL_PPP_ProcessCpltCallback()* function is declared as weak function in the driver that means, the user can declare it again in the application code. The function in the driver should not be modified.

An example of use is illustrated below:

main.c file:

```
UART_HandleTypeDef UartHandle;
int main(void)
{
/* Set User Paramaters */
UartHandle.Init.BaudRate = 9600;
UartHandle.Init.WordLength = UART_DATABITS_8;
UartHandle.Init.StopBits = UART_STOPBITS_1;
UartHandle.Init.Parity = UART_PARITY_NONE;
UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
UartHandle.Init.Mode = UART_MODE_TX_RX; UartHandle.Init.Instance = USART1;
HAL_UART_Init(&UartHandle);
HAL_UART_Send_DMA(&UartHandle, TxBuffer, sizeof(TxBuffer));
while (1);
}
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *phuart)
{
}
void HAL_UART_TxErrorCallback(UART_HandleTypeDef *phuart)
{
}
```

stm32h7xx_it.c file:

```
extern UART_HandleTypeDef UartHandle;
void DMAx_IRQHandler(void)
{
HAL_DMA_IRQHandler(&UartHandle.DMA_Handle_tx);
}
```

HAL_USART_TxCpltCallback() and *HAL_USART_ErrorCallback()* should be linked in the *HAL_PPP_Process_DMA()* function to the DMA transfer complete callback and the DMA transfer Error callback by using the following statement:

```
HAL_PPP_Process_DMA (PPP_HandleTypeDef *hppp, Params….)
{
(…)
hppp->DMA_Handle->XferCpltCallback = HAL_UART_TxCpltCallback ;
hppp->DMA_Handle->XferErrorCallback = HAL_UART_ErrorCallback ;
(…)
}
```

4.12.4 Timeout and error management

4.12.4.1 *Timeout management*

The timeout is often used for the APIs that operate in Polling mode. It defines the delay during which a blocking process should wait till an error is returned. An example is provided below:

```
HAL_StatusTypeDef HAL_DMA_PollForTransfer(DMA_HandleTypeDef *hdma, uint32_t CompleteLevel,
uint32_t Timeout)
```

The timeout possible values are the following:

Table 15. Timeout values

Timeout value	Description
0	No poll : Immediate process check and exit
1 ... (HAL_MAX_DELAY -1) (1)	Timeout in ms
HAL_MAX_DELAY	Infinite poll till process is successful

1. *HAL_MAX_DELAY is defined in the stm32h7xx_hal_def.h as 0xFFFFFFFF*

However, in some cases, a fixed timeout is used for system peripherals or internal HAL driver processes. In these cases, the timeout has the same meaning and is used in the same way, except when it is defined locally in the drivers and cannot be modified or introduced as an argument in the user application.

Example of fixed timeout:

```
#define LOCAL_PROCESS_TIMEOUT 100
HAL_StatusTypeDef HAL_PPP_Process(PPP_HandleTypeDef)
{
(…)
timeout = HAL_GetTick() + LOCAL_PROCESS_TIMEOUT;
(…)
while(ProcessOngoing)
{
(…)
if(HAL_GetTick() ≥ timeout)
{
/* Process unlocked */
__HAL_UNLOCK(hppp);
hppp->State= HAL_PPP_STATE_TIMEOUT;
return HAL_PPP_STATE_TIMEOUT;
}
}
(…)
}
```

The following example shows how to use the timeout inside the polling functions:

```
HAL_PPP_StateTypeDef HAL_PPP_Poll (PPP_HandleTypeDef *hppp, uint32_t Timeout)
{
(…)
timeout = HAL_GetTick() + Timeout;
(…)
while(ProcessOngoing)
{
(…)
if(Timeout != HAL_MAX_DELAY)
{
if(HAL_GetTick() ≥ timeout)
{
/* Process unlocked */
__HAL_UNLOCK(hppp);
hppp->State= HAL_PPP_STATE_TIMEOUT;
return hppp->State;
}
}
(…)
}
```

4.12.4.2 *Error management*

The HAL drivers implement a check on the following items:

- Valid parameters: for some process the used parameters should be valid and already defined, otherwise the system may crash or go into an undefined state. These critical parameters are checked before being used (see example below).

```
HAL_StatusTypeDef HAL_PPP_Process(PPP_HandleTypeDef* hppp, uint32_t *pdata, uint32
Size)
{
if ((pData == NULL ) || (Size == 0))
{
return HAL_ERROR;
}
}
```

- Valid handle: the PPP peripheral handle is the most important argument since it keeps the PPP driver vital parameters. It is always checked in the beginning of the *HAL_PPP_Init()* function.

```
HAL_StatusTypeDef HAL_PPP_Init(PPP_HandleTypeDef* hppp)
{
if (hppp == NULL) //the handle should be already allocated
{
return HAL_ERROR;
}
}
```

- Timeout error: the following statement is used when a timeout error occurs:

```
while (Process ongoing)
{
timeout = HAL_GetTick() + Timeout; while (data processing is running)
{
if(timeout) { return HAL_TIMEOUT;
}
}
```

When an error occurs during a peripheral process, *HAL_PPP_Process ()* returns with a *HAL_ERROR* status. The HAL PPP driver implements the *HAL_PPP_GetError ()* to allow retrieving the origin of the error.

```
HAL_PPP_ErrorTypeDef HAL_PPP_GetError (PPP_HandleTypeDef *hppp);
```

In all peripheral handles, a *HAL_PPP_ErrorTypeDef* is defined and used to store the last error code.

```
typedef struct
{
PPP_TypeDef * Instance; /* PPP registers base address */
PPP_InitTypeDef Init; /* PPP initialization parameters */
HAL_LockTypeDef Lock; /* PPP locking object */
__IO HAL_PPP_StateTypeDef State; /* PPP state */
__IO HAL_PPP_ErrorTypeDef ErrorCode; /* PPP Error code */
(…)
/* PPP specific parameters */
}
PPP_HandleTypeDef;
```

The error state and the peripheral global state are always updated before returning an error:

```
PPP->State = HAL_PPP_READY;  /* Set the peripheral ready */
PP->ErrorCode = HAL_ERRORCODE ; /* Set the error code */
_HAL_UNLOCK(PPP) ; /* Unlock the PPP resources */
return HAL_ERROR;  /*return with HAL error */
```

HAL_PPP_GetError () must be used in interrupt mode in the error callback:

```
void HAL_PPP_ProcessCpltCallback(PPP_HandleTypeDef *hspi)
{
ErrorCode = HAL_PPP_GetError (hppp); /* retreive error code */
}
```

4.12.4.3 *Run-time checking*

The HAL implements run-time failure detection by checking the input values of all HAL driver functions. The run-time checking is achieved by using an *assert_param* macro. This macro is used in all the HAL driver functions which have an input parameter. It allows verifying that the input value lies within the parameter allowed values.

To enable the run-time checking, use the *assert_param* macro, and leave the define ***USE_FULL_ASSERT*** uncommented in *stm32h7xx_hal_conf.h* file.

```
void HAL_UART_Init(UART_HandleTypeDef *huart)
{
(..) /* Check the parameters */
assert_param(IS_UART_INSTANCE(huart->Instance));
assert_param(IS_UART_BAUDRATE(huart->Init.BaudRate));
assert_param(IS_UART_WORD_LENGTH(huart->Init.WordLength));
assert_param(IS_UART_STOPBITS(huart->Init.StopBits));
assert_param(IS_UART_PARITY(huart->Init.Parity));
assert_param(IS_UART_MODE(huart->Init.Mode));
assert_param(IS_UART_HARDWARE_FLOW_CONTROL(huart->Init.HwFlowCtl));
(..)
```

```
/** @defgroup UART_Word_Length *
@{
*/
#define UART_WORDLENGTH_8B ((uint32_t)0x00000000)
#define UART_WORDLENGTH_9B ((uint32_t)USART_CR1_M)
#define IS_UART_WORD_LENGTH(LENGTH) (((LENGTH) == UART_WORDLENGTH_8B) ||
\ ((LENGTH) == UART_WORDLENGTH_9B))
```

If the expression passed to the assert_param macro is false, the*assert_failed* function is called and returns the name of the source file and the source line number of the call that failed. If the expression is true, no value is returned.

The *assert_param* macro is implemented in stm32h7xx_hal_conf.h:

```
/* Exported macro ------------------------------------------------------------*/
#ifdef USE_FULL_ASSERT
/**
* @brief The assert_param macro is used for function's parameters check.
* @param expr: If expr is false, it calls assert_failed function
* which reports the name of the source file and the source
* line number of the call that failed.
* If expr is true, it returns no value.
* @retval None */
#define assert_param(expr) ((expr)?(void)0:assert_failed((uint8_t *)__FILE__, __LINE__))
/* Exported functions --------------------------------------*/
void assert_failed(uint8_t* file, uint32_t line);
#else
#define assert_param(expr)((void)0)
#endif /* USE_FULL_ASSERT */
```

The *assert_failed* function is implemented in the main.c file or in any other user C file:

```
#ifdef USE_FULL_ASSERT /**
* @brief Reports the name of the source file and the source line number
* where the assert_param error has occurred.
* @param file: pointer to the source file name
* @param line: assert_param error line source number
* @retval None */
void assert_failed(uint8_t* file, uint32_t line)
{
/* User can add his own implementation to report the file name and line number,
ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
/* Infinite loop */
while (1)
{
}
}
```

Attention: ***Because of the overhead run-time checking introduces, it is recommended to use it during application code development and debugging, and to remove it from the final application to improve code size and speed.***

5 Overview of low-layer drivers

The low-layer (LL) drivers are designed to offer a fast light-weight expert-oriented layer which is closer to the hardware than the HAL. Contrary to the HAL, LL APIs are not provided for peripherals where optimized access is not a key feature, or those requiring heavy software configuration and/or complex upper-level stack (such as USB).

The LL drivers feature:

- A set of functions to initialize peripheral main features according to the parameters specified in data structures
- A set of functions used to fill initialization data structures with the reset values of each field
- Functions to perform peripheral de-initialization (peripheral registers restored to their default values)
- A set of inline functions for direct and atomic register access
- Full independence from HAL since LL drivers can be used either in standalone mode (without HAL drivers) or in mixed mode (with HAL drivers)
- Full coverage of the supported peripheral features

The low-layer drivers provide hardware services based on the available features of the STM32 peripherals. These services reflect exactly the hardware capabilities and provide one-shot operations that must be called following the programming model described in the microcontroller line reference manual. As a result, the LL services do not implement any processing and do not require any additional memory resources to save their states, counter or data pointers: all the operations are performed by changing the associated peripheral registers content.

5.1 Low-layer files

The low-layer drivers are built around header/C files (one per each supported peripheral) plus five header files for some System and Cortex related features.

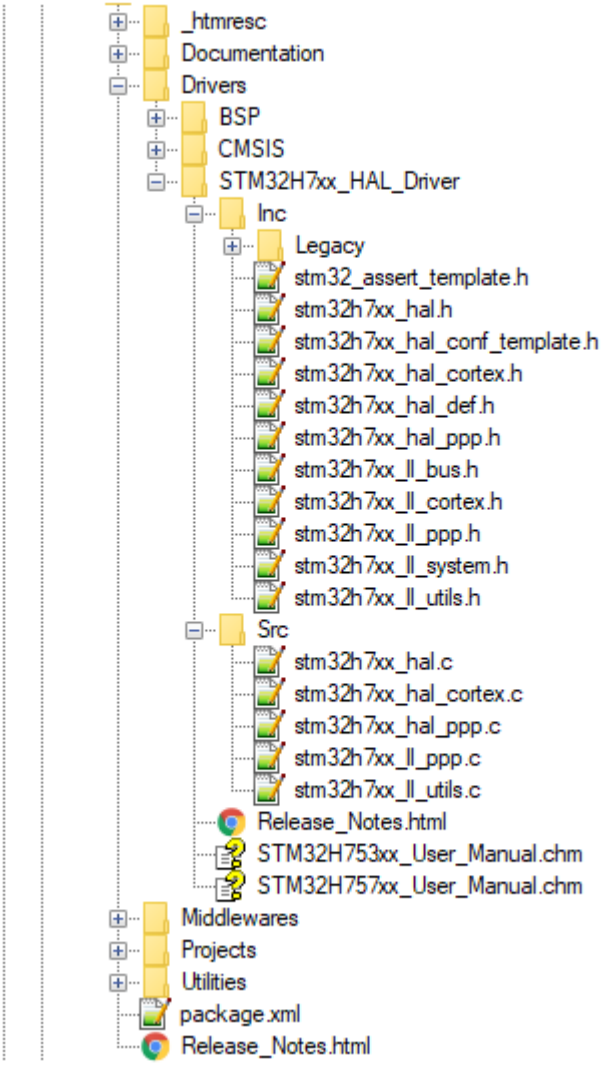
Table 16. LL driver files

File	Description
stm32h7xx_ll_bus.h	This is the h-source file for core bus control and peripheral clock activation and deactivation *Example: LL_AHB2_GRP1_EnableClock*
stm32h7xx_ll_ppp.h/.c	stm32h7xx_ll_ppp.c provides peripheral initialization functions such as LL_PPP_Init(), LL_PPP_StructInit(), LL_PPP_DeInit(). All the other APIs are definined within stm32h7xx_ll_ppp.h file. The low-layer PPP driver is a standalone module. To use it, the application must include it in the stm32h7xx_ll_ppp.h file.
stm32h7xx_ll_cortex.h	Cortex-M related register operation APIs including the Systick, Low power (such as LL_SYSTICK_xxxxx and LL_LPM_xxxxx "Low Power Mode")
stm32h7xx_ll_utils.h/.c	This file covers the generic APIs: • Read of device unique ID and electronic signature • Timebase and delay management • System clock configuration.
stm32h7xx_ll_system.h	System related operations. Example: LL_SYSCFG_xxx, LL_DBGMCU_xxx and LL_FLASH_xxx and LL_VREFBUF_xxx
stm32_assert_template.h	Template file allowing to define the assert_param macro, that is used when run-time checking is enabled. This file is required only when the LL drivers are used in standalone mode (without calling the HAL APIs). It should be copied to the application folder and renamed to stm32_assert.h.

Note: There is no configuration file for the LL drivers.

The low-layer files are located in the same HAL driver folder.

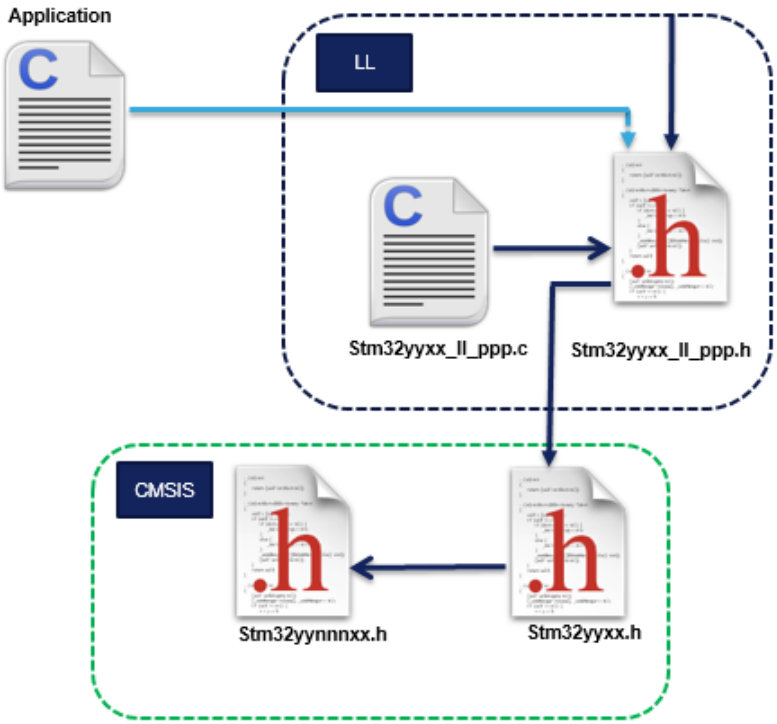
Figure 10. **Low-layer driver folders**



In general, low-layer drivers include only the STM32 CMSIS device file.

```
#include "stm32yyxx.h"
```

Figure 11. **Low-layer driver CMSIS files**



Application files have to include only the used low-layer driver header files.

5.2 Overview of low-layer APIs and naming rules

5.2.1 Peripheral initialization functions

The LL drivers offer three sets of initialization functions. They are defined in stm32h7xx_ll_ppp.c file:

- Functions to initialize peripheral main features according to the parameters specified in data structures
- A set of functions used to fill initialization data structures with the reset values of each field
- Function for peripheral de-initialization (peripheral registers restored to their default values)

The definition of these LL initialization functions and associated resources (structure, literals and prototypes) is conditioned by a compilation switch: *USE_FULL_LL_DRIVER*. To use these functions, this switch must be added in the toolchain compiler preprocessor or to any generic header file which is processed before the LL drivers.

The below table shows the list of the common functions provided for all the supported peripherals:

Table 17. Common peripheral initialization functions

Functions	Return Type	Parameters	Description
LL_PPP_Init	*ErrorStatus*	• *PPP_TypeDef* PPPx* • *LL_PPP_InitTypeDef* PPP_InitStruct*	Initializes the peripheral main features according to the parameters specified in PPP_InitStruct. Example: LL_USART_Init(USART_TypeDef *USARTx, LL_USART_InitTypeDef *USART_InitStruct)
LL_PPP_StructInit	*void*	• *LL_PPP_InitTypeDef* PPP_InitStruct*	Fills each PPP_InitStruct member with its default value. Example. LL_USART_StructInit(LL_USART_InitTypeDef *USART_InitStruct)
LL_PPP_DeInit	*ErrorStatus*	• *PPP_TypeDef* PPPx*	De-initializes the peripheral registers, that is restore them to their default reset values. Example. LL_USART_DeInit(USART_TypeDef *USARTx)

Additional functions are available for some peripherals (refer to Table 18. Optional peripheral initialization functions).

Table 18. Optional peripheral initialization functions

Functions	Return Type	Parameters	Examples
LL_PPP{_CATEGORY}_Init	*ErrorStatus*	• *PPP_TypeDef* PPPx* • *LL_PPP{_CATEGORY}_InitTypeDef* PPP{_CATEGORY}_InitStruct*	Initializes peripheral features according to the parameters specified in PPP_InitStruct. Example: LL_ADC_INJ_Init(ADC_TypeDef *ADCx, LL_ADC_INJ_InitTypeDef *ADC_INJ_InitStruct) LL_RTC_TIME_Init(RTC_TypeDef *RTCx, uint32_t RTC_Format, LL_RTC_TimeTypeDef *RTC_TimeStruct) LL_RTC_DATE_Init(RTC_TypeDef *RTCx, uint32_t RTC_Format, LL_RTC_DateTypeDef *RTC_DateStruct) LL_TIM_IC_Init(TIM_TypeDef* TIMx, uint32_t Channel, LL_TIM_IC_InitTypeDef* TIM_IC_InitStruct) LL_TIM_ENCODER_Init(TIM_TypeDef* TIMx, LL_TIM_ENCODER_InitTypeDef* TIM_EncoderInitStruct)
LL_PPP{_CATEGORY}_StructInit	*void*	*LL_PPP{_CATEGORY}_InitTypeDef* PPP{_CATEGORY}_InitStruct*	Fills each *PPP{_CATEGORY}_InitStruct* member with its default value. Example: LL_ADC_INJ_StructInit(LL_ADC_INJ_InitTypeDef *ADC_INJ_InitStruct)
LL_PPP_CommonInit	*ErrorStatus*	• *PPP_TypeDef* PPPx* • *LL_PPP_CommonInitTypeDef* PPP_CommonInitStruct*	Initializes the common features shared between different instances of the same peripheral. Example: LL_ADC_CommonInit(ADC_Common_TypeDef *ADCxy_COMMON, LL_ADC_CommonInitTypeDef *ADC_CommonInitStruct)
LL_PPP_CommonStructInit	*void*	*LL_PPP_CommonInitTypeDef* PPP_CommonInitStruct*	Fills each *PPP_CommonInitStruct* member with its default value Example: LL_ADC_CommonStructInit(LL_ADC_CommonInitTypeDef *ADC_CommonInitStruct)
LL_PPP_ClockInit	*ErrorStatus*	• *PPP_TypeDef* PPPx* • *LL_PPP_ClockInitTypeDef* PPP_ClockInitStruct*	Initializes the peripheral clock configuration in synchronous mode.

Functions	Return Type	Parameters	Examples
			Example: LL_USART_ClockInit(USART_TypeDef *USARTx, LL_USART_ClockInitTypeDef *USART_ClockInitStruct)
LL_PPP_ClockStructInit	*void*	*LL_PPP_ClockInitTypeDef* PPP_ClockInitStruct*	Fills each *PPP_ClockInitStruct* member with its default value Example: LL_USART_ClockStructInit(LL_USART_ClockInitTypeDef *USART_ClockInitStruct)

5.2.1.1 *Run-time checking*

Like HAL drivers, LL initialization functions implement run-time failure detection by checking the input values of all LL driver functions. For more details, refer to Section 4.12.4.3 Run-time checking.

When using the LL drivers in standalone mode (without calling HAL functions), the following actions are required to use run-time checking:

1. Copy stm32_assert_template.h to the application folder and rename it to stm32_assert.h. This file defines the assert_param macro which is used when run-time checking is enabled.
2. Include stm32_assert.h file within the application main header file.
3. Add the USE_FULL_ASSERT compilation switch in the toolchain compiler preprocessor or in any generic header file which is processed before the stm32_assert.h driver.

Note: Run-time checking is not available for LL inline functions.

5.2.2 Peripheral register-level configuration functions

On top of the peripheral initialization functions, the LL drivers offer a set of inline functions for direct atomic register access. Their format is as follows:

```
__STATIC_INLINE return_type LL_PPP_Function (PPPx_TypeDef *PPPx, args)
```

The “Function” naming is defined depending to the action category:

- **Specific Interrupt, DMA request and status flags management**: Set/Get/Clear/Enable/Disable flags on interrupt and status registers

Table 19. Specific Interrupt, DMA request and status flags management

Name	Examples
LL_PPP_{_CATEGORY}_ActionItem_BITNAME *LL_PPP{_CATEGORY}_IsItem_BITNAME_Action*	• LL_RCC_IsActiveFlag_LSIRDY • LL_RCC_IsActiveFlag_FWRST() • LL_ADC_ClearFlag_EOC(ADC1) • LL_DMA_ClearFlag_TCx(DMA_TypeDef* DMAx)

Table 20. Available function formats

Item	Action	Format
Flag	Get	*LL_PPP_IsActiveFlag_BITNAME*
	Clear	*LL_PPP_ClearFlag_BITNAME*
Interrupts	Enable	*LL_PPP_EnableIT_BITNAME*
	Disable	*LL_PPP_DisableIT_BITNAME*
	Get	*LL_PPP_IsEnabledIT_BITNAME*
DMA	Enable	*LL_PPP_EnableDMAReq_BITNAME*
	Disable	*LL_PPP_DisableDMAReq_BITNAME*
	Get	*LL_PPP_IsEnabledDMAReq_BITNAME*

Note: BITNAME refers to the peripheral register bit name as described in the product line reference manual.

- **Peripheral clock activation/deactivation management**: Enable/Disable/Reset a peripheral clock

Table 21. Peripheral clock activation/deactivation management

Name	Examples
LL_BUS_GRPx_ActionClock{Mode}	• *LL_AHB2_GRP1_EnableClock (LL_AHB2_GRP1_PERIPH_GPIOA| LL_AHB2_GRP1_PERIPH_GPIOB)* • *LL_APB1_GRP1_EnableClockSleep (LL_APB1_GRP1_PERIPH_DAC1)*

Note: 'x' corresponds to the group index and refers to the index of the modified register on a given bus. 'bus' corresponds to the bus name.

- ***Peripheral activation/deactivation management*** : Enable/disable a peripheral or activate/deactivate specific peripheral features

Table 22. Peripheral activation/deactivation management

Name	Examples
LL_PPP{_CATEGORY}_Action{Item} *LL_PPP{_CATEGORY}_IsItemAction*	• *LL_ADC_Enable ()* • *LL_ADC_StartCalibration();* • *LL_ADC_IsCalibrationOnGoing;* • *LL_RCC_HSI_Enable ()* • *LL_RCC_HSI_IsReady()*

- ***Peripheral configuration management*** : Set/get a peripheral configuration settings

Table 23. Peripheral configuration management

Name	Examples
LL_PPP{_CATEGORY}_Set{ or Get}ConfigItem	*LL_USART_SetBaudRate (USART2, Clock, LL_USART_BAUDRATE_9600)*

- ***Peripheral register management*** : Write/read the content of a register/retrun DMA relative register address

Table 24. Peripheral register management

Name
LL_PPP_WriteReg(__INSTANCE__, __REG__, __VALUE__)
LL_PPP_ReadReg(__INSTANCE__, __REG__)
*LL_PPP_DMA_GetRegAddr (PPP_TypeDef *PPPx,{Sub Instance if any ex: Channel} , {uint32_t Propriety})*

Note: The Propriety is a variable used to identify the DMA transfer direction or the data register type.

6 Cohabiting of HAL and LL

The low-ayer APIs are designed to be used in standalone mode or combined with the HAL. They cannot be automatically used with the HAL for the same peripheral instance. If you use the LL APIs for a specific instance, you can still use the HAL APIs for other instances. Be careful that the low-layer APIs might overwrite some registers which content is mirrored in the HAL handles.

6.1 Low-layer driver used in Standalone mode

The low-layer APIs can be used without calling the HAL driver services. This is done by simply including stm32h7xx_ll_ppp.h in the application files. The LL APIs for a given peripheral are called by executing the same sequence as the one recommended by the programming model in the corresponding product line reference manual. In this case the HAL drivers associated to the used peripheral can be removed from the workspace. However the STM32CubeH7 framework should be used in the same way as in the HAL drivers case which means that System file, startup file and CMSIS should always be used.

Note: When the BSP drivers are included, the used HAL drivers associated with the BSP functions drivers should be included in the workspace, even if they are not used by the application layer.

6.2 Mixed use of low-layer APIs and HAL drivers

In this case the low-layer APIs are used in conjunction with the HAL drivers to achieve direct and register level based operations.

Mixed use is allowed, however some consideration should be taken into account:

- It is recommended to avoid using simultaneously the HAL APIs and the combination of low-layer APIs for a given peripheral instance. If this is the case, one or more private fields in the HAL PPP handle structure should be updated accordingly.
- For operations and processes that do not alter the handle fields including the initialization structure, the HAL driver APIs and the low-layer services can be used together for the same peripheral instance.
- The low-layer drivers can be used without any restriction with all the HAL drivers that are not based on handle objects (RCC, common HAL, Flash and GPIO).

Several examples showing how to use HAL and LL in the same application are provided within stm32h7 firmware package (refer to Examples_MIX projects).

Note:

1. *When the HAL Init/DeInit APIs are not used and are replaced by the low-layer macros, the InitMsp() functions are not called and the MSP initialization should be done in the user application.*
2. *When process APIs are not used and the corresponding function is performed through the low-layer APIs, the callbacks are not called and post processing or error management should be done by the user application.*
3. *When the LL APIs is used for process operations, the IRQ handler HAL APIs cannot be called and the IRQ should be implemented by the user application. Each LL driver implements the macros needed to read and clear the associated interrupt flags.*

7 HAL System Driver

7.1 HAL Firmware driver API description

The following section lists the various functions of the HAL library.

7.1.1 How to use this driver

The common HAL driver contains a set of generic and common APIs that can be used by the PPP peripheral drivers and the user to start using the HAL.

The HAL contains two APIs' categories:

- Common HAL APIs
- Services HAL APIs

7.1.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initializes the Flash interface the NVIC allocation and initial clock configuration. It initializes the systick also when timeout is needed and the backup domain when enabled.
- De-Initializes common part of the HAL.
- Configure The time base source to have 1ms time base with a dedicated Tick interrupt priority.
  - SysTick timer is used by default as source of time base, but user can eventually implement his proper time base source (a general purpose timer for example or other time source), keeping in mind that Time base duration should be kept 1ms since PPP_TIMEOUT_VALUEs are defined and handled in milliseconds basis.
  - Time base configuration function (HAL_InitTick ()) is called automatically at the beginning of the program after reset by HAL_Init() or at any time when clock is configured, by HAL_RCC_ClockConfig().
  - Source of time base is configured to generate interrupts at regular time intervals. Care must be taken if HAL_Delay() is called from a peripheral ISR process, the Tick interrupt line must have higher priority (numerically lower) than the peripheral interrupt. Otherwise the caller ISR process will be blocked.
  - functions affecting time base configurations are declared as __weak to make override possible in case of other implementations in user file.

This section contains the following APIs:

- ***HAL_Init()***
- ***HAL_DeInit()***
- ***HAL_MspInit()***
- ***HAL_MspDeInit()***
- ***HAL_InitTick()***

7.1.3 HAL Control functions

This section provides functions allowing to:

- Provide a tick value in millisecond
- Provide a blocking delay in millisecond
- Suspend the time base source interrupt
- Resume the time base source interrupt
- Get the HAL API driver version
- Get the device identifier
- Get the device revision identifier
- Enable/Disable Debug module during SLEEP mode
- Enable/Disable Debug module during STOP mode

- Enable/Disable Debug module during STANDBY mode

This section contains the following APIs:

- *HAL_IncTick()*
- *HAL_GetTick()*
- *HAL_GetTickPrio()*
- *HAL_SetTickFreq()*
- *HAL_GetTickFreq()*
- *HAL_Delay()*
- *HAL_SuspendTick()*
- *HAL_ResumeTick()*
- *HAL_GetHalVersion()*
- *HAL_GetREVID()*
- *HAL_GetDEVID()*
- *HAL_GetUIDw0()*
- *HAL_GetUIDw1()*
- *HAL_GetUIDw2()*
- *HAL_SYSCFG_VREFBUF_VoltageScalingConfig()*
- *HAL_SYSCFG_VREFBUF_HighImpedanceConfig()*
- *HAL_SYSCFG_VREFBUF_TrimmingConfig()*
- *HAL_SYSCFG_EnableVREFBUF()*
- *HAL_SYSCFG_DisableVREFBUF()*
- *HAL_SYSCFG_ETHInterfaceSelect()*
- *HAL_SYSCFG_AnalogSwitchConfig()*
- *HAL_SYSCFG_EnableBOOST()*
- *HAL_SYSCFG_DisableBOOST()*
- *HAL_SYSCFG_CM7BootAddConfig()*
- *HAL_SYSCFG_CM4BootAddConfig()*
- *HAL_SYSCFG_EnableCM7BOOT()*
- *HAL_SYSCFG_DisableCM7BOOT()*
- *HAL_SYSCFG_EnableCM4BOOT()*
- *HAL_SYSCFG_DisableCM4BOOT()*
- *HAL_EnableCompensationCell()*
- *HAL_DisableCompensationCell()*
- *HAL_SYSCFG_EnableIOSpeedOptimize()*
- *HAL_SYSCFG_DisableIOSpeedOptimize()*
- *HAL_SYSCFG_CompensationCodeSelect()*
- *HAL_SYSCFG_CompensationCodeConfig()*
- *HAL_DBGMCU_EnableDBGSleepMode()*
- *HAL_DBGMCU_DisableDBGSleepMode()*
- *HAL_DBGMCU_EnableDBGStopMode()*
- *HAL_DBGMCU_DisableDBGStopMode()*
- *HAL_DBGMCU_EnableDBGStandbyMode()*
- *HAL_DBGMCU_DisableDBGStandbyMode()*
- *HAL_EnableDomain2DBGSleepMode()*
- *HAL_DisableDomain2DBGSleepMode()*
- *HAL_EnableDomain2DBGStopMode()*
- *HAL_DisableDomain2DBGStopMode()*
- *HAL_EnableDomain2DBGStandbyMode()*
- *HAL_DisableDomain2DBGStandbyMode()*

- *HAL_SetFMCMemorySwappingConfig()*
- *HAL_GetFMCMemorySwappingConfig()*
- *HAL_EXTI_EdgeConfig()*
- *HAL_EXTI_GenerateSWInterrupt()*
- *HAL_EXTI_D1_ClearFlag()*
- *HAL_EXTI_D2_ClearFlag()*
- *HAL_EXTI_D1_EventInputConfig()*
- *HAL_EXTI_D2_EventInputConfig()*
- *HAL_EXTI_D3_EventInputConfig()*

7.1.4 Detailed description of functions

HAL_Init

Function name

HAL_StatusTypeDef HAL_Init (void)

Function description

This function is used to initialize the HAL Library; it must be the first instruction to be executed in the main program (before to call any other HAL function), it performs the following: Configures the SysTick to generate an interrupt each 1 millisecond, which is clocked by the HSI (at this stage, the clock is not yet configured and thus the system is running from the internal HSI at 16 MHz).

Return values

- **HAL:** status

Notes

- SysTick is used as time base for the HAL_Delay() function, the application need to ensure that the SysTick time base is always set to 1 millisecond to have correct HAL operation.

HAL_DeInit

Function name

HAL_StatusTypeDef HAL_DeInit (void)

Function description

This function de-Initializes common part of the HAL and stops the systick.

Return values

- **HAL:** status

HAL_MspInit

Function name

void HAL_MspInit (void)

Function description

Initializes the MSP.

Return values

- **None:**

HAL_MspDeInit

Function name

void HAL_MspDeInit (void)

Function description

DeInitializes the MSP.

Return values

- **None:**

HAL_InitTick

Function name

HAL_StatusTypeDef HAL_InitTick (uint32_t TickPriority)

Function description

This function configures the source of the time base.

Parameters

- **TickPriority:** Tick interrupt priority.

Return values

- **HAL:** status

Notes

- This function is called automatically at the beginning of program after reset by HAL_Init() or at any time when clock is reconfigured by HAL_RCC_ClockConfig().
- In the default implementation, SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Care must be taken if HAL_Delay() is called from a peripheral ISR process, the SysTick interrupt must have higher priority (numerically lower) than the peripheral interrupt. Otherwise the caller ISR process will be blocked. The function is declared as __weak to be overwritten in case of other implementation in user file.

HAL_IncTick

Function name

void HAL_IncTick (void)

Function description

This function is called to increment a global variable "uwTick" used as application time base.

Return values

- **None:**

Notes

- In the default implementation, this variable is incremented each 1ms in Systick ISR.
- This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_Delay

Function name

void HAL_Delay (uint32_t Delay)

Function description

This function provides minimum delay (in milliseconds) based on variable incremented.

Parameters

- **Delay:** specifies the delay time length, in milliseconds.

Return values

- **None:**

Notes

- In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals where uwTick is incremented.
- This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_GetTick

Function name

uint32_t HAL_GetTick (void)

Function description

Provides a tick value in millisecond.

Return values

- **tick:** value

Notes

- This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_GetTickPrio

Function name

uint32_t HAL_GetTickPrio (void)

Function description

This function returns a tick priority.

Return values

- **tick:** priority

HAL_SetTickFreq

Function name

HAL_StatusTypeDef HAL_SetTickFreq (HAL_TickFreqTypeDef Freq)

Function description

Set new tick Freq.

Return values

- **Status:**

HAL_GetTickFreq

Function name

HAL_TickFreqTypeDef HAL_GetTickFreq (void)

Function description

Return tick frequency.

Return values

- **tick:** period in Hz

HAL_SuspendTick

Function name

void HAL_SuspendTick (void)

Function description

Suspend Tick increment.

Return values

- **None:**

Notes

- In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Once HAL_SuspendTick() is called, the SysTick interrupt will be disabled and so Tick increment is suspended.
- This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_ResumeTick

Function name

void HAL_ResumeTick (void)

Function description

Resume Tick increment.

Return values

- **None:**

Notes

- In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Once HAL_ResumeTick() is called, the SysTick interrupt will be enabled and so Tick increment is resumed.
- This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_GetHalVersion

Function name

uint32_t HAL_GetHalVersion (void)

Function description

Returns the HAL revision.

Return values

- **version:** : 0xXYZR (8bits for each decimal, R for RC)

HAL_GetREVID

Function name

uint32_t HAL_GetREVID (void)

Function description

Returns the device revision identifier.

Return values

- **Device:** revision identifier

HAL_GetDEVID

Function name

uint32_t HAL_GetDEVID (void)

Function description

Returns the device identifier.

Return values

- **Device:** identifier

HAL_GetUIDw0

Function name

uint32_t HAL_GetUIDw0 (void)

Function description

Return the first word of the unique device identifier (UID based on 96 bits)

Return values

- **Device:** identifier

HAL_GetUIDw1

Function name

uint32_t HAL_GetUIDw1 (void)

Function description

Return the second word of the unique device identifier (UID based on 96 bits)

Return values

- **Device:** identifier

HAL_GetUIDw2

Function name

uint32_t HAL_GetUIDw2 (void)

Function description

Return the third word of the unique device identifier (UID based on 96 bits)

Return values

- **Device:** identifier

HAL_SYSCFG_ETHInterfaceSelect

Function name

void HAL_SYSCFG_ETHInterfaceSelect (uint32_t SYSCFG_ETHInterface)

Function description

Ethernet PHY Interface Selection either MII or RMII.

Parameters

- **SYSCFG_ETHInterface:** Selects the Ethernet PHY interface This parameter can be one of the following values:
  - SYSCFG_ETH_MII : Select the Media Independent Interface
  - SYSCFG_ETH_RMII: Select the Reduced Media Independent Interface

Return values

- **None:**

HAL_SYSCFG_AnalogSwitchConfig

Function name

void HAL_SYSCFG_AnalogSwitchConfig (uint32_t SYSCFG_AnalogSwitch, uint32_t SYSCFG_SwitchState)

Function description

Analog Switch control for dual analog pads.

Parameters

- **SYSCFG_AnalogSwitch:** Selects the analog pad This parameter can be one or a combination of the following values:
  - SYSCFG_SWITCH_PA0 : Select PA0 analog switch
  - SYSCFG_SWITCH_PA1: Select PA1 analog switch
  - SYSCFG_SWITCH_PC2 : Select PC2 analog switch
  - SYSCFG_SWITCH_PC3: Select PC3 analog switch
- **SYSCFG_SwitchState:** Open or Close the analog switch between dual pads (PXn and PXn_C) This parameter can be one or a combination of the following values:
  - SYSCFG_SWITCH_PA0_OPEN
  - SYSCFG_SWITCH_PA0_CLOSE
  - SYSCFG_SWITCH_PA1_OPEN
  - SYSCFG_SWITCH_PA1_CLOSE
  - SYSCFG_SWITCH_PC2_OPEN
  - SYSCFG_SWITCH_PC2_CLOSE
  - SYSCFG_SWITCH_PC3_OPEN
  - SYSCFG_SWITCH_PC3_CLOSE

Return values

- **None:**

HAL_SYSCFG_EnableBOOST

Function name

void HAL_SYSCFG_EnableBOOST (void)

Function description

Enables the booster to reduce the total harmonic distortion of the analog switch when the supply voltage is lower than 2.7 V.

Return values

- **None:**

Notes

- Activating the booster allows to guaranty the analog switch AC performance when the supply voltage is below 2.7 V: in this case, the analog switch performance is the same on the full voltage range

HAL_SYSCFG_DisableBOOST

Function name

void HAL_SYSCFG_DisableBOOST (void)

Function description

Disables the booster.

Return values

- **None:**

Notes

- Activating the booster allows to guaranty the analog switch AC performance when the supply voltage is below 2.7 V: in this case, the analog switch performance is the same on the full voltage range

HAL_SYSCFG_CM7BootAddConfig

Function name

void HAL_SYSCFG_CM7BootAddConfig (uint32_t BootRegister, uint32_t BootAddress)

Function description

BootCM7 address 0 configuration.

Parameters

- **BootRegister:** :Specifies the Boot Address register (Address0 or Address1) This parameter can be one of the following values:
  - SYSCFG_BOOT_ADDR0 : Select the boot address0
  - SYSCFG_BOOT_ADDR1: Select the boot address1
- **BootAddress:** :Specifies the CM7 Boot Address to be loaded in Address0 or Address1

Return values

- **None:**

HAL_SYSCFG_CM4BootAddConfig

Function name

void HAL_SYSCFG_CM4BootAddConfig (uint32_t BootRegister, uint32_t BootAddress)

Function description

BootCM4 address 0 configuration.

Parameters

- **BootRegister:** :Specifies the Boot Address register (Address0 or Address1) This parameter can be one of the following values:
  - SYSCFG_BOOT_ADDR0 : Select the boot address0
  - SYSCFG_BOOT_ADDR1: Select the boot address1
- **BootAddress:** :Specifies the CM4 Boot Address to be loaded in Address0 or Address1

Return values

- **None:**

HAL_SYSCFG_EnableCM7BOOT

Function name

void HAL_SYSCFG_EnableCM7BOOT (void)

Function description

Enables the Cortex-M7 boot.

Return values

- **None:**

HAL_SYSCFG_DisableCM7BOOT

Function name

void HAL_SYSCFG_DisableCM7BOOT (void)

Function description

Disables the Cortex-M7 boot.

Return values

- **None:**

Notes

- Disabling the boot will gate the CPU clock

HAL_SYSCFG_EnableCM4BOOT

Function name

void HAL_SYSCFG_EnableCM4BOOT (void)

Function description

Enables the Cortex-M4 boot.

Return values

- **None:**

HAL_SYSCFG_DisableCM4BOOT

Function name

void HAL_SYSCFG_DisableCM4BOOT (void)

Function description

Disables the Cortex-M4 boot.

Return values

- **None:**

Notes

- Disabling the boot will gate the CPU clock

HAL_EnableCompensationCell

Function name

void HAL_EnableCompensationCell (void)

Function description

Enables the I/O Compensation Cell.

Return values

- **None:**

Notes

- The I/O compensation cell can be used only when the device supply voltage ranges from 1.62 to 2.0 V and from 2.7 to 3.6 V.

HAL_DisableCompensationCell

Function name

void HAL_DisableCompensationCell (void)

Function description

Power-down the I/O Compensation Cell.

Return values

- **None:**

Notes

- The I/O compensation cell can be used only when the device supply voltage ranges from 1.62 to 2.0 V and from 2.7 to 3.6 V.

HAL_SYSCFG_EnableIOSpeedOptimize

Function name

void HAL_SYSCFG_EnableIOSpeedOptimize (void)

Function description

To Enable optimize the I/O speed when the product voltage is low.

Return values

- **None:**

Notes

- This bit is active only if PRODUCT_BELOW_25V user option bit is set. It must be used only if the product supply voltage is below 2.5 V. Setting this bit when VDD is higher than 2.5 V might be destructive.

HAL_SYSCFG_DisableIOSpeedOptimize

Function name

void HAL_SYSCFG_DisableIOSpeedOptimize (void)

Function description

To Disable optimize the I/O speed when the product voltage is low.

Return values

- **None:**

Notes

- This bit is active only if PRODUCT_BELOW_25V user option bit is set. It must be used only if the product supply voltage is below 2.5 V. Setting this bit when VDD is higher than 2.5 V might be destructive.

HAL_SYSCFG_CompensationCodeSelect

Function name

void HAL_SYSCFG_CompensationCodeSelect (uint32_t SYSCFG_CompCode)

Function description

Code selection for the I/O Compensation cell.

Parameters

- **SYSCFG_CompCode:** Selects the code to be applied for the I/O compensation cell This parameter can be one of the following values:
  - SYSCFG_CELL_CODE : Select Code from the cell (available in the SYSCFG_CCVR)
  - SYSCFG_REGISTER_CODE: Select Code from the SYSCFG compensation cell code register (SYSCFG_CCCR)

Return values

- **None:**

HAL_SYSCFG_CompensationCodeConfig

Function name

void HAL_SYSCFG_CompensationCodeConfig (uint32_t SYSCFG_PMOSCode, uint32_t SYSCFG_NMOSCode)

Function description

Code selection for the I/O Compensation cell.

Parameters

- **SYSCFG_PMOSCode:** PMOS compensation code This code is applied to the I/O compensation cell when the CS bit of the SYSCFG_CMPCR is set
- **SYSCFG_NMOSCode:** NMOS compensation code This code is applied to the I/O compensation cell when the CS bit of the SYSCFG_CMPCR is set

Return values

- **None:**

HAL_DBGMCU_EnableDBGSleepMode

Function name

void HAL_DBGMCU_EnableDBGSleepMode (void)

Function description

Enable the Debug Module during Domain1/CDomain SLEEP mode.

Return values

- **None:**

HAL_DBGMCU_DisableDBGSleepMode

Function name

void HAL_DBGMCU_DisableDBGSleepMode (void)

Function description

Disable the Debug Module during Domain1/CDomain SLEEP mode.

Return values

- **None:**

HAL_DBGMCU_EnableDBGStopMode

Function name

void HAL_DBGMCU_EnableDBGStopMode (void)

Function description

Enable the Debug Module during Domain1/CDomain STOP mode.

Return values

- **None:**

HAL_DBGMCU_DisableDBGStopMode

Function name

void HAL_DBGMCU_DisableDBGStopMode (void)

Function description

Disable the Debug Module during Domain1/CDomain STOP mode.

Return values

- **None:**

HAL_DBGMCU_EnableDBGStandbyMode

Function name

void HAL_DBGMCU_EnableDBGStandbyMode (void)

Function description

Enable the Debug Module during Domain1/CDomain STANDBY mode.

Return values

- **None:**

HAL_DBGMCU_DisableDBGStandbyMode

Function name

void HAL_DBGMCU_DisableDBGStandbyMode (void)

Function description

Disable the Debug Module during Domain1/CDomain STANDBY mode.

Return values

- **None:**

HAL_EnableDomain2DBGSleepMode

Function name

void HAL_EnableDomain2DBGSleepMode (void)

Function description

Enable the Debug Module during Domain1 SLEEP mode.

Return values

- **None:**

HAL_DisableDomain2DBGSleepMode

Function name

void HAL_DisableDomain2DBGSleepMode (void)

Function description

Disable the Debug Module during Domain2 SLEEP mode.

Return values

- **None:**

HAL_EnableDomain2DBGStopMode

Function name

void HAL_EnableDomain2DBGStopMode (void)

Function description

Enable the Debug Module during Domain2 STOP mode.

Return values

- **None:**

HAL_DisableDomain2DBGStopMode

Function name

void HAL_DisableDomain2DBGStopMode (void)

Function description

Disable the Debug Module during Domain2 STOP mode.

Return values

- **None:**

HAL_EnableDomain2DBGStandbyMode

Function name

void HAL_EnableDomain2DBGStandbyMode (void)

Function description

Enable the Debug Module during Domain2 STANDBY mode.

Return values

- **None:**

HAL_DisableDomain2DBGStandbyMode

Function name

void HAL_DisableDomain2DBGStandbyMode (void)

Function description

Disable the Debug Module during Domain2 STANDBY mode.

Return values

- **None:**

HAL_EXTI_EdgeConfig

Function name

void HAL_EXTI_EdgeConfig (uint32_t EXTI_Line, uint32_t EXTI_Edge)

Function description

Configure the EXTI input event line edge.

Parameters

- **EXTI_Line:** Specifies the EXTI LINE, it can be one of the following values, (EXTI_LINE0....EXTI_LINE87)excluding :line45, line81,line83 which are reserved
- **EXTI_Edge:** Specifies EXTI line Edge used. This parameter can be one of the following values :
  - EXTI_RISING_EDGE : Configurable line, with Rising edge trigger detection
  - EXTI_FALLING_EDGE: Configurable line, with Falling edge trigger detection

Return values

- **None:**

Notes

- No edge configuration for direct lines but for configurable lines:(EXTI_LINE0..EXTI_LINE21), EXTI_LINE49,EXTI_LINE51,EXTI_LINE82,EXTI_LINE84,EXTI_LINE85 and EXTI_LINE86.

HAL_EXTI_GenerateSWInterrupt

Function name

void HAL_EXTI_GenerateSWInterrupt (uint32_t EXTI_Line)

Function description

Generates a Software interrupt on selected EXTI line.

Parameters

- **EXTI_Line:** Specifies the EXTI LINE, it can be one of the following values, (EXTI_LINE0..EXTI_LINE21),EXTI_LINE49,EXTI_LINE51,EXTI_LINE82,EXTI_LINE84,EXTI_LINE85 and EXTI_LINE86.

Return values

- **None:**

HAL_EXTI_D2_ClearFlag

Function name

void HAL_EXTI_D2_ClearFlag (uint32_t EXTI_Line)

Function description

Clears the EXTI's line pending flags for Domain D2.

Parameters

- **EXTI_Line:** Specifies the EXTI LINE, it can be one of the following values, (EXTI_LINE0....EXTI_LINE87)excluding :line45, line81,line83 which are reserved

Return values

- **None:**

HAL_EXTI_D1_ClearFlag

Function name

void HAL_EXTI_D1_ClearFlag (uint32_t EXTI_Line)

Function description

Clears the EXTI's line pending flags for Domain D1.

Parameters

- **EXTI_Line:** Specifies the EXTI LINE, it can be one of the following values, (EXTI_LINE0....EXTI_LINE87)excluding :line45, line81,line83 which are reserved

Return values

- **None:**

HAL_EXTI_D1_EventInputConfig

Function name

void HAL_EXTI_D1_EventInputConfig (uint32_t EXTI_Line, uint32_t EXTI_Mode, uint32_t EXTI_LineCmd)

Function description

Configure the EXTI input event line for Domain D1.

Parameters

- **EXTI_Line:** Specifies the EXTI LINE, it can be one of the following values, (EXTI_LINE0....EXTI_LINE87)excluding :line45, line81,line83 which are reserved
- **EXTI_Mode:** Specifies which EXTI line is used as interrupt or an event. This parameter can be one or a combination of the following values :
  - EXTI_MODE_IT : Interrupt Mode selected
  - EXTI_MODE_EVT : Event Mode selected
- **EXTI_LineCmd:** controls (Enable/Disable) the EXTI line.

Return values

- **None:**

HAL_EXTI_D2_EventInputConfig

Function name

void HAL_EXTI_D2_EventInputConfig (uint32_t EXTI_Line, uint32_t EXTI_Mode, uint32_t EXTI_LineCmd)

Function description

Configure the EXTI input event line for Domain D2.

Parameters

- **EXTI_Line:** Specifies the EXTI LINE, it can be one of the following values, (EXTI_LINE0....EXTI_LINE87)excluding :line45, line81,line83 which are reserved
- **EXTI_Mode:** Specifies which EXTI line is used as interrupt or an event. This parameter can be one or a combination of the following values :
  - EXTI_MODE_IT : Interrupt Mode selected
  - EXTI_MODE_EVT : Event Mode selected
- **EXTI_LineCmd:** controls (Enable/Disable) the EXTI line.

Return values

- **None:**

HAL_EXTI_D3_EventInputConfig

Function name

void HAL_EXTI_D3_EventInputConfig (uint32_t EXTI_Line, uint32_t EXTI_LineCmd, uint32_t EXTI_ClearSrc)

Function description

Configure the EXTI input event line for Domain D3.

Parameters

- **EXTI_Line:** Specifies the EXTI LINE, it can be one of the following values, (EXTI_LINE0...EXTI_LINE15),(EXTI_LINE19...EXTI_LINE21),EXTI_LINE25, EXTI_LINE34, EXTI_LINE35,EXTI_LINE41,(EXTI_LINE48...EXTI_LINE53)
- **EXTI_LineCmd:** controls (Enable/Disable) the EXTI line.
- **EXTI_ClearSrc:** Specifies the clear source of D3 pending event. This parameter can be one of the following values :
  - BDMA_CH6_CLEAR : BDMA ch6 event selected as D3 domain pendclear source
  - BDMA_CH7_CLEAR : BDMA ch7 event selected as D3 domain pendclear source
  - LPTIM4_OUT_CLEAR : LPTIM4 out selected as D3 domain pendclear source
  - LPTIM5_OUT_CLEAR : LPTIM5 out selected as D3 domain pendclear source

Return values

- **None:**

HAL_SetFMCMemorySwappingConfig

Function name

void HAL_SetFMCMemorySwappingConfig (uint32_t BankMapConfig)

Function description

Set the FMC Memory Mapping Swapping config.

Parameters

- **BankMapConfig:** Defines the FMC Bank mapping configuration. This parameter can be FMC_SWAPBMAP_DISABLE, FMC_SWAPBMAP_SDRAM_SRAM, FMC_SWAPBMAP_SDRAMB2

Return values

- **HAL:** state

HAL_GetFMCMemorySwappingConfig

Function name

uint32_t HAL_GetFMCMemorySwappingConfig (void)

Function description

Get FMC Bank mapping mode.

Return values

- **The:** FMC Bank mapping mode. This parameter can be FMC_SWAPBMAP_DISABLE, FMC_SWAPBMAP_SDRAM_SRAM, FMC_SWAPBMAP_SDRAMB2

HAL_SYSCFG_VREFBUF_VoltageScalingConfig

Function name

void HAL_SYSCFG_VREFBUF_VoltageScalingConfig (uint32_t VoltageScaling)

Function description

Configure the internal voltage reference buffer voltage scale.

Parameters

- **VoltageScaling:** specifies the output voltage to achieve This parameter can be one of the following values:
  - SYSCFG_VREFBUF_VOLTAGE_SCALE0: VREF_OUT1 around 2.5 V. This requires VDDA equal to or higher than 2.8 V.
  - SYSCFG_VREFBUF_VOLTAGE_SCALE1: VREF_OUT2 around 2.048 V. This requires VDDA equal to or higher than 2.4 V.
  - SYSCFG_VREFBUF_VOLTAGE_SCALE2: VREF_OUT3 around 1.8 V. This requires VDDA equal to or higher than 2.1 V.
  - SYSCFG_VREFBUF_VOLTAGE_SCALE3: VREF_OUT4 around 1.5 V. This requires VDDA equal to or higher than 1.8 V.

Return values

- **None:**

HAL_SYSCFG_VREFBUF_HighImpedanceConfig

Function name

void HAL_SYSCFG_VREFBUF_HighImpedanceConfig (uint32_t Mode)

Function description

Configure the internal voltage reference buffer high impedance mode.

Parameters

- **Mode:** specifies the high impedance mode This parameter can be one of the following values:
  - SYSCFG_VREFBUF_HIGH_IMPEDANCE_DISABLE: VREF+ pin is internally connect to VREFINT output.
  - SYSCFG_VREFBUF_HIGH_IMPEDANCE_ENABLE: VREF+ pin is high impedance.

Return values

- **None:**

HAL_SYSCFG_VREFBUF_TrimmingConfig

Function name

void HAL_SYSCFG_VREFBUF_TrimmingConfig (uint32_t TrimmingValue)

Function description

Tune the Internal Voltage Reference buffer (VREFBUF).

Return values

- **None:**

HAL_SYSCFG_EnableVREFBUF

Function name

HAL_StatusTypeDef HAL_SYSCFG_EnableVREFBUF (void)

Function description

Enable the Internal Voltage Reference buffer (VREFBUF).

Return values

- **HAL_OK/HAL_TIMEOUT:**

HAL_SYSCFG_DisableVREFBUF

Function name

void HAL_SYSCFG_DisableVREFBUF (void)

Function description

Disable the Internal Voltage Reference buffer (VREFBUF).

Return values

- **None:**

7.2 HAL Firmware driver defines

The following section lists the various define and macros of the module.

7.2.1 HAL

HAL

ETH States

HAL_ETH_STATE_RESET

Peripheral not yet Initialized or disabled

HAL_ETH_STATE_READY

Peripheral Communication started

HAL_ETH_STATE_BUSY

an internal process is ongoing

HAL_ETH_STATE_STARTED

an internal process is started

HAL_ETH_STATE_ERROR

Error State

HAL FDCAN Error Code

HAL_FDCAN_ERROR_NONE

No error

HAL_FDCAN_ERROR_TIMEOUT

Timeout error

HAL_FDCAN_ERROR_NOT_INITIALIZED

Peripheral not initialized

HAL_FDCAN_ERROR_NOT_READY

Peripheral not ready

HAL_FDCAN_ERROR_NOT_STARTED

Peripheral not started

HAL_FDCAN_ERROR_NOT_SUPPORTED

Mode not supported

HAL_FDCAN_ERROR_PARAM

Parameter error

HAL_FDCAN_ERROR_PENDING

Pending operation

HAL_FDCAN_ERROR_RAM_ACCESS

Message RAM Access Failure

HAL_FDCAN_ERROR_FIFO_EMPTY

Put element in full FIFO

HAL_FDCAN_ERROR_FIFO_FULL

Get element from empty FIFO

HAL_FDCAN_ERROR_LOG_OVERFLOW

Overflow of CAN Error Logging Counter

HAL_FDCAN_ERROR_RAM_WDG

Message RAM Watchdog event occurred

HAL_FDCAN_ERROR_PROTOCOL_ARBT

Protocol Error in Arbitration Phase (Nominal Bit Time is used)

HAL_FDCAN_ERROR_PROTOCOL_DATA

Protocol Error in Data Phase (Data Bit Time is used)

HAL_FDCAN_ERROR_RESERVED_AREA

Access to Reserved Address

HAL_FDCAN_ERROR_TT_GLOBAL_TIME

Global Time Error : Synchronization deviation exceeded limit

HAL_FDCAN_ERROR_TT_TX_UNDERFLOW

Tx Count Underflow : Less Tx trigger than expected in one matrix cycle

HAL_FDCAN_ERROR_TT_TX_OVERFLOW

Tx Count Overflow : More Tx trigger than expected in one matrix cycle

HAL_FDCAN_ERROR_TT_SCHEDULE1

Scheduling error 1

HAL_FDCAN_ERROR_TT_SCHEDULE2

Scheduling error 2

HAL_FDCAN_ERROR_TT_NO_INIT_REF

No system startup due to missing reference message

HAL_FDCAN_ERROR_TT_NO_REF

Missing reference message

HAL_FDCAN_ERROR_TT_APPL_WDG

Application watchdog not served in time

HAL_FDCAN_ERROR_TT_CONFIG

Error found in trigger list

HAL_FDCAN_ERROR_INVALID_CALLBACK

Invalid Callback error

HAL state definition

HAL_SMBUS_STATE_RESET

SMBUS not yet initialized or disabled

HAL_SMBUS_STATE_READY

SMBUS initialized and ready for use

HAL_SMBUS_STATE_BUSY

SMBUS internal process is ongoing

HAL_SMBUS_STATE_MASTER_BUSY_TX

Master Data Transmission process is ongoing

HAL_SMBUS_STATE_MASTER_BUSY_RX

Master Data Reception process is ongoing

HAL_SMBUS_STATE_SLAVE_BUSY_TX

Slave Data Transmission process is ongoing

HAL_SMBUS_STATE_SLAVE_BUSY_RX

Slave Data Reception process is ongoing

HAL_SMBUS_STATE_TIMEOUT

Timeout state

HAL_SMBUS_STATE_ERROR

Reception process is ongoing

HAL_SMBUS_STATE_LISTEN

Address Listen Mode is ongoing

Analog Switch Config

SYSCFG_SWITCH_PA0

Select PA0 analog switch

SYSCFG_SWITCH_PA1

Select PA1 analog switch

SYSCFG_SWITCH_PC2

Select PC2 analog switch

SYSCFG_SWITCH_PC3

Select PC3 analog switch

SYSCFG_SWITCH_PA0_OPEN

PA0 analog switch opened

SYSCFG_SWITCH_PA0_CLOSE

PA0 analog switch closed

SYSCFG_SWITCH_PA1_OPEN

PA1 analog switch opened

SYSCFG_SWITCH_PA1_CLOSE

PA1 analog switch closed

SYSCFG_SWITCH_PC2_OPEN

PC2 analog switch opened

SYSCFG_SWITCH_PC2_CLOSE

PC2 analog switch closed

SYSCFG_SWITCH_PC3_OPEN

PC3 analog switch opened

SYSCFG_SWITCH_PC3_CLOSE

PC3 analog switch closed

Boot Config

SYSCFG_BOOT_ADDR0

Select Boot address0

SYSCFG_BOOT_ADDR1

Select Boot address1

IS_SYSCFG_BOOT_REGISTER

IS_SYSCFG_BOOT_ADDRESS

Ethernet Config

SYSCFG_ETH_MII

Select the Media Independent Interface

SYSCFG_ETH_RMII

Select the Reduced Media Independent Interface

IS_SYSCFG_ETHERNET_CONFIG

SYSCFG Exported Constants

IS_SYSCFG_ANALOG_SWITCH

IS_SYSCFG_SWITCH_STATE

SYSCFG Exported Macros

__HAL_SYSCFG_BREAK_AXISRAM_DBL_ECC_LOCK

Notes:

- The selected configuration is locked and can be unlocked only by system reset. This feature is available on STM32H7 rev.B and above.

Enable and lock the connection of AXIRAM double ECC error to TIM1/8/15/16/17 and HRTIMER Break input.

__HAL_SYSCFG_BREAK_ITCM_DBL_ECC_LOCK

Notes:

- The selected configuration is locked and can be unlocked only by system reset. This feature is available on STM32H7 rev.B and above.

Enable and lock the connection of ITCM double ECC error to TIM1/8/15/16/17 and HRTIMER Break input.

__HAL_SYSCFG_BREAK_DTCM_DBL_ECC_LOCK

Notes:

- The selected configuration is locked and can be unlocked only by system reset. This feature is available on STM32H7 rev.B and above.

Enable and lock the connection of DTCM double ECC error to TIM1/8/15/16/17 and HRTIMER Break input.

__HAL_SYSCFG_BREAK_SRAM1_DBL_ECC_LOCK

Notes:

- The selected configuration is locked and can be unlocked only by system reset. This feature is available on STM32H7 rev.B and above.

Enable and lock the connection of SRAM1 double ECC error to TIM1/8/15/16/17 and HRTIMER Break input.

__HAL_SYSCFG_BREAK_SRAM2_DBL_ECC_LOCK

Notes:

- The selected configuration is locked and can be unlocked only by system reset. This feature is available on STM32H7 rev.B and above.

Enable and lock the connection of SRAM2 double ECC error to TIM1/8/15/16/17 and HRTIMER Break input.

__HAL_SYSCFG_BREAK_SRAM3_DBL_ECC_LOCK

Notes:

- The selected configuration is locked and can be unlocked only by system reset. This feature is available on STM32H7 rev.B and above.

Enable and lock the connection of SRAM3 double ECC error to TIM1/8/15/16/17 and HRTIMER Break input.

__HAL_SYSCFG_BREAK_SRAM4_DBL_ECC_LOCK

Notes:

- The selected configuration is locked and can be unlocked only by system reset. This feature is available on STM32H7 rev.B and above.

Enable and lock the connection of SRAM4 double ECC error to TIM1/8/15/16/17 and HRTIMER Break input.

__HAL_SYSCFG_BREAK_BKRAM_DBL_ECC_LOCK

Notes:

- The selected configuration is locked and can be unlocked only by system reset. This feature is available on STM32H7 rev.B and above.

Enable and lock the connection of Backup SRAM double ECC error to TIM1/8/15/16/17 and HRTIMER Break input.

__HAL_SYSCFG_BREAK_CM7_LOCKUP_LOCK

Notes:

- The selected configuration is locked and can be unlocked only by system reset. This feature is available on STM32H7 rev.B and above.

Enable and lock the connection of Cortex-M7 LOCKUP output to TIM1/8/15/16/17 and HRTIMER Break input.

__HAL_SYSCFG_BREAK_FLASH_DBL_ECC_LOCK

Notes:

- The selected configuration is locked and can be unlocked only by system reset. This feature is available on STM32H7 rev.B and above.

Enable and lock the connection of Flash double ECC error connection to TIM1/8/15/16/17 and HRTIMER Break input.

__HAL_SYSCFG_BREAK_PVD_LOCK

Notes:

- The selected configuration is locked and can be unlocked only by system reset. This feature is available on STM32H7 rev.B and above.

Enable and lock the PVD connection to Timer1/8/15/16/17 and HRTIMER Break input, as well as the PVDE and PLS[2:0] in the PWR_CR1 register.

__HAL_SYSCFG_BREAK_CM4_LOCKUP_LOCK

Notes:

- The selected configuration is locked and can be unlocked only by system reset. This feature is available on STM32H7 rev.B and above.

Enable and lock the connection of Cortex-M4 LOCKUP output to TIM1/8/15/16/17 and HRTIMER Break input.

IOCompenstionCell Config

SYSCFG_CELL_CODE

Select Code from the cell

SYSCFG_REGISTER_CODE

Code from the SYSCFG compensation cell code register

IS_SYSCFG_CODE_SELECT

IS_SYSCFG_CODE_CONFIG

VREFBUF High Impedance

SYSCFG_VREFBUF_HIGH_IMPEDANCE_DISABLE

VREF_plus pin is internally connected to Voltage reference buffer output

SYSCFG_VREFBUF_HIGH_IMPEDANCE_ENABLE

VREF_plus pin is high impedance

IS_SYSCFG_VREFBUF_HIGH_IMPEDANCE

IS_SYSCFG_VREFBUF_TRIMMING

VREFBUF Voltage Scale

SYSCFG_VREFBUF_VOLTAGE_SCALE0

Voltage reference scale 0 (VREF_OUT1)

SYSCFG_VREFBUF_VOLTAGE_SCALE1

Voltage reference scale 1 (VREF_OUT2)

SYSCFG_VREFBUF_VOLTAGE_SCALE2

Voltage reference scale 2 (VREF_OUT3)

SYSCFG_VREFBUF_VOLTAGE_SCALE3

Voltage reference scale 3 (VREF_OUT4)

IS_SYSCFG_VREFBUF_VOLTAGE_SCALE

8 HAL ADC Generic Driver

8.1 ADC Firmware driver registers structures

8.1.1 ADC_OversamplingTypeDef

ADC_OversamplingTypeDef is defined in the stm32h7xx_hal_adc.h

Data Fields

- ***uint32_t Ratio***
- ***uint32_t RightBitShift***
- ***uint32_t TriggeredMode***
- ***uint32_t OversamplingStopReset***

Field Documentation

- ***uint32_t ADC_OversamplingTypeDef::Ratio***
  Configures the oversampling ratio.
- ***uint32_t ADC_OversamplingTypeDef::RightBitShift***
  Configures the division coefficient for the Oversampler. This parameter can be a value of ***ADC_HAL_EC_OVS_SHIFT***
- ***uint32_t ADC_OversamplingTypeDef::TriggeredMode***
  Selects the regular triggered oversampling mode. This parameter can be a value of ***ADC_HAL_EC_OVS_DISCONT_MODE***
- ***uint32_t ADC_OversamplingTypeDef::OversamplingStopReset***
  Selects the regular oversampling mode. The oversampling is either temporary stopped or reset upon an injected sequence interruption. If oversampling is enabled on both regular and injected groups, this parameter is discarded and forced to setting "ADC_REGOVERSAMPLING_RESUMED_MODE" (the oversampling buffer is zeroed during injection sequence). This parameter can be a value of ***ADC_HAL_EC_OVS_SCOPE_REG***

8.1.2 ADC_InitTypeDef

ADC_InitTypeDef is defined in the stm32h7xx_hal_adc.h

Data Fields

- ***uint32_t ClockPrescaler***
- ***uint32_t Resolution***
- ***uint32_t ScanConvMode***
- ***uint32_t EOCSelection***
- ***FunctionalState LowPowerAutoWait***
- ***FunctionalState ContinuousConvMode***
- ***uint32_t NbrOfConversion***
- ***FunctionalState DiscontinuousConvMode***
- ***uint32_t NbrOfDiscConversion***
- ***uint32_t ExternalTrigConv***
- ***uint32_t ExternalTrigConvEdge***
- ***uint32_t ConversionDataManagement***
- ***uint32_t Overrun***
- ***uint32_t LeftBitShift***
- ***FunctionalState OversamplingMode***
- ***ADC_OversamplingTypeDef Oversampling***

Field Documentation

- ***uint32_t ADC_InitTypeDef::ClockPrescaler***

  Select ADC clock source (synchronous clock derived from APB clock or asynchronous clock derived from system clock or PLL (Refer to reference manual for list of clocks available)) and clock prescaler. This parameter can be a value of ***ADC_HAL_EC_COMMON_CLOCK_SOURCE***. Note: The ADC clock configuration is common to all ADC instances. Note: In case of usage of channels on injected group, ADC frequency should be lower than AHB clock frequency /4 for resolution 12 or 10 bits, AHB clock frequency /3 for resolution 8 bits, AHB clock frequency /2 for resolution 6 bits. Note: In case of synchronous clock mode based on HCLK/1, the configuration must be enabled only if the system clock has a 50% duty clock cycle (APB prescaler configured inside RCC must be bypassed and PCLK clock must have 50% duty cycle). Refer to reference manual for details. Note: In case of usage of asynchronous clock, the selected clock must be preliminarily enabled at RCC top level. Note: This parameter can be modified only if all ADC instances are disabled.

- ***uint32_t ADC_InitTypeDef::Resolution***

  Configure the ADC resolution. This parameter can be a value of ***ADC_HAL_EC_RESOLUTION***

- ***uint32_t ADC_InitTypeDef::ScanConvMode***

  Configure the sequencer of ADC groups regular and injected. This parameter can be associated to parameter 'DiscontinuousConvMode' to have main sequence subdivided in successive parts. If disabled: Conversion is performed in single mode (one channel converted, the one defined in rank 1). Parameters 'NbrOfConversion' and 'InjectedNbrOfConversion' are discarded (equivalent to set to 1). If enabled: Conversions are performed in sequence mode (multiple ranks defined by 'NbrOfConversion' or 'InjectedNbrOfConversion' and rank of each channel in sequencer). Scan direction is upward: from rank 1 to rank 'n'. This parameter can be a value of ***ADC_Scan_mode***

- ***uint32_t ADC_InitTypeDef::EOCSelection***

  Specify which EOC (End Of Conversion) flag is used for conversion by polling and interruption: end of unitary conversion or end of sequence conversions. This parameter can be a value of ***ADC_EOCSelection***.

- ***FunctionalState ADC_InitTypeDef::LowPowerAutoWait***

  Select the dynamic low power Auto Delay: new conversion start only when the previous conversion (for ADC group regular) or previous sequence (for ADC group injected) has been retrieved by user software, using function **HAL_ADC_GetValue()** or **HAL_ADCEx_InjectedGetValue()**. This feature automatically adapts the frequency of ADC conversions triggers to the speed of the system that reads the data. Moreover, this avoids risk of overrun for low frequency applications. This parameter can be set to ENABLE or DISABLE. Note: It is not recommended to use with interruption or DMA (**HAL_ADC_Start_IT()**, **HAL_ADC_Start_DMA()**) since these modes have to clear immediately the EOC flag (by CPU to free the IRQ pending event or by DMA). Auto wait will work but fort a very short time, discarding its intended benefit (except specific case of high load of CPU or DMA transfers which can justify usage of auto wait). Do use with polling: 1. Start conversion with **HAL_ADC_Start()**, 2. Later on, when ADC conversion data is needed: and use **HAL_ADC_GetValue()** to retrieve conversion result and trig another conversion (in case of usage of injected group, use the equivalent functions HAL_ADCExInjected_Start(), **HAL_ADCEx_InjectedGetValue()**, ...).

- ***FunctionalState ADC_InitTypeDef::ContinuousConvMode***

  Specify whether the conversion is performed in single mode (one conversion) or continuous mode for ADC group regular, after the first ADC conversion start trigger occurred (software start or external trigger). This parameter can be set to ENABLE or DISABLE.

- ***uint32_t ADC_InitTypeDef::NbrOfConversion***

  Specify the number of ranks that will be converted within the regular group sequencer. To use the regular group sequencer and convert several ranks, parameter 'ScanConvMode' must be enabled. This parameter must be a number between Min_Data = 1 and Max_Data = 16. Note: This parameter must be modified when no conversion is on going on regular group (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).

- ***FunctionalState ADC_InitTypeDef::DiscontinuousConvMode***

  Specify whether the conversions sequence of ADC group regular is performed in Complete-sequence/ Discontinuous-sequence (main sequence subdivided in successive parts). Discontinuous mode is used only if sequencer is enabled (parameter 'ScanConvMode'). If sequencer is disabled, this parameter is discarded. Discontinuous mode can be enabled only if continuous mode is disabled. If continuous mode is enabled, this parameter setting is discarded. This parameter can be set to ENABLE or DISABLE.

- ***uint32_t ADC_InitTypeDef::NbrOfDiscConversion***

  Specifies the number of discontinuous conversions in which the main sequence of ADC group regular (parameter NbrOfConversion) will be subdivided. If parameter 'DiscontinuousConvMode' is disabled, this parameter is discarded. This parameter must be a number between Min_Data = 1 and Max_Data = 8.
- ***uint32_t ADC_InitTypeDef::ExternalTrigConv***

  Select the external event source used to trigger ADC group regular conversion start. If set to ADC_SOFTWARE_START, external triggers are disabled and software trigger is used instead. This parameter can be a value of ***ADC_regular_external_trigger_source***. Caution: external trigger source is common to all ADC instances.
- ***uint32_t ADC_InitTypeDef::ExternalTrigConvEdge***

  Select the external event edge used to trigger ADC group regular conversion start. If trigger source is set to ADC_SOFTWARE_START, this parameter is discarded. This parameter can be a value of ***ADC_regular_external_trigger_edge***
- ***uint32_t ADC_InitTypeDef::ConversionDataManagement***

  Specifies whether the Data conversion data is managed: using the DMA (oneshot or circular), or stored in the DR register or transferred to DFSDM register. Note: In continuous mode, DMA must be configured in circular mode. Otherwise an overrun will be triggered when DMA buffer maximum pointer is reached. This parameter can be a value of ***ADC_ConversionDataManagement***. Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).
- ***uint32_t ADC_InitTypeDef::Overrun***

  Select the behavior in case of overrun: data overwritten or preserved (default). This parameter applies to ADC group regular only. This parameter can be a value of ***ADC_HAL_EC_REG_OVR_DATA_BEHAVIOR***. Note: In case of overrun set to data preserved and usage with programming model with interruption (HAL_Start_IT()): ADC IRQ handler has to clear end of conversion flags, this induces the release of the preserved data. If needed, this data can be saved in function **HAL_ADC_ConvCpltCallback()**, placed in user program code (called before end of conversion flags clear). Note: Error reporting with respect to the conversion mode:
  - Usage with ADC conversion by polling for event or interruption: Error is reported only if overrun is set to data preserved. If overrun is set to data overwritten, user can willingly not read all the converted data, this is not considered as an erroneous case.
  - Usage with ADC conversion by DMA: Error is reported whatever overrun setting (DMA is expected to process all data from data register).
- ***uint32_t ADC_InitTypeDef::LeftBitShift***

  Configures the left shifting applied to the final result with or without oversampling. This parameter can be a value of ***ADCEx_Left_Bit_Shift***
- ***FunctionalState ADC_InitTypeDef::OversamplingMode***

  Specify whether the oversampling feature is enabled or disabled. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing on ADC groups regular and injected
- ***ADC_OversamplingTypeDef ADC_InitTypeDef::Oversampling***

  Specify the Oversampling parameters. Caution: this setting overwrites the previous oversampling configuration if oversampling is already enabled.

8.1.3 ADC_ChannelConfTypeDef

ADC_ChannelConfTypeDef is defined in the stm32h7xx_hal_adc.h

Data Fields

- ***uint32_t Channel***
- ***uint32_t Rank***
- ***uint32_t SamplingTime***
- ***uint32_t SingleDiff***
- ***uint32_t OffsetNumber***
- ***uint32_t Offset***
- ***FunctionalState OffsetRightShift***

- ***FunctionalState OffsetSignedSaturation***

Field Documentation

- ***uint32_t ADC_ChannelConfTypeDef::Channel***

  Specify the channel to configure into ADC regular group. This parameter can be a value of ***ADC_HAL_EC_CHANNEL*** Note: Depending on devices and ADC instances, some channels may not be available on device package pins. Refer to device datasheet for channels availability.

- ***uint32_t ADC_ChannelConfTypeDef::Rank***

  Specify the rank in the regular group sequencer. This parameter can be a value of ***ADC_HAL_EC_REG_SEQ_RANKS*** Note: to disable a channel or change order of conversion sequencer, rank containing a previous channel setting can be overwritten by the new channel setting (or parameter number of conversions adjusted)

- ***uint32_t ADC_ChannelConfTypeDef::SamplingTime***

  Sampling time value to be set for the selected channel. Unit: ADC clock cycles Conversion time is the addition of sampling time and processing time (12.5 ADC clock cycles at ADC resolution 12 bits, 10.5 cycles at 10 bits, 8.5 cycles at 8 bits, 6.5 cycles at 6 bits). This parameter can be a value of ***ADC_HAL_EC_CHANNEL_SAMPLINGTIME*** Caution: This parameter applies to a channel that can be used into regular and/or injected group. It overwrites the last setting. Note: In case of usage of internal measurement channels (VrefInt/Vbat/TempSensor), sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting) Refer to device datasheet for timings values.

- ***uint32_t ADC_ChannelConfTypeDef::SingleDiff***

  Select single-ended or differential input. In differential mode: Differential measurement is carried out between the selected channel 'i' (positive input) and channel 'i+1' (negative input). Only channel 'i' has to be configured, channel 'i+1' is configured automatically. This parameter must be a value of ***ADC_HAL_EC_CHANNEL_SINGLE_DIFF_ENDING*** Caution: This parameter applies to a channel that can be used in a regular and/or injected group. It overwrites the last setting. Note: Refer to Reference Manual to ensure the selected channel is available in differential mode. Note: When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately. Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion). If ADC is enabled, this parameter setting is bypassed without error reporting (as it can be the expected behavior in case of another parameter update on the fly)

- ***uint32_t ADC_ChannelConfTypeDef::OffsetNumber***

  Select the offset number This parameter can be a value of ***ADC_HAL_EC_OFFSET_NB*** Caution: Only one offset is allowed per channel. This parameter overwrites the last setting.

- ***uint32_t ADC_ChannelConfTypeDef::Offset***

  Define the offset to be subtracted from the raw converted data. Offset value must be a positive number. Maximum value depends on ADC resolution and oversampling ratio (in case of oversampling used). This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x3FFFC00 (corresponding to resolution 16 bit and oversampling ratio 1024). Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).

- ***FunctionalState ADC_ChannelConfTypeDef::OffsetRightShift***

  Define the Right-shift data after Offset correction. This parameter is applied only for 16-bit or 8-bit resolution. This parameter can be set to ENABLE or DISABLE.

- ***FunctionalState ADC_ChannelConfTypeDef::OffsetSignedSaturation***

  Specify whether the Signed saturation feature is used or not. This parameter is applied only for 16-bit or 8-bit resolution. This parameter can be set to ENABLE or DISABLE.

8.1.4 ADC_AnalogWDGConfTypeDef

ADC_AnalogWDGConfTypeDef is defined in the stm32h7xx_hal_adc.h

Data Fields

- ***uint32_t WatchdogNumber***
- ***uint32_t WatchdogMode***
- ***uint32_t Channel***
- ***FunctionalState ITMode***

- ***uint32_t HighThreshold***
- ***uint32_t LowThreshold***

Field Documentation

- ***uint32_t ADC_AnalogWDGConfTypeDef::WatchdogNumber***

  Select which ADC analog watchdog is monitoring the selected channel. For Analog Watchdog 1: Only 1 channel can be monitored (or overall group of channels by setting parameter 'WatchdogMode') For Analog Watchdog 2 and 3: Several channels can be monitored (by successive calls of '**HAL_ADC_AnalogWDGConfig()**' for each channel) This parameter can be a value of ***ADC_HAL_EC_AWD_NUMBER***.
- ***uint32_t ADC_AnalogWDGConfTypeDef::WatchdogMode***

  Configure the ADC analog watchdog mode: single/all/none channels. For Analog Watchdog 1: Configure the ADC analog watchdog mode: single channel or all channels, ADC groups regular and-or injected. For Analog Watchdog 2 and 3: Several channels can be monitored by applying successively the AWD init structure. Channels on ADC group regular and injected are not differentiated: Set value 'ADC_ANALOGWATCHDOG_SINGLE_xxx' to monitor 1 channel, value 'ADC_ANALOGWATCHDOG_ALL_xxx' to monitor all channels, 'ADC_ANALOGWATCHDOG_NONE' to monitor no channel. This parameter can be a value of ***ADC_analog_watchdog_mode***.
- ***uint32_t ADC_AnalogWDGConfTypeDef::Channel***

  Select which ADC channel to monitor by analog watchdog. For Analog Watchdog 1: this parameter has an effect only if parameter 'WatchdogMode' is configured on single channel (only 1 channel can be monitored). For Analog Watchdog 2 and 3: Several channels can be monitored. To use this feature, call successively the function **HAL_ADC_AnalogWDGConfig()** for each channel to be added (or removed with value 'ADC_ANALOGWATCHDOG_NONE'). This parameter can be a value of ***ADC_HAL_EC_CHANNEL***.
- ***FunctionalState ADC_AnalogWDGConfTypeDef::ITMode***

  Specify whether the analog watchdog is configured in interrupt or polling mode. This parameter can be set to ENABLE or DISABLE
- ***uint32_t ADC_AnalogWDGConfTypeDef::HighThreshold***

  Configure the ADC analog watchdog High threshold value. Depending of ADC resolution selected (16, 14, 12, 10, 8 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFFF, 0x3FFF, 0xFFF, 0x3FF or 0xFF respectively. Note: Analog watchdog 2 and 3 are limited to a resolution of 8 bits: if ADC resolution is 12 bits the 4 LSB are ignored, if ADC resolution is 10 bits the 2 LSB are ignored. Note: If ADC oversampling is enabled, ADC analog watchdog thresholds are impacted: the comparison of analog watchdog thresholds is done on oversampling intermediate computation (after ratio, before shift application): intermediate register bitfield [32:7] (26 most significant bits).
- ***uint32_t ADC_AnalogWDGConfTypeDef::LowThreshold***

  Configures the ADC analog watchdog Low threshold value. Depending of ADC resolution selected (16, 14, 12, 10, 8 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFFF, 0x3FFF, 0xFFF, 0x3FF or 0xFF respectively. Note: Analog watchdog 2 and 3 are limited to a resolution of 8 bits: if ADC resolution is 12 bits the 4 LSB are ignored, if ADC resolution is 10 bits the 2 LSB are ignored. Note: If ADC oversampling is enabled, ADC analog watchdog thresholds are impacted: the comparison of analog watchdog thresholds is done on oversampling intermediate computation (after ratio, before shift application): intermediate register bitfield [32:7] (26 most significant bits).

8.1.5 ADC_InjectionConfigTypeDef

ADC_InjectionConfigTypeDef is defined in the stm32h7xx_hal_adc.h

Data Fields

- ***uint32_t ContextQueue***
- ***uint32_t ChannelCount***

Field Documentation

- ***uint32_t ADC_InjectionConfigTypeDef::ContextQueue***

  Injected channel configuration context: build-up over each **HAL_ADCEx_InjectedConfigChannel()** call to finally initialize JSQR register at **HAL_ADCEx_InjectedConfigChannel()** last call
- ***uint32_t ADC_InjectionConfigTypeDef::ChannelCount***

  Number of channels in the injected sequence

8.1.6 __ADC_HandleTypeDef

__ADC_HandleTypeDef is defined in the stm32h7xx_hal_adc.h

Data Fields

- ***ADC_TypeDef * Instance***
- ***ADC_InitTypeDef Init***
- ***DMA_HandleTypeDef * DMA_Handle***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t State***
- ***__IO uint32_t ErrorCode***
- ***ADC_InjectionConfigTypeDef InjectionConfig***
- ***void(* ConvCpltCallback***
- ***void(* ConvHalfCpltCallback***
- ***void(* LevelOutOfWindowCallback***
- ***void(* ErrorCallback***
- ***void(* InjectedConvCpltCallback***
- ***void(* InjectedQueueOverflowCallback***
- ***void(* LevelOutOfWindow2Callback***
- ***void(* LevelOutOfWindow3Callback***
- ***void(* EndOfSamplingCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***ADC_TypeDef* __ADC_HandleTypeDef::Instance***
  Register base address
- ***ADC_InitTypeDef __ADC_HandleTypeDef::Init***
  ADC initialization parameters and regular conversions setting
- ***DMA_HandleTypeDef* __ADC_HandleTypeDef::DMA_Handle***
  Pointer DMA Handler
- ***HAL_LockTypeDef __ADC_HandleTypeDef::Lock***
  ADC locking object
- ***__IO uint32_t __ADC_HandleTypeDef::State***
  ADC communication state (bitmap of ADC states)
- ***__IO uint32_t __ADC_HandleTypeDef::ErrorCode***
  ADC Error code
- ***ADC_InjectionConfigTypeDef __ADC_HandleTypeDef::InjectionConfig***
  ADC injected channel configuration build-up structure
- ***void(* __ADC_HandleTypeDef::ConvCpltCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC conversion complete callback
- ***void(* __ADC_HandleTypeDef::ConvHalfCpltCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC conversion DMA half-transfer callback
- ***void(* __ADC_HandleTypeDef::LevelOutOfWindowCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC analog watchdog 1 callback
- ***void(* __ADC_HandleTypeDef::ErrorCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC error callback
- ***void(* __ADC_HandleTypeDef::InjectedConvCpltCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC group injected conversion complete callback
- ***void(* __ADC_HandleTypeDef::InjectedQueueOverflowCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC group injected context queue overflow callback

- ***void(* __ADC_HandleTypeDef::LevelOutOfWindow2Callback)(struct __ADC_HandleTypeDef *hadc)***
  ADC analog watchdog 2 callback
- ***void(* __ADC_HandleTypeDef::LevelOutOfWindow3Callback)(struct __ADC_HandleTypeDef *hadc)***
  ADC analog watchdog 3 callback
- ***void(* __ADC_HandleTypeDef::EndOfSamplingCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC end of sampling callback
- ***void(* __ADC_HandleTypeDef::MspInitCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC Msp Init callback
- ***void(* __ADC_HandleTypeDef::MspDeInitCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC Msp DeInit callback

8.2 ADC Firmware driver API description

The following section lists the various functions of the ADC library.

8.2.1 ADC peripheral features

- 16-bit, 14-bit, 12-bit, 10-bit or 8-bit configurable resolution. Note: On devices STM32H72xx and STM32H73xx, these resolution are applicable to instances ADC1 and ADC2. ADC3 is featuring resolutions 12-bit, 10-bit, 8-bit, 6-bit.
- Interrupt generation at the end of regular conversion and in case of analog watchdog or overrun events.
- Single and continuous conversion modes.
- Scan mode for conversion of several channels sequentially.
- Data alignment with in-built data coherency.
- Programmable sampling time (channel wise)
- External trigger (timer or EXTI) with configurable polarity
- DMA request generation for transfer of conversions data of regular group.
- Configurable delay between conversions in Dual interleaved mode.
- ADC channels selectable single/differential input.
- ADC offset shared on 4 offset instances.
- ADC calibration
- ADC conversion of regular group.
- ADC supply requirements: 1.62 V to 3.6 V.
- ADC input range: from Vref- (connected to Vssa) to Vref+ (connected to Vdda or to an external voltage reference).

8.2.2 How to use this driver

Configuration of top level parameters related to ADC

1. Enable the ADC interface
   - As prerequisite, ADC clock must be configured at RCC top level.
   - Two clock settings are mandatory:
     - ADC clock (core clock, also possibly conversion clock).
     - ADC clock (conversions clock). Two possible clock sources: synchronous clock derived from AHB clock or asynchronous clock derived from system clock, the PLL2 or the PLL3 running up to 400MHz.
     - Example: Into HAL_ADC_MspInit() (recommended code location) or with other device clock parameters configuration:
     - __HAL_RCC_ADC_CLK_ENABLE(); (mandatory) RCC_ADCCLKSOURCE_PLL2 enable: (optional: if asynchronous clock selected)
     - RCC_PeriphClkInitTypeDef RCC_PeriphClkInit;
     - PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
     - PeriphClkInit.AdcClockSelection = RCC_ADCCLKSOURCE_PLL2;
     - HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit);
   - ADC clock source and clock prescaler are configured at ADC level with parameter "ClockPrescaler" using function HAL_ADC_Init().
2. ADC pins configuration
   - Enable the clock for the ADC GPIOs using macro __HAL_RCC_GPIOx_CLK_ENABLE()
   - Configure these ADC pins in analog mode using function HAL_GPIO_Init()
3. Optionally, in case of usage of ADC with interruptions:
   - Configure the NVIC for ADC using function HAL_NVIC_EnableIRQ(ADCx_IRQn)
   - Insert the ADC interruption handler function HAL_ADC_IRQHandler() into the function of corresponding ADC interruption vector ADCx_IRQHandler().
4. Optionally, in case of usage of DMA:
   - Configure the DMA (DMA channel, mode normal or circular, ...) using function HAL_DMA_Init().
   - Configure the NVIC for DMA using function HAL_NVIC_EnableIRQ(DMAx_Channelx_IRQn)
   - Insert the ADC interruption handler function HAL_ADC_IRQHandler() into the function of corresponding DMA interruption vector DMAx_Channelx_IRQHandler().

Configuration of ADC, group regular, channels parameters

1. Configure the ADC parameters (resolution, data alignment, ...) and regular group parameters (conversion trigger, sequencer, ...) using function HAL_ADC_Init().
2. Configure the channels for regular group parameters (channel number, channel rank into sequencer, ..., into regular group) using function HAL_ADC_ConfigChannel().
3. Optionally, configure the analog watchdog parameters (channels monitored, thresholds, ...) using function HAL_ADC_AnalogWDGConfig().

Execution of ADC conversions

1. Optionally, perform an automatic ADC calibration to improve the conversion accuracy using function HAL_ADCEx_Calibration_Start().

2. ADC driver can be used among three modes: polling, interruption, transfer by DMA.
   - ADC conversion by polling:
     - Activate the ADC peripheral and start conversions using function HAL_ADC_Start()
     - Wait for ADC conversion completion using function HAL_ADC_PollForConversion()
     - Retrieve conversion results using function HAL_ADC_GetValue()
     - Stop conversion and disable the ADC peripheral using function HAL_ADC_Stop()
   - ADC conversion by interruption:
     - Activate the ADC peripheral and start conversions using function HAL_ADC_Start_IT()
     - Wait for ADC conversion completion by call of function HAL_ADC_ConvCpltCallback() (this function must be implemented in user program)
     - Retrieve conversion results using function HAL_ADC_GetValue()
     - Stop conversion and disable the ADC peripheral using function HAL_ADC_Stop_IT()
   - ADC conversion with transfer by DMA:
     - Activate the ADC peripheral and start conversions using function HAL_ADC_Start_DMA()
     - Wait for ADC conversion completion by call of function HAL_ADC_ConvCpltCallback() or HAL_ADC_ConvHalfCpltCallback() (these functions must be implemented in user program)
     - Conversion results are automatically transferred by DMA into destination variable address.
     - Stop conversion and disable the ADC peripheral using function HAL_ADC_Stop_DMA()

Note: Callback functions must be implemented in user program:

- *HAL_ADC_ErrorCallback()*
- *HAL_ADC_LevelOutOfWindowCallback() (callback of analog watchdog)*
- *HAL_ADC_ConvCpltCallback()*
- *HAL_ADC_ConvHalfCpltCallback*

Deinitialization of ADC

1. Disable the ADC interface
   - ADC clock can be hard reset and disabled at RCC top level.
   - Hard reset of ADC peripherals using macro __HAL_RCC_ADCx_FORCE_RESET(), __HAL_RCC_ADCx_RELEASE_RESET().
   - ADC clock disable using the equivalent macro/functions as configuration step.
     - Example: Into HAL_ADC_MspDeInit() (recommended code location) or with other device clock parameters configuration:
     - __HAL_RCC_ADC_CLK_DISABLE(); (if not used anymore) RCC_ADCCLKSOURCE_CLKP restore: (optional)
     - RCC_PeriphClkInitTypeDef RCC_PeriphClkInit;
     - PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
     - PeriphClkInit.AdcClockSelection = RCC_ADCCLKSOURCE_CLKP;
     - HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit);
2. ADC pins configuration
   - Disable the clock for the ADC GPIOs using macro __HAL_RCC_GPIOx_CLK_DISABLE()
3. Optionally, in case of usage of ADC with interruptions:
   - Disable the NVIC for ADC using function HAL_NVIC_EnableIRQ(ADCx_IRQn)
4. Optionally, in case of usage of DMA:
   - Deinitialize the DMA using function HAL_DMA_Init().
   - Disable the NVIC for DMA using function HAL_NVIC_EnableIRQ(DMAx_Channelx_IRQn)

Callback registration

The compilation flag USE_HAL_ADC_REGISTER_CALLBACKS, when set to 1, allows the user to configure dynamically the driver callbacks. Use Functions HAL_ADC_RegisterCallback() to register an interrupt callback.

Function HAL_ADC_RegisterCallback() allows to register following callbacks:

- ConvCpltCallback : ADC conversion complete callback
- ConvHalfCpltCallback : ADC conversion DMA half-transfer callback
- LevelOutOfWindowCallback : ADC analog watchdog 1 callback
- ErrorCallback : ADC error callback
- InjectedConvCpltCallback : ADC group injected conversion complete callback
- InjectedQueueOverflowCallback : ADC group injected context queue overflow callback
- LevelOutOfWindow2Callback : ADC analog watchdog 2 callback
- LevelOutOfWindow3Callback : ADC analog watchdog 3 callback
- EndOfSamplingCallback : ADC end of sampling callback
- MspInitCallback : ADC Msp Init callback
- MspDeInitCallback : ADC Msp DeInit callback This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_ADC_UnRegisterCallback to reset a callback to the default weak function.

HAL_ADC_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- ConvCpltCallback : ADC conversion complete callback
- ConvHalfCpltCallback : ADC conversion DMA half-transfer callback
- LevelOutOfWindowCallback : ADC analog watchdog 1 callback
- ErrorCallback : ADC error callback
- InjectedConvCpltCallback : ADC group injected conversion complete callback
- InjectedQueueOverflowCallback : ADC group injected context queue overflow callback
- LevelOutOfWindow2Callback : ADC analog watchdog 2 callback
- LevelOutOfWindow3Callback : ADC analog watchdog 3 callback
- EndOfSamplingCallback : ADC end of sampling callback
- MspInitCallback : ADC Msp Init callback
- MspDeInitCallback : ADC Msp DeInit callback

By default, after the HAL_ADC_Init() and when the state is HAL_ADC_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_ADC_ConvCpltCallback(), HAL_ADC_ErrorCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_ADC_Init()/ HAL_ADC_DeInit() only when these callbacks are null (not registered beforehand).

If MspInit or MspDeInit are not null, the HAL_ADC_Init()/ HAL_ADC_DeInit() keep and use the user MspInit/ MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in HAL_ADC_STATE_READY state only. Exception done MspInit/MspDeInit functions that can be registered/unregistered in HAL_ADC_STATE_READY or HAL_ADC_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/ DeInit.

Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_ADC_RegisterCallback() before calling HAL_ADC_DeInit() or HAL_ADC_Init() function.

When the compilation flag USE_HAL_ADC_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

8.2.3 Peripheral Control functions

This section provides functions allowing to:

- Configure channels on regular group
- Configure the analog watchdog

This section contains the following APIs:

- ***HAL_ADC_ConfigChannel()***
- ***HAL_ADC_AnalogWDGConfig()***

8.2.4 Peripheral state and errors functions

This subsection provides functions to get in run-time the status of the peripheral.

- Check the ADC state
- Check the ADC error code

This section contains the following APIs:

- *HAL_ADC_GetState()*
- *HAL_ADC_GetError()*

8.2.5 Detailed description of functions

HAL_ADC_Init

Function name

HAL_StatusTypeDef HAL_ADC_Init (ADC_HandleTypeDef * hadc)

Function description

Initialize the ADC peripheral and regular group according to parameters specified in structure "ADC_InitTypeDef".

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- As prerequisite, ADC clock must be configured at RCC top level (refer to description of RCC configuration for ADC in header of this file).
- Possibility to update parameters on the fly: This function initializes the ADC MSP (HAL_ADC_MspInit()) only when coming from ADC state reset. Following calls to this function can be used to reconfigure some parameters of ADC_InitTypeDef structure on the fly, without modifying MSP configuration. If ADC MSP has to be modified again, HAL_ADC_DeInit() must be called before HAL_ADC_Init(). The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_InitTypeDef".
- This function configures the ADC within 2 scopes: scope of entire ADC and scope of regular group. For parameters details, see comments of structure "ADC_InitTypeDef".
- Parameters related to common ADC registers (ADC clock mode) are set only if all ADCs are disabled. If this is not the case, these common parameters setting are bypassed without error reporting: it can be the intended behaviour in case of update of a parameter of ADC_InitTypeDef on the fly, without disabling the other ADCs.

HAL_ADC_DeInit

Function name

HAL_StatusTypeDef HAL_ADC_DeInit (ADC_HandleTypeDef * hadc)

Function description

Deinitialize the ADC peripheral registers to their default reset values, with deinitialization of the ADC MSP.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- For devices with several ADCs: reset of ADC common registers is done only if all ADCs sharing the same common group are disabled. (function "HAL_ADC_MspDeInit()" is also called under the same conditions: all ADC instances use the same core clock at RCC level, disabling the core clock reset all ADC instances). If this is not the case, reset of these common parameters reset is bypassed without error reporting: it can be the intended behavior in case of reset of a single ADC while the other ADCs sharing the same common group is still running.
- By default, HAL_ADC_DeInit() set ADC in mode deep power-down: this saves more power by reducing leakage currents and is particularly interesting before entering MCU low-power modes.

HAL_ADC_MspInit

Function name

void HAL_ADC_MspInit (ADC_HandleTypeDef * hadc)

Function description

Initialize the ADC MSP.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADC_MspDeInit

Function name

void HAL_ADC_MspDeInit (ADC_HandleTypeDef * hadc)

Function description

DeInitialize the ADC MSP.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

Notes

- All ADC instances use the same core clock at RCC level, disabling the core clock reset all ADC instances).

HAL_ADC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_ADC_RegisterCallback (ADC_HandleTypeDef * hadc, HAL_ADC_CallbackIDTypeDef CallbackID, pADC_CallbackTypeDef pCallback)

Function description

Register a User ADC Callback To be used instead of the weak predefined callback.

Parameters

- **hadc:** Pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_ADC_CONVERSION_COMPLETE_CB_ID ADC conversion complete callback ID
  - HAL_ADC_CONVERSION_HALF_CB_ID ADC conversion DMA half-transfer callback ID
  - HAL_ADC_LEVEL_OUT_OF_WINDOW_1_CB_ID ADC analog watchdog 1 callback ID
  - HAL_ADC_ERROR_CB_ID ADC error callback ID
  - HAL_ADC_INJ_CONVERSION_COMPLETE_CB_ID ADC group injected conversion complete callback ID
  - HAL_ADC_INJ_QUEUE_OVEFLOW_CB_ID ADC group injected context queue overflow callback ID
  - HAL_ADC_LEVEL_OUT_OF_WINDOW_2_CB_ID ADC analog watchdog 2 callback ID
  - HAL_ADC_LEVEL_OUT_OF_WINDOW_3_CB_ID ADC analog watchdog 3 callback ID
  - HAL_ADC_END_OF_SAMPLING_CB_ID ADC end of sampling callback ID
  - HAL_ADC_MSPINIT_CB_ID ADC Msp Init callback ID
  - HAL_ADC_MSPDEINIT_CB_ID ADC Msp DeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_ADC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_ADC_UnRegisterCallback (ADC_HandleTypeDef * hadc, HAL_ADC_CallbackIDTypeDef CallbackID)

Function description

Unregister a ADC Callback ADC callback is redirected to the weak predefined callback.

Parameters

- **hadc:** Pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_ADC_CONVERSION_COMPLETE_CB_ID ADC conversion complete callback ID
  - HAL_ADC_CONVERSION_HALF_CB_ID ADC conversion DMA half-transfer callback ID
  - HAL_ADC_LEVEL_OUT_OF_WINDOW_1_CB_ID ADC analog watchdog 1 callback ID
  - HAL_ADC_ERROR_CB_ID ADC error callback ID
  - HAL_ADC_INJ_CONVERSION_COMPLETE_CB_ID ADC group injected conversion complete callback ID
  - HAL_ADC_INJ_QUEUE_OVEFLOW_CB_ID ADC group injected context queue overflow callback ID
  - HAL_ADC_LEVEL_OUT_OF_WINDOW_2_CB_ID ADC analog watchdog 2 callback ID
  - HAL_ADC_LEVEL_OUT_OF_WINDOW_3_CB_ID ADC analog watchdog 3 callback ID
  - HAL_ADC_END_OF_SAMPLING_CB_ID ADC end of sampling callback ID
  - HAL_ADC_MSPINIT_CB_ID ADC Msp Init callback ID
  - HAL_ADC_MSPDEINIT_CB_ID ADC Msp DeInit callback ID

Return values

- **HAL:** status

HAL_ADC_Start

Function name

HAL_StatusTypeDef HAL_ADC_Start (ADC_HandleTypeDef * hadc)

Function description

Enable ADC, start conversion of regular group.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- Interruptions enabled in this function: None.
- Case of multimode enabled (when multimode feature is available): if ADC is Slave, ADC is enabled but conversion is not started, if ADC is master, ADC is enabled and multimode conversion is started.

HAL_ADC_Stop

Function name

HAL_StatusTypeDef HAL_ADC_Stop (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion of regular group (and injected channels in case of auto_injection mode), disable ADC peripheral.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

- : ADC peripheral disable is forcing stop of potential conversion on injected group. If injected group is under use, it should be preliminarily stopped using HAL_ADCEx_InjectedStop function.

HAL_ADC_PollForConversion

Function name

HAL_StatusTypeDef HAL_ADC_PollForConversion (ADC_HandleTypeDef * hadc, uint32_t Timeout)

Function description

Wait for regular group conversion to be completed.

Parameters

- **hadc:** ADC handle
- **Timeout:** Timeout value in millisecond.

Return values

- **HAL:** status

Notes

- ADC conversion flags EOS (end of sequence) and EOC (end of conversion) are cleared by this function, with an exception: if low power feature "LowPowerAutoWait" is enabled, flags are not cleared to not interfere with this feature until data register is read using function HAL_ADC_GetValue().
- This function cannot be used in a particular setup: ADC configured in DMA mode and polling for end of each conversion (ADC init parameter "EOCSelection" set to ADC_EOC_SINGLE_CONV). In this case, DMA resets the flag EOC and polling cannot be performed on each conversion. Nevertheless, polling can still be performed on the complete sequence (ADC init parameter "EOCSelection" set to ADC_EOC_SEQ_CONV).

HAL_ADC_PollForEvent

Function name

HAL_StatusTypeDef HAL_ADC_PollForEvent (ADC_HandleTypeDef * hadc, uint32_t EventType, uint32_t Timeout)

Function description

Poll for ADC event.

Parameters

- **hadc:** ADC handle
- **EventType:** the ADC event type. This parameter can be one of the following values:
  - ADC_EOSMP_EVENT ADC End of Sampling event
  - ADC_AWD1_EVENT ADC Analog watchdog 1 event (main analog watchdog, present on all STM32 devices)
  - ADC_AWD2_EVENT ADC Analog watchdog 2 event (additional analog watchdog, not present on all STM32 families)
  - ADC_AWD3_EVENT ADC Analog watchdog 3 event (additional analog watchdog, not present on all STM32 families)
  - ADC_OVR_EVENT ADC Overrun event
  - ADC_JQOVF_EVENT ADC Injected context queue overflow event
- **Timeout:** Timeout value in millisecond.

Return values

- **HAL:** status

Notes

- The relevant flag is cleared if found to be set, except for ADC_FLAG_OVR. Indeed, the latter is reset only if hadc->Init.Overrun field is set to ADC_OVR_DATA_OVERWRITTEN. Otherwise, data register may be potentially overwritten by a new converted data as soon as OVR is cleared. To reset OVR flag once the preserved data is retrieved, the user can resort to macro __HAL_ADC_CLEAR_FLAG(hadc, ADC_FLAG_OVR);

HAL_ADC_Start_IT

Function name

HAL_StatusTypeDef HAL_ADC_Start_IT (ADC_HandleTypeDef * hadc)

Function description

Enable ADC, start conversion of regular group with interruption.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- Interruptions enabled in this function according to initialization setting : EOC (end of conversion), EOS (end of sequence), OVR overrun. Each of these interruptions has its dedicated callback function.
- Case of multimode enabled (when multimode feature is available): HAL_ADC_Start_IT() must be called for ADC Slave first, then for ADC Master. For ADC Slave, ADC is enabled only (conversion is not started). For ADC Master, ADC is enabled and multimode conversion is started.
- To guarantee a proper reset of all interruptions once all the needed conversions are obtained, HAL_ADC_Stop_IT() must be called to ensure a correct stop of the IT-based conversions.
- By default, HAL_ADC_Start_IT() does not enable the End Of Sampling interruption. If required (e.g. in case of oversampling with trigger mode), the user must: 1. first clear the EOSMP flag if set with macro __HAL_ADC_CLEAR_FLAG(hadc, ADC_FLAG_EOSMP) 2. then enable the EOSMP interrupt with macro __HAL_ADC_ENABLE_IT(hadc, ADC_IT_EOSMP) before calling HAL_ADC_Start_IT().

HAL_ADC_Stop_IT

Function name

HAL_StatusTypeDef HAL_ADC_Stop_IT (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion of regular group (and injected group in case of auto_injection mode), disable interrution of end-of-conversion, disable ADC peripheral.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

HAL_ADC_Start_DMA

Function name

HAL_StatusTypeDef HAL_ADC_Start_DMA (ADC_HandleTypeDef * hadc, uint32_t * pData, uint32_t Length)

Function description

Enable ADC, start conversion of regular group and transfer result through DMA.

Parameters

- **hadc:** ADC handle
- **pData:** Destination Buffer address.
- **Length:** Number of data to be transferred from ADC peripheral to memory

Return values

- **HAL:** status.

Notes

- Interruptions enabled in this function: overrun (if applicable), DMA half transfer, DMA transfer complete. Each of these interruptions has its dedicated callback function.
- Case of multimode enabled (when multimode feature is available): HAL_ADC_Start_DMA() is designed for single-ADC mode only. For multimode, the dedicated HAL_ADCEx_MultiModeStart_DMA() function must be used.

HAL_ADC_Stop_DMA

Function name

HAL_StatusTypeDef HAL_ADC_Stop_DMA (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion of regular group (and injected group in case of auto_injection mode), disable ADC DMA transfer, disable ADC peripheral.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

- : ADC peripheral disable is forcing stop of potential conversion on ADC group injected. If ADC group injected is under use, it should be preliminarily stopped using HAL_ADCEx_InjectedStop function.
- Case of multimode enabled (when multimode feature is available): HAL_ADC_Stop_DMA() function is dedicated to single-ADC mode only. For multimode, the dedicated HAL_ADCEx_MultiModeStop_DMA() API must be used.

HAL_ADC_GetValue

Function name

uint32_t HAL_ADC_GetValue (ADC_HandleTypeDef * hadc)

Function description

Get ADC regular group conversion result.

Parameters

- **hadc:** ADC handle

Return values

- **ADC:** group regular conversion data

Notes

- Reading register DR automatically clears ADC flag EOC (ADC group regular end of unitary conversion).
- This function does not clear ADC flag EOS (ADC group regular end of sequence conversion). Occurrence of flag EOS rising: If sequencer is composed of 1 rank, flag EOS is equivalent to flag EOC.If sequencer is composed of several ranks, during the scan sequence flag EOC only is raised, at the end of the scan sequence both flags EOC and EOS are raised. To clear this flag, either use function: in programming model IT: HAL_ADC_IRQHandler(), in programming model polling: HAL_ADC_PollForConversion() or __HAL_ADC_CLEAR_FLAG(&hadc, ADC_FLAG_EOS).

HAL_ADC_IRQHandler

Function name

void HAL_ADC_IRQHandler (ADC_HandleTypeDef * hadc)

Function description

Handle ADC interrupt request.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADC_ConvCpltCallback

Function name

void HAL_ADC_ConvCpltCallback (ADC_HandleTypeDef * hadc)

Function description

Conversion complete callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADC_ConvHalfCpltCallback

Function name

void HAL_ADC_ConvHalfCpltCallback (ADC_HandleTypeDef * hadc)

Function description

Conversion DMA half-transfer callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADC_LevelOutOfWindowCallback

Function name

void HAL_ADC_LevelOutOfWindowCallback (ADC_HandleTypeDef * hadc)

Function description

Analog watchdog 1 callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADC_ErrorCallback

Function name

void HAL_ADC_ErrorCallback (ADC_HandleTypeDef * hadc)

Function description

ADC error callback in non-blocking mode (ADC conversion with interruption or transfer by DMA).

Parameters

- **hadc:** ADC handle

Return values

- **None:**

Notes

- In case of error due to overrun when using ADC with DMA transfer (HAL ADC handle parameter "ErrorCode" to state "HAL_ADC_ERROR_OVR"): Reinitialize the DMA using function "HAL_ADC_Stop_DMA()".If needed, restart a new ADC conversion using function "HAL_ADC_Start_DMA()" (this function is also clearing overrun flag)

HAL_ADC_ConfigChannel

Function name

HAL_StatusTypeDef HAL_ADC_ConfigChannel (ADC_HandleTypeDef * hadc, ADC_ChannelConfTypeDef * sConfig)

Function description

Configure a channel to be assigned to ADC group regular.

Parameters

- **hadc:** ADC handle
- **sConfig:** Structure of ADC channel assigned to ADC group regular.

Return values

- **HAL:** status

Notes

- In case of usage of internal measurement channels: Vbat/VrefInt/TempSensor. These internal paths can be disabled using function HAL_ADC_DeInit().
- Possibility to update parameters on the fly: This function initializes channel into ADC group regular, following calls to this function can be used to reconfigure some parameters of structure "ADC_ChannelConfTypeDef" on the fly, without resetting the ADC. The setting of these parameters is conditioned to ADC state: Refer to comments of structure "ADC_ChannelConfTypeDef".

HAL_ADC_AnalogWDGConfig

Function name

HAL_StatusTypeDef HAL_ADC_AnalogWDGConfig (ADC_HandleTypeDef * hadc, ADC_AnalogWDGConfTypeDef * AnalogWDGConfig)

Function description

Configure the analog watchdog.

Parameters

- **hadc:** ADC handle
- **AnalogWDGConfig:** Structure of ADC analog watchdog configuration

Return values

- **HAL:** status

Notes

- Possibility to update parameters on the fly: This function initializes the selected analog watchdog, successive calls to this function can be used to reconfigure some parameters of structure "ADC_AnalogWDGConfTypeDef" on the fly, without resetting the ADC. The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_AnalogWDGConfTypeDef".
- On this STM32 series, analog watchdog thresholds cannot be modified while ADC conversion is on going.

HAL_ADC_GetState

Function name

uint32_t HAL_ADC_GetState (ADC_HandleTypeDef * hadc)

Function description

Return the ADC handle state.

Parameters

- **hadc:** ADC handle

Return values

- **ADC:** handle state (bitfield on 32 bits)

Notes

- ADC state machine is managed by bitfields, ADC status must be compared with states bits.
  For example: " if ((HAL_ADC_GetState(hadc1) & HAL_ADC_STATE_REG_BUSY) != 0UL) " " if ((HAL_ADC_GetState(hadc1) & HAL_ADC_STATE_AWD1) != 0UL) "

HAL_ADC_GetError

Function name

uint32_t HAL_ADC_GetError (ADC_HandleTypeDef * hadc)

Function description

Return the ADC error code.

Parameters

- **hadc:** ADC handle

Return values

- **ADC:** error code (bitfield on 32 bits)

ADC_ConversionStop

Function name

HAL_StatusTypeDef ADC_ConversionStop (ADC_HandleTypeDef * hadc, uint32_t ConversionGroup)

Function description

Stop ADC conversion.

Parameters

- **hadc:** ADC handle
- **ConversionGroup:** ADC group regular and/or injected. This parameter can be one of the following values:
  - ADC_REGULAR_GROUP ADC regular conversion type.
  - ADC_INJECTED_GROUP ADC injected conversion type.
  - ADC_REGULAR_INJECTED_GROUP ADC regular and injected conversion type.

Return values

- **HAL:** status.

ADC_Enable

Function name

HAL_StatusTypeDef ADC_Enable (ADC_HandleTypeDef * hadc)

Function description

Enable the selected ADC.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

- Prerequisite condition to use this function: ADC must be disabled and voltage regulator must be enabled (done into HAL_ADC_Init()).

ADC_Disable

Function name

HAL_StatusTypeDef ADC_Disable (ADC_HandleTypeDef * hadc)

Function description

Disable the selected ADC.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

- Prerequisite condition to use this function: ADC conversions must be stopped.

ADC_DMAConvCplt

Function name

void ADC_DMAConvCplt (DMA_HandleTypeDef * hdma)

Function description

DMA transfer complete callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

ADC_DMAHalfConvCplt

Function name

void ADC_DMAHalfConvCplt (DMA_HandleTypeDef * hdma)

Function description

DMA half transfer complete callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

ADC_DMAError

Function name

void ADC_DMAError (DMA_HandleTypeDef * hdma)

Function description

DMA error callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

ADC_ConfigureBoostMode

Function name

void ADC_ConfigureBoostMode (ADC_HandleTypeDef * hadc)

Function description

Configure boost mode of selected ADC.

Parameters

- **hadc:** ADC handle

Return values

- **None.:**

Notes

- Prerequisite condition to use this function: ADC conversions must be stopped.

8.3 ADC Firmware driver defines

The following section lists the various define and macros of the module.

8.3.1 ADC

ADC

ADC Analog Watchdog Mode

ADC_ANALOGWATCHDOG_NONE

No analog watchdog selected

ADC_ANALOGWATCHDOG_SINGLE_REG

Analog watchdog applied to a regular group single channel

ADC_ANALOGWATCHDOG_SINGLE_INJEC

Analog watchdog applied to an injected group single channel

ADC_ANALOGWATCHDOG_SINGLE_REGINJEC

Analog watchdog applied to a regular and injected groups single channel

ADC_ANALOGWATCHDOG_ALL_REG

Analog watchdog applied to regular group all channels

ADC_ANALOGWATCHDOG_ALL_INJEC

Analog watchdog applied to injected group all channels

ADC_ANALOGWATCHDOG_ALL_REGINJEC

Analog watchdog applied to regular and injected groups all channels

ADCx CFGR fields

ADC_CFGR_FIELDS

ADCx CFGR sub fields

ADC_CFGR_FIELDS_2

ADC Conversion Data Management

ADC_CONVERSIONDATA_DR

Regular Conversion data stored in DR register only

ADC_CONVERSIONDATA_DFSDM

DFSDM mode selected

ADC_CONVERSIONDATA_DMA_ONESHOT

DMA one shot mode selected

ADC_CONVERSIONDATA_DMA_CIRCULAR

DMA circular mode selected

ADC sequencer end of unitary conversion or sequence conversions

ADC_EOC_SINGLE_CONV

End of unitary conversion flag

ADC_EOC_SEQ_CONV

End of sequence conversions flag

ADC Error Code

HAL_ADC_ERROR_NONE

No error

HAL_ADC_ERROR_INTERNAL

ADC peripheral internal error (problem of clocking, enable/disable, erroneous state, ...)

HAL_ADC_ERROR_OVR

Overrun error

HAL_ADC_ERROR_DMA

DMA transfer error

HAL_ADC_ERROR_JQOVF

Injected context queue overflow error

HAL_ADC_ERROR_INVALID_CALLBACK

Invalid Callback error

ADC Event type

ADC_EOSMP_EVENT

ADC End of Sampling event

ADC_AWD1_EVENT

ADC Analog watchdog 1 event (main analog watchdog, present on all STM32 series)

ADC_AWD2_EVENT

ADC Analog watchdog 2 event (additional analog watchdog, not present on all STM32 series)

ADC_AWD3_EVENT

ADC Analog watchdog 3 event (additional analog watchdog, not present on all STM32 series)

ADC_OVR_EVENT

ADC overrun event

ADC_JQOVF_EVENT

ADC Injected Context Queue Overflow event

ADC Exported Constants

ADC_AWD_EVENT

ADC Analog watchdog 1 event: Naming for compatibility with other STM32 devices having only one analog watchdog

ADC flags definition

ADC_FLAG_RDY

ADC Ready flag

ADC_FLAG_EOSMP

ADC End of Sampling flag

ADC_FLAG_EOC

ADC End of Regular Conversion flag

ADC_FLAG_EOS

ADC End of Regular sequence of Conversions flag

ADC_FLAG_OVR

ADC overrun flag

ADC_FLAG_JEOC

ADC End of Injected Conversion flag

ADC_FLAG_JEOS

ADC End of Injected sequence of Conversions flag

ADC_FLAG_AWD1

ADC Analog watchdog 1 flag (main analog watchdog)

ADC_FLAG_AWD2

ADC Analog watchdog 2 flag (additional analog watchdog)

ADC_FLAG_AWD3

ADC Analog watchdog 3 flag (additional analog watchdog)

ADC_FLAG_JQOVF

ADC Injected Context Queue Overflow flag

ADC_FLAG_LDORDY

ADC LDO output voltage ready bit

Analog watchdog - Analog watchdog number

ADC_ANALOGWATCHDOG_1

ADC analog watchdog number 1

ADC_ANALOGWATCHDOG_2

ADC analog watchdog number 2

ADC_ANALOGWATCHDOG_3

ADC analog watchdog number 3

ADC instance - Channel number

ADC_CHANNEL_0

ADC external channel (channel connected to GPIO pin) ADCx_IN0

ADC_CHANNEL_1

ADC external channel (channel connected to GPIO pin) ADCx_IN1

ADC_CHANNEL_2

ADC external channel (channel connected to GPIO pin) ADCx_IN2

ADC_CHANNEL_3

ADC external channel (channel connected to GPIO pin) ADCx_IN3

ADC_CHANNEL_4

ADC external channel (channel connected to GPIO pin) ADCx_IN4

ADC_CHANNEL_5

ADC external channel (channel connected to GPIO pin) ADCx_IN5

ADC_CHANNEL_6

ADC external channel (channel connected to GPIO pin) ADCx_IN6

ADC_CHANNEL_7

ADC external channel (channel connected to GPIO pin) ADCx_IN7

ADC_CHANNEL_8

ADC external channel (channel connected to GPIO pin) ADCx_IN8

ADC_CHANNEL_9

ADC external channel (channel connected to GPIO pin) ADCx_IN9

ADC_CHANNEL_10

ADC external channel (channel connected to GPIO pin) ADCx_IN10

ADC_CHANNEL_11

ADC external channel (channel connected to GPIO pin) ADCx_IN11

ADC_CHANNEL_12

ADC external channel (channel connected to GPIO pin) ADCx_IN12

ADC_CHANNEL_13

ADC external channel (channel connected to GPIO pin) ADCx_IN13

ADC_CHANNEL_14

ADC external channel (channel connected to GPIO pin) ADCx_IN14

ADC_CHANNEL_15

ADC external channel (channel connected to GPIO pin) ADCx_IN15

ADC_CHANNEL_16

ADC external channel (channel connected to GPIO pin) ADCx_IN16

ADC_CHANNEL_17

ADC external channel (channel connected to GPIO pin) ADCx_IN17

ADC_CHANNEL_18

ADC external channel (channel connected to GPIO pin) ADCx_IN18

ADC_CHANNEL_19

ADC external channel (channel connected to GPIO pin) ADCx_IN19

ADC_CHANNEL_VREFINT

ADC internal channel connected to VrefInt: Internal voltage reference, channel specific to ADC3.

ADC_CHANNEL_TEMPSENSOR

ADC internal channel connected to Temperature sensor, channel specific to ADC3.

ADC_CHANNEL_VBAT

ADC internal channel connected to Vbat/4: Vbat voltage through a divider ladder of factor 1/4 to have Vbat always below Vdda, channel specific to ADC3.

ADC_CHANNEL_DAC1CH1_ADC2

ADC internal channel connected to DAC1 channel 1, channel specific to ADC2

ADC_CHANNEL_DAC1CH2_ADC2

ADC internal channel connected to DAC1 channel 2, channel specific to ADC2

Channel - Sampling time

ADC_SAMPLETIME_1CYCLE_5

Sampling time 1.5 ADC clock cycles, On devices STM32H72xx and STM32H73xx, parameter available only on ADC instance: ADC1, ADC2

ADC_SAMPLETIME_2CYCLES_5

Sampling time 2.5 ADC clock cycles, On devices STM32H72xx and STM32H73xx, parameter available only on ADC instance: ADC1, ADC2

ADC_SAMPLETIME_8CYCLES_5

Sampling time 8.5 ADC clock cycles, On devices STM32H72xx and STM32H73xx, parameter available only on ADC instance: ADC1, ADC2

ADC_SAMPLETIME_16CYCLES_5

Sampling time 16.5 ADC clock cycles, On devices STM32H72xx and STM32H73xx, parameter available only on ADC instance: ADC1, ADC2

ADC_SAMPLETIME_32CYCLES_5

Sampling time 32.5 ADC clock cycles, On devices STM32H72xx and STM32H73xx, parameter available only on ADC instance: ADC1, ADC2

ADC_SAMPLETIME_64CYCLES_5

Sampling time 64.5 ADC clock cycles, On devices STM32H72xx and STM32H73xx, parameter available only on ADC instance: ADC1, ADC2

ADC_SAMPLETIME_387CYCLES_5

Sampling time 387.5 ADC clock cycles, On devices STM32H72xx and STM32H73xx, parameter available only on ADC instance: ADC1, ADC2

ADC_SAMPLETIME_810CYCLES_5

Sampling time 810.5 ADC clock cycles, On devices STM32H72xx and STM32H73xx, parameter available only on ADC instance: ADC1, ADC2

Channel - Single or differential ending

ADC_SINGLE_ENDED

ADC channel ending set to single ended (literal also used to set calibration mode)

ADC_DIFFERENTIAL_ENDED

ADC channel ending set to differential (literal also used to set calibration mode)

ADC common - Clock source

ADC_CLOCK_SYNC_PCLK_DIV1

ADC synchronous clock derived from AHB clock without prescaler

ADC_CLOCK_SYNC_PCLK_DIV2

ADC synchronous clock derived from AHB clock with prescaler division by 2

ADC_CLOCK_SYNC_PCLK_DIV4

ADC synchronous clock derived from AHB clock with prescaler division by 4

ADC_CLOCK_ASYNC_DIV1

ADC asynchronous clock without prescaler

ADC_CLOCK_ASYNC_DIV2

ADC asynchronous clock with prescaler division by 2

ADC_CLOCK_ASYNC_DIV4

ADC asynchronous clock with prescaler division by 4

ADC_CLOCK_ASYNC_DIV6

ADC asynchronous clock with prescaler division by 6

ADC_CLOCK_ASYNC_DIV8

ADC asynchronous clock with prescaler division by 8

ADC_CLOCK_ASYNC_DIV10

ADC asynchronous clock with prescaler division by 10

ADC_CLOCK_ASYNC_DIV12

ADC asynchronous clock with prescaler division by 12

ADC_CLOCK_ASYNC_DIV16

ADC asynchronous clock with prescaler division by 16

ADC_CLOCK_ASYNC_DIV32

ADC asynchronous clock with prescaler division by 32

ADC_CLOCK_ASYNC_DIV64

ADC asynchronous clock with prescaler division by 64

ADC_CLOCK_ASYNC_DIV128

ADC asynchronous clock with prescaler division by 128

ADC_CLOCK_ASYNC_DIV256

ADC asynchronous clock with prescaler division by 256

ADC instance - Groups

ADC_REGULAR_GROUP

ADC group regular (available on all STM32 devices)

ADC_INJECTED_GROUP

ADC group injected (not available on all STM32 devices)

ADC_REGULAR_INJECTED_GROUP

ADC both groups regular and injected

Multimode - Mode

ADC_MODE_INDEPENDENT

ADC dual mode disabled (ADC independent mode)

ADC_DUALMODE_REGSIMULT

ADC dual mode enabled: group regular simultaneous

ADC_DUALMODE_INTERL

ADC dual mode enabled: Combined group regular interleaved

ADC_DUALMODE_INJECSIMULT

ADC dual mode enabled: group injected simultaneous

ADC_DUALMODE_ALTERTRIG

ADC dual mode enabled: group injected alternate trigger. Works only with external triggers (not internal SW start)

ADC_DUALMODE_REGSIMULT_INJECSIMULT

ADC dual mode enabled: Combined group regular simultaneous + group injected simultaneous

ADC_DUALMODE_REGSIMULT_ALTERTRIG

ADC dual mode enabled: Combined group regular simultaneous + group injected alternate trigger

ADC_DUALMODE_REGINTERL_INJECSIMULT

ADC dual mode enabled: Combined group regular interleaved + group injected simultaneous

Multimode - Delay between two sampling phases

ADC_TWOSAMPLINGDELAY_1CYCLE

ADC multimode delay between two sampling phases: 1 ADC clock cycle

ADC_TWOSAMPLINGDELAY_2CYCLES

ADC multimode delay between two sampling phases: 2 ADC clock cycles

ADC_TWOSAMPLINGDELAY_3CYCLES

ADC multimode delay between two sampling phases: 3 ADC clock cycles

ADC_TWOSAMPLINGDELAY_4CYCLES

ADC multimode delay between two sampling phases: 4 ADC clock cycles

ADC_TWOSAMPLINGDELAY_5CYCLES

ADC multimode delay between two sampling phases: 5 ADC clock cycles

ADC_TWOSAMPLINGDELAY_6CYCLES

ADC multimode delay between two sampling phases: 6 ADC clock cycles

ADC_TWOSAMPLINGDELAY_7CYCLES

ADC multimode delay between two sampling phases: 7 ADC clock cycles

ADC_TWOSAMPLINGDELAY_8CYCLES

ADC multimode delay between two sampling phases: 8 ADC clock cycles

ADC_TWOSAMPLINGDELAY_9CYCLES

ADC multimode delay between two sampling phases: 9 ADC clock cycles

ADC instance - Offset number

ADC_OFFSET_NONE

ADC offset disabled: no offset correction for the selected ADC channel

ADC_OFFSET_1

ADC offset number 1: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

ADC_OFFSET_2

ADC offset number 2: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

ADC_OFFSET_3

ADC offset number 3: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

ADC_OFFSET_4

ADC offset number 4: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

Oversampling - Discontinuous mode

ADC_TRIGGEREDMODE_SINGLE_TRIGGER

ADC oversampling discontinuous mode: continuous mode (all conversions of oversampling ratio are done from 1 trigger)

ADC_TRIGGEREDMODE_MULTI_TRIGGER

ADC oversampling discontinuous mode: discontinuous mode (each conversion of oversampling ratio needs a trigger)

Oversampling - Oversampling scope for ADC group regular

ADC_REGOVERSAMPLING_CONTINUED_MODE

Oversampling buffer maintained during injection sequence

ADC_REGOVERSAMPLING_RESUMED_MODE

Oversampling buffer zeroed during injection sequence

Oversampling - Data shift

ADC_RIGHTBITSHIFT_NONE

ADC oversampling no shift (sum of the ADC conversions data is not divided to result as the ADC oversampling conversion data)

ADC_RIGHTBITSHIFT_1

ADC oversampling shift of 1 (sum of the ADC conversions data is divided by 2 to result as the ADC oversampling conversion data)

ADC_RIGHTBITSHIFT_2

ADC oversampling shift of 2 (sum of the ADC conversions data is divided by 4 to result as the ADC oversampling conversion data)

ADC_RIGHTBITSHIFT_3

ADC oversampling shift of 3 (sum of the ADC conversions data is divided by 8 to result as the ADC oversampling conversion data)

ADC_RIGHTBITSHIFT_4

ADC oversampling shift of 4 (sum of the ADC conversions data is divided by 16 to result as the ADC oversampling conversion data)

ADC_RIGHTBITSHIFT_5

ADC oversampling shift of 5 (sum of the ADC conversions data is divided by 32 to result as the ADC oversampling conversion data)

ADC_RIGHTBITSHIFT_6

ADC oversampling shift of 6 (sum of the ADC conversions data is divided by 64 to result as the ADC oversampling conversion data)

ADC_RIGHTBITSHIFT_7

ADC oversampling shift of 7 (sum of the ADC conversions data is divided by 128 to result as the ADC oversampling conversion data)

ADC_RIGHTBITSHIFT_8

ADC oversampling shift of 8 (sum of the ADC conversions data is divided by 256 to result as the ADC oversampling conversion data)

ADC_RIGHTBITSHIFT_9

ADC oversampling shift of 9 (sum of the ADC conversions data is divided by 512 to result as the ADC oversampling conversion data)

ADC_RIGHTBITSHIFT_10

ADC oversampling shift of 10 (sum of the ADC conversions data is divided by 1024 to result as the ADC oversampling conversion data)

ADC_RIGHTBITSHIFT_11

ADC oversampling shift of 11 (sum of the ADC conversions data is divided by 2048 to result as the ADC oversampling conversion data)

ADC group regular - DFSDM transfer of ADC conversion data

ADC_DFSDM_MODE_DISABLE

ADC conversions are not transferred by DFSDM.

ADC_DFSDM_MODE_ENABLE

ADC conversion data are transferred to DFSDM for post processing. The ADC conversion data format must be 16-bit signed and right aligned, refer to reference manual. DFSDM transfer cannot be used if DMA transfer is enabled.

ADC group regular - Overrun behavior on conversion data

ADC_OVR_DATA_PRESERVED

ADC group regular behavior in case of overrun: data preserved

ADC_OVR_DATA_OVERWRITTEN

ADC group regular behavior in case of overrun: data overwritten

ADC group regular - Sequencer ranks

ADC_REGULAR_RANK_1

ADC group regular sequencer rank 1

ADC_REGULAR_RANK_2

ADC group regular sequencer rank 2

ADC_REGULAR_RANK_3

ADC group regular sequencer rank 3

ADC_REGULAR_RANK_4

ADC group regular sequencer rank 4

ADC_REGULAR_RANK_5

ADC group regular sequencer rank 5

ADC_REGULAR_RANK_6

ADC group regular sequencer rank 6

ADC_REGULAR_RANK_7

ADC group regular sequencer rank 7

ADC_REGULAR_RANK_8

ADC group regular sequencer rank 8

ADC_REGULAR_RANK_9

ADC group regular sequencer rank 9

ADC_REGULAR_RANK_10

ADC group regular sequencer rank 10

ADC_REGULAR_RANK_11

ADC group regular sequencer rank 11

ADC_REGULAR_RANK_12

ADC group regular sequencer rank 12

ADC_REGULAR_RANK_13

ADC group regular sequencer rank 13

ADC_REGULAR_RANK_14

ADC group regular sequencer rank 14

ADC_REGULAR_RANK_15

ADC group regular sequencer rank 15

ADC_REGULAR_RANK_16

ADC group regular sequencer rank 16

ADC instance - Resolution

ADC_RESOLUTION_16B

ADC resolution 16 bits, On devices STM32H72xx and STM32H73xx, parameter available only on ADC instance: ADC1, ADC2

ADC_RESOLUTION_14B

ADC resolution 14 bits, On devices STM32H72xx and STM32H73xx, parameter available only on ADC instance: ADC1, ADC2

ADC_RESOLUTION_12B

ADC resolution 12 bits

ADC_RESOLUTION_10B

ADC resolution 10 bits

ADC_RESOLUTION_8B

ADC resolution 8 bits

ADC_RESOLUTION_14B_OPT

ADC resolution 14 bits optimized for power consumption, available on for devices revision V only

ADC_RESOLUTION_12B_OPT

ADC resolution 12 bits optimized for power consumption, available on for devices revision V only

HAL ADC macro to manage HAL ADC handle, IT and flags.

__HAL_ADC_RESET_HANDLE_STATE

Description:

- Reset ADC handle state.

Parameters:

- __HANDLE__: ADC handle

Return value:

- None

__HAL_ADC_ENABLE_IT

Description:

- Enable ADC interrupt.

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC Interrupt This parameter can be one of the following values:
  - ADC_IT_RDY ADC Ready interrupt source
  - ADC_IT_EOSMP ADC End of Sampling interrupt source
  - ADC_IT_EOC ADC End of Regular Conversion interrupt source
  - ADC_IT_EOS ADC End of Regular sequence of Conversions interrupt source
  - ADC_IT_OVR ADC overrun interrupt source
  - ADC_IT_JEOC ADC End of Injected Conversion interrupt source
  - ADC_IT_JEOS ADC End of Injected sequence of Conversions interrupt source
  - ADC_IT_AWD1 ADC Analog watchdog 1 interrupt source (main analog watchdog)
  - ADC_IT_AWD2 ADC Analog watchdog 2 interrupt source (additional analog watchdog)
  - ADC_IT_AWD3 ADC Analog watchdog 3 interrupt source (additional analog watchdog)
  - ADC_IT_JQOVF ADC Injected Context Queue Overflow interrupt source.

Return value:

- None

__HAL_ADC_DISABLE_IT

Description:

- Disable ADC interrupt.

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC Interrupt This parameter can be one of the following values:
  - ADC_IT_RDY ADC Ready interrupt source
  - ADC_IT_EOSMP ADC End of Sampling interrupt source
  - ADC_IT_EOC ADC End of Regular Conversion interrupt source
  - ADC_IT_EOS ADC End of Regular sequence of Conversions interrupt source
  - ADC_IT_OVR ADC overrun interrupt source
  - ADC_IT_JEOC ADC End of Injected Conversion interrupt source
  - ADC_IT_JEOS ADC End of Injected sequence of Conversions interrupt source
  - ADC_IT_AWD1 ADC Analog watchdog 1 interrupt source (main analog watchdog)
  - ADC_IT_AWD2 ADC Analog watchdog 2 interrupt source (additional analog watchdog)
  - ADC_IT_AWD3 ADC Analog watchdog 3 interrupt source (additional analog watchdog)
  - ADC_IT_JQOVF ADC Injected Context Queue Overflow interrupt source.

Return value:

- None

__HAL_ADC_GET_IT_SOURCE

Description:

- Checks if the specified ADC interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC interrupt source to check This parameter can be one of the following values:
  - ADC_IT_RDY ADC Ready interrupt source
  - ADC_IT_EOSMP ADC End of Sampling interrupt source
  - ADC_IT_EOC ADC End of Regular Conversion interrupt source
  - ADC_IT_EOS ADC End of Regular sequence of Conversions interrupt source
  - ADC_IT_OVR ADC overrun interrupt source
  - ADC_IT_JEOC ADC End of Injected Conversion interrupt source
  - ADC_IT_JEOS ADC End of Injected sequence of Conversions interrupt source
  - ADC_IT_AWD1 ADC Analog watchdog 1 interrupt source (main analog watchdog)
  - ADC_IT_AWD2 ADC Analog watchdog 2 interrupt source (additional analog watchdog)
  - ADC_IT_AWD3 ADC Analog watchdog 3 interrupt source (additional analog watchdog)
  - ADC_IT_JQOVF ADC Injected Context Queue Overflow interrupt source.

Return value:

- State: of interruption (SET or RESET)

__HAL_ADC_GET_FLAG

Description:

- Check whether the specified ADC flag is set or not.

Parameters:

- __HANDLE__: ADC handle
- __FLAG__: ADC flag This parameter can be one of the following values:
  - ADC_FLAG_RDY ADC Ready flag
  - ADC_FLAG_EOSMP ADC End of Sampling flag
  - ADC_FLAG_EOC ADC End of Regular Conversion flag
  - ADC_FLAG_EOS ADC End of Regular sequence of Conversions flag
  - ADC_FLAG_OVR ADC overrun flag
  - ADC_FLAG_JEOC ADC End of Injected Conversion flag
  - ADC_FLAG_JEOS ADC End of Injected sequence of Conversions flag
  - ADC_FLAG_AWD1 ADC Analog watchdog 1 flag (main analog watchdog)
  - ADC_FLAG_AWD2 ADC Analog watchdog 2 flag (additional analog watchdog)
  - ADC_FLAG_AWD3 ADC Analog watchdog 3 flag (additional analog watchdog)
  - ADC_FLAG_JQOVF ADC Injected Context Queue Overflow flag
  - ADC_FLAG_LDORDY ADC LDO output voltage ready bit.

Return value:

- State: of flag (TRUE or FALSE).

__HAL_ADC_CLEAR_FLAG

Description:

- Clear the specified ADC flag.

Parameters:

- __HANDLE__: ADC handle
- __FLAG__: ADC flag This parameter can be one of the following values:
  - ADC_FLAG_RDY ADC Ready flag
  - ADC_FLAG_EOSMP ADC End of Sampling flag
  - ADC_FLAG_EOC ADC End of Regular Conversion flag
  - ADC_FLAG_EOS ADC End of Regular sequence of Conversions flag
  - ADC_FLAG_OVR ADC overrun flag
  - ADC_FLAG_JEOC ADC End of Injected Conversion flag
  - ADC_FLAG_JEOS ADC End of Injected sequence of Conversions flag
  - ADC_FLAG_AWD1 ADC Analog watchdog 1 flag (main analog watchdog)
  - ADC_FLAG_AWD2 ADC Analog watchdog 2 flag (additional analog watchdog)
  - ADC_FLAG_AWD3 ADC Analog watchdog 3 flag (additional analog watchdog)
  - ADC_FLAG_JQOVF ADC Injected Context Queue Overflow flag.

Return value:

- None

HAL ADC helper macro

__HAL_ADC_CHANNEL_TO_DECIMAL_NB

Description:

- Helper macro to get ADC channel number in decimal format from literals ADC_CHANNEL_x.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - ADC_CHANNEL_0 (3)
  - ADC_CHANNEL_1 (3)
  - ADC_CHANNEL_2 (3)
  - ADC_CHANNEL_3 (3)
  - ADC_CHANNEL_4 (3)
  - ADC_CHANNEL_5 (3)
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8
  - ADC_CHANNEL_9
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15
  - ADC_CHANNEL_16
  - ADC_CHANNEL_17
  - ADC_CHANNEL_18
  - ADC_CHANNEL_VREFINT (1)
  - ADC_CHANNEL_TEMPSENSOR (1)
  - ADC_CHANNEL_VBAT (1)
  - ADC_CHANNEL_DAC1CH1_ADC2 (2)
  - ADC_CHANNEL_DAC1CH2_ADC2 (2)

Return value:

- Value: between Min_Data=0 and Max_Data=18

Notes:

- Example: __HAL_ADC_CHANNEL_TO_DECIMAL_NB(ADC_CHANNEL_4) will return decimal number "4". The input can be a value from functions where a channel number is returned, either defined with number or with bitfield (only one bit must be set).

__HAL_ADC_DECIMAL_NB_TO_CHANNEL

Description:

- Helper macro to get ADC channel in literal format ADC_CHANNEL_x from number in decimal format.

Parameters:

- __DECIMAL_NB__: Value between Min_Data=0 and Max_Data=18

Return value:

- Returned: value can be one of the following values:
  - ADC_CHANNEL_0 (3)
  - ADC_CHANNEL_1 (3)
  - ADC_CHANNEL_2 (3)
  - ADC_CHANNEL_3 (3)
  - ADC_CHANNEL_4 (3)
  - ADC_CHANNEL_5 (3)
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8
  - ADC_CHANNEL_9
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15
  - ADC_CHANNEL_16
  - ADC_CHANNEL_17
  - ADC_CHANNEL_18
  - ADC_CHANNEL_VREFINT (1)
  - ADC_CHANNEL_TEMPSENSOR (1)
  - ADC_CHANNEL_VBAT (1)
  - ADC_CHANNEL_DAC1CH1_ADC2 (2)
  - ADC_CHANNEL_DAC1CH2_ADC2 (2)

Notes:

- Example: __HAL_ADC_DECIMAL_NB_TO_CHANNEL(4) will return a data equivalent to "ADC_CHANNEL_4".

__HAL_ADC_IS_CHANNEL_INTERNAL

Description:

- Helper macro to determine whether the selected channel corresponds to literal definitions of driver.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - ADC_CHANNEL_0 (3)
  - ADC_CHANNEL_1 (3)
  - ADC_CHANNEL_2 (3)
  - ADC_CHANNEL_3 (3)
  - ADC_CHANNEL_4 (3)
  - ADC_CHANNEL_5 (3)
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8
  - ADC_CHANNEL_9
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15
  - ADC_CHANNEL_16
  - ADC_CHANNEL_17
  - ADC_CHANNEL_18
  - ADC_CHANNEL_VREFINT (1)
  - ADC_CHANNEL_TEMPSENSOR (1)
  - ADC_CHANNEL_VBAT (1)
  - ADC_CHANNEL_DAC1CH1_ADC2 (2)
  - ADC_CHANNEL_DAC1CH2_ADC2 (2)

Return value:

- Value: "0" if the channel corresponds to a parameter definition of a ADC external channel (channel connected to a GPIO pin). Value "1" if the channel corresponds to a parameter definition of a ADC internal channel.

Notes:

- The different literal definitions of ADC channels are: ADC internal channel: ADC_CHANNEL_VREFINT, ADC_CHANNEL_TEMPSENSOR, ...ADC external channel (channel connected to a GPIO pin): ADC_CHANNEL_1, ADC_CHANNEL_2, ... The channel parameter must be a value defined from literal definition of a ADC internal channel (ADC_CHANNEL_VREFINT, ADC_CHANNEL_TEMPSENSOR, ...), ADC external channel (ADC_CHANNEL_1, ADC_CHANNEL_2, ...), must not be a value from functions where a channel number is returned from ADC registers, because internal and external channels share the same channel number in ADC registers. The differentiation is made only with parameters definitions of driver.

__HAL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL

Description:

- Helper macro to convert a channel defined from parameter definition of a ADC internal channel (ADC_CHANNEL_VREFINT, ADC_CHANNEL_TEMPSENSOR, ...), to its equivalent parameter definition of a ADC external channel (ADC_CHANNEL_1, ADC_CHANNEL_2, ...).

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - ADC_CHANNEL_0 (3)
  - ADC_CHANNEL_1 (3)
  - ADC_CHANNEL_2 (3)
  - ADC_CHANNEL_3 (3)
  - ADC_CHANNEL_4 (3)
  - ADC_CHANNEL_5 (3)
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8
  - ADC_CHANNEL_9
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15
  - ADC_CHANNEL_16
  - ADC_CHANNEL_17
  - ADC_CHANNEL_18
  - ADC_CHANNEL_VREFINT (1)
  - ADC_CHANNEL_TEMPSENSOR (1)
  - ADC_CHANNEL_VBAT (1)
  - ADC_CHANNEL_DAC1CH1_ADC2 (2)
  - ADC_CHANNEL_DAC1CH2_ADC2 (2)

Return value:

- Returned: value can be one of the following values:
  - ADC_CHANNEL_0
  - ADC_CHANNEL_1
  - ADC_CHANNEL_2
  - ADC_CHANNEL_3
  - ADC_CHANNEL_4
  - ADC_CHANNEL_5
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8
  - ADC_CHANNEL_9
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15
  - ADC_CHANNEL_16
  - ADC_CHANNEL_17
  - ADC_CHANNEL_18

Notes:

- The channel parameter can be, additionally to a value defined from parameter definition of a ADC internal channel (ADC_CHANNEL_VREFINT, ADC_CHANNEL_TEMPSENSOR, ...), a value defined from parameter definition of ADC external channel (ADC_CHANNEL_1, ADC_CHANNEL_2, ...) or a value from functions where a channel number is returned from ADC registers.

__HAL_ADC_IS_CHANNEL_INTERNAL_AVAILABLE

Description:

- Helper macro to determine whether the internal channel selected is available on the ADC instance selected.

Parameters:

- __ADC_INSTANCE__: ADC instance
- __CHANNEL__: This parameter can be one of the following values:
  - ADC_CHANNEL_VREFINT (1)
  - ADC_CHANNEL_TEMPSENSOR (1)
  - ADC_CHANNEL_VBAT (1)
  - ADC_CHANNEL_DAC1CH1_ADC2 (2)
  - ADC_CHANNEL_DAC1CH2_ADC2 (2)

Return value:

- Value: "0" if the internal channel selected is not available on the ADC instance selected. Value "1" if the internal channel selected is available on the ADC instance selected.

Notes:

- The channel parameter must be a value defined from parameter definition of a ADC internal channel (ADC_CHANNEL_VREFINT, ADC_CHANNEL_TEMPSENSOR, ...), must not be a value defined from parameter definition of ADC external channel (ADC_CHANNEL_1, ADC_CHANNEL_2, ...) or a value from functions where a channel number is returned from ADC registers, because internal and external channels share the same channel number in ADC registers. The differentiation is made only with parameters definitions of driver.

__HAL_ADC_MULTI_CONV_DATA_MASTER_SLAVE

Description:

- Helper macro to get the ADC multimode conversion data of ADC master or ADC slave from raw value with both ADC conversion data concatenated.

Parameters:

- __ADC_MULTI_MASTER_SLAVE__: This parameter can be one of the following values:
  - LL_ADC_MULTI_MASTER
  - LL_ADC_MULTI_SLAVE
- __ADC_MULTI_CONV_DATA__: Value between Min_Data=0x000 and Max_Data=0xFFF

Return value:

- Value: between Min_Data=0x000 and Max_Data=0xFFF

Notes:

- This macro is intended to be used when multimode transfer by DMA is enabled: refer to function LL_ADC_SetMultiDMATransfer(). In this case the transferred data need to processed with this macro to separate the conversion data of ADC master and ADC slave.

__HAL_ADC_COMMON_INSTANCE

Description:

- Helper macro to select the ADC common instance to which is belonging the selected ADC instance.

Parameters:

- __ADCx__: ADC instance

Return value:

- ADC: common register instance

Notes:

- ADC common register instance can be used for: Set parameters common to several ADC instancesMultimode (for devices with several ADC instances) Refer to functions having argument "ADCxy_COMMON" as parameter.

__HAL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE

Description:

- Helper macro to check if all ADC instances sharing the same ADC common instance are disabled.

Parameters:

- __ADCXY_COMMON__: ADC common instance (can be set directly from CMSIS definition or by using helper macro

Return value:

- Value: "0" if all ADC instances sharing the same ADC common instance are disabled. Value "1" if at least one ADC instance sharing the same ADC common instance is enabled.

Notes:

- This check is required by functions with setting conditioned to ADC state: All ADC instances of the ADC common group must be disabled. Refer to functions having argument "ADCxy_COMMON" as parameter. On devices with only 1 ADC common instance, parameter of this macro is useless and can be ignored (parameter kept for compatibility with devices featuring several ADC common instances).

__HAL_ADC_DIGITAL_SCALE

Description:

- Helper macro to define the ADC conversion data full-scale digital value corresponding to the selected ADC resolution.

Parameters:

- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - ADC_RESOLUTION_16B
  - ADC_RESOLUTION_14B
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B

Return value:

- ADC: conversion data full-scale digital value

Notes:

- ADC conversion data full-scale corresponds to voltage range determined by analog voltage references Vref+ and Vref- (refer to reference manual).

__HAL_ADC_CONVERT_DATA_RESOLUTION

Description:

- Helper macro to convert the ADC conversion data from a resolution to another resolution.

Parameters:

- __DATA__: ADC conversion data to be converted
- __ADC_RESOLUTION_CURRENT__: Resolution of to the data to be converted This parameter can be one of the following values:
  - ADC_RESOLUTION_16B
  - ADC_RESOLUTION_14B
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
- __ADC_RESOLUTION_TARGET__: Resolution of the data after conversion This parameter can be one of the following values:
  - ADC_RESOLUTION_16B
  - ADC_RESOLUTION_14B
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B

Return value:

- ADC: conversion data to the requested resolution

__HAL_ADC_CALC_DATA_TO_VOLTAGE

Description:

- Helper macro to calculate the voltage (unit: mVolt) corresponding to a ADC conversion data (unit: digital value).

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __ADC_DATA__: ADC conversion data (resolution 12 bits) (unit: digital value).
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - ADC_RESOLUTION_16B
  - ADC_RESOLUTION_14B
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B

Return value:

- ADC: conversion data equivalent voltage value (unit: mVolt)

Notes:

- Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE().

__HAL_ADC_CALC_VREFANALOG_VOLTAGE

Description:

- Helper macro to calculate analog reference voltage (Vref+) (unit: mVolt) from ADC conversion data of internal voltage reference VrefInt.

Parameters:

- __VREFINT_ADC_DATA__: ADC conversion data (resolution 12 bits) of internal voltage reference VrefInt (unit: digital value).
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - ADC_RESOLUTION_16B
  - ADC_RESOLUTION_14B
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B

Return value:

- Analog: reference voltage (unit: mV)

Notes:

- Computation is using VrefInt calibration value stored in system memory for each device during production. This voltage depends on user board environment: voltage level connected to pin Vref+. On devices with small package, the pin Vref+ is not present and internally bonded to pin Vdda. On this STM32 series, calibration data of internal voltage reference VrefInt corresponds to a resolution of 12 bits, this is the recommended ADC resolution to convert voltage of internal voltage reference VrefInt. Otherwise, this macro performs the processing to scale ADC conversion data to 12 bits.

__HAL_ADC_CALC_TEMPERATURE

Description:

- Helper macro to calculate the temperature (unit: degree Celsius) from ADC conversion data of internal temperature sensor.

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __TEMPSENSOR_ADC_DATA__: ADC conversion data of internal temperature sensor (unit: digital value).
- __ADC_RESOLUTION__: ADC resolution at which internal temperature sensor voltage has been measured. This parameter can be one of the following values:
  - ADC_RESOLUTION_16B
  - ADC_RESOLUTION_14B
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B

Return value:

- Temperature: (unit: degree Celsius)

Notes:

- Computation is using temperature sensor calibration values stored in system memory for each device during production. Calculation formula: Temperature = ((TS_ADC_DATA - TS_CAL1) * (TS_CAL2_TEMP - TS_CAL1_TEMP)) / (TS_CAL2 - TS_CAL1) + TS_CAL1_TEMP with TS_ADC_DATA = temperature sensor raw data measured by ADC Avg_Slope = (TS_CAL2 - TS_CAL1) / (TS_CAL2_TEMP - TS_CAL1_TEMP) TS_CAL1 = equivalent TS_ADC_DATA at temperature TEMP_DEGC_CAL1 (calibrated in factory) TS_CAL2 = equivalent TS_ADC_DATA at temperature TEMP_DEGC_CAL2 (calibrated in factory) Caution: Calculation relevancy under reserve that calibration parameters are correct (address and data). To calculate temperature using temperature sensor datasheet typical values (generic values less, therefore less accurate than calibrated values), use helper macro __LL_ADC_CALC_TEMPERATURE_TYP_PARAMS(). As calculation input, the analog reference voltage (Vref+) must be defined as it impacts the ADC LSB equivalent voltage. Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE(). On this STM32 series, calibration data of temperature sensor corresponds to a resolution of 12 bits, this is the recommended ADC resolution to convert voltage of temperature sensor. Otherwise, this macro performs the processing to scale ADC conversion data to 12 bits.

__HAL_ADC_CALC_TEMPERATURE_TYP_PARAMS

Description:

- Helper macro to calculate the temperature (unit: degree Celsius) from ADC conversion data of internal temperature sensor.

Parameters:

- __TEMPSENSOR_TYP_AVGSLOPE__: Device datasheet data: Temperature sensor slope typical value (unit: uV/DegCelsius). On STM32H7, refer to device datasheet parameter "Avg_Slope".
- __TEMPSENSOR_TYP_CALX_V__: Device datasheet data: Temperature sensor voltage typical value (at temperature and Vref+ defined in parameters below) (unit: mV). On STM32H7, refer to device datasheet parameter "V30" (corresponding to TS_CAL1).
- __TEMPSENSOR_CALX_TEMP__: Device datasheet data: Temperature at which temperature sensor voltage (see parameter above) is corresponding (unit: mV)
- __VREFANALOG_VOLTAGE__: Analog voltage reference (Vref+) voltage (unit: mV)
- __TEMPSENSOR_ADC_DATA__: ADC conversion data of internal temperature sensor (unit: digital value).
- __ADC_RESOLUTION__: ADC resolution at which internal temperature sensor voltage has been measured. This parameter can be one of the following values:
  - ADC_RESOLUTION_16B
  - ADC_RESOLUTION_14B
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B

Return value:

- Temperature: (unit: degree Celsius)

Notes:

- Computation is using temperature sensor typical values (refer to device datasheet). Calculation formula: Temperature = (TS_TYP_CALx_VOLT(uV) - TS_ADC_DATA * Conversion_uV) / Avg_Slope + CALx_TEMP with TS_ADC_DATA = temperature sensor raw data measured by ADC (unit: digital value) Avg_Slope = temperature sensor slope (unit: uV/Degree Celsius) TS_TYP_CALx_VOLT = temperature sensor digital value at temperature CALx_TEMP (unit: mV) Caution: Calculation relevancy under reserve the temperature sensor of the current device has characteristics in line with datasheet typical values. If temperature sensor calibration values are available on on this device (presence of macro __LL_ADC_CALC_TEMPERATURE()), temperature calculation will be more accurate using helper macro __LL_ADC_CALC_TEMPERATURE(). As calculation input, the analog reference voltage (Vref+) must be defined as it impacts the ADC LSB equivalent voltage. Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE(). ADC measurement data must correspond to a resolution of 12bits (full scale digital value 4095). If not the case, the data must be preliminarily rescaled to an equivalent resolution of 12 bits.

ADC group injected trigger edge (when external trigger is selected)

ADC_EXTERNALTRIGINJECCONV_EDGE_NONE

Injected conversions hardware trigger detection disabled

ADC_EXTERNALTRIGINJECCONV_EDGE_RISING

Injected conversions hardware trigger detection on the rising edge

ADC_EXTERNALTRIGINJECCONV_EDGE_FALLING

Injected conversions hardware trigger detection on the falling edge

ADC_EXTERNALTRIGINJECCONV_EDGE_RISINGFALLING

Injected conversions hardware trigger detection on both the rising and falling edges

ADC group injected trigger source

ADC_INJECTED_SOFTWARE_START

Software triggers injected group conversion start

ADC_EXTERNALTRIGINJEC_T1_TRGO

ADC group injected conversion trigger from external peripheral: TIM1 TRGO event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T1_CC4

ADC group injected conversion trigger from external peripheral: TIM1 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T2_TRGO

ADC group injected conversion trigger from external peripheral: TIM2 TRGO event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T2_CC1

ADC group injected conversion trigger from external peripheral: TIM2 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T3_CC4

ADC group injected conversion trigger from external peripheral: TIM3 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T4_TRGO

ADC group injected conversion trigger from external peripheral: TIM4 TRGO event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_EXT_IT15

ADC group injected conversion trigger from external peripheral: external interrupt line 15. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T8_CC4

ADC group injected conversion trigger from external peripheral: TIM8 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T1_TRGO2

ADC group injected conversion trigger from external peripheral: TIM1 TRGO2 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T8_TRGO

ADC group injected conversion trigger from external peripheral: TIM8 TRGO event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T8_TRGO2

ADC group injected conversion trigger from external peripheral: TIM8 TRGO2 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T3_CC3

ADC group injected conversion trigger from external peripheral: TIM3 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T3_TRGO

ADC group injected conversion trigger from external peripheral: TIM3 TRGO event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T3_CC1

ADC group injected conversion trigger from external peripheral: TIM3 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T6_TRGO

ADC group injected conversion trigger from external peripheral: TIM6 TRGO event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T15_TRGO

ADC group injected conversion trigger from external peripheral: TIM15 TRGO event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_HR1_ADCTRG2

ADC group injected conversion trigger from external peripheral: HRTIM1 TRG2 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_HR1_ADCTRG4

ADC group injected conversion trigger from external peripheral: HRTIM1 TRG4 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_LPTIM1_OUT

ADC group injected conversion trigger from external peripheral: LPTIM1 OUT event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_LPTIM2_OUT

ADC group injected conversion trigger from external peripheral: LPTIM2 OUT event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_LPTIM3_OUT

ADC group injected conversion trigger from external peripheral: LPTIM3 OUT event. Trigger edge set to rising edge (default setting).

ADC group injected - Sequencer ranks

ADC_INJECTED_RANK_1

ADC group injected sequencer rank 1

ADC_INJECTED_RANK_2

ADC group injected sequencer rank 2

ADC_INJECTED_RANK_3

ADC group injected sequencer rank 3

ADC_INJECTED_RANK_4

ADC group injected sequencer rank 4

ADC interrupts definition

ADC_IT_RDY

ADC Ready interrupt source

ADC_IT_EOSMP

ADC End of sampling interrupt source

ADC_IT_EOC

ADC End of regular conversion interrupt source

ADC_IT_EOS

ADC End of regular sequence of conversions interrupt source

ADC_IT_OVR

ADC overrun interrupt source

ADC_IT_JEOC

ADC End of injected conversion interrupt source

ADC_IT_JEOS

ADC End of injected sequence of conversions interrupt source

ADC_IT_AWD1

ADC Analog watchdog 1 interrupt source (main analog watchdog)

ADC_IT_AWD2

ADC Analog watchdog 2 interrupt source (additional analog watchdog)

ADC_IT_AWD3

ADC Analog watchdog 3 interrupt source (additional analog watchdog)

ADC_IT_JQOVF

ADC Injected Context Queue Overflow interrupt source

ADC_IT_AWD

ADC Analog watchdog 1 interrupt source: naming for compatibility with other STM32 devices having only one analog watchdog

ADC group regular trigger edge (when external trigger is selected)

ADC_EXTERNALTRIGCONVEDGE_NONE

Regular conversions hardware trigger detection disabled

ADC_EXTERNALTRIGCONVEDGE_RISING

ADC group regular conversion trigger polarity set to rising edge

ADC_EXTERNALTRIGCONVEDGE_FALLING

ADC group regular conversion trigger polarity set to falling edge

ADC_EXTERNALTRIGCONVEDGE_RISINGFALLING

ADC group regular conversion trigger polarity set to both rising and falling edges

ADC group regular trigger source

ADC_SOFTWARE_START

ADC group regular conversion trigger internal: SW start.

ADC_EXTERNALTRIG_T1_CC1

ADC group regular conversion trigger from external peripheral: TIM1 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T1_CC2

ADC group regular conversion trigger from external peripheral: TIM1 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T1_CC3

ADC group regular conversion trigger from external peripheral: TIM1 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T2_CC2

ADC group regular conversion trigger from external peripheral: TIM2 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T3_TRGO

ADC group regular conversion trigger from external peripheral: TIM3 TRGO event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T4_CC4

ADC group regular conversion trigger from external peripheral: TIM4 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_EXT_IT11

ADC group regular conversion trigger from external peripheral: external interrupt line 11 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T8_TRGO

ADC group regular conversion trigger from external peripheral: TIM8 TRGO event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T8_TRGO2

ADC group regular conversion trigger from external peripheral: TIM8 TRGO2 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T1_TRGO

ADC group regular conversion trigger from external peripheral: TIM1 TRGO event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T1_TRGO2

ADC group regular conversion trigger from external peripheral: TIM1 TRGO2 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T2_TRGO

ADC group regular conversion trigger from external peripheral: TIM2 TRGO event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T4_TRGO

ADC group regular conversion trigger from external peripheral: TIM4 TRGO event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T6_TRGO

ADC group regular conversion trigger from external peripheral: TIM6 TRGO event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T15_TRGO

ADC group regular conversion trigger from external peripheral: TIM15 TRGO event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T3_CC4

ADC group regular conversion trigger from external peripheral: TIM3 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_HR1_ADCTRG1

ADC group regular conversion trigger from external peripheral: HRTIM TRG1 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_HR1_ADCTRG3

ADC group regular conversion trigger from external peripheral: HRTIM TRG3 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_LPTIM1_OUT

ADC group regular conversion trigger from external peripheral: LPTIM1 OUT event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_LPTIM2_OUT

ADC group regular conversion trigger from external peripheral: LPTIM2 OUT event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_LPTIM3_OUT

ADC group regular conversion trigger from external peripheral: LPTIM3 event OUT. Trigger edge set to rising edge (default setting).

ADC sequencer scan mode

ADC_SCAN_DISABLE

Scan mode disabled

ADC_SCAN_ENABLE

Scan mode enabled

ADCx SMPR1 fields

ADC_SMPR1_FIELDS

ADC States

HAL_ADC_STATE_RESET

Notes:

- ADC state machine is managed by bitfields, state must be compared with bit by bit.
  For example: " if ((HAL_ADC_GetState(hadc1) & HAL_ADC_STATE_REG_BUSY) != 0UL) " " if ((HAL_ADC_GetState(hadc1) & HAL_ADC_STATE_AWD1) != 0UL) " ADC not yet initialized or disabled

HAL_ADC_STATE_READY

ADC peripheral ready for use

HAL_ADC_STATE_BUSY_INTERNAL

ADC is busy due to an internal process (initialization, calibration)

HAL_ADC_STATE_TIMEOUT

TimeOut occurrence

HAL_ADC_STATE_ERROR_INTERNAL

Internal error occurrence

HAL_ADC_STATE_ERROR_CONFIG

Configuration error occurrence

HAL_ADC_STATE_ERROR_DMA

DMA error occurrence

HAL_ADC_STATE_REG_BUSY

A conversion on ADC group regular is ongoing or can occur (either by continuous mode, external trigger, low power auto power-on (if feature available), multimode ADC master control (if feature available))

HAL_ADC_STATE_REG_EOC

Conversion data available on group regular

HAL_ADC_STATE_REG_OVR

Overrun occurrence

HAL_ADC_STATE_REG_EOSMP

Not available on this STM32 series: End Of Sampling flag raised

HAL_ADC_STATE_INJ_BUSY

A conversion on ADC group injected is ongoing or can occur (either by auto-injection mode, external trigger, low power auto power-on (if feature available), multimode ADC master control (if feature available))

HAL_ADC_STATE_INJ_EOC

Conversion data available on group injected

HAL_ADC_STATE_INJ_JQOVF

Injected queue overflow occurrence

HAL_ADC_STATE_AWD1

Out-of-window occurrence of ADC analog watchdog 1

HAL_ADC_STATE_AWD2

Out-of-window occurrence of ADC analog watchdog 2

HAL_ADC_STATE_AWD3

Out-of-window occurrence of ADC analog watchdog 3

HAL_ADC_STATE_MULTIMODE_SLAVE

ADC in multimode slave state, controlled by another ADC master (when feature available)

9 HAL ADC Extension Driver

9.1 ADCEx Firmware driver registers structures

9.1.1 ADC_InjOversamplingTypeDef

ADC_InjOversamplingTypeDef is defined in the stm32h7xx_hal_adc_ex.h

Data Fields

- ***uint32_t Ratio***
- ***uint32_t RightBitShift***

Field Documentation

- ***uint32_t ADC_InjOversamplingTypeDef::Ratio***
  Configures the oversampling ratio.
- ***uint32_t ADC_InjOversamplingTypeDef::RightBitShift***
  Configures the division coefficient for the Oversampler. This parameter can be a value of ***ADC_HAL_EC_OVS_SHIFT***

9.1.2 ADC_InjectionConfTypeDef

ADC_InjectionConfTypeDef is defined in the stm32h7xx_hal_adc_ex.h

Data Fields

- ***uint32_t InjectedChannel***
- ***uint32_t InjectedRank***
- ***uint32_t InjectedSamplingTime***
- ***uint32_t InjectedSingleDiff***
- ***uint32_t InjectedOffsetNumber***
- ***uint32_t InjectedOffset***
- ***uint32_t InjectedOffsetRightShift***
- ***FunctionalState InjectedOffsetSignedSaturation***
- ***uint32_t InjectedNbrOfConversion***
- ***FunctionalState InjectedDiscontinuousConvMode***
- ***FunctionalState AutoInjectedConv***
- ***FunctionalState QueueInjectedContext***
- ***uint32_t ExternalTrigInjecConv***
- ***uint32_t ExternalTrigInjecConvEdge***
- ***FunctionalState InjecOversamplingMode***
- ***ADC_InjOversamplingTypeDef InjecOversampling***

Field Documentation

- ***uint32_t ADC_InjectionConfTypeDef::InjectedChannel***
  Specifies the channel to configure into ADC group injected. This parameter can be a value of ***ADC_HAL_EC_CHANNEL*** Note: Depending on devices and ADC instances, some channels may not be available on device package pins. Refer to device datasheet for channels availability.
- ***uint32_t ADC_InjectionConfTypeDef::InjectedRank***
  Specifies the rank in the ADC group injected sequencer. This parameter must be a value of ***ADC_INJ_SEQ_RANKS***. Note: to disable a channel or change order of conversion sequencer, rank containing a previous channel setting can be overwritten by the new channel setting (or parameter number of conversions adjusted)

- ***uint32_t ADC_InjectionConfTypeDef::InjectedSamplingTime***

  Sampling time value to be set for the selected channel. Unit: ADC clock cycles. Conversion time is the addition of sampling time and processing time (12.5 ADC clock cycles at ADC resolution 12 bits, 10.5 cycles at 10 bits, 8.5 cycles at 8 bits, 6.5 cycles at 6 bits). This parameter can be a value of ***ADC_HAL_EC_CHANNEL_SAMPLINGTIME***. Caution: This parameter applies to a channel that can be used in a regular and/or injected group. It overwrites the last setting. Note: In case of usage of internal measurement channels (VrefInt/Vbat/TempSensor), sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting) Refer to device datasheet for timings values.

- ***uint32_t ADC_InjectionConfTypeDef::InjectedSingleDiff***

  Selection of single-ended or differential input. In differential mode: Differential measurement is between the selected channel 'i' (positive input) and channel 'i+1' (negative input). Only channel 'i' has to be configured, channel 'i+1' is configured automatically. This parameter must be a value of ***ADC_HAL_EC_CHANNEL_SINGLE_DIFF_ENDING***. Caution: This parameter applies to a channel that can be used in a regular and/or injected group. It overwrites the last setting. Note: Refer to Reference Manual to ensure the selected channel is available in differential mode. Note: When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately. Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion). If ADC is enabled, this parameter setting is bypassed without error reporting (as it can be the expected behavior in case of another parameter update on the fly)

- ***uint32_t ADC_InjectionConfTypeDef::InjectedOffsetNumber***

  Selects the offset number. This parameter can be a value of ***ADC_HAL_EC_OFFSET_NB***. Caution: Only one offset is allowed per channel. This parameter overwrites the last setting.

- ***uint32_t ADC_InjectionConfTypeDef::InjectedOffset***

  Defines the offset to be subtracted from the raw converted data. Offset value must be a positive number. Maximum value depends on ADC resolution and oversampling ratio (in case of oversampling used). This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x3FFFC00 (corresponding to resolution 16 bit and oversampling ratio 1024). Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).

- ***uint32_t ADC_InjectionConfTypeDef::InjectedOffsetRightShift***

  Specifies whether the 1 bit Right-shift feature is used or not. This parameter is applied only for 16-bit or 8-bit resolution. This parameter can be set to ENABLE or DISABLE.

- ***FunctionalState ADC_InjectionConfTypeDef::InjectedOffsetSignedSaturation***

  Specifies whether the Signed saturation feature is used or not. This parameter is applied only for 16-bit or 8-bit resolution. This parameter can be set to ENABLE or DISABLE.

- ***uint32_t ADC_InjectionConfTypeDef::InjectedNbrOfConversion***

  Specifies the number of ranks that will be converted within the ADC group injected sequencer. To use the injected group sequencer and convert several ranks, parameter 'ScanConvMode' must be enabled. This parameter must be a number between Min_Data = 1 and Max_Data = 4. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.

- ***FunctionalState ADC_InjectionConfTypeDef::InjectedDiscontinuousConvMode***

  Specifies whether the conversions sequence of ADC group injected is performed in Complete-sequence/ Discontinuous-sequence (main sequence subdivided in successive parts). Discontinuous mode is used only if sequencer is enabled (parameter 'ScanConvMode'). If sequencer is disabled, this parameter is discarded. Discontinuous mode can be enabled only if continuous mode is disabled. This parameter can be set to ENABLE or DISABLE. Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion). Note: For injected group, discontinuous mode converts the sequence channel by channel (discontinuous length fixed to 1 rank). Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.

- ***FunctionalState ADC_InjectionConfTypeDef::AutoInjectedConv***

  Enables or disables the selected ADC group injected automatic conversion after regular one This parameter can be set to ENABLE or DISABLE. Note: To use Automatic injected conversion, discontinuous mode must be disabled ('DiscontinuousConvMode' and 'InjectedDiscontinuousConvMode' set to DISABLE) Note: To use Automatic injected conversion, injected group external triggers must be disabled ('ExternalTrigInjecConv' set to ADC_INJECTED_SOFTWARE_START) Note: In case of DMA used with regular group: if DMA configured in normal mode (single shot) JAUTO will be stopped upon DMA transfer complete. To maintain JAUTO always enabled, DMA must be configured in circular mode. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***FunctionalState ADC_InjectionConfTypeDef::QueueInjectedContext***

  Specifies whether the context queue feature is enabled. This parameter can be set to ENABLE or DISABLE. If context queue is enabled, injected sequencer&channels configurations are queued on up to 2 contexts. If a new injected context is set when queue is full, error is triggered by interruption and through function 'HAL_ADCEx_InjectedQueueOverflowCallback'. Caution: This feature request that the sequence is fully configured before injected conversion start. Therefore, configure channels with as many calls to **HAL_ADCEx_InjectedConfigChannel()** as the 'InjectedNbrOfConversion' parameter. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set. Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion).
- ***uint32_t ADC_InjectionConfTypeDef::ExternalTrigInjecConv***

  Selects the external event used to trigger the conversion start of injected group. If set to ADC_INJECTED_SOFTWARE_START, external triggers are disabled and software trigger is used instead. This parameter can be a value of ***ADC_injected_external_trigger_source***. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***uint32_t ADC_InjectionConfTypeDef::ExternalTrigInjecConvEdge***

  Selects the external trigger edge of injected group. This parameter can be a value of ***ADC_injected_external_trigger_edge***. If trigger source is set to ADC_INJECTED_SOFTWARE_START, this parameter is discarded. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***FunctionalState ADC_InjectionConfTypeDef::InjecOversamplingMode***

  Specifies whether the oversampling feature is enabled or disabled. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing (both ADSTART and JADSTART cleared).
- ***ADC_InjOversamplingTypeDef ADC_InjectionConfTypeDef::InjecOversampling***

  Specifies the Oversampling parameters. Caution: this setting overwrites the previous oversampling configuration if oversampling already enabled. Note: This parameter can be modified only if there is no conversion is ongoing (both ADSTART and JADSTART cleared).

9.1.3 ADC_MultiModeTypeDef

ADC_MultiModeTypeDef is defined in the stm32h7xx_hal_adc_ex.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t DualModeData***
- ***uint32_t TwoSamplingDelay***

Field Documentation

- ***uint32_t ADC_MultiModeTypeDef::Mode***

  Configures the ADC to operate in independent or multimode. This parameter can be a value of ***ADC_HAL_EC_MULTI_MODE***.
- ***uint32_t ADC_MultiModeTypeDef::DualModeData***

  Configures the Dual ADC Mode Data Format: This parameter can be a value of ***ADCEx_Dual_Mode_Data_Format***

- ***uint32_t ADC_MultiModeTypeDef::TwoSamplingDelay***

  Configures the Delay between 2 sampling phases. This parameter can be a value of ***ADC_HAL_EC_MULTI_TWOSMP_DELAY***. Delay range depends on selected resolution: from 1 to 9 clock cycles for 16 bits, from 1 to 9 clock cycles for 14 bits from 1 to 8 clock cycles for 12 bits from 1 to 6 clock cycles for 10 bits from 1 to 6 clock cycles for 8 bits

9.2 ADCEx Firmware driver API description

The following section lists the various functions of the ADCEx library.

9.2.1 IO operation functions

This section provides functions allowing to:

- Perform the ADC self-calibration for single or differential ending.
- Get calibration factors for single or differential ending.
- Set calibration factors for single or differential ending.
- Start conversion of ADC group injected.
- Stop conversion of ADC group injected.
- Poll for conversion complete on ADC group injected.
- Get result of ADC group injected channel conversion.
- Start conversion of ADC group injected and enable interruptions.
- Stop conversion of ADC group injected and disable interruptions.
- When multimode feature is available, start multimode and enable DMA transfer.
- Stop multimode and disable ADC DMA transfer.
- Get result of multimode conversion.

This section contains the following APIs:

- ***HAL_ADCEx_Calibration_Start()***
- ***HAL_ADCEx_Calibration_GetValue()***
- ***HAL_ADCEx_LinearCalibration_GetValue()***
- ***HAL_ADCEx_Calibration_SetValue()***
- ***HAL_ADCEx_LinearCalibration_SetValue()***
- ***HAL_ADCEx_LinearCalibration_FactorLoad()***
- ***HAL_ADCEx_InjectedStart()***
- ***HAL_ADCEx_InjectedStop()***
- ***HAL_ADCEx_InjectedPollForConversion()***
- ***HAL_ADCEx_InjectedStart_IT()***
- ***HAL_ADCEx_InjectedStop_IT()***
- ***HAL_ADCEx_MultiModeStart_DMA()***
- ***HAL_ADCEx_MultiModeStop_DMA()***
- ***HAL_ADCEx_MultiModeGetValue()***
- ***HAL_ADCEx_InjectedGetValue()***
- ***HAL_ADCEx_InjectedConvCpltCallback()***
- ***HAL_ADCEx_InjectedQueueOverflowCallback()***
- ***HAL_ADCEx_LevelOutOfWindow2Callback()***
- ***HAL_ADCEx_LevelOutOfWindow3Callback()***
- ***HAL_ADCEx_EndOfSamplingCallback()***
- ***HAL_ADCEx_RegularStop()***
- ***HAL_ADCEx_RegularStop_IT()***
- ***HAL_ADCEx_RegularStop_DMA()***
- ***HAL_ADCEx_RegularMultiModeStop_DMA()***

9.2.2 Peripheral Control functions

This section provides functions allowing to:

- Configure channels on injected group
- Configure multimode when multimode feature is available
- Enable or Disable Injected Queue
- Disable ADC voltage regulator
- Enter ADC deep-power-down mode

This section contains the following APIs:

- ***HAL_ADCEx_InjectedConfigChannel()***
- ***HAL_ADCEx_MultiModeConfigChannel()***
- ***HAL_ADCEx_EnableInjectedQueue()***
- ***HAL_ADCEx_DisableInjectedQueue()***
- ***HAL_ADCEx_DisableVoltageRegulator()***
- ***HAL_ADCEx_EnterADCDeepPowerDownMode()***

9.2.3 Detailed description of functions

HAL_ADCEx_Calibration_Start

Function name

HAL_StatusTypeDef HAL_ADCEx_Calibration_Start (ADC_HandleTypeDef * hadc, uint32_t CalibrationMode, uint32_t SingleDiff)

Function description

Perform an ADC automatic self-calibration Calibration prerequisite: ADC must be disabled (execute this function before HAL_ADC_Start() or after HAL_ADC_Stop()).

Parameters

- **hadc:** ADC handle
- **CalibrationMode:** Selection of calibration offset or linear calibration offset.
  - ADC_CALIB_OFFSET Channel in mode calibration offset
  - ADC_CALIB_OFFSET_LINEARITY Channel in mode linear calibration offset
- **SingleDiff:** Selection of single-ended or differential input This parameter can be one of the following values:
  - ADC_SINGLE_ENDED Channel in mode input single ended
  - ADC_DIFFERENTIAL_ENDED Channel in mode input differential ended

Return values

- **HAL:** status

HAL_ADCEx_Calibration_GetValue

Function name

uint32_t HAL_ADCEx_Calibration_GetValue (ADC_HandleTypeDef * hadc, uint32_t SingleDiff)

Function description

Get the calibration factor.

Parameters

- **hadc:** ADC handle.
- **SingleDiff:** This parameter can be only:
  - ADC_SINGLE_ENDED Channel in mode input single ended
  - ADC_DIFFERENTIAL_ENDED Channel in mode input differential ended

Return values

- **Calibration:** value.

HAL_ADCEx_LinearCalibration_GetValue

Function name

HAL_StatusTypeDef HAL_ADCEx_LinearCalibration_GetValue (ADC_HandleTypeDef * hadc, uint32_t * LinearCalib_Buffer)

Function description

Get the calibration factor from automatic conversion result.

Parameters

- **hadc:** ADC handle
- **LinearCalib_Buffer:** Linear calibration factor

Return values

- **HAL:** state

HAL_ADCEx_Calibration_SetValue

Function name

HAL_StatusTypeDef HAL_ADCEx_Calibration_SetValue (ADC_HandleTypeDef * hadc, uint32_t SingleDiff, uint32_t CalibrationFactor)

Function description

Set the calibration factor to overwrite automatic conversion result.

Parameters

- **hadc:** ADC handle
- **SingleDiff:** This parameter can be only:
  - ADC_SINGLE_ENDED Channel in mode input single ended
  - ADC_DIFFERENTIAL_ENDED Channel in mode input differential ended
- **CalibrationFactor:** Calibration factor (coded on 7 bits maximum)

Return values

- **HAL:** state

HAL_ADCEx_LinearCalibration_SetValue

Function name

HAL_StatusTypeDef HAL_ADCEx_LinearCalibration_SetValue (ADC_HandleTypeDef * hadc, uint32_t * LinearCalib_Buffer)

Function description

Set the linear calibration factor.

Parameters

- **hadc:** ADC handle
- **LinearCalib_Buffer:** Linear calibration factor

Return values

- **HAL:** state

HAL_ADCEx_LinearCalibration_FactorLoad

Function name

HAL_StatusTypeDef HAL_ADCEx_LinearCalibration_FactorLoad (ADC_HandleTypeDef * hadc)

Function description

Load the calibration factor from engi bytes.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** state

HAL_ADCEx_InjectedStart

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedStart (ADC_HandleTypeDef * hadc)

Function description

Enable ADC, start conversion of injected group.

Parameters

- **hadc:** ADC handle.

Return values

- **HAL:** status

Notes

- Interruptions enabled in this function: None.
- Case of multimode enabled when multimode feature is available: HAL_ADCEx_InjectedStart() API must be called for ADC slave first, then for ADC master. For ADC slave, ADC is enabled only (conversion is not started). For ADC master, ADC is enabled and multimode conversion is started.

HAL_ADCEx_InjectedStop

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedStop (ADC_HandleTypeDef * hadc)

Function description

Stop conversion of injected channels.

Parameters

- **hadc:** ADC handle.

Return values

- **HAL:** status

Notes

- If ADC must be disabled and if conversion is on going on regular group, function HAL_ADC_Stop must be used to stop both injected and regular groups, and disable the ADC.
- If injected group mode auto-injection is enabled, function HAL_ADC_Stop must be used.
- In case of multimode enabled (when multimode feature is available), HAL_ADCEx_InjectedStop() must be called for ADC master first, then for ADC slave. For ADC master, conversion is stopped and ADC is disabled. For ADC slave, ADC is disabled only (conversion stop of ADC master has already stopped conversion of ADC slave).

HAL_ADCEx_InjectedPollForConversion

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedPollForConversion (ADC_HandleTypeDef * hadc, uint32_t Timeout)

Function description

Wait for injected group conversion to be completed.

Parameters

- **hadc:** ADC handle
- **Timeout:** Timeout value in millisecond.

Return values

- **HAL:** status

Notes

- Depending on hadc->Init.EOCSelection, JEOS or JEOC is checked and cleared depending on AUTDLY bit status.

HAL_ADCEx_InjectedStart_IT

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedStart_IT (ADC_HandleTypeDef * hadc)

Function description

Enable ADC, start conversion of injected group with interruption.

Parameters

- **hadc:** ADC handle.

Return values

- **HAL:** status.

Notes

- Interruptions enabled in this function according to initialization setting : JEOC (end of conversion) or JEOS (end of sequence)
- Case of multimode enabled (when multimode feature is enabled): HAL_ADCEx_InjectedStart_IT() API must be called for ADC slave first, then for ADC master. For ADC slave, ADC is enabled only (conversion is not started). For ADC master, ADC is enabled and multimode conversion is started.

HAL_ADCEx_InjectedStop_IT

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedStop_IT (ADC_HandleTypeDef * hadc)

Function description

Stop conversion of injected channels, disable interruption of end-of-conversion.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- If ADC must be disabled and if conversion is on going on regular group, function HAL_ADC_Stop must be used to stop both injected and regular groups, and disable the ADC.
- If injected group mode auto-injection is enabled, function HAL_ADC_Stop must be used.
- Case of multimode enabled (when multimode feature is available): HAL_ADCEx_InjectedStop_IT() API must be called for ADC master first, then for ADC slave. For ADC master, conversion is stopped and ADC is disabled. For ADC slave, ADC is disabled only (conversion stop of ADC master has already stopped conversion of ADC slave).
- In case of auto-injection mode, HAL_ADC_Stop() must be used.

HAL_ADCEx_MultiModeStart_DMA

Function name

HAL_StatusTypeDef HAL_ADCEx_MultiModeStart_DMA (ADC_HandleTypeDef * hadc, uint32_t * pData, uint32_t Length)

Function description

Enable ADC, start MultiMode conversion and transfer regular results through DMA.

Parameters

- **hadc:** ADC handle of ADC master (handle of ADC slave must not be used)
- **pData:** Destination Buffer address.
- **Length:** Length of data to be transferred from ADC peripheral to memory (in bytes).

Return values

- **HAL:** status

Notes

- Multimode must have been previously configured using HAL_ADCEx_MultiModeConfigChannel() function. Interruptions enabled in this function: overrun, DMA half transfer, DMA transfer complete. Each of these interruptions has its dedicated callback function.
- State field of Slave ADC handle is not updated in this configuration: user should not rely on it for information related to Slave regular conversions.

HAL_ADCEx_MultiModeStop_DMA

Function name

HAL_StatusTypeDef HAL_ADCEx_MultiModeStop_DMA (ADC_HandleTypeDef * hadc)

Function description

Stop multimode ADC conversion, disable ADC DMA transfer, disable ADC peripheral.

Parameters

- **hadc:** ADC handle of ADC master (handle of ADC slave must not be used)

Return values

- **HAL:** status

Notes

- Multimode is kept enabled after this function. MultiMode DMA bits (MDMA and DMACFG bits of common CCR register) are maintained. To disable Multimode (set with HAL_ADCEx_MultiModeConfigChannel()), ADC must be reinitialized using HAL_ADC_Init() or HAL_ADC_DeInit(), or the user can resort to HAL_ADCEx_DisableMultiMode() API.
- In case of DMA configured in circular mode, function HAL_ADC_Stop_DMA() must be called after this function with handle of ADC slave, to properly disable the DMA channel.

HAL_ADCEx_MultiModeGetValue

Function name

uint32_t HAL_ADCEx_MultiModeGetValue (ADC_HandleTypeDef * hadc)

Function description

Return the last ADC Master and Slave regular conversions results when in multimode configuration.

Parameters

- **hadc:** ADC handle of ADC Master (handle of ADC Slave must not be used)

Return values

- **The:** converted data values.

HAL_ADCEx_InjectedGetValue

Function name

uint32_t HAL_ADCEx_InjectedGetValue (ADC_HandleTypeDef * hadc, uint32_t InjectedRank)

Function description

Get ADC injected group conversion result.

Parameters

- **hadc:** ADC handle
- **InjectedRank:** the converted ADC injected rank. This parameter can be one of the following values:
  - ADC_INJECTED_RANK_1 ADC group injected rank 1
  - ADC_INJECTED_RANK_2 ADC group injected rank 2
  - ADC_INJECTED_RANK_3 ADC group injected rank 3
  - ADC_INJECTED_RANK_4 ADC group injected rank 4

Return values

- **ADC:** group injected conversion data

Notes

- Reading register JDRx automatically clears ADC flag JEOC (ADC group injected end of unitary conversion).
- This function does not clear ADC flag JEOS (ADC group injected end of sequence conversion) Occurrence of flag JEOS rising: If sequencer is composed of 1 rank, flag JEOS is equivalent to flag JEOC.If sequencer is composed of several ranks, during the scan sequence flag JEOC only is raised, at the end of the scan sequence both flags JEOC and EOS are raised. Flag JEOS must not be cleared by this function because it would not be compliant with low power features (feature low power auto-wait, not available on all STM32 families). To clear this flag, either use function: in programming model IT: HAL_ADC_IRQHandler(), in programming model polling: HAL_ADCEx_InjectedPollForConversion() or __HAL_ADC_CLEAR_FLAG(&hadc, ADC_FLAG_JEOS).

HAL_ADCEx_InjectedConvCpltCallback

Function name

void HAL_ADCEx_InjectedConvCpltCallback (ADC_HandleTypeDef * hadc)

Function description

Injected conversion complete callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADCEx_InjectedQueueOverflowCallback

Function name

void HAL_ADCEx_InjectedQueueOverflowCallback (ADC_HandleTypeDef * hadc)

Function description

Injected context queue overflow callback.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

Notes

- This callback is called if injected context queue is enabled (parameter "QueueInjectedContext" in injected channel configuration) and if a new injected context is set when queue is full (maximum 2 contexts).

HAL_ADCEx_LevelOutOfWindow2Callback

Function name

void HAL_ADCEx_LevelOutOfWindow2Callback (ADC_HandleTypeDef * hadc)

Function description

Analog watchdog 2 callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADCEx_LevelOutOfWindow3Callback

Function name

void HAL_ADCEx_LevelOutOfWindow3Callback (ADC_HandleTypeDef * hadc)

Function description

Analog watchdog 3 callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADCEx_EndOfSamplingCallback

Function name

void HAL_ADCEx_EndOfSamplingCallback (ADC_HandleTypeDef * hadc)

Function description

End Of Sampling callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADCEx_RegularStop

Function name

HAL_StatusTypeDef HAL_ADCEx_RegularStop (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion of regular group (and injected channels in case of auto_injection mode), disable ADC peripheral if no conversion is on going on injected group.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

HAL_ADCEx_RegularStop_IT

Function name

HAL_StatusTypeDef HAL_ADCEx_RegularStop_IT (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion of ADC groups regular and injected, disable interrution of end-of-conversion, disable ADC peripheral if no conversion is on going on injected group.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

HAL_ADCEx_RegularStop_DMA

Function name

HAL_StatusTypeDef HAL_ADCEx_RegularStop_DMA (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion of regular group (and injected group in case of auto_injection mode), disable ADC DMA transfer, disable ADC peripheral if no conversion is on going on injected group.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

- HAL_ADCEx_RegularStop_DMA() function is dedicated to single-ADC mode only. For multimode (when multimode feature is available), HAL_ADCEx_RegularMultiModeStop_DMA() API must be used.

HAL_ADCEx_RegularMultiModeStop_DMA

Function name

HAL_StatusTypeDef HAL_ADCEx_RegularMultiModeStop_DMA (ADC_HandleTypeDef * hadc)

Function description

Stop DMA-based multimode ADC conversion, disable ADC DMA transfer, disable ADC peripheral if no injected conversion is on-going.

Parameters

- **hadc:** ADC handle of ADC master (handle of ADC slave must not be used)

Return values

- **HAL:** status

Notes

- Multimode is kept enabled after this function. Multimode DMA bits (MDMA and DMACFG bits of common CCR register) are maintained. To disable multimode (set with HAL_ADCEx_MultiModeConfigChannel()), ADC must be reinitialized using HAL_ADC_Init() or HAL_ADC_DeInit(), or the user can resort to HAL_ADCEx_DisableMultiMode() API.
- In case of DMA configured in circular mode, function HAL_ADCEx_RegularStop_DMA() must be called after this function with handle of ADC slave, to properly disable the DMA channel.

HAL_ADCEx_InjectedConfigChannel

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedConfigChannel (ADC_HandleTypeDef * hadc, ADC_InjectionConfTypeDef * sConfigInjected)

Function description

Configure a channel to be assigned to ADC group injected.

Parameters

- **hadc:** ADC handle
- **sConfigInjected:** Structure of ADC injected group and ADC channel for injected group.

Return values

- **HAL:** status

Notes

- Possibility to update parameters on the fly: This function initializes injected group, following calls to this function can be used to reconfigure some parameters of structure "ADC_InjectionConfTypeDef" on the fly, without resetting the ADC. The setting of these parameters is conditioned to ADC state: Refer to comments of structure "ADC_InjectionConfTypeDef".
- In case of usage of internal measurement channels: Vbat/VrefInt/TempSensor. These internal paths can be disabled using function HAL_ADC_DeInit().
- Caution: For Injected Context Queue use, a context must be fully defined before start of injected conversion. All channels are configured consecutively for the same ADC instance. Therefore, the number of calls to HAL_ADCEx_InjectedConfigChannel() must be equal to the value of parameter InjectedNbrOfConversion for each context. Example 1: If 1 context is intended to be used (or if there is no use of the Injected Queue Context feature) and if the context contains 3 injected ranks (InjectedNbrOfConversion = 3), HAL_ADCEx_InjectedConfigChannel() must be called once for each channel (i.e. 3 times) before starting a conversion. This function must not be called to configure a 4th injected channel: it would start a new context into context queue.Example 2: If 2 contexts are intended to be used and each of them contains 3 injected ranks (InjectedNbrOfConversion = 3), HAL_ADCEx_InjectedConfigChannel() must be called once for each channel and for each context (3 channels x 2 contexts = 6 calls). Conversion can start once the 1st context is set, that is after the first three HAL_ADCEx_InjectedConfigChannel() calls. The 2nd context can be set on the fly.

HAL_ADCEx_MultiModeConfigChannel

Function name

HAL_StatusTypeDef HAL_ADCEx_MultiModeConfigChannel (ADC_HandleTypeDef * hadc, ADC_MultiModeTypeDef * multimode)

Function description

Enable ADC multimode and configure multimode parameters.

Parameters

- **hadc:** Master ADC handle
- **multimode:** Structure of ADC multimode configuration

Return values

- **HAL:** status

Notes

- Possibility to update parameters on the fly: This function initializes multimode parameters, following calls to this function can be used to reconfigure some parameters of structure "ADC_MultiModeTypeDef" on the fly, without resetting the ADCs. The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_MultiModeTypeDef".
- To move back configuration from multimode to single mode, ADC must be reset (using function HAL_ADC_Init()).

HAL_ADCEx_EnableInjectedQueue

Function name

HAL_StatusTypeDef HAL_ADCEx_EnableInjectedQueue (ADC_HandleTypeDef * hadc)

Function description

Enable Injected Queue.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- This function resets CFGR register JQDIS bit in order to enable the Injected Queue. JQDIS can be written only when ADSTART and JDSTART are both equal to 0 to ensure that no regular nor injected conversion is ongoing.

HAL_ADCEx_DisableInjectedQueue

Function name

HAL_StatusTypeDef HAL_ADCEx_DisableInjectedQueue (ADC_HandleTypeDef * hadc)

Function description

Disable Injected Queue.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- This function sets CFGR register JQDIS bit in order to disable the Injected Queue. JQDIS can be written only when ADSTART and JDSTART are both equal to 0 to ensure that no regular nor injected conversion is ongoing.

HAL_ADCEx_DisableVoltageRegulator

Function name

HAL_StatusTypeDef HAL_ADCEx_DisableVoltageRegulator (ADC_HandleTypeDef * hadc)

Function description

Disable ADC voltage regulator.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- Disabling voltage regulator allows to save power. This operation can be carried out only when ADC is disabled.
- To enable again the voltage regulator, the user is expected to resort to HAL_ADC_Init() API.

HAL_ADCEx_EnterADCDeepPowerDownMode

Function name

HAL_StatusTypeDef HAL_ADCEx_EnterADCDeepPowerDownMode (ADC_HandleTypeDef * hadc)

Function description

Enter ADC deep-power-down mode.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- This mode is achieved in setting DEEPPWD bit and allows to save power in reducing leakage currents. It is particularly interesting before entering stop modes.
- Setting DEEPPWD automatically clears ADVREGEN bit and disables the ADC voltage regulator. This means that this API encompasses HAL_ADCEx_DisableVoltageRegulator(). Additionally, the internal calibration is lost.
- To exit the ADC deep-power-down mode, the user is expected to resort to HAL_ADC_Init() API as well as to relaunch a calibration with HAL_ADCEx_Calibration_Start() API or to re-apply a previously saved calibration factor.

9.3 ADCEx Firmware driver defines

The following section lists the various define and macros of the module.

9.3.1 ADCEx

ADCEx

ADC Extended Calibration mode offset mode or linear mode

ADC_CALIB_OFFSET

ADC_CALIB_OFFSET_LINEARITY

ADC Extended Dual Mode Data Formatting

ADC_DUALMODEDATAFORMAT_DISABLED

Dual ADC mode without data packing: ADCx_CDR and ADCx_CDR2 registers not used

ADC_DUALMODEDATAFORMAT_32_10_BITS

Data formatting mode for 32 down to 10-bit resolution

ADC_DUALMODEDATAFORMAT_8_BITS

Data formatting mode for 8-bit resolution

ADC Extended Exported Macros

ADC_FORCE_MODE_INDEPENDENT

Description:

- Force ADC instance in multimode mode independent (multimode disable).

Parameters:

- __HANDLE__: ADC handle.

Return value:

- None

Notes:

- This macro must be used only in case of transition from multimode to mode independent and in case of unknown previous state, to ensure ADC configuration is in mode independent. Standard way of multimode configuration change is done from HAL ADC handle of ADC master using function "HAL_ADCEx_MultiModeConfigChannel(..., ADC_MODE_INDEPENDENT)")". Usage of this macro is not the Standard way of multimode configuration and can lead to have HAL ADC handles status misaligned. Usage of this macro must be limited to cases mentioned above.

ADC Extended Oversampling left Shift

ADC_LEFTBITSHIFT_NONE

ADC No bit shift

ADC_LEFTBITSHIFT_1

ADC 1 bit shift

ADC_LEFTBITSHIFT_2

ADC 2 bits shift

ADC_LEFTBITSHIFT_3

ADC 3 bits shift

ADC_LEFTBITSHIFT_4

ADC 4 bits shift

ADC_LEFTBITSHIFT_5

ADC 5 bits shift

ADC_LEFTBITSHIFT_6

ADC 6 bits shift

ADC_LEFTBITSHIFT_7

ADC 7 bits shift

ADC_LEFTBITSHIFT_8

ADC 8 bits shift

ADC_LEFTBITSHIFT_9

ADC 9 bits shift

ADC_LEFTBITSHIFT_10

ADC 10 bits shift

ADC_LEFTBITSHIFT_11

ADC 11 bits shift

ADC_LEFTBITSHIFT_12

ADC 12 bits shift

ADC_LEFTBITSHIFT_13

ADC 13 bits shift

ADC_LEFTBITSHIFT_14

ADC 14 bits shift

ADC_LEFTBITSHIFT_15

ADC 15 bits shift

10 HAL CEC Generic Driver

10.1 CEC Firmware driver registers structures

10.1.1 CEC_InitTypeDef

CEC_InitTypeDef is defined in the stm32h7xx_hal_cec.h

Data Fields

- ***uint32_t SignalFreeTime***
- ***uint32_t Tolerance***
- ***uint32_t BRERxStop***
- ***uint32_t BREErrorBitGen***
- ***uint32_t LBPEErrorBitGen***
- ***uint32_t BroadcastMsgNoErrorBitGen***
- ***uint32_t SignalFreeTimeOption***
- ***uint32_t ListenMode***
- ***uint16_t OwnAddress***
- ***uint8_t * RxBuffer***

Field Documentation

- ***uint32_t CEC_InitTypeDef::SignalFreeTime***

  Set SFT field, specifies the Signal Free Time. It can be one of CEC_Signal_Free_Time and belongs to the set {0,...,7} where 0x0 is the default configuration else means 0.5 + (SignalFreeTime - 1) nominal data bit periods
- ***uint32_t CEC_InitTypeDef::Tolerance***

  Set RXTOL bit, specifies the tolerance accepted on the received waveforms, it can be a value of CEC_Tolerance : it is either CEC_STANDARD_TOLERANCE or CEC_EXTENDED_TOLERANCE
- ***uint32_t CEC_InitTypeDef::BRERxStop***

  Set BRESTP bit CEC_BRERxStop : specifies whether or not a Bit Rising Error stops the reception. CEC_NO_RX_STOP_ON_BRE: reception is not stopped. CEC_RX_STOP_ON_BRE: reception is stopped.
- ***uint32_t CEC_InitTypeDef::BREErrorBitGen***

  Set BREGEN bit CEC_BREErrorBitGen : specifies whether or not an Error-Bit is generated on the CEC line upon Bit Rising Error detection. CEC_BRE_ERRORBIT_NO_GENERATION: no error-bit generation. CEC_BRE_ERRORBIT_GENERATION: error-bit generation if BRESTP is set.
- ***uint32_t CEC_InitTypeDef::LBPEErrorBitGen***

  Set LBPEGEN bit CEC_LBPEErrorBitGen : specifies whether or not an Error-Bit is generated on the CEC line upon Long Bit Period Error detection. CEC_LBPE_ERRORBIT_NO_GENERATION: no error-bit generation. CEC_LBPE_ERRORBIT_GENERATION: error-bit generation.
- ***uint32_t CEC_InitTypeDef::BroadcastMsgNoErrorBitGen***

  Set BRDNOGEN bit CEC_BroadCastMsgErrorBitGen : allows to avoid an Error-Bit generation on the CEC line upon an error detected on a broadcast message.It supersedes BREGEN and LBPEGEN bits for a broadcast message error handling. It can take two values:1) CEC_BROADCASTERROR_ERRORBIT_GENERATION. a) BRE detection: error-bit generation on the CEC line if BRESTP=CEC_RX_STOP_ON_BRE and BREGEN=CEC_BRE_ERRORBIT_NO_GENERATION. b) LBPE detection: error-bit generation on the CEC line if LBPGEN=CEC_LBPE_ERRORBIT_NO_GENERATION.2) CEC_BROADCASTERROR_NO_ERRORBIT_GENERATION. no error-bit generation in case neither a) nor b) are satisfied. Additionally, there is no error-bit generation in case of Short Bit Period Error detection in a broadcast message while LSTN bit is set.

- ***uint32_t CEC_InitTypeDef::SignalFreeTimeOption***
  Set SFTOP bit CEC_SFT_Option : specifies when SFT timer starts. CEC_SFT_START_ON_TXSOM SFT: timer starts when TXSOM is set by software. CEC_SFT_START_ON_TX_RX_END: SFT timer starts automatically at the end of message transmission/reception.
- ***uint32_t CEC_InitTypeDef::ListenMode***
  Set LSTN bit CEC_Listening_Mode : specifies device listening mode. It can take two values:CEC_REDUCED_LISTENING_MODE: CEC peripheral receives only message addressed to its own address (OAR). Messages addressed to different destination are ignored. Broadcast messages are always received.CEC_FULL_LISTENING_MODE: CEC peripheral receives messages addressed to its own address (OAR) with positive acknowledge. Messages addressed to different destination are received, but without interfering with the CEC bus: no acknowledge sent.
- ***uint16_t CEC_InitTypeDef::OwnAddress***
  Own addresses configuration This parameter can be a value of CEC_OWN_ADDRESS
- ***uint8_t* CEC_InitTypeDef::RxBuffer***
  CEC Rx buffer pointer

10.1.2 __CEC_HandleTypeDef

__CEC_HandleTypeDef is defined in the stm32h7xx_hal_cec.h

Data Fields

- ***CEC_TypeDef * Instance***
- ***CEC_InitTypeDef Init***
- ***const uint8_t * pTxBuffPtr***
- ***uint16_t TxXferCount***
- ***uint16_t RxXferSize***
- ***HAL_LockTypeDef Lock***
- ***HAL_CEC_StateTypeDef gState***
- ***HAL_CEC_StateTypeDef RxState***
- ***uint32_t ErrorCode***
- ***void(* TxCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***CEC_TypeDef* __CEC_HandleTypeDef::Instance***
  CEC registers base address
- ***CEC_InitTypeDef __CEC_HandleTypeDef::Init***
  CEC communication parameters
- ***const uint8_t* __CEC_HandleTypeDef::pTxBuffPtr***
  Pointer to CEC Tx transfer Buffer
- ***uint16_t __CEC_HandleTypeDef::TxXferCount***
  CEC Tx Transfer Counter
- ***uint16_t __CEC_HandleTypeDef::RxXferSize***
  CEC Rx Transfer size, 0: header received only
- ***HAL_LockTypeDef __CEC_HandleTypeDef::Lock***
  Locking object
- ***HAL_CEC_StateTypeDef __CEC_HandleTypeDef::gState***
  CEC state information related to global Handle management and also related to Tx operations. This parameter can be a value of HAL_CEC_StateTypeDef
- ***HAL_CEC_StateTypeDef __CEC_HandleTypeDef::RxState***
  CEC state information related to Rx operations. This parameter can be a value of HAL_CEC_StateTypeDef

- ***uint32_t __CEC_HandleTypeDef::ErrorCode***
  For errors handling purposes, copy of ISR register in case error is reported
- ***void(* __CEC_HandleTypeDef::TxCpltCallback)(struct __CEC_HandleTypeDef *hcec)***
  CEC Tx Transfer completed callback
- ***void(* __CEC_HandleTypeDef::RxCpltCallback)(struct __CEC_HandleTypeDef *hcec, uint32_t RxFrameSize)***
  CEC Rx Transfer completed callback
- ***void(* __CEC_HandleTypeDef::ErrorCallback)(struct __CEC_HandleTypeDef *hcec)***
  CEC error callback
- ***void(* __CEC_HandleTypeDef::MspInitCallback)(struct __CEC_HandleTypeDef *hcec)***
  CEC Msp Init callback
- ***void(* __CEC_HandleTypeDef::MspDeInitCallback)(struct __CEC_HandleTypeDef *hcec)***
  CEC Msp DeInit callback

10.2 CEC Firmware driver API description

The following section lists the various functions of the CEC library.

10.2.1 How to use this driver

The CEC HAL driver can be used as follow:

1. Declare a CEC_HandleTypeDef handle structure.
2. Initialize the CEC low level resources by implementing the HAL_CEC_MspInit ()API:
   a. Enable the CEC interface clock.
   b. CEC pins configuration:
      ◦ Enable the clock for the CEC GPIOs.
      ◦ Configure these CEC pins as alternate function pull-up.
   c. NVIC configuration if you need to use interrupt process (HAL_CEC_Transmit_IT() and HAL_CEC_Receive_IT() APIs):
      ◦ Configure the CEC interrupt priority.
      ◦ Enable the NVIC CEC IRQ handle.
      ◦ The specific CEC interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_CEC_ENABLE_IT() and __HAL_CEC_DISABLE_IT() inside the transmit and receive process.
3. Program the Signal Free Time (SFT) and SFT option, Tolerance, reception stop in in case of Bit Rising Error, Error-Bit generation conditions, device logical address and Listen mode in the hcec Init structure.
4. Initialize the CEC registers by calling the HAL_CEC_Init() API.

Note: This API (HAL_CEC_Init()) configures also the low level Hardware (GPIO, CLOCK, CORTEX...etc) by calling the customed HAL_CEC_MspInit() API.

Callback registration

10.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the CEC

- The following parameters need to be configured:
  - SignalFreeTime
  - Tolerance
  - BRERxStop (RX stopped or not upon Bit Rising Error)
  - BREErrorBitGen (Error-Bit generation in case of Bit Rising Error)
  - LBPEErrorBitGen (Error-Bit generation in case of Long Bit Period Error)
  - BroadcastMsgNoErrorBitGen (Error-bit generation in case of broadcast message error)
  - SignalFreeTimeOption (SFT Timer start definition)
  - OwnAddress (CEC device address)
  - ListenMode

This section contains the following APIs:

- ***HAL_CEC_Init()***
- ***HAL_CEC_DeInit()***
- ***HAL_CEC_SetDeviceAddress()***
- ***HAL_CEC_MspInit()***
- ***HAL_CEC_MspDeInit()***
- ***HAL_CEC_RegisterCallback()***
- ***HAL_CEC_UnRegisterCallback()***
- ***HAL_CEC_RegisterRxCpltCallback()***
- ***HAL_CEC_UnRegisterRxCpltCallback()***

10.2.3 IO operation functions

This section contains the following APIs:

- ***HAL_CEC_Transmit_IT()***
- ***HAL_CEC_GetLastReceivedFrameSize()***
- ***HAL_CEC_ChangeRxBuffer()***
- ***HAL_CEC_IRQHandler()***
- ***HAL_CEC_TxCpltCallback()***
- ***HAL_CEC_RxCpltCallback()***
- ***HAL_CEC_ErrorCallback()***

10.2.4 Peripheral Control function

This subsection provides a set of functions allowing to control the CEC.

- HAL_CEC_GetState() API can be helpful to check in run-time the state of the CEC peripheral.
- HAL_CEC_GetError() API can be helpful to check in run-time the error of the CEC peripheral.

This section contains the following APIs:

- ***HAL_CEC_GetState()***
- ***HAL_CEC_GetError()***

10.2.5 Detailed description of functions

HAL_CEC_Init

Function name

HAL_StatusTypeDef HAL_CEC_Init (CEC_HandleTypeDef * hcec)

Function description

Initializes the CEC mode according to the specified parameters in the CEC_InitTypeDef and creates the associated handle .

Parameters

- **hcec:** CEC handle

Return values

- **HAL:** status

HAL_CEC_DeInit

Function name

HAL_StatusTypeDef HAL_CEC_DeInit (CEC_HandleTypeDef * hcec)

Function description

DeInitializes the CEC peripheral.

Parameters

- **hcec:** CEC handle

Return values

- **HAL:** status

HAL_CEC_SetDeviceAddress

Function name

HAL_StatusTypeDef HAL_CEC_SetDeviceAddress (CEC_HandleTypeDef * hcec, uint16_t CEC_OwnAddress)

Function description

Initializes the Own Address of the CEC device.

Parameters

- **hcec:** CEC handle
- **CEC_OwnAddress:** The CEC own address.

Return values

- **HAL:** status

HAL_CEC_MspInit

Function name

void HAL_CEC_MspInit (CEC_HandleTypeDef * hcec)

Function description

CEC MSP Init.

Parameters

- **hcec:** CEC handle

Return values

- **None:**

HAL_CEC_MspDeInit

Function name

void HAL_CEC_MspDeInit (CEC_HandleTypeDef * hcec)

Function description

CEC MSP DeInit.

Parameters

- **hcec:** CEC handle

Return values

- **None:**

HAL_CEC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_CEC_RegisterCallback (CEC_HandleTypeDef * hcec, HAL_CEC_CallbackIDTypeDef CallbackID, pCEC_CallbackTypeDef pCallback)

Function description

Register a User CEC Callback To be used instead of the weak predefined callback.

Parameters

- **hcec:** CEC handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_CEC_TX_CPLT_CB_ID Tx Complete callback ID
  - HAL_CEC_ERROR_CB_ID Error callback ID
  - HAL_CEC_MSPINIT_CB_ID MspInit callback ID
  - HAL_CEC_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_CEC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_CEC_UnRegisterCallback (CEC_HandleTypeDef * hcec, HAL_CEC_CallbackIDTypeDef CallbackID)

Function description

Unregister an CEC Callback CEC callback is redirected to the weak predefined callback.

Parameters

- **hcec:** uart handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_CEC_TX_CPLT_CB_ID Tx Complete callback ID
  - HAL_CEC_ERROR_CB_ID Error callback ID
  - HAL_CEC_MSPINIT_CB_ID MspInit callback ID
  - HAL_CEC_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **status:**

HAL_CEC_RegisterRxCpltCallback

Function name

HAL_StatusTypeDef HAL_CEC_RegisterRxCpltCallback (CEC_HandleTypeDef * hcec, pCEC_RxCallbackTypeDef pCallback)

Function description

Register CEC RX complete Callback To be used instead of the weak HAL_CEC_RxCpltCallback() predefined callback.

Parameters

- **hcec:** CEC handle
- **pCallback:** pointer to the Rx transfer compelete Callback function

Return values

- **HAL:** status

HAL_CEC_UnRegisterRxCpltCallback

Function name

HAL_StatusTypeDef HAL_CEC_UnRegisterRxCpltCallback (CEC_HandleTypeDef * hcec)

Function description

UnRegister CEC RX complete Callback CEC RX complete Callback is redirected to the weak HAL_CEC_RxCpltCallback() predefined callback.

Parameters

- **hcec:** CEC handle

Return values

- **HAL:** status

HAL_CEC_Transmit_IT

Function name

HAL_StatusTypeDef HAL_CEC_Transmit_IT (CEC_HandleTypeDef * hcec, uint8_t InitiatorAddress, uint8_t DestinationAddress, const uint8_t * pData, uint32_t Size)

Function description

Send data in interrupt mode.

Parameters

- **hcec:** CEC handle
- **InitiatorAddress:** Initiator address
- **DestinationAddress:** destination logical address
- **pData:** pointer to input byte data buffer
- **Size:** amount of data to be sent in bytes (without counting the header). 0 means only the header is sent (ping operation). Maximum TX size is 15 bytes (1 opcode and up to 14 operands).

Return values

- **HAL:** status

HAL_CEC_GetLastReceivedFrameSize

Function name

uint32_t HAL_CEC_GetLastReceivedFrameSize (const CEC_HandleTypeDef * hcec)

Function description

Get size of the received frame.

Parameters

- **hcec:** CEC handle

Return values

- **Frame:** size

HAL_CEC_ChangeRxBuffer

Function name

void HAL_CEC_ChangeRxBuffer (CEC_HandleTypeDef * hcec, uint8_t * Rxbuffer)

Function description

Change Rx Buffer.

Parameters

- **hcec:** CEC handle
- **Rxbuffer:** Rx Buffer

Return values

- **Frame:** size

Notes

- This function can be called only inside the HAL_CEC_RxCpltCallback()

HAL_CEC_IRQHandler

Function name

void HAL_CEC_IRQHandler (CEC_HandleTypeDef * hcec)

Function description

This function handles CEC interrupt requests.

Parameters

- **hcec:** CEC handle

Return values

- **None:**

HAL_CEC_TxCpltCallback

Function name

void HAL_CEC_TxCpltCallback (CEC_HandleTypeDef * hcec)

Function description

Tx Transfer completed callback.

Parameters

- **hcec:** CEC handle

Return values

- **None:**

HAL_CEC_RxCpltCallback

Function name

void HAL_CEC_RxCpltCallback (CEC_HandleTypeDef * hcec, uint32_t RxFrameSize)

Function description

Rx Transfer completed callback.

Parameters

- **hcec:** CEC handle
- **RxFrameSize:** Size of frame

Return values

- **None:**

HAL_CEC_ErrorCallback

Function name

void HAL_CEC_ErrorCallback (CEC_HandleTypeDef * hcec)

Function description

CEC error callbacks.

Parameters

- **hcec:** CEC handle

Return values

- **None:**

HAL_CEC_GetState

Function name

HAL_CEC_StateTypeDef HAL_CEC_GetState (const CEC_HandleTypeDef * hcec)

Function description

return the CEC state

Parameters

- **hcec:** pointer to a CEC_HandleTypeDef structure that contains the configuration information for the specified CEC module.

Return values

- **HAL:** state

HAL_CEC_GetError

Function name

uint32_t HAL_CEC_GetError (const CEC_HandleTypeDef * hcec)

Function description

Return the CEC error code.

Parameters

- **hcec:** pointer to a CEC_HandleTypeDef structure that contains the configuration information for the specified CEC.

Return values

- **CEC:** Error Code

10.3 CEC Firmware driver defines

The following section lists the various define and macros of the module.

10.3.1 CEC

CEC

CEC all RX or TX errors flags

CEC_ISR_ALL_ERROR

CEC Error Bit Generation if Bit Rise Error reported

CEC_BRE_ERRORBIT_NO_GENERATION

CEC_BRE_ERRORBIT_GENERATION

CEC Reception Stop on Error

CEC_NO_RX_STOP_ON_BRE

CEC_RX_STOP_ON_BRE

CEC Error Bit Generation on Broadcast message

CEC_BROADCASTERROR_ERRORBIT_GENERATION

CEC_BROADCASTERROR_NO_ERRORBIT_GENERATION

CEC Error Code

HAL_CEC_ERROR_NONE
no error

HAL_CEC_ERROR_RXOVR
CEC Rx-Overrun

HAL_CEC_ERROR_BRE
CEC Rx Bit Rising Error

HAL_CEC_ERROR_SBPE
CEC Rx Short Bit period Error

HAL_CEC_ERROR_LBPE
CEC Rx Long Bit period Error

HAL_CEC_ERROR_RXACKE
CEC Rx Missing Acknowledge

HAL_CEC_ERROR_ARBLST
CEC Arbitration Lost

HAL_CEC_ERROR_TXUDR
CEC Tx-Buffer Underrun

HAL_CEC_ERROR_TXERR
CEC Tx-Error

HAL_CEC_ERROR_TXACKE
CEC Tx Missing Acknowledge

HAL_CEC_ERROR_INVALID_CALLBACK
Invalid Callback Error

CEC Exported Macros

__HAL_CEC_RESET_HANDLE_STATE

Description:

- Reset CEC handle gstate & RxState.

Parameters:

- __HANDLE__: CEC handle.

Return value:

- None

__HAL_CEC_GET_FLAG

Description:

- Checks whether or not the specified CEC interrupt flag is set.

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __FLAG__: specifies the flag to check.
  - CEC_FLAG_TXACKE: Tx Missing acknowledge Error
  - CEC_FLAG_TXERR: Tx Error.
  - CEC_FLAG_TXUDR: Tx-Buffer Underrun.
  - CEC_FLAG_TXEND: End of transmission (successful transmission of the last byte).
  - CEC_FLAG_TXBR: Tx-Byte Request.
  - CEC_FLAG_ARBLST: Arbitration Lost
  - CEC_FLAG_RXACKE: Rx-Missing Acknowledge
  - CEC_FLAG_LBPE: Rx Long period Error
  - CEC_FLAG_SBPE: Rx Short period Error
  - CEC_FLAG_BRE: Rx Bit Rising Error
  - CEC_FLAG_RXOVR: Rx Overrun.
  - CEC_FLAG_RXEND: End Of Reception.
  - CEC_FLAG_RXBR: Rx-Byte Received.

Return value:

- ITStatus

__HAL_CEC_CLEAR_FLAG

Description:

- Clears the interrupt or status flag when raised (write at 1)

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __FLAG__: specifies the interrupt/status flag to clear. This parameter can be one of the following values:
  - CEC_FLAG_TXACKE: Tx Missing acknowledge Error
  - CEC_FLAG_TXERR: Tx Error.
  - CEC_FLAG_TXUDR: Tx-Buffer Underrun.
  - CEC_FLAG_TXEND: End of transmission (successful transmission of the last byte).
  - CEC_FLAG_TXBR: Tx-Byte Request.
  - CEC_FLAG_ARBLST: Arbitration Lost
  - CEC_FLAG_RXACKE: Rx-Missing Acknowledge
  - CEC_FLAG_LBPE: Rx Long period Error
  - CEC_FLAG_SBPE: Rx Short period Error
  - CEC_FLAG_BRE: Rx Bit Rising Error
  - CEC_FLAG_RXOVR: Rx Overrun.
  - CEC_FLAG_RXEND: End Of Reception.
  - CEC_FLAG_RXBR: Rx-Byte Received.

Return value:

- none

__HAL_CEC_ENABLE_IT

Description:

- Enables the specified CEC interrupt.

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __INTERRUPT__: specifies the CEC interrupt to enable. This parameter can be one of the following values:
  - CEC_IT_TXACKE: Tx Missing acknowledge Error IT Enable
  - CEC_IT_TXERR: Tx Error IT Enable
  - CEC_IT_TXUDR: Tx-Buffer Underrun IT Enable
  - CEC_IT_TXEND: End of transmission IT Enable
  - CEC_IT_TXBR: Tx-Byte Request IT Enable
  - CEC_IT_ARBLST: Arbitration Lost IT Enable
  - CEC_IT_RXACKE: Rx-Missing Acknowledge IT Enable
  - CEC_IT_LBPE: Rx Long period Error IT Enable
  - CEC_IT_SBPE: Rx Short period Error IT Enable
  - CEC_IT_BRE: Rx Bit Rising Error IT Enable
  - CEC_IT_RXOVR: Rx Overrun IT Enable
  - CEC_IT_RXEND: End Of Reception IT Enable
  - CEC_IT_RXBR: Rx-Byte Received IT Enable

Return value:

- none

__HAL_CEC_DISABLE_IT

Description:

- Disables the specified CEC interrupt.

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __INTERRUPT__: specifies the CEC interrupt to disable. This parameter can be one of the following values:
  - CEC_IT_TXACKE: Tx Missing acknowledge Error IT Enable
  - CEC_IT_TXERR: Tx Error IT Enable
  - CEC_IT_TXUDR: Tx-Buffer Underrun IT Enable
  - CEC_IT_TXEND: End of transmission IT Enable
  - CEC_IT_TXBR: Tx-Byte Request IT Enable
  - CEC_IT_ARBLST: Arbitration Lost IT Enable
  - CEC_IT_RXACKE: Rx-Missing Acknowledge IT Enable
  - CEC_IT_LBPE: Rx Long period Error IT Enable
  - CEC_IT_SBPE: Rx Short period Error IT Enable
  - CEC_IT_BRE: Rx Bit Rising Error IT Enable
  - CEC_IT_RXOVR: Rx Overrun IT Enable
  - CEC_IT_RXEND: End Of Reception IT Enable
  - CEC_IT_RXBR: Rx-Byte Received IT Enable

Return value:

- none

__HAL_CEC_GET_IT_SOURCE

Description:

- Checks whether or not the specified CEC interrupt is enabled.

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __INTERRUPT__: specifies the CEC interrupt to check. This parameter can be one of the following values:
  - CEC_IT_TXACKE: Tx Missing acknowledge Error IT Enable
  - CEC_IT_TXERR: Tx Error IT Enable
  - CEC_IT_TXUDR: Tx-Buffer Underrun IT Enable
  - CEC_IT_TXEND: End of transmission IT Enable
  - CEC_IT_TXBR: Tx-Byte Request IT Enable
  - CEC_IT_ARBLST: Arbitration Lost IT Enable
  - CEC_IT_RXACKE: Rx-Missing Acknowledge IT Enable
  - CEC_IT_LBPE: Rx Long period Error IT Enable
  - CEC_IT_SBPE: Rx Short period Error IT Enable
  - CEC_IT_BRE: Rx Bit Rising Error IT Enable
  - CEC_IT_RXOVR: Rx Overrun IT Enable
  - CEC_IT_RXEND: End Of Reception IT Enable
  - CEC_IT_RXBR: Rx-Byte Received IT Enable

Return value:

- FlagStatus

__HAL_CEC_ENABLE

Description:

- Enables the CEC device.

Parameters:

- __HANDLE__: specifies the CEC Handle.

Return value:

- none

__HAL_CEC_DISABLE

Description:

- Disables the CEC device.

Parameters:

- __HANDLE__: specifies the CEC Handle.

Return value:

- none

__HAL_CEC_FIRST_BYTE_TX_SET

Description:

- Set Transmission Start flag.

Parameters:

- __HANDLE__: specifies the CEC Handle.

Return value:

- none

__HAL_CEC_LAST_BYTE_TX_SET

Description:

- Set Transmission End flag.

Parameters:

- __HANDLE__: specifies the CEC Handle.

Return value:

- none: If the CEC message consists of only one byte, TXEOM must be set before of TXSOM.

__HAL_CEC_GET_TRANSMISSION_START_FLAG

Description:

- Get Transmission Start flag.

Parameters:

- __HANDLE__: specifies the CEC Handle.

Return value:

- FlagStatus

__HAL_CEC_GET_TRANSMISSION_END_FLAG

Description:

- Get Transmission End flag.

Parameters:

- __HANDLE__: specifies the CEC Handle.

Return value:

- FlagStatus

__HAL_CEC_CLEAR_OAR

Description:

- Clear OAR register.

Parameters:

- __HANDLE__: specifies the CEC Handle.

Return value:

- none

__HAL_CEC_SET_OAR

Description:

- Set OAR register (without resetting previously set address in case of multi-address mode) To reset OAR,

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __ADDRESS__: Own Address value (CEC logical address is identified by bit position)

Return value:

- none

CEC Flags definition

CEC_FLAG_TXACKE

CEC_FLAG_TXERR

CEC_FLAG_TXUDR

CEC_FLAG_TXEND

CEC_FLAG_TXBR

CEC_FLAG_ARBLST

CEC_FLAG_RXACKE

CEC_FLAG_LBPE

CEC_FLAG_SBPE

CEC_FLAG_BRE

CEC_FLAG_RXOVR

CEC_FLAG_RXEND

CEC_FLAG_RXBR

CEC all RX errors interrupts enabling flag

CEC_IER_RX_ALL_ERR

CEC all TX errors interrupts enabling flag

CEC_IER_TX_ALL_ERR

CEC Initiator logical address position in message header

CEC_INITIATOR_LSB_POS

CEC Interrupts definition

CEC_IT_TXACKE

CEC_IT_TXERR

CEC_IT_TXUDR

CEC_IT_TXEND

CEC_IT_TXBR

CEC_IT_ARBLST

CEC_IT_RXACKE

CEC_IT_LBPE

CEC_IT_SBPE

CEC_IT_BRE

CEC_IT_RXOVR

CEC_IT_RXEND

CEC_IT_RXBR

CEC Error Bit Generation if Long Bit Period Error reported

CEC_LBPE_ERRORBIT_NO_GENERATION

CEC_LBPE_ERRORBIT_GENERATION

CEC Listening mode option

CEC_REDUCED_LISTENING_MODE

CEC_FULL_LISTENING_MODE

CEC Device Own Address position in CEC CFGR register

CEC_CFGR_OAR_LSB_POS

CEC Own Address

CEC_OWN_ADDRESS_NONE

CEC_OWN_ADDRESS_0

CEC_OWN_ADDRESS_1

CEC_OWN_ADDRESS_2

CEC_OWN_ADDRESS_3

CEC_OWN_ADDRESS_4

CEC_OWN_ADDRESS_5

CEC_OWN_ADDRESS_6

CEC_OWN_ADDRESS_7

CEC_OWN_ADDRESS_8

CEC_OWN_ADDRESS_9

CEC_OWN_ADDRESS_10

CEC_OWN_ADDRESS_11

CEC_OWN_ADDRESS_12

CEC_OWN_ADDRESS_13

CEC_OWN_ADDRESS_14

CEC Signal Free Time start option

CEC_SFT_START_ON_TXSOM

CEC_SFT_START_ON_TX_RX_END

CEC Signal Free Time setting parameter

CEC_DEFAULT_SFT

CEC_0_5_BITPERIOD_SFT

CEC_1_5_BITPERIOD_SFT

CEC_2_5_BITPERIOD_SFT

CEC_3_5_BITPERIOD_SFT

CEC_4_5_BITPERIOD_SFT

CEC_5_5_BITPERIOD_SFT

CEC_6_5_BITPERIOD_SFT

CEC State Code Definition

HAL_CEC_STATE_RESET

Peripheral is not yet Initialized Value is allowed for gState and RxState

HAL_CEC_STATE_READY

Peripheral Initialized and ready for use Value is allowed for gState and RxState

HAL_CEC_STATE_BUSY

an internal process is ongoing Value is allowed for gState only

HAL_CEC_STATE_BUSY_RX

Data Reception process is ongoing Value is allowed for RxState only

HAL_CEC_STATE_BUSY_TX

Data Transmission process is ongoing Value is allowed for gState only

HAL_CEC_STATE_BUSY_RX_TX

an internal process is ongoing Value is allowed for gState only

HAL_CEC_STATE_ERROR

Error Value is allowed for gState only

CEC Receiver Tolerance

CEC_STANDARD_TOLERANCE

CEC_EXTENDED_TOLERANCE

11 HAL COMP Generic Driver

11.1 COMP Firmware driver registers structures

11.1.1 COMP_InitTypeDef

COMP_InitTypeDef is defined in the stm32h7xx_hal_comp.h

Data Fields

- ***uint32_t WindowMode***
- ***uint32_t Mode***
- ***uint32_t NonInvertingInput***
- ***uint32_t InvertingInput***
- ***uint32_t Hysteresis***
- ***uint32_t OutputPol***
- ***uint32_t BlankingSrce***
- ***uint32_t TriggerMode***

Field Documentation

- ***uint32_t COMP_InitTypeDef::WindowMode***

  Set window mode of a pair of comparators instances (2 consecutive instances odd and even COMP<x> and COMP<x+1>). Note: HAL COMP driver allows to set window mode from any COMP instance of the pair of COMP instances composing window mode. This parameter can be a value of ***COMP_WindowMode***
- ***uint32_t COMP_InitTypeDef::Mode***

  Set comparator operating mode to adjust power and speed. Note: For the characteritics of comparator power modes (propagation delay and power consumption), refer to device datasheet. This parameter can be a value of ***COMP_PowerMode***
- ***uint32_t COMP_InitTypeDef::NonInvertingInput***

  Set comparator input plus (non-inverting input). This parameter can be a value of ***COMP_InputPlus***
- ***uint32_t COMP_InitTypeDef::InvertingInput***

  Set comparator input minus (inverting input). This parameter can be a value of ***COMP_InputMinus***
- ***uint32_t COMP_InitTypeDef::Hysteresis***

  Set comparator hysteresis mode of the input minus. This parameter can be a value of ***COMP_Hysteresis***
- ***uint32_t COMP_InitTypeDef::OutputPol***

  Set comparator output polarity. This parameter can be a value of ***COMP_OutputPolarity***
- ***uint32_t COMP_InitTypeDef::BlankingSrce***

  Set comparator blanking source. This parameter can be a value of ***COMP_BlankingSrce***
- ***uint32_t COMP_InitTypeDef::TriggerMode***

  Set the comparator output triggering External Interrupt Line (EXTI). This parameter can be a value of ***COMP_EXTI_TriggerMode***

11.1.2 __COMP_HandleTypeDef

__COMP_HandleTypeDef is defined in the stm32h7xx_hal_comp.h

Data Fields

- ***COMP_TypeDef * Instance***
- ***COMP_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_COMP_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* TriggerCallback***

- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***COMP_TypeDef* __COMP_HandleTypeDef::Instance***
  Register base address
- ***COMP_InitTypeDef __COMP_HandleTypeDef::Init***
  COMP required parameters
- ***HAL_LockTypeDef __COMP_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_COMP_StateTypeDef __COMP_HandleTypeDef::State***
  COMP communication state
- ***__IO uint32_t __COMP_HandleTypeDef::ErrorCode***
  COMP error code
- ***void(* __COMP_HandleTypeDef::TriggerCallback)(struct __COMP_HandleTypeDef *hcomp)***
  COMP trigger callback
- ***void(* __COMP_HandleTypeDef::MspInitCallback)(struct __COMP_HandleTypeDef *hcomp)***
  COMP Msp Init callback
- ***void(* __COMP_HandleTypeDef::MspDeInitCallback)(struct __COMP_HandleTypeDef *hcomp)***
  COMP Msp DeInit callback

11.2 COMP Firmware driver API description

The following section lists the various functions of the COMP library.

11.2.1 COMP Peripheral features

The STM32H7xx device family integrates two analog comparators instances COMP1 and COMP2:

1. The COMP input minus (inverting input) and input plus (non inverting input) can be set to internal references or to GPIO pins (refer to GPIO list in reference manual).
2. The COMP output level is available using HAL_COMP_GetOutputLevel() and can be redirected to other peripherals: GPIO pins (in mode alternate functions for comparator), timers. (refer to GPIO list in reference manual).
3. Pairs of comparators instances can be combined in window mode (2 consecutive instances odd and even COMP<x> and COMP<x+1>).
4. The comparators have interrupt capability through the EXTI controller with wake-up from sleep and stop modes:
   – COMP1 is internally connected to EXTI Line 20
   – COMP2 is internally connected to EXTI Line 21

From the corresponding IRQ handler, the right interrupt source can be retrieved using macro __HAL_COMP_COMP1_EXTI_GET_FLAG() and __HAL_COMP_COMP2_EXTI_GET_FLAG().

11.2.2 How to use this driver

This driver provides functions to configure and program the comparator instances of STM32H7xx devices. To use the comparator, perform the following steps:

1. Initialize the COMP low level resources by implementing the HAL_COMP_MspInit():
   – Configure the GPIO connected to comparator inputs plus and minus in analog mode using HAL_GPIO_Init().
   – If needed, configure the GPIO connected to comparator output in alternate function mode using HAL_GPIO_Init().
   – If required enable the COMP interrupt by configuring and enabling EXTI line in Interrupt mode and selecting the desired sensitivity level using HAL_GPIO_Init() function. After that enable the comparator interrupt vector using HAL_NVIC_EnableIRQ() function.

2. Configure the comparator using HAL_COMP_Init() function:
   – Select the input minus (inverting input)
   – Select the input plus (non-inverting input)
   – Select the hysteresis
   – Select the blanking source
   – Select the output polarity
   – Select the power mode
   – Select the window mode

Note: HAL_COMP_Init() calls internally __HAL_RCC_SYSCFG_CLK_ENABLE() to enable internal control clock of the comparators. However, this is a legacy strategy. Therefore, for compatibility anticipation, it is recommended to implement __HAL_RCC_SYSCFG_CLK_ENABLE() in "HAL_COMP_MspInit()". In STM32H7,COMP clock enable __HAL_RCC_COMP12_CLK_ENABLE() must be implemented by user in "HAL_COMP_MspInit()".

3. Reconfiguration on-the-fly of comparator can be done by calling again function HAL_COMP_Init() with new input structure parameters values.
4. Enable the comparator using HAL_COMP_Start() or HAL_COMP_Start_IT()to be enabled with the interrupt through NVIC of the CPU. Note: HAL_COMP_Start_IT() must be called after each interrupt otherwise the interrupt mode will stay disabled.
5. Use HAL_COMP_GetOutputLevel() or HAL_COMP_TriggerCallback() functions to manage comparator outputs(output level or events)
6. Disable the comparator using HAL_COMP_Stop() or HAL_COMP_Stop_IT() to disable the interrupt too.
7. De-initialize the comparator using HAL_COMP_DeInit() function.
8. For safety purpose, comparator configuration can be locked using HAL_COMP_Lock() function. The only way to unlock the comparator is a device hardware reset.

Callback registration

The compilation flag USE_HAL_COMP_REGISTER_CALLBACKS, when set to 1, allows the user to configure dynamically the driver callbacks. Use Functions HAL_COMP_RegisterCallback() to register an interrupt callback.

Function HAL_COMP_RegisterCallback() allows to register following callbacks:

- TriggerCallback : callback for COMP trigger.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_COMP_UnRegisterCallback to reset a callback to the default weak function.

HAL_COMP_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- TriggerCallback : callback for COMP trigger.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit.

By default, after the HAL_COMP_Init() and when the state is HAL_COMP_STATE_RESET all callbacks are set to the corresponding weak functions: example HAL_COMP_TriggerCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_COMP_Init()/ HAL_COMP_DeInit() only when these callbacks are null (not registered beforehand).

If MspInit or MspDeInit are not null, the HAL_COMP_Init()/ HAL_COMP_DeInit() keep and use the user MspInit/ MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in HAL_COMP_STATE_READY state only. Exception done MspInit/MspDeInit functions that can be registered/unregistered in HAL_COMP_STATE_READY or HAL_COMP_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/ DeInit.

Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_COMP_RegisterCallback() before calling HAL_COMP_DeInit() or HAL_COMP_Init() function.

When the compilation flag USE_HAL_COMP_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

11.2.3 Initialization and de-initialization functions

This section provides functions to initialize and de-initialize comparators

This section contains the following APIs:

- *HAL_COMP_Init()*
- *HAL_COMP_DeInit()*
- *HAL_COMP_MspInit()*
- *HAL_COMP_MspDeInit()*
- *HAL_COMP_RegisterCallback()*
- *HAL_COMP_UnRegisterCallback()*

11.2.4 IO operation functions

This section provides functions allowing to:

- Start a Comparator instance without interrupt.
- Stop a Comparator instance without interrupt.
- Start a Comparator instance with interrupt generation.
- Stop a Comparator instance with interrupt generation.

This section contains the following APIs:

- *HAL_COMP_Start()*
- *HAL_COMP_Stop()*
- *HAL_COMP_Start_IT()*
- *HAL_COMP_Stop_IT()*
- *HAL_COMP_IRQHandler()*

11.2.5 Peripheral Control functions

This subsection provides a set of functions allowing to control the comparators.

This section contains the following APIs:

- *HAL_COMP_Lock()*
- *HAL_COMP_GetOutputLevel()*
- *HAL_COMP_TriggerCallback()*

11.2.6 Peripheral State functions

This subsection permit to get in run-time the status of the peripheral.

This section contains the following APIs:

- *HAL_COMP_GetState()*
- *HAL_COMP_GetError()*

11.2.7 Detailed description of functions

HAL_COMP_Init

Function name

HAL_StatusTypeDef HAL_COMP_Init (COMP_HandleTypeDef * hcomp)

Function description

Initialize the COMP according to the specified parameters in the COMP_InitTypeDef and initialize the associated handle.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

Notes

- If the selected comparator is locked, initialization can't be performed. To unlock the configuration, perform a system reset.

HAL_COMP_DeInit

Function name

HAL_StatusTypeDef HAL_COMP_DeInit (COMP_HandleTypeDef * hcomp)

Function description

DeInitialize the COMP peripheral.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

Notes

- Deinitialization cannot be performed if the COMP configuration is locked. To unlock the configuration, perform a system reset.

HAL_COMP_MspInit

Function name

void HAL_COMP_MspInit (COMP_HandleTypeDef * hcomp)

Function description

Initialize the COMP MSP.

Parameters

- **hcomp:** COMP handle

Return values

- **None:**

HAL_COMP_MspDeInit

Function name

void HAL_COMP_MspDeInit (COMP_HandleTypeDef * hcomp)

Function description

DeInitialize the COMP MSP.

Parameters

- **hcomp:** COMP handle

Return values

- **None:**

HAL_COMP_RegisterCallback

Function name

HAL_StatusTypeDef HAL_COMP_RegisterCallback (COMP_HandleTypeDef * hcomp, HAL_COMP_CallbackIDTypeDef CallbackID, pCOMP_CallbackTypeDef pCallback)

Function description

Register a User COMP Callback To be used instead of the weak predefined callback.

Parameters

- **hcomp:** Pointer to a COMP_HandleTypeDef structure that contains the configuration information for the specified COMP.
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_COMP_TRIGGER_CB_ID Trigger callback ID
  - HAL_COMP_MSPINIT_CB_ID MspInit callback ID
  - HAL_COMP_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_COMP_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_COMP_UnRegisterCallback (COMP_HandleTypeDef * hcomp, HAL_COMP_CallbackIDTypeDef CallbackID)

Function description

Unregister a COMP Callback COMP callback is redirected to the weak predefined callback.

Parameters

- **hcomp:** Pointer to a COMP_HandleTypeDef structure that contains the configuration information for the specified COMP.
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_COMP_TRIGGER_CB_ID Trigger callback ID
  - HAL_COMP_MSPINIT_CB_ID MspInit callback ID
  - HAL_COMP_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_COMP_Start

Function name

HAL_StatusTypeDef HAL_COMP_Start (COMP_HandleTypeDef * hcomp)

Function description

Start the comparator.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

HAL_COMP_Stop

Function name

HAL_StatusTypeDef HAL_COMP_Stop (COMP_HandleTypeDef * hcomp)

Function description

Stop the comparator.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

HAL_COMP_Start_IT

Function name

HAL_StatusTypeDef HAL_COMP_Start_IT (COMP_HandleTypeDef * hcomp)

Function description

Enable the interrupt and start the comparator.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

HAL_COMP_Stop_IT

Function name

HAL_StatusTypeDef HAL_COMP_Stop_IT (COMP_HandleTypeDef * hcomp)

Function description

Disable the interrupt and Stop the comparator.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

HAL_COMP_IRQHandler

Function name

void HAL_COMP_IRQHandler (COMP_HandleTypeDef * hcomp)

Function description

Comparator IRQ Handler.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

HAL_COMP_Lock

Function name

HAL_StatusTypeDef HAL_COMP_Lock (COMP_HandleTypeDef * hcomp)

Function description

Lock the selected comparator configuration.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

Notes

- A system reset is required to unlock the comparator configuration.

HAL_COMP_GetOutputLevel

Function name

uint32_t HAL_COMP_GetOutputLevel (COMP_HandleTypeDef * hcomp)

Function description

Return the output level (high or low) of the selected comparator.

Parameters

- **hcomp:** COMP handle

Return values

- **Returns:** the selected comparator output level:
  - COMP_OUTPUT_LEVEL_LOW
  - COMP_OUTPUT_LEVEL_HIGH

Notes

- The output level depends on the selected polarity. If the polarity is not inverted: Comparator output is low when the input plus is at a lower voltage than the input minusComparator output is high when the input plus is at a higher voltage than the input minus If the polarity is inverted:Comparator output is high when the input plus is at a lower voltage than the input minusComparator output is low when the input plus is at a higher voltage than the input minus

HAL_COMP_TriggerCallback

Function name

void HAL_COMP_TriggerCallback (COMP_HandleTypeDef * hcomp)

Function description

Comparator trigger callback.

Parameters

- **hcomp:** COMP handle

Return values

- **None:**

HAL_COMP_GetState

Function name

HAL_COMP_StateTypeDef HAL_COMP_GetState (COMP_HandleTypeDef * hcomp)

Function description

Return the COMP handle state.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** state

HAL_COMP_GetError

Function name

uint32_t HAL_COMP_GetError (COMP_HandleTypeDef * hcomp)

Function description

Return the COMP error code.

Parameters

- **hcomp:** COMP handle

Return values

- **COMP:** error code

11.3 COMP Firmware driver defines

The following section lists the various define and macros of the module.

11.3.1 COMP

COMP

COMP Blanking Source

COMP_BLANKINGSRC_NONE

No blanking source

COMP_BLANKINGSRC_TIM1_OC5

TIM1 OC5 selected as blanking source for comparator

COMP_BLANKINGSRC_TIM2_OC3

TIM2 OC3 selected as blanking source for comparator

COMP_BLANKINGSRC_TIM3_OC3

TIM3 OC3 selected as blanking source for compartor

COMP_BLANKINGSRC_TIM3_OC4

TIM3 OC4 selected as blanking source for comparator

COMP_BLANKINGSRC_TIM8_OC5

TIM8 OC5 selected as blanking source for comparator

COMP_BLANKINGSRC_TIM15_OC1

TIM15 OC1 selected as blanking source for comparator

COMP Error Code

HAL_COMP_ERROR_NONE

No error

HAL_COMP_ERROR_INVALID_CALLBACK

Invalid Callback error

COMP Exported Macros

__HAL_COMP_ENABLE_OR

Description:

- Enable the specified bit in the Option register.

Parameters:

- __AF__: specifies the Alternate Function source selection . This parameter can be one of the following values:
  - COMP_OR_AFOPA6 : Alternate Function PA6 source selection
  - COMP_OR_AFOPA8 : Alternate Function PA8 source selection
  - COMP_OR_AFOPB12 : Alternate Function PB12 source selection
  - COMP_OR_AFOPE6 : Alternate Function PE6 source selection
  - COMP_OR_AFOPE15 : Alternate Function PE15 source selection
  - COMP_OR_AFOPG2 : Alternate Function PG2 source selection
  - COMP_OR_AFOPG3 : Alternate Function PG3 source selection
  - COMP_OR_AFOPG4 : Alternate Function PG4 source selection
  - COMP_OR_AFOPI1 : Alternate Function PI1 source selection
  - COMP_OR_AFOPI4 : Alternate Function PI4 source selection
  - COMP_OR_AFOPK2 : Alternate Function PK2 source selection

Return value:

- None

__HAL_COMP_DISABLE_OR

Description:

- Disable the specified bit in the Option register.

Parameters:

- __AF__: specifies the Alternate Function source selection . This parameter can be one of the following values:
  - COMP_OR_AFOPA6 : Alternate Function PA6 source selection
  - COMP_OR_AFOPA8 : Alternate Function PA8 source selection
  - COMP_OR_AFOPB12 : Alternate Function PB12 source selection
  - COMP_OR_AFOPE6 : Alternate Function PE6 source selection
  - COMP_OR_AFOPE15 : Alternate Function PE15 source selection
  - COMP_OR_AFOPG2 : Alternate Function PG2 source selection
  - COMP_OR_AFOPG3 : Alternate Function PG3 source selection
  - COMP_OR_AFOPG4 : Alternate Function PG4 source selection
  - COMP_OR_AFOPI1 : Alternate Function PI1 source selection
  - COMP_OR_AFOPI4 : Alternate Function PI4 source selection
  - COMP_OR_AFOPK2 : Alternate Function PK2 source selection

Return value:

- None

COMP Exported Types

COMP_STATE_BITFIELD_LOCK

COMP EXTI Lines

COMP_EXTI_LINE_COMP1

EXTI line 20 connected to COMP1 output

COMP_EXTI_LINE_COMP2

EXTI line 21 connected to COMP2 output

COMP_EXTI_IT

EXTI line event with interruption

COMP_EXTI_EVENT

EXTI line event only (without interruption)

COMP_EXTI_RISING

EXTI line event on rising edge

COMP_EXTI_FALLING

EXTI line event on falling edge

COMP external interrupt line management

__HAL_COMP_COMP1_EXTI_ENABLE_RISING_EDGE

Description:

- Enable the COMP1 EXTI line rising edge trigger.

Return value:

- None

__HAL_COMP_COMP1_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the COMP1 EXTI line rising edge trigger.

Return value:

- None

__HAL_COMP_COMP1_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the COMP1 EXTI line falling edge trigger.

Return value:

- None

__HAL_COMP_COMP1_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable the COMP1 EXTI line falling edge trigger.

Return value:

- None

__HAL_COMP_COMP1_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable the COMP1 EXTI line rising & falling edge trigger.

Return value:

- None

__HAL_COMP_COMP1_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable the COMP1 EXTI line rising & falling edge trigger.

Return value:

- None

__HAL_COMP_COMP1_EXTI_ENABLE_IT

Description:

- Enable the COMP1 EXTI line in interrupt mode.

Return value:

- None

__HAL_COMP_COMP1_EXTI_DISABLE_IT

Description:

- Disable the COMP1 EXTI line in interrupt mode.

Return value:

- None

__HAL_COMP_COMP1_EXTI_ENABLE_EVENT

Description:

- Enable the COMP1 EXTI Line in event mode.

Return value:

- None

__HAL_COMP_COMP1_EXTI_DISABLE_EVENT

Description:

- Disable the COMP1 EXTI Line in event mode.

Return value:

- None

__HAL_COMP_COMP1_EXTI_GET_FLAG

Description:

- Check whether the COMP1 EXTI line flag is set or not.

Return value:

- RESET: or SET

__HAL_COMP_COMP1_EXTI_CLEAR_FLAG

Description:

- Clear the COMP1 EXTI flag.

Return value:

- None

__HAL_COMP_COMP1_EXTI_GENERATE_SWIT

Description:

- Generate a software interrupt on the COMP1 EXTI line.

Return value:

- None

__HAL_COMP_COMP1_EXTID3_ENABLE_EVENT

Description:

- Enable the COMP1 D3 EXTI Line in event mode.

Return value:

- None

__HAL_COMP_COMP1_EXTID3_DISABLE_EVENT

Description:

- Disable the COMP1 D3 EXTI Line in event mode.

Return value:

- None

__HAL_COMP_COMP1_EXTID2_ENABLE_IT

Description:

- Enable the COMP1 D2 EXTI line in interrupt mode.

Return value:

- None

__HAL_COMP_COMP1_EXTID2_DISABLE_IT

Description:

- Disable the COMP1 D2 EXTI line in interrupt mode.

Return value:

- None

__HAL_COMP_COMP1_EXTID2_ENABLE_EVENT

Description:

- Enable the COMP1 D2 EXTI Line in event mode.

Return value:

- None

__HAL_COMP_COMP1_EXTID2_DISABLE_EVENT

Description:

- Disable the COMP1 D2 EXTI Line in event mode.

Return value:

- None

__HAL_COMP_COMP1_EXTID2_GET_FLAG

Description:

- Check whether the COMP1 D2 EXTI line flag is set or not.

Return value:

- RESET: or SET

__HAL_COMP_COMP1_EXTID2_CLEAR_FLAG

Description:

- Clear the COMP1 D2 EXTI flag.

Return value:

- None

__HAL_COMP_COMP2_EXTI_ENABLE_RISING_EDGE

Description:

- Enable the COMP2 EXTI line rising edge trigger.

Return value:

- None

__HAL_COMP_COMP2_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the COMP2 EXTI line rising edge trigger.

Return value:

- None

__HAL_COMP_COMP2_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the COMP2 EXTI line falling edge trigger.

Return value:

- None

__HAL_COMP_COMP2_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable the COMP2 EXTI line falling edge trigger.

Return value:

- None

__HAL_COMP_COMP2_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable the COMP2 EXTI line rising & falling edge trigger.

Return value:

- None

__HAL_COMP_COMP2_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable the COMP2 EXTI line rising & falling edge trigger.

Return value:

- None

__HAL_COMP_COMP2_EXTI_ENABLE_IT

Description:

- Enable the COMP2 EXTI line.

Return value:

- None

__HAL_COMP_COMP2_EXTI_DISABLE_IT

Description:

- Disable the COMP2 EXTI line.

Return value:

- None

__HAL_COMP_COMP2_EXTI_ENABLE_EVENT

Description:

- Enable the COMP2 EXTI Line in event mode.

Return value:

- None

__HAL_COMP_COMP2_EXTI_DISABLE_EVENT

Description:

- Disable the COMP2 EXTI Line in event mode.

Return value:

- None

__HAL_COMP_COMP2_EXTI_GET_FLAG

Description:

- Check whether the COMP2 EXTI line flag is set or not.

Return value:

- RESET: or SET

__HAL_COMP_COMP2_EXTI_CLEAR_FLAG

Description:

- Clear the the COMP2 EXTI flag.

Return value:

- None

__HAL_COMP_COMP2_EXTID3_ENABLE_EVENT

Description:

- Enable the COMP2 D3 EXTI Line in event mode.

Return value:

- None

__HAL_COMP_COMP2_EXTID3_DISABLE_EVENT

Description:

- Disable the COMP2 D3 EXTI Line in event mode.

Return value:

- None

__HAL_COMP_COMP2_EXTI_GENERATE_SWIT

Description:

- Generate a software interrupt on the COMP2 EXTI line.

Return value:

- None

__HAL_COMP_COMP2_EXTID2_ENABLE_IT

Description:

- Enable the COMP2 D2 EXTI line.

Return value:

- None

__HAL_COMP_COMP2_EXTID2_DISABLE_IT

Description:

- Disable the COMP2 D2 EXTI line.

Return value:

- None

__HAL_COMP_COMP2_EXTID2_ENABLE_EVENT

Description:

- Enable the COMP2 D2 EXTI Line in event mode.

Return value:

- None

__HAL_COMP_COMP2_EXTID2_DISABLE_EVENT

Description:

- Disable the COMP2 D2 EXTI Line in event mode.

Return value:

- None

__HAL_COMP_COMP2_EXTID2_GET_FLAG

Description:

- Check whether the COMP2 D2 EXTI line flag is set or not.

Return value:

- RESET: or SET

__HAL_COMP_COMP2_EXTID2_CLEAR_FLAG

Description:

- Clear the the COMP2 D2 EXTI flag.

Return value:

- None

__HAL_COMP_GET_IT_SOURCE

Description:

- Checks if the specified COMP interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: specifies the COMP Handle. This parameter can be COMP1 where x: 1 or 2 to select the COMP peripheral.
- __INTERRUPT__: specifies the COMP interrupt source to check. This parameter can be one of the following values:
  - COMP_IT_EN: Comparator interrupt enable

Return value:

- The: new state of __IT__ (TRUE or FALSE)

__HAL_COMP_GET_FLAG

Description:

- Checks whether the specified COMP flag is set or not.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - COMP_FLAG_C1I: Comparator 1 Interrupt Flag
  - COMP_FLAG_C2I: Comparator 2 Interrupt Flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE)

__HAL_COMP_CLEAR_FLAG

Description:

- Clears the specified COMP pending flag.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - COMP_CLEAR_C1IF : Clear Comparator 1 Interrupt Flag
  - COMP_CLEAR_C2IF : Clear Comparator 2 Interrupt Flag

Return value:

- None

__HAL_COMP_CLEAR_C1IFLAG

Description:

- Clear the COMP C1I flag.

Return value:

- None

__HAL_COMP_CLEAR_C2IFLAG

Description:

- Clear the COMP C2I flag.

Return value:

- None

__HAL_COMP_ENABLE_IT

Description:

- Enable the specified COMP interrupt.

Parameters:

- __HANDLE__: specifies the COMP Handle.
- __INTERRUPT__: specifies the COMP interrupt source to enable. This parameter can be one of the following values:
  - COMP_CFGRx_ITEN : Comparator interrupt

Return value:

- None

__HAL_COMP_DISABLE_IT

Description:

- Disable the specified COMP interrupt.

Parameters:

- __HANDLE__: specifies the COMP Handle.
- __INTERRUPT__: specifies the COMP interrupt source to enable. This parameter can be one of the following values:
  - COMP_CFGRx_ITEN : Comparator interrupt

Return value:

- None

COMP output to EXTI

COMP_TRIGGERMODE_NONE

Comparator output triggering no External Interrupt Line

COMP_TRIGGERMODE_IT_RISING

Comparator output triggering External Interrupt Line event with interruption, on rising edge

COMP_TRIGGERMODE_IT_FALLING

Comparator output triggering External Interrupt Line event with interruption, on falling edge

COMP_TRIGGERMODE_IT_RISING_FALLING

Comparator output triggering External Interrupt Line event with interruption, on both rising and falling edges

COMP_TRIGGERMODE_EVENT_RISING

Comparator output triggering External Interrupt Line event only (without interruption), on rising edge

COMP_TRIGGERMODE_EVENT_FALLING

Comparator output triggering External Interrupt Line event only (without interruption), on falling edge

COMP_TRIGGERMODE_EVENT_RISING_FALLING

Comparator output triggering External Interrupt Line event only (without interruption), on both rising and falling edges

COMP Flag

COMP_FLAG_C1I

Comparator 1 Interrupt Flag

COMP_FLAG_C2I

Comparator 2 Interrupt Flag

COMP_FLAG_LOCK

Lock flag

COMP Private macros to get EXTI line associated with Comparators

COMP_GET_EXTI_LINE

Description:

- Get the specified EXTI line for a comparator instance.

Parameters:

- __INSTANCE__: specifies the COMP instance.

Return value:

- value: of

COMP Handle Management

__HAL_COMP_RESET_HANDLE_STATE

Description:

- Reset COMP handle state.

Parameters:

- __HANDLE__: COMP handle

Return value:

- None

COMP_CLEAR_ERRORCODE

Description:

- Clear COMP error code (set it to no error code "HAL_COMP_ERROR_NONE").

Parameters:

- __HANDLE__: COMP handle

Return value:

- None

__HAL_COMP_ENABLE

Description:

- Enable the specified comparator.

Parameters:

- __HANDLE__: COMP handle

Return value:

- None

__HAL_COMP_DISABLE

Description:

- Disable the specified comparator.

Parameters:

- __HANDLE__: COMP handle

Return value:

- None

__HAL_COMP_LOCK

Description:

- Lock the specified comparator configuration.

Parameters:

- __HANDLE__: COMP handle

Return value:

- None

Notes:

- Using this macro induce HAL COMP handle state machine being no more in line with COMP instance state. To keep HAL COMP handle state machine updated, it is recommended to use function "HAL_COMP_Lock')".

__HAL_COMP_IS_LOCKED

Description:

- Check whether the specified comparator is locked.

Parameters:

- __HANDLE__: COMP handle

Return value:

- Value: 0 if COMP instance is not locked, value 1 if COMP instance is locked

COMP hysteresis

COMP_HYSTERESIS_NONE

No hysteresis

COMP_HYSTERESIS_LOW

Hysteresis level low

COMP_HYSTERESIS_MEDIUM

Hysteresis level medium

COMP_HYSTERESIS_HIGH

Hysteresis level high

COMP input minus (inverting input)

COMP_INPUT_MINUS_1_4VREFINT

Comparator input minus connected to 1/4 VrefInt

COMP_INPUT_MINUS_1_2VREFINT

Comparator input minus connected to 1/2 VrefInt

COMP_INPUT_MINUS_3_4VREFINT

Comparator input minus connected to 3/4 VrefInt

COMP_INPUT_MINUS_VREFINT

Comparator input minus connected to VrefInt

COMP_INPUT_MINUS_DAC1_CH1

Comparator input minus connected to DAC1 channel 1 (DAC_OUT1)

COMP_INPUT_MINUS_DAC1_CH2

Comparator input minus connected to DAC1 channel 2 (DAC_OUT2)

COMP_INPUT_MINUS_IO1

Comparator input minus connected to IO1 (pin PB1 for COMP1, pin PE10 for COMP2)

COMP_INPUT_MINUS_IO2

Comparator input minus connected to IO2 (pin PC4 for COMP1, pin PE7 for COMP2)

COMP input plus (non-inverting input)

COMP_INPUT_PLUS_IO1

Comparator input plus connected to IO1 (pin PB0 for COMP1, pin PE9 for COMP2)

COMP_INPUT_PLUS_IO2

Comparator input plus connected to IO2 (pin PB2 for COMP1, pin PE11 for COMP2)

COMP Interrupts Definitions

COMP_IT_EN

COMP private macros to check input parameters

IS_COMP_WINDOWMODE

IS_COMP_POWERMODE

IS_COMP_INPUT_PLUS

IS_COMP_INPUT_MINUS

IS_COMP_HYSTERESIS

IS_COMP_OUTPUTPOL

IS_COMP_BLANKINGSRCE

IS_COMP_TRIGGERMODE

IS_COMP_OUTPUT_LEVEL

COMP Interruption Clear Flags

COMP_CLEAR_C1IF

Clear Comparator 1 Interrupt Flag

COMP_CLEAR_C2IF

Clear Comparator 2 Interrupt Flag

COMP Output Level

COMP_OUTPUT_LEVEL_LOW

COMP_OUTPUT_LEVEL_HIGH

COMP Output Polarity

COMP_OUTPUTPOL_NONINVERTED

COMP output level is not inverted (comparator output is high when the input plus is at a higher voltage than the input minus)

COMP_OUTPUTPOL_INVERTED

COMP output level is inverted (comparator output is low when the input plus is at a higher voltage than the input minus)

COMP power mode

COMP_POWERMODE_HIGHSPEED

High Speed

COMP_POWERMODE_MEDIUMSPEED

Medium Speed

COMP_POWERMODE_ULTRALOWPOWER

Ultra-low power mode

COMP Window Mode

COMP_WINDOWMODE_DISABLE

Window mode disable: Comparators instances pair COMP1 and COMP2 are independent

COMP_WINDOWMODE_COMP1_INPUT_PLUS_COMMON

Window mode enable: Comparators instances pair COMP1 and COMP2 have their input plus connected together. The common input is COMP1 input plus (COMP2 input plus is no more accessible).

12 HAL CORDIC Generic Driver

12.1 CORDIC Firmware driver registers structures

12.1.1 __CORDIC_HandleTypeDef

__CORDIC_HandleTypeDef is defined in the stm32h7xx_hal_cordic.h

Data Fields

- ***CORDIC_TypeDef * Instance***
- ***const int32_t * pInBuff***
- ***int32_t * pOutBuff***
- ***uint32_t NbCalcToOrder***
- ***uint32_t NbCalcToGet***
- ***uint32_t DMADirection***
- ***DMA_HandleTypeDef * hdmaIn***
- ***DMA_HandleTypeDef * hdmaOut***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_CORDIC_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* ErrorCallback***
- ***void(* CalculateCpltCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***CORDIC_TypeDef* __CORDIC_HandleTypeDef::Instance***
  Register base address
- ***const int32_t* __CORDIC_HandleTypeDef::pInBuff***
  Pointer to CORDIC input data buffer
- ***int32_t* __CORDIC_HandleTypeDef::pOutBuff***
  Pointer to CORDIC output data buffer
- ***uint32_t __CORDIC_HandleTypeDef::NbCalcToOrder***
  Remaining number of calculation to order
- ***uint32_t __CORDIC_HandleTypeDef::NbCalcToGet***
  Remaining number of calculation result to get
- ***uint32_t __CORDIC_HandleTypeDef::DMADirection***
  Direction of CORDIC DMA transfers
- ***DMA_HandleTypeDef* __CORDIC_HandleTypeDef::hdmaIn***
  CORDIC peripheral input data DMA handle parameters
- ***DMA_HandleTypeDef* __CORDIC_HandleTypeDef::hdmaOut***
  CORDIC peripheral output data DMA handle parameters
- ***HAL_LockTypeDef __CORDIC_HandleTypeDef::Lock***
  CORDIC locking object
- ***__IO HAL_CORDIC_StateTypeDef __CORDIC_HandleTypeDef::State***
  CORDIC state
- ***__IO uint32_t __CORDIC_HandleTypeDef::ErrorCode***
  CORDIC peripheral error code This parameter can be a value of ***CORDIC_Error_Code***

- ***void(* __CORDIC_HandleTypeDef::ErrorCallback)(struct __CORDIC_HandleTypeDef *hcordic)***
  CORDIC error callback
- ***void(* __CORDIC_HandleTypeDef::CalculateCpltCallback)(struct __CORDIC_HandleTypeDef *hcordic)***
  CORDIC calculate complete callback
- ***void(* __CORDIC_HandleTypeDef::MspInitCallback)(struct __CORDIC_HandleTypeDef *hcordic)***
  CORDIC Msp Init callback
- ***void(* __CORDIC_HandleTypeDef::MspDeInitCallback)(struct __CORDIC_HandleTypeDef *hcordic)***
  CORDIC Msp DeInit callback

12.1.2 CORDIC_ConfigTypeDef

CORDIC_ConfigTypeDef is defined in the stm32h7xx_hal_cordic.h

Data Fields

- ***uint32_t Function***
- ***uint32_t Scale***
- ***uint32_t InSize***
- ***uint32_t OutSize***
- ***uint32_t NbWrite***
- ***uint32_t NbRead***
- ***uint32_t Precision***

Field Documentation

- ***uint32_t CORDIC_ConfigTypeDef::Function***
  Function This parameter can be a value of ***CORDIC_Function***
- ***uint32_t CORDIC_ConfigTypeDef::Scale***
  Scaling factor This parameter can be a value of ***CORDIC_Scale***
- ***uint32_t CORDIC_ConfigTypeDef::InSize***
  Width of input data This parameter can be a value of ***CORDIC_In_Size***
- ***uint32_t CORDIC_ConfigTypeDef::OutSize***
  Width of output data This parameter can be a value of ***CORDIC_Out_Size***
- ***uint32_t CORDIC_ConfigTypeDef::NbWrite***
  Number of 32-bit write expected for one calculation This parameter can be a value of ***CORDIC_Nb_Write***
- ***uint32_t CORDIC_ConfigTypeDef::NbRead***
  Number of 32-bit read expected after one calculation This parameter can be a value of ***CORDIC_Nb_Read***
- ***uint32_t CORDIC_ConfigTypeDef::Precision***
  Number of cycles for calculation This parameter can be a value of ***CORDIC_Precision_In_Cycles_Number***

12.2 CORDIC Firmware driver API description

The following section lists the various functions of the CORDIC library.

12.2.1 How to use this driver

The CORDIC HAL driver can be used as follows:

1. Initialize the CORDIC low level resources by implementing the HAL_CORDIC_MspInit():
   - Enable the CORDIC interface clock using __HAL_RCC_CORDIC_CLK_ENABLE()
   - In case of using interrupts (e.g. HAL_CORDIC_Calculate_IT())
     - Configure the CORDIC interrupt priority using HAL_NVIC_SetPriority()
     - Enable the CORDIC IRQ handler using HAL_NVIC_EnableIRQ()
     - In CORDIC IRQ handler, call HAL_CORDIC_IRQHandler()
   - In case of using DMA to control data transfer (e.g. HAL_CORDIC_Calculate_DMA())
     - Enable the DMA2 interface clock using __HAL_RCC_DMA2_CLK_ENABLE()
     - Configure and enable two DMA channels one for managing data transfer from memory to peripheral (input channel) and another channel for managing data transfer from peripheral to memory (output channel)
     - Associate the initialized DMA handle to the CORDIC DMA handle using __HAL_LINKDMA()
     - Configure the priority and enable the NVIC for the transfer complete interrupt on the two DMA channels. Resort to HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ()
2. Initialize the CORDIC HAL using HAL_CORDIC_Init(). This function
   - resorts to HAL_CORDIC_MspInit() for low-level initialization,
3. Configure CORDIC processing (calculation) using HAL_CORDIC_Configure(). This function configures:
   - Processing functions: Cosine, Sine, Phase, Modulus, Arctangent, Hyperbolic cosine, Hyperbolic sine, Hyperbolic arctangent, Natural log, Square root
   - Scaling factor: 1 to 2exp(-7)
   - Width of input data: 32 bits input data size (Q1.31 format) or 16 bits input data size (Q1.15 format)
   - Width of output data: 32 bits output data size (Q1.31 format) or 16 bits output data size (Q1.15 format)
   - Number of 32-bit write expected for one calculation: One 32-bits write or Two 32-bit write
   - Number of 32-bit read expected after one calculation: One 32-bits read or Two 32-bit read
   - Precision: 1 to 15 cycles for calculation (the more cycles, the better precision)
4. Four processing (calculation) functions are available:
   - Polling mode: processing API is blocking function i.e. it processes the data and wait till the processing is finished API is HAL_CORDIC_Calculate
   - Polling Zero-overhead mode: processing API is blocking function i.e. it processes the data and wait till the processing is finished A bit faster than standard polling mode, but blocking also AHB bus API is HAL_CORDIC_CalculateZO
   - Interrupt mode: processing API is not blocking functions i.e. it processes the data under interrupt API is HAL_CORDIC_Calculate_IT
   - DMA mode: processing API is not blocking functions and the CPU is not used for data transfer, i.e. the data transfer is ensured by DMA API is HAL_CORDIC_Calculate_DMA
5. Call HAL_CORDIC_DeInit() to de-initialize the CORDIC peripheral. This function
   - resorts to HAL_CORDIC_MspDeInit() for low-level de-initialization,

Callback registration

12.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the CORDIC peripheral and the associated handle
- DeInitialize the CORDIC peripheral
- Initialize the CORDIC MSP (MCU Specific Package)
- De-Initialize the CORDIC MSP

This section contains the following APIs:

- ***HAL_CORDIC_Init()***
- ***HAL_CORDIC_DeInit()***
- ***HAL_CORDIC_MspInit()***
- ***HAL_CORDIC_MspDeInit()***

- *HAL_CORDIC_RegisterCallback()*
- *HAL_CORDIC_UnRegisterCallback()*
- *HAL_CORDIC_RegisterCallback()*

12.2.3 Peripheral Control functions

This section provides functions allowing to:

- Configure the CORDIC peripheral: function, precision, scaling factor, number of input data and output data, size of input data and output data.
- Calculate output data of CORDIC processing on input date, using the existing CORDIC configuration

Four processing functions are available for calculation:

- Polling mode
- Polling mode, with Zero-Overhead register access
- Interrupt mode
- DMA mode

This section contains the following APIs:

- *HAL_CORDIC_Configure()*
- *HAL_CORDIC_Calculate()*
- *HAL_CORDIC_CalculateZO()*
- *HAL_CORDIC_Calculate_IT()*
- *HAL_CORDIC_Calculate_DMA()*

12.2.4 Callback functions

This section provides Interruption and DMA callback functions:

- DMA or Interrupt calculate complete
- DMA or Interrupt error

This section contains the following APIs:

- *HAL_CORDIC_ErrorCallback()*
- *HAL_CORDIC_CalculateCpltCallback()*

12.2.5 IRQ handler management

This section provides IRQ handler function.

This section contains the following APIs:

- *HAL_CORDIC_IRQHandler()*

12.2.6 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral.

This section contains the following APIs:

- *HAL_CORDIC_GetState()*
- *HAL_CORDIC_GetError()*

12.2.7 Detailed description of functions

HAL_CORDIC_Init

Function name

HAL_StatusTypeDef HAL_CORDIC_Init (CORDIC_HandleTypeDef * hcordic)

Function description

Initialize the CORDIC peripheral and the associated handle.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure.

Return values

- **HAL:** status

HAL_CORDIC_DeInit

Function name

HAL_StatusTypeDef HAL_CORDIC_DeInit (CORDIC_HandleTypeDef * hcordic)

Function description

DeInitialize the CORDIC peripheral.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure.

Return values

- **HAL:** status

HAL_CORDIC_MspInit

Function name

void HAL_CORDIC_MspInit (CORDIC_HandleTypeDef * hcordic)

Function description

Initialize the CORDIC MSP.

Parameters

- **hcordic:** CORDIC handle

Return values

- **None:**

HAL_CORDIC_MspDeInit

Function name

void HAL_CORDIC_MspDeInit (CORDIC_HandleTypeDef * hcordic)

Function description

DeInitialize the CORDIC MSP.

Parameters

- **hcordic:** CORDIC handle

Return values

- **None:**

HAL_CORDIC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_CORDIC_RegisterCallback (CORDIC_HandleTypeDef * hcordic, HAL_CORDIC_CallbackIDTypeDef CallbackID, pCORDIC_CallbackTypeDef pCallback)

Function description

HAL_CORDIC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_CORDIC_UnRegisterCallback (CORDIC_HandleTypeDef * hcordic, HAL_CORDIC_CallbackIDTypeDef CallbackID)

Function description

Unregister a CORDIC CallBack.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_CORDIC_ERROR_CB_ID error Callback ID
  - HAL_CORDIC_CALCULATE_CPLT_CB_ID calculate complete Callback ID
  - HAL_CORDIC_MSPINIT_CB_ID MspInit callback ID
  - HAL_CORDIC_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_CORDIC_Configure

Function name

HAL_StatusTypeDef HAL_CORDIC_Configure (CORDIC_HandleTypeDef * hcordic, const CORDIC_ConfigTypeDef * sConfig)

Function description

Configure the CORDIC processing according to the specified parameters in the CORDIC_ConfigTypeDef structure.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module
- **sConfig:** pointer to a CORDIC_ConfigTypeDef structure that contains the CORDIC configuration information.

Return values

- **HAL:** status

HAL_CORDIC_Calculate

Function name

HAL_StatusTypeDef HAL_CORDIC_Calculate (CORDIC_HandleTypeDef * hcordic, const int32_t * pInBuff, int32_t * pOutBuff, uint32_t NbCalc, uint32_t Timeout)

Function description

Carry out data of CORDIC processing in polling mode, according to the existing CORDIC configuration.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module.
- **pInBuff:** Pointer to buffer containing input data for CORDIC processing.
- **pOutBuff:** Pointer to buffer where output data of CORDIC processing will be stored.
- **NbCalc:** Number of CORDIC calculation to process.
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_CORDIC_CalculateZO

Function name

HAL_StatusTypeDef HAL_CORDIC_CalculateZO (CORDIC_HandleTypeDef * hcordic, const int32_t * pInBuff, int32_t * pOutBuff, uint32_t NbCalc, uint32_t Timeout)

Function description

Carry out data of CORDIC processing in Zero-Overhead mode (output data being read soon as input data are written), according to the existing CORDIC configuration.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module.
- **pInBuff:** Pointer to buffer containing input data for CORDIC processing.
- **pOutBuff:** Pointer to buffer where output data of CORDIC processing will be stored.
- **NbCalc:** Number of CORDIC calculation to process.
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_CORDIC_Calculate_IT

Function name

HAL_StatusTypeDef HAL_CORDIC_Calculate_IT (CORDIC_HandleTypeDef * hcordic, const int32_t * pInBuff, int32_t * pOutBuff, uint32_t NbCalc)

Function description

Carry out data of CORDIC processing in interrupt mode, according to the existing CORDIC configuration.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module.
- **pInBuff:** Pointer to buffer containing input data for CORDIC processing.
- **pOutBuff:** Pointer to buffer where output data of CORDIC processing will be stored.
- **NbCalc:** Number of CORDIC calculation to process.

Return values

- **HAL:** status

HAL_CORDIC_Calculate_DMA

Function name

HAL_StatusTypeDef HAL_CORDIC_Calculate_DMA (CORDIC_HandleTypeDef * hcordic, const int32_t * pInBuff, int32_t * pOutBuff, uint32_t NbCalc, uint32_t DMADirection)

Function description

Carry out input and/or output data of CORDIC processing in DMA mode, according to the existing CORDIC configuration.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module.
- **pInBuff:** Pointer to buffer containing input data for CORDIC processing.
- **pOutBuff:** Pointer to buffer where output data of CORDIC processing will be stored.
- **NbCalc:** Number of CORDIC calculation to process.
- **DMADirection:** Direction of DMA transfers. This parameter can be one of the following values:
  – CORDIC DMA direction CORDIC DMA direction

Return values

- **HAL:** status

Notes

- pInBuff or pOutBuff is unused in case of unique DMADirection transfer, and can be set to NULL value in this case.
- pInBuff and pOutBuff buffers must be 32-bit aligned to ensure a correct DMA transfer to and from the Peripheral.

HAL_CORDIC_ErrorCallback

Function name

void HAL_CORDIC_ErrorCallback (CORDIC_HandleTypeDef * hcordic)

Function description

CORDIC error callback.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module

Return values

- **None:**

HAL_CORDIC_CalculateCpltCallback

Function name

void HAL_CORDIC_CalculateCpltCallback (CORDIC_HandleTypeDef * hcordic)

Function description

CORDIC calculate complete callback.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module

Return values

- **None:**

HAL_CORDIC_IRQHandler

Function name

void HAL_CORDIC_IRQHandler (CORDIC_HandleTypeDef * hcordic)

Function description

Handle CORDIC interrupt request.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module

Return values

- **None:**

HAL_CORDIC_GetState

Function name

HAL_CORDIC_StateTypeDef HAL_CORDIC_GetState (const CORDIC_HandleTypeDef * hcordic)

Function description

Return the CORDIC handle state.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module

Return values

- **HAL:** state

HAL_CORDIC_GetError

Function name

uint32_t HAL_CORDIC_GetError (const CORDIC_HandleTypeDef * hcordic)

Function description

Return the CORDIC peripheral error.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module

Return values

- **Error:** bit-map

Notes

- The returned error is a bit-map combination of possible errors

12.3 CORDIC Firmware driver defines

The following section lists the various define and macros of the module.

12.3.1 CORDIC

CORDIC

DMA Read Request Enable bit

CORDIC_DMA_REN

DMA Read requests enable

DMA Write Request Enable bit

CORDIC_DMA_WEN

DMA Write channel enable

CORDIC DMA direction

CORDIC_DMA_DIR_NONE

DMA direction : none

CORDIC_DMA_DIR_IN

DMA direction : Input of CORDIC

CORDIC_DMA_DIR_OUT

DMA direction : Output of CORDIC

CORDIC_DMA_DIR_IN_OUT

DMA direction : Input and Output of CORDIC

CORDIC Error code

HAL_CORDIC_ERROR_NONE

No error

HAL_CORDIC_ERROR_PARAM

Wrong parameter error

HAL_CORDIC_ERROR_NOT_READY

Peripheral not ready

HAL_CORDIC_ERROR_TIMEOUT

Timeout error

HAL_CORDIC_ERROR_DMA

DMA error

HAL_CORDIC_ERROR_INVALID_CALLBACK

Invalid Callback error

CORDIC Exported Macros

__HAL_CORDIC_RESET_HANDLE_STATE

Description:

- Reset CORDIC handle state.

Parameters:

- __HANDLE__: CORDIC handle

Return value:

- None

__HAL_CORDIC_ENABLE_IT

Description:

- Enable the CORDIC interrupt when result is ready.

Parameters:

- __HANDLE__: CORDIC handle.
- __INTERRUPT__: CORDIC Interrupt. This parameter can be one of the following values:
  - CORDIC_IT_IEN Enable Interrupt

Return value:

- None

__HAL_CORDIC_DISABLE_IT

Description:

- Disable the CORDIC interrupt.

Parameters:

- __HANDLE__: CORDIC handle.
- __INTERRUPT__: CORDIC Interrupt. This parameter can be one of the following values:
  - CORDIC_IT_IEN Enable Interrupt

Return value:

- None

__HAL_CORDIC_GET_IT

Description:

- Check whether the specified CORDIC interrupt occurred or not.

Parameters:

- __HANDLE__: CORDIC handle.
- __INTERRUPT__: CORDIC interrupt to check

Return value:

- SET: (interrupt occurred) or RESET (interrupt did not occurred)

__HAL_CORDIC_CLEAR_IT

Description:

- Clear specified CORDIC interrupt status.

Parameters:

- __HANDLE__: CORDIC handle.
- __INTERRUPT__: CORDIC interrupt to clear

Return value:

- None

__HAL_CORDIC_GET_FLAG

Description:

- Check whether the specified CORDIC status flag is set or not.

Parameters:

- __HANDLE__: CORDIC handle.
- __FLAG__: CORDIC flag to check This parameter can be one of the following values:
  - CORDIC_FLAG_RRDY Result Ready Flag

Return value:

- SET: (flag is set) or RESET (flag is reset)

__HAL_CORDIC_CLEAR_FLAG

Description:

- Clear specified CORDIC status flag.

Parameters:

- __HANDLE__: CORDIC handle.
- __FLAG__: CORDIC flag to clear This parameter can be one of the following values:
  - CORDIC_FLAG_RRDY Result Ready Flag

Return value:

- None

__HAL_CORDIC_GET_IT_SOURCE

Description:

- Check whether the specified CORDIC interrupt is enabled or not.

Parameters:

- __HANDLE__: CORDIC handle.
- __INTERRUPT__: CORDIC interrupt to check This parameter can be one of the following values:
  - CORDIC_IT_IEN Enable Interrupt

Return value:

- FlagStatus

CORDIC status flags

CORDIC_FLAG_RRDY

Result Ready Flag

CORDIC Function

CORDIC_FUNCTION_COSINE

Cosine

CORDIC_FUNCTION_SINE

Sine

CORDIC_FUNCTION_PHASE

Phase

CORDIC_FUNCTION_MODULUS

Modulus

CORDIC_FUNCTION_ARCTANGENT

Arctangent

CORDIC_FUNCTION_HCOSINE

Hyperbolic Cosine

CORDIC_FUNCTION_HSINE

Hyperbolic Sine

CORDIC_FUNCTION_HARCTANGENT

Hyperbolic Arctangent

CORDIC_FUNCTION_NATURALLOG

Natural Logarithm

CORDIC_FUNCTION_SQUAREROOT

Square Root

CORDIC Interrupts Enable bit

CORDIC_IT_IEN

Result ready interrupt enable

CORDIC input data size

CORDIC_INSIZE_32BITS

32 bits input data size (Q1.31 format)

CORDIC_INSIZE_16BITS

16 bits input data size (Q1.15 format)

CORDIC Number of 32-bit read required after one calculation

CORDIC_NBREAD_1

One 32-bits read containing either only one 32-bit data output (Q1.31 format), or two 16-bit data output (Q1.15 format) packed in one 32 bits Data

CORDIC_NBREAD_2

Two 32-bit Data containing two 32-bits data output (Q1.31 format)

CORDIC Number of 32-bit write required for one calculation

CORDIC_NBWRITE_1

One 32-bits write containing either only one 32-bit data input (Q1.31 format), or two 16-bit data input (Q1.15 format) packed in one 32 bits Data

CORDIC_NBWRITE_2

Two 32-bit write containing two 32-bits data input (Q1.31 format)

CORDIC Results Size

CORDIC_OUTSIZE_32BITS

32 bits output data size (Q1.31 format)

CORDIC_OUTSIZE_16BITS

16 bits output data size (Q1.15 format)

CORDIC Precision in Cycles Number

CORDIC_PRECISION_1CYCLE

CORDIC_PRECISION_2CYCLES

CORDIC_PRECISION_3CYCLES

CORDIC_PRECISION_4CYCLES

CORDIC_PRECISION_5CYCLES

CORDIC_PRECISION_6CYCLES

CORDIC_PRECISION_7CYCLES

CORDIC_PRECISION_8CYCLES

CORDIC_PRECISION_9CYCLES

CORDIC_PRECISION_10CYCLES

CORDIC_PRECISION_11CYCLES

CORDIC_PRECISION_12CYCLES

CORDIC_PRECISION_13CYCLES

CORDIC_PRECISION_14CYCLES

CORDIC_PRECISION_15CYCLES

CORDIC Scaling factor

CORDIC_SCALE_0

CORDIC_SCALE_1

CORDIC_SCALE_2

CORDIC_SCALE_3

CORDIC_SCALE_4

CORDIC_SCALE_5

CORDIC_SCALE_6

CORDIC_SCALE_7

13 HAL CORTEX Generic Driver

13.1 CORTEX Firmware driver API description

The following section lists the various functions of the CORTEX library.

13.1.1 How to use this driver

How to configure Interrupts using CORTEX HAL driver

This section provides functions allowing to configure the NVIC interrupts (IRQ). The Cortex-M exceptions are managed by CMSIS functions.

1. Configure the NVIC Priority Grouping using HAL_NVIC_SetPriorityGrouping() function according to the following table.
2. Configure the priority of the selected IRQ Channels using HAL_NVIC_SetPriority().
3. Enable the selected IRQ Channels using HAL_NVIC_EnableIRQ().
4. please refer to programming manual for details in how to configure priority.

Note: When the NVIC_PRIORITYGROUP_0 is selected, IRQ preemption is no more possible. The pending IRQ priority will be managed only by the sub priority.

Note: IRQ priority order (sorted by highest to lowest priority):

- *Lowest preemption priority*
- *Lowest sub priority*
- *Lowest hardware priority (IRQ number)*

How to configure Systick using CORTEX HAL driver

Setup SysTick Timer for time base.

- The HAL_SYSTICK_Config() function calls the SysTick_Config() function which is a CMSIS function that:
  - Configures the SysTick Reload register with value passed as function parameter.
  - Configures the SysTick IRQ priority to the lowest value (0x0F).
  - Resets the SysTick Counter register.
  - Configures the SysTick Counter clock source to be Core Clock Source (HCLK).
  - Enables the SysTick Interrupt.
  - Starts the SysTick Counter.
- You can change the SysTick Clock source to be HCLK_Div8 by calling the macro HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK_DIV8) just after the HAL_SYSTICK_Config() function call. The HAL_SYSTICK_CLKSourceConfig() macro is defined inside the stm32h7xx_hal_cortex.h file.
- You can change the SysTick IRQ priority by calling the HAL_NVIC_SetPriority(SysTick_IRQn,...) function just after the HAL_SYSTICK_Config() function call. The HAL_NVIC_SetPriority() call the NVIC_SetPriority() function which is a CMSIS function.
- To adjust the SysTick time base, use the following formula: Reload Value = SysTick Counter Clock (Hz) x Desired Time base (s)
  - Reload Value is the parameter to be passed for HAL_SYSTICK_Config() function
  - Reload Value should not exceed 0xFFFFFF

13.1.2 Initialization and de-initialization functions

This section provides the CORTEX HAL driver functions allowing to configure Interrupts Systick functionalities

This section contains the following APIs:

- ***HAL_NVIC_SetPriorityGrouping()***

- *HAL_NVIC_SetPriority()*
- *HAL_NVIC_EnableIRQ()*
- *HAL_NVIC_DisableIRQ()*
- *HAL_NVIC_SystemReset()*
- *HAL_SYSTICK_Config()*

13.1.3 Peripheral Control functions

This subsection provides a set of functions allowing to control the CORTEX (NVIC, SYSTICK, MPU) functionalities.

This section contains the following APIs:

- *HAL_NVIC_GetPriorityGrouping()*
- *HAL_NVIC_GetPriority()*
- *HAL_NVIC_SetPendingIRQ()*
- *HAL_NVIC_GetPendingIRQ()*
- *HAL_NVIC_ClearPendingIRQ()*
- *HAL_NVIC_GetActive()*
- *HAL_SYSTICK_CLKSourceConfig()*
- *HAL_SYSTICK_IRQHandler()*
- *HAL_SYSTICK_Callback()*
- *HAL_GetCurrentCPUID()*

13.1.4 Detailed description of functions

HAL_NVIC_SetPriorityGrouping

Function name

void HAL_NVIC_SetPriorityGrouping (uint32_t PriorityGroup)

Function description

Sets the priority grouping field (preemption priority and subpriority) using the required unlock sequence.

Parameters

- **PriorityGroup:** The priority grouping bits length. This parameter can be one of the following values:
  - NVIC_PRIORITYGROUP_0: 0 bits for preemption priority 4 bits for subpriority
  - NVIC_PRIORITYGROUP_1: 1 bits for preemption priority 3 bits for subpriority
  - NVIC_PRIORITYGROUP_2: 2 bits for preemption priority 2 bits for subpriority
  - NVIC_PRIORITYGROUP_3: 3 bits for preemption priority 1 bits for subpriority
  - NVIC_PRIORITYGROUP_4: 4 bits for preemption priority 0 bits for subpriority

Return values

- **None:**

Notes

- When the NVIC_PriorityGroup_0 is selected, IRQ preemption is no more possible. The pending IRQ priority will be managed only by the subpriority.

HAL_NVIC_SetPriority

Function name

void HAL_NVIC_SetPriority (IRQn_Type IRQn, uint32_t PreemptPriority, uint32_t SubPriority)

Function description

Sets the priority of an interrupt.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32h7xxxx.h))
- **PreemptPriority:** The preemption priority for the IRQn channel. This parameter can be a value between 0 and 15 A lower priority value indicates a higher priority
- **SubPriority:** the subpriority level for the IRQ channel. This parameter can be a value between 0 and 15 A lower priority value indicates a higher priority.

Return values

- **None:**

HAL_NVIC_EnableIRQ

Function name

void HAL_NVIC_EnableIRQ (IRQn_Type IRQn)

Function description

Enables a device specific interrupt in the NVIC interrupt controller.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32h7xxxx.h))

Return values

- **None:**

Notes

- To configure interrupts priority correctly, the NVIC_PriorityGroupConfig() function should be called before.

HAL_NVIC_DisableIRQ

Function name

void HAL_NVIC_DisableIRQ (IRQn_Type IRQn)

Function description

Disables a device specific interrupt in the NVIC interrupt controller.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32h7xxxx.h))

Return values

- **None:**

HAL_NVIC_SystemReset

Function name

void HAL_NVIC_SystemReset (void)

Function description

Initiates a system reset request to reset the MCU.

Return values

- **None:**

HAL_SYSTICK_Config

Function name

uint32_t HAL_SYSTICK_Config (uint32_t TicksNumb)

Function description

Initializes the System Timer and its interrupt, and starts the System Tick Timer.

Parameters

- **TicksNumb:** Specifies the ticks Number of ticks between two interrupts.

Return values

- **status:** - 0 Function succeeded.
  - 1 Function failed.

HAL_NVIC_GetPriorityGrouping

Function name

uint32_t HAL_NVIC_GetPriorityGrouping (void)

Function description

Gets the priority grouping field from the NVIC Interrupt Controller.

Return values

- **Priority:** grouping field (SCB->AIRCR [10:8] PRIGROUP field)

HAL_NVIC_GetPriority

Function name

void HAL_NVIC_GetPriority (IRQn_Type IRQn, uint32_t PriorityGroup, uint32_t * pPreemptPriority, uint32_t * pSubPriority)

Function description

Gets the priority of an interrupt.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32h7xxxx.h))
- **PriorityGroup:** the priority grouping bits length. This parameter can be one of the following values:
  - NVIC_PRIORITYGROUP_0: 0 bits for preemption priority 4 bits for subpriority
  - NVIC_PRIORITYGROUP_1: 1 bits for preemption priority 3 bits for subpriority
  - NVIC_PRIORITYGROUP_2: 2 bits for preemption priority 2 bits for subpriority
  - NVIC_PRIORITYGROUP_3: 3 bits for preemption priority 1 bits for subpriority
  - NVIC_PRIORITYGROUP_4: 4 bits for preemption priority 0 bits for subpriority
- **pPreemptPriority:** Pointer on the Preemptive priority value (starting from 0).
- **pSubPriority:** Pointer on the Subpriority value (starting from 0).

Return values

- **None:**

HAL_NVIC_GetPendingIRQ

Function name

uint32_t HAL_NVIC_GetPendingIRQ (IRQn_Type IRQn)

Function description

Gets Pending Interrupt (reads the pending register in the NVIC and returns the pending bit for the specified interrupt).

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32h7xxxx.h))

Return values

- **status:** - 0 Interrupt status is not pending.
  - 1 Interrupt status is pending.

HAL_NVIC_SetPendingIRQ

Function name

void HAL_NVIC_SetPendingIRQ (IRQn_Type IRQn)

Function description

Sets Pending bit of an external interrupt.

Parameters

- **IRQn:** External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32h7xxxx.h))

Return values

- **None:**

HAL_NVIC_ClearPendingIRQ

Function name

void HAL_NVIC_ClearPendingIRQ (IRQn_Type IRQn)

Function description

Clears the pending bit of an external interrupt.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32h7xxxx.h))

Return values

- **None:**

HAL_NVIC_GetActive

Function name

uint32_t HAL_NVIC_GetActive (IRQn_Type IRQn)

Function description

Gets active interrupt (reads the active register in NVIC and returns the active bit).

Parameters

- **IRQn:** External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32h7xxxx.h))

Return values

- **status:** - 0 Interrupt status is not pending.
  - 1 Interrupt status is pending.

HAL_SYSTICK_CLKSourceConfig

Function name

void HAL_SYSTICK_CLKSourceConfig (uint32_t CLKSource)

Function description

Configures the SysTick clock source.

Parameters

- **CLKSource:** specifies the SysTick clock source. This parameter can be one of the following values:
  - SYSTICK_CLKSOURCE_HCLK_DIV8: AHB clock divided by 8 selected as SysTick clock source.
  - SYSTICK_CLKSOURCE_HCLK: AHB clock selected as SysTick clock source.

Return values

- **None:**

HAL_SYSTICK_IRQHandler

Function name

void HAL_SYSTICK_IRQHandler (void)

Function description

This function handles SYSTICK interrupt request.

Return values

- **None:**

HAL_SYSTICK_Callback

Function name

void HAL_SYSTICK_Callback (void)

Function description

SYSTICK callback.

Return values

- **None:**

HAL_GetCurrentCPUID

Function name

uint32_t HAL_GetCurrentCPUID (void)

Function description

Returns the current CPU ID.

Return values

- **CPU:** identifier

13.2 CORTEX Firmware driver defines

The following section lists the various define and macros of the module.

13.2.1 CORTEX

CORTEX

CORTEX_CPU_Identifier

CM7_CPUID

CM4_CPUID

CORTEX Preemption Priority Group

NVIC_PRIORITYGROUP_0

0 bits for pre-emption priority 4 bits for subpriority

NVIC_PRIORITYGROUP_1

1 bits for pre-emption priority 3 bits for subpriority

NVIC_PRIORITYGROUP_2

2 bits for pre-emption priority 2 bits for subpriority

NVIC_PRIORITYGROUP_3

3 bits for pre-emption priority 1 bits for subpriority

NVIC_PRIORITYGROUP_4

4 bits for pre-emption priority 0 bits for subpriority

CORTEX _SysTick clock source

SYSTICK_CLKSOURCE_HCLK_DIV8

SYSTICK_CLKSOURCE_HCLK

14 HAL CRC Generic Driver

14.1 CRC Firmware driver registers structures

14.1.1 CRC_InitTypeDef

CRC_InitTypeDef is defined in the stm32h7xx_hal_crc.h

Data Fields

- ***uint8_t DefaultPolynomialUse***
- ***uint8_t DefaultInitValueUse***
- ***uint32_t GeneratingPolynomial***
- ***uint32_t CRCLength***
- ***uint32_t InitValue***
- ***uint32_t InputDataInversionMode***
- ***uint32_t OutputDataInversionMode***

Field Documentation

- ***uint8_t CRC_InitTypeDef::DefaultPolynomialUse***

  This parameter is a value of ***CRC_Default_Polynomial*** and indicates if default polynomial is used. If set to DEFAULT_POLYNOMIAL_ENABLE, resort to default X^32 + X^26 + X^23 + X^22 + X^16 + X^12 + X^11 + X^10 +X^8 + X^7 + X^5 + X^4 + X^2+ X +1. In that case, there is no need to set GeneratingPolynomial field. If otherwise set to DEFAULT_POLYNOMIAL_DISABLE, GeneratingPolynomial and CRCLength fields must be set.

- ***uint8_t CRC_InitTypeDef::DefaultInitValueUse***

  This parameter is a value of ***CRC_Default_InitValue_Use*** and indicates if default init value is used. If set to DEFAULT_INIT_VALUE_ENABLE, resort to default 0xFFFFFFFF value. In that case, there is no need to set InitValue field. If otherwise set to DEFAULT_INIT_VALUE_DISABLE, InitValue field must be set.

- ***uint32_t CRC_InitTypeDef::GeneratingPolynomial***

  Set CRC generating polynomial as a 7, 8, 16 or 32-bit long value for a polynomial degree respectively equal to 7, 8, 16 or 32. This field is written in normal, representation e.g., for a polynomial of degree 7, X^7 + X^6 + X^5 + X^2 + 1 is written 0x65. No need to specify it if DefaultPolynomialUse is set to DEFAULT_POLYNOMIAL_ENABLE.

- ***uint32_t CRC_InitTypeDef::CRCLength***

  This parameter is a value of ***CRC_Polynomial_Sizes*** and indicates CRC length. Value can be either one of
  - **CRC_POLYLENGTH_32B** (32-bit CRC),
  - **CRC_POLYLENGTH_16B** (16-bit CRC),
  - **CRC_POLYLENGTH_8B** (8-bit CRC),
  - **CRC_POLYLENGTH_7B** (7-bit CRC).

- ***uint32_t CRC_InitTypeDef::InitValue***

  Init value to initiate CRC computation. No need to specify it if DefaultInitValueUse is set to DEFAULT_INIT_VALUE_ENABLE.

- ***uint32_t CRC_InitTypeDef::InputDataInversionMode***

  This parameter is a value of ***CRCEx_Input_Data_Inversion*** and specifies input data inversion mode. Can be either one of the following values
  - **CRC_INPUTDATA_INVERSION_NONE** no input data inversion
  - **CRC_INPUTDATA_INVERSION_BYTE** byte-wise inversion, 0x1A2B3C4D becomes 0x58D43CB2
  - **CRC_INPUTDATA_INVERSION_HALFWORD** halfword-wise inversion, 0x1A2B3C4D becomes 0xD458B23C
  - **CRC_INPUTDATA_INVERSION_WORD** word-wise inversion, 0x1A2B3C4D becomes 0xB23CD458

- ***uint32_t CRC_InitTypeDef::OutputDataInversionMode***

  This parameter is a value of ***CRCEx_Output_Data_Inversion*** and specifies output data (i.e. CRC) inversion mode. Can be either
  - **CRC_OUTPUTDATA_INVERSION_DISABLE** no CRC inversion,
  - **CRC_OUTPUTDATA_INVERSION_ENABLE** CRC 0x11223344 is converted into 0x22CC4488

14.1.2 CRC_HandleTypeDef

CRC_HandleTypeDef is defined in the stm32h7xx_hal_crc.h

Data Fields

- ***CRC_TypeDef * Instance***
- ***CRC_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_CRC_StateTypeDef State***
- ***uint32_t InputDataFormat***

Field Documentation

- ***CRC_TypeDef* CRC_HandleTypeDef::Instance***

  Register base address
- ***CRC_InitTypeDef CRC_HandleTypeDef::Init***

  CRC configuration parameters
- ***HAL_LockTypeDef CRC_HandleTypeDef::Lock***

  CRC Locking object
- ***__IO HAL_CRC_StateTypeDef CRC_HandleTypeDef::State***

  CRC communication state
- ***uint32_t CRC_HandleTypeDef::InputDataFormat***

  This parameter is a value of ***CRC_Input_Buffer_Format*** and specifies input data format. Can be either
  - **CRC_INPUTDATA_FORMAT_BYTES** input data is a stream of bytes (8-bit data)
  - **CRC_INPUTDATA_FORMAT_HALFWORDS** input data is a stream of half-words (16-bit data)
  - **CRC_INPUTDATA_FORMAT_WORDS** input data is a stream of words (32-bit data)

  Note that constant CRC_INPUT_FORMAT_UNDEFINED is defined but an initialization error must occur if InputBufferFormat is not one of the three values listed above

14.2 CRC Firmware driver API description

The following section lists the various functions of the CRC library.

14.2.1 How to use this driver

- Enable CRC AHB clock using __HAL_RCC_CRC_CLK_ENABLE();
- Initialize CRC calculator
  - specify generating polynomial (peripheral default or non-default one)
  - specify initialization value (peripheral default or non-default one)
  - specify input data format
  - specify input or output data inversion mode if any
- Use HAL_CRC_Accumulate() function to compute the CRC value of the input data buffer starting with the previously computed CRC as initialization value
- Use HAL_CRC_Calculate() function to compute the CRC value of the input data buffer starting with the defined initialization value (default or non-default) to initiate CRC calculation

14.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the CRC according to the specified parameters in the CRC_InitTypeDef and create the associated handle
- DeInitialize the CRC peripheral
- Initialize the CRC MSP (MCU Specific Package)
- DeInitialize the CRC MSP

This section contains the following APIs:

- ***HAL_CRC_Init()***
- ***HAL_CRC_DeInit()***
- ***HAL_CRC_MspInit()***
- ***HAL_CRC_MspDeInit()***

14.2.3 Peripheral Control functions

This section provides functions allowing to:

- compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer using combination of the previous CRC value and the new one.

or

- compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer independently of the previous CRC value.

This section contains the following APIs:

- ***HAL_CRC_Accumulate()***
- ***HAL_CRC_Calculate()***

14.2.4 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral.

This section contains the following APIs:

- ***HAL_CRC_GetState()***

14.2.5 Detailed description of functions

HAL_CRC_Init

Function name

HAL_StatusTypeDef HAL_CRC_Init (CRC_HandleTypeDef * hcrc)

Function description

Initialize the CRC according to the specified parameters in the CRC_InitTypeDef and create the associated handle.

Parameters

- **hcrc:** CRC handle

Return values

- **HAL:** status

HAL_CRC_DeInit

Function name

HAL_StatusTypeDef HAL_CRC_DeInit (CRC_HandleTypeDef * hcrc)

Function description

DeInitialize the CRC peripheral.

Parameters

- **hcrc:** CRC handle

Return values

- **HAL:** status

HAL_CRC_MspInit

Function name

void HAL_CRC_MspInit (CRC_HandleTypeDef * hcrc)

Function description

Initializes the CRC MSP.

Parameters

- **hcrc:** CRC handle

Return values

- **None:**

HAL_CRC_MspDeInit

Function name

void HAL_CRC_MspDeInit (CRC_HandleTypeDef * hcrc)

Function description

DeInitialize the CRC MSP.

Parameters

- **hcrc:** CRC handle

Return values

- **None:**

HAL_CRC_Accumulate

Function name

uint32_t HAL_CRC_Accumulate (CRC_HandleTypeDef * hcrc, uint32_t pBuffer, uint32_t BufferLength)

Function description

Compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer starting with the previously computed CRC as initialization value.

Parameters

- **hcrc:** CRC handle
- **pBuffer:** pointer to the input data buffer, exact input data format is provided by hcrc->InputDataFormat.
- **BufferLength:** input data buffer length (number of bytes if pBuffer type is * uint8_t, number of half-words if pBuffer type is * uint16_t, number of words if pBuffer type is * uint32_t).

Return values

- **uint32_t:** CRC (returned value LSBs for CRC shorter than 32 bits)

Notes

- By default, the API expects a uint32_t pointer as input buffer parameter. Input buffer pointers with other types simply need to be cast in uint32_t and the API will internally adjust its input data processing based on the handle field hcrc->InputDataFormat.

HAL_CRC_Calculate

Function name

uint32_t HAL_CRC_Calculate (CRC_HandleTypeDef * hcrc, uint32_t pBuffer, uint32_t BufferLength)

Function description

Compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer starting with hcrc->Instance->INIT as initialization value.

Parameters

- **hcrc:** CRC handle
- **pBuffer:** pointer to the input data buffer, exact input data format is provided by hcrc->InputDataFormat.
- **BufferLength:** input data buffer length (number of bytes if pBuffer type is * uint8_t, number of half-words if pBuffer type is * uint16_t, number of words if pBuffer type is * uint32_t).

Return values

- **uint32_t:** CRC (returned value LSBs for CRC shorter than 32 bits)

Notes

- By default, the API expects a uint32_t pointer as input buffer parameter. Input buffer pointers with other types simply need to be cast in uint32_t and the API will internally adjust its input data processing based on the handle field hcrc->InputDataFormat.

HAL_CRC_GetState

Function name

HAL_CRC_StateTypeDef HAL_CRC_GetState (CRC_HandleTypeDef * hcrc)

Function description

Return the CRC handle state.

Parameters

- **hcrc:** CRC handle

Return values

- **HAL:** state

14.3 CRC Firmware driver defines

The following section lists the various define and macros of the module.

14.3.1 CRC

CRC

Default CRC computation initialization value

DEFAULT_CRC_INITVALUE

Initial CRC default value

Indicates whether or not default init value is used

DEFAULT_INIT_VALUE_ENABLE

Enable initial CRC default value

DEFAULT_INIT_VALUE_DISABLE

Disable initial CRC default value

Indicates whether or not default polynomial is used

DEFAULT_POLYNOMIAL_ENABLE

Enable default generating polynomial 0x04C11DB7

DEFAULT_POLYNOMIAL_DISABLE

Disable default generating polynomial 0x04C11DB7

Default CRC generating polynomial

DEFAULT_CRC32_POLY

X^32 + X^26 + X^23 + X^22 + X^16 + X^12 + X^11 + X^10 +X^8 + X^7 + X^5 + X^4 + X^2+ X +1

CRC Exported Macros

__HAL_CRC_RESET_HANDLE_STATE

Description:

- Reset CRC handle state.

Parameters:

- __HANDLE__: CRC handle.

Return value:

- None

__HAL_CRC_DR_RESET

Description:

- Reset CRC Data Register.

Parameters:

- __HANDLE__: CRC handle

Return value:

- None

__HAL_CRC_INITIALCRCVALUE_CONFIG

Description:

- Set CRC INIT non-default value.

Parameters:

- __HANDLE__: CRC handle
- __INIT__: 32-bit initial value

Return value:

- None

__HAL_CRC_SET_IDR

Description:

- Store data in the Independent Data (ID) register.

Parameters:

- __HANDLE__: CRC handle
- __VALUE__: Value to be stored in the ID register

Return value:

- None

Notes:

- Refer to the Reference Manual to get the authorized __VALUE__ length in bits

__HAL_CRC_GET_IDR

Description:

- Return the data stored in the Independent Data (ID) register.

Parameters:

- __HANDLE__: CRC handle

Return value:

- Value: of the ID register

Notes:

- Refer to the Reference Manual to get the authorized __VALUE__ length in bits

Input Buffer Format

CRC_INPUTDATA_FORMAT_UNDEFINED

Undefined input data format

CRC_INPUTDATA_FORMAT_BYTES

Input data in byte format

CRC_INPUTDATA_FORMAT_HALFWORDS

Input data in half-word format

CRC_INPUTDATA_FORMAT_WORDS

Input data in word format

Polynomial sizes to configure the peripheral

CRC_POLYLENGTH_32B

Resort to a 32-bit long generating polynomial

CRC_POLYLENGTH_16B

Resort to a 16-bit long generating polynomial

CRC_POLYLENGTH_8B

Resort to a 8-bit long generating polynomial

CRC_POLYLENGTH_7B

Resort to a 7-bit long generating polynomial

CRC polynomial possible sizes actual definitions

HAL_CRC_LENGTH_32B

32-bit long CRC

HAL_CRC_LENGTH_16B

16-bit long CRC

HAL_CRC_LENGTH_8B

8-bit long CRC

HAL_CRC_LENGTH_7B

7-bit long CRC

15 HAL CRC Extension Driver

15.1 CRCEx Firmware driver API description

The following section lists the various functions of the CRCEx library.

15.1.1 How to use this driver

- Set user-defined generating polynomial through HAL_CRCEx_Polynomial_Set()
- Configure Input or Output data inversion

15.1.2 Extended configuration functions

This section provides functions allowing to:

- Configure the generating polynomial
- Configure the input data inversion
- Configure the output data inversion

This section contains the following APIs:

- ***HAL_CRCEx_Polynomial_Set()***
- ***HAL_CRCEx_Input_Data_Reverse()***
- ***HAL_CRCEx_Output_Data_Reverse()***

15.1.3 Detailed description of functions

HAL_CRCEx_Polynomial_Set

Function name

HAL_StatusTypeDef HAL_CRCEx_Polynomial_Set (CRC_HandleTypeDef * hcrc, uint32_t Pol, uint32_t PolyLength)

Function description

Initialize the CRC polynomial if different from default one.

Parameters

- **hcrc:** CRC handle
- **Pol:** CRC generating polynomial (7, 8, 16 or 32-bit long). This parameter is written in normal representation, e.g.
  - for a polynomial of degree 7, X^7 + X^6 + X^5 + X^2 + 1 is written 0x65
  - for a polynomial of degree 16, X^16 + X^12 + X^5 + 1 is written 0x1021
- **PolyLength:** CRC polynomial length. This parameter can be one of the following values:
  - CRC_POLYLENGTH_7B 7-bit long CRC (generating polynomial of degree 7)
  - CRC_POLYLENGTH_8B 8-bit long CRC (generating polynomial of degree 8)
  - CRC_POLYLENGTH_16B 16-bit long CRC (generating polynomial of degree 16)
  - CRC_POLYLENGTH_32B 32-bit long CRC (generating polynomial of degree 32)

Return values

- **HAL:** status

HAL_CRCEx_Input_Data_Reverse

Function name

HAL_StatusTypeDef HAL_CRCEx_Input_Data_Reverse (CRC_HandleTypeDef * hcrc, uint32_t InputReverseMode)

Function description

Set the Reverse Input data mode.

Parameters

- **hcrc:** CRC handle
- **InputReverseMode:** Input Data inversion mode. This parameter can be one of the following values:
  - CRC_INPUTDATA_INVERSION_NONE no change in bit order (default value)
  - CRC_INPUTDATA_INVERSION_BYTE Byte-wise bit reversal
  - CRC_INPUTDATA_INVERSION_HALFWORD HalfWord-wise bit reversal
  - CRC_INPUTDATA_INVERSION_WORD Word-wise bit reversal

Return values

- **HAL:** status

HAL_CRCEx_Output_Data_Reverse

Function name

HAL_StatusTypeDef HAL_CRCEx_Output_Data_Reverse (CRC_HandleTypeDef * hcrc, uint32_t OutputReverseMode)

Function description

Set the Reverse Output data mode.

Parameters

- **hcrc:** CRC handle
- **OutputReverseMode:** Output Data inversion mode. This parameter can be one of the following values:
  - CRC_OUTPUTDATA_INVERSION_DISABLE no CRC inversion (default value)
  - CRC_OUTPUTDATA_INVERSION_ENABLE bit-level inversion (e.g. for a 8-bit CRC: 0xB5 becomes 0xAD)

Return values

- **HAL:** status

15.2 CRCEx Firmware driver defines

The following section lists the various define and macros of the module.

15.2.1 CRCEx

CRCEx

CRC Extended Exported Macros

__HAL_CRC_OUTPUTREVERSAL_ENABLE

Description:

- Set CRC output reversal.

Parameters:

- __HANDLE__: CRC handle

Return value:

- None

__HAL_CRC_OUTPUTREVERSAL_DISABLE

Description:

- Unset CRC output reversal.

Parameters:

- __HANDLE__: CRC handle

Return value:

- None

__HAL_CRC_POLYNOMIAL_CONFIG

Description:

- Set CRC non-default polynomial.

Parameters:

- __HANDLE__: CRC handle
- __POLYNOMIAL__: 7, 8, 16 or 32-bit polynomial

Return value:

- None

Input Data Inversion Modes

CRC_INPUTDATA_INVERSION_NONE

No input data inversion

CRC_INPUTDATA_INVERSION_BYTE

Byte-wise input data inversion

CRC_INPUTDATA_INVERSION_HALFWORD

HalfWord-wise input data inversion

CRC_INPUTDATA_INVERSION_WORD

Word-wise input data inversion

Output Data Inversion Modes

CRC_OUTPUTDATA_INVERSION_DISABLE

No output data inversion

CRC_OUTPUTDATA_INVERSION_ENABLE

Bit-wise output data inversion

16 HAL CRYP Generic Driver

16.1 CRYP Firmware driver registers structures

16.1.1 CRYP_ConfigTypeDef

CRYP_ConfigTypeDef is defined in the stm32h7xx_hal_cryp.h

Data Fields

- ***uint32_t DataType***
- ***uint32_t KeySize***
- ***uint32_t * pKey***
- ***uint32_t * pInitVect***
- ***uint32_t Algorithm***
- ***uint32_t * Header***
- ***uint32_t HeaderSize***
- ***uint32_t * B0***
- ***uint32_t DataWidthUnit***
- ***uint32_t HeaderWidthUnit***
- ***uint32_t KeyIVConfigSkip***

Field Documentation

- ***uint32_t CRYP_ConfigTypeDef::DataType***
  no swap(32-bit data), halfword swap(16-bit data), byte swap(8-bit data) or bit swap(1-bit data).this parameter can be a value of ***CRYP_Data_Type***
- ***uint32_t CRYP_ConfigTypeDef::KeySize***
  Used only in AES mode : 128, 192 or 256 bit key length in CRYP1. This parameter can be a value of ***CRYP_Key_Size***
- ***uint32_t* CRYP_ConfigTypeDef::pKey***
  The key used for encryption/decryption
- ***uint32_t* CRYP_ConfigTypeDef::pInitVect***
  The initialization vector used also as initialization counter in CTR mode
- ***uint32_t CRYP_ConfigTypeDef::Algorithm***
  DES/ TDES Algorithm ECB/CBC AES Algorithm ECB/CBC/CTR/GCM or CCM This parameter can be a value of ***CRYP_Algorithm_Mode***
- ***uint32_t* CRYP_ConfigTypeDef::Header***
  used only in AES GCM and CCM Algorithm for authentication, GCM : also known as Additional Authentication Data CCM : named B1 composed of the associated data length and Associated Data.
- ***uint32_t CRYP_ConfigTypeDef::HeaderSize***
  The size of header buffer
- ***uint32_t* CRYP_ConfigTypeDef::B0***
  B0 is first authentication block used only in AES CCM mode
- ***uint32_t CRYP_ConfigTypeDef::DataWidthUnit***
  Payload data Width Unit, this parameter can be value of ***CRYP_Data_Width_Unit***
- ***uint32_t CRYP_ConfigTypeDef::HeaderWidthUnit***
  Header Width Unit, this parameter can be value of ***CRYP_Header_Width_Unit***
- ***uint32_t CRYP_ConfigTypeDef::KeyIVConfigSkip***
  CRYP peripheral Key and IV configuration skip, to configure Key and Initialization Vector only once and to skip configuration for consecutive processing. This parameter can be a value of ***CRYP_Configuration_Skip***

16.1.2 __CRYP_HandleTypeDef

__CRYP_HandleTypeDef is defined in the stm32h7xx_hal_cryp.h

Data Fields

- ***CRYP_TypeDef * Instance***
- ***CRYP_ConfigTypeDef Init***
- ***uint32_t * pCrypInBuffPtr***
- ***uint32_t * pCrypOutBuffPtr***
- ***__IO uint16_t CrypHeaderCount***
- ***__IO uint16_t CrypInCount***
- ***__IO uint16_t CrypOutCount***
- ***uint16_t Size***
- ***uint32_t Phase***
- ***DMA_HandleTypeDef * hdmain***
- ***DMA_HandleTypeDef * hdmaout***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_CRYP_STATETypeDef State***
- ***__IO uint32_t ErrorCode***
- ***uint32_t Version***
- ***uint32_t KeyIVConfig***
- ***uint32_t SizesSum***
- ***void(* InCpltCallback***
- ***void(* OutCpltCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***CRYP_TypeDef* __CRYP_HandleTypeDef::Instance***
  CRYP registers base address
- ***CRYP_ConfigTypeDef __CRYP_HandleTypeDef::Init***
  CRYP required parameters
- ***uint32_t* __CRYP_HandleTypeDef::pCrypInBuffPtr***
  Pointer to CRYP processing (encryption, decryption,...) buffer
- ***uint32_t* __CRYP_HandleTypeDef::pCrypOutBuffPtr***
  Pointer to CRYP processing (encryption, decryption,...) buffer
- ***__IO uint16_t __CRYP_HandleTypeDef::CrypHeaderCount***
  Counter of header data
- ***__IO uint16_t __CRYP_HandleTypeDef::CrypInCount***
  Counter of input data
- ***__IO uint16_t __CRYP_HandleTypeDef::CrypOutCount***
  Counter of output data
- ***uint16_t __CRYP_HandleTypeDef::Size***
  length of input data in word or in byte, according to DataWidthUnit
- ***uint32_t __CRYP_HandleTypeDef::Phase***
  CRYP peripheral phase
- ***DMA_HandleTypeDef* __CRYP_HandleTypeDef::hdmain***
  CRYP In DMA handle parameters
- ***DMA_HandleTypeDef* __CRYP_HandleTypeDef::hdmaout***
  CRYP Out DMA handle parameters

- ***HAL_LockTypeDef __CRYP_HandleTypeDef::Lock***
  CRYP locking object
- ***__IO HAL_CRYP_STATETypeDef __CRYP_HandleTypeDef::State***
  CRYP peripheral state
- ***__IO uint32_t __CRYP_HandleTypeDef::ErrorCode***
  CRYP peripheral error code
- ***uint32_t __CRYP_HandleTypeDef::Version***
  CRYP1 IP version
- ***uint32_t __CRYP_HandleTypeDef::KeyIVConfig***
  CRYP peripheral Key and IV configuration flag, used when configuration can be skipped
- ***uint32_t __CRYP_HandleTypeDef::SizesSum***
  Sum of successive payloads lengths (in bytes), stored for a single signature computation after several messages processing
- ***void(* __CRYP_HandleTypeDef::InCpltCallback)(struct __CRYP_HandleTypeDef *hcryp)***
  CRYP Input FIFO transfer completed callback
- ***void(* __CRYP_HandleTypeDef::OutCpltCallback)(struct __CRYP_HandleTypeDef *hcryp)***
  CRYP Output FIFO transfer completed callback
- ***void(* __CRYP_HandleTypeDef::ErrorCallback)(struct __CRYP_HandleTypeDef *hcryp)***
  CRYP Error callback
- ***void(* __CRYP_HandleTypeDef::MspInitCallback)(struct __CRYP_HandleTypeDef *hcryp)***
  CRYP Msp Init callback
- ***void(* __CRYP_HandleTypeDef::MspDeInitCallback)(struct __CRYP_HandleTypeDef *hcryp)***
  CRYP Msp DeInit callback

16.2 CRYP Firmware driver API description

The following section lists the various functions of the CRYP library.

16.2.1 How to use this driver

The CRYP HAL driver can be used in CRYP IP as follows:

1. Initialize the CRYP low level resources by implementing the HAL_CRYP_MspInit():
   a. Enable the CRYP interface clock using __HAL_RCC_CRYP_CLK_ENABLE()
   b. In case of using interrupts (e.g. HAL_CRYP_Encrypt_IT())
      - Configure the CRYP interrupt priority using HAL_NVIC_SetPriority()
      - Enable the CRYP IRQ handler using HAL_NVIC_EnableIRQ()
      - In CRYP IRQ handler, call HAL_CRYP_IRQHandler()
   c. In case of using DMA to control data transfer (e.g. HAL_CRYP_Encrypt_DMA())
      - Enable the DMAx interface clock using __RCC_DMAx_CLK_ENABLE()
      - Configure and enable two DMA streams one for managing data transfer from memory to peripheral (input stream) and another stream for managing data transfer from peripheral to memory (output stream)
      - Associate the initialized DMA handle to the CRYP DMA handle using __HAL_LINKDMA()
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the two DMA Streams. The output stream should have higher priority than the input stream HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ()

2. Initialize the CRYP according to the specified parameters :
   a. The data type: bit swap(1-bit data), byte swap(8-bit data), half word swap(16-bit data) or no swap(32-bit data).
   b. The key size: 128, 192 or 256.
   c. The AlgoMode DES/ TDES Algorithm ECB/CBC or AES Algorithm ECB/CBC/CTR/GCM or CCM.
   d. The initialization vector (counter). It is not used in ECB mode.
   e. The key buffer used for encryption/decryption.
   f. The Header used only in AES GCM and CCM Algorithm for authentication.
   g. The HeaderSize The size of header buffer in word.
   h. The B0 block is the first authentication block used only in AES CCM mode.
3. Three processing (encryption/decryption) functions are available:
   a. Polling mode: encryption and decryption APIs are blocking functions i.e. they process the data and wait till the processing is finished, e.g. HAL_CRYP_Encrypt & HAL_CRYP_Decrypt
   b. Interrupt mode: encryption and decryption APIs are not blocking functions i.e. they process the data under interrupt, e.g. HAL_CRYP_Encrypt_IT & HAL_CRYP_Decrypt_IT
   c. DMA mode: encryption and decryption APIs are not blocking functions i.e. the data transfer is ensured by DMA, e.g. HAL_CRYP_Encrypt_DMA & HAL_CRYP_Decrypt_DMA
4. When the processing function is called at first time after HAL_CRYP_Init() the CRYP peripheral is configured and processes the buffer in input. At second call, no need to Initialize the CRYP, user have to get current configuration via HAL_CRYP_GetConfig() API, then only HAL_CRYP_SetConfig() is requested to set new parametres, finally user can start encryption/decryption.
5. Call HAL_CRYP_DeInit() to deinitialize the CRYP peripheral.
6. To process a single message with consecutive calls to HAL_CRYP_Encrypt() or HAL_CRYP_Decrypt() without having to configure again the Key or the Initialization Vector between each API call, the field KeyIVConfigSkip of the initialization structure must be set to CRYP_KEYIVCONFIG_ONCE. Same is true for consecutive calls of HAL_CRYP_Encrypt_IT(), HAL_CRYP_Decrypt_IT(), HAL_CRYP_Encrypt_DMA() or HAL_CRYP_Decrypt_DMA().

The cryptographic processor supports following standards:

1. The data encryption standard (DES) and Triple-DES (TDES) supported only by CRYP1 IP:
   a. 64-bit data block processing
   b. chaining modes supported :
      - Electronic Code Book(ECB)
      - Cipher Block Chaining (CBC)
   c. keys length supported :64-bit, 128-bit and 192-bit.
2. The advanced encryption standard (AES) supported by CRYP1:
   a. 128-bit data block processing
   b. chaining modes supported :
      - Electronic Code Book(ECB)
      - Cipher Block Chaining (CBC)
      - Counter mode (CTR)
      - Galois/counter mode (GCM/GMAC)
      - Counter with Cipher Block Chaining-Message(CCM)
   c. keys length Supported :
      - for CRYP1 IP: 128-bit, 192-bit and 256-bit.

This section describes the AES Galois/counter mode (GCM) supported by both CRYP1 IP:

1. Algorithm supported :
   a. Galois/counter mode (GCM)
   b. Galois message authentication code (GMAC) :is exactly the same as GCM algorithm composed only by an header.

2. Four phases are performed in GCM :
   a. Init phase: IP prepares the GCM hash subkey (H) and do the IV processing
   b. Header phase: IP processes the Additional Authenticated Data (AAD), with hash computation only.
   c. Payload phase: IP processes the plaintext (P) with hash computation + keystream encryption + data XORing. It works in a similar way for ciphertext (C).
   d. Final phase: IP generates the authenticated tag (T) using the last block of data. HAL_CRYPEx_AESGCM_GenerateAuthTAG API used in this phase to generate 4 words which correspond to the Tag. user should consider only part of this 4 words, if Tag length is less than 128 bits.
3. structure of message construction in GCM is defined as below :
   a. 16 bytes Initial Counter Block (ICB)composed of IV and counter
   b. The authenticated header A (also knows as Additional Authentication Data AAD) this part of the message is only authenticated, not encrypted.
   c. The plaintext message P is both authenticated and encrypted as ciphertext. GCM standard specifies that ciphertext has same bit length as the plaintext.
   d. The last block is composed of the length of A (on 64 bits) and the length of ciphertext (on 64 bits)

This section describe The AES Counter with Cipher Block Chaining-Message Authentication Code (CCM) supported by both CRYP1 IP:

1. Specific parameters for CCM :
   a. B0 block : According to NIST Special Publication 800-38C, The first block B0 is formatted as follows, where l(m) is encoded in most-significant-byte first order(see below table 3)
      - Q: a bit string representation of the octet length of P (plaintext)
      - q The octet length of the binary representation of the octet length of the payload
      - A nonce (N), n The octet length of the where n+q=15.
      - Flags: most significant octet containing four flags for control information,
      - t The octet length of the MAC.
   b. B1 block (header) : associated data length(a) concatenated with Associated Data (A) the associated data length expressed in bytes (a) defined as below:
      - If 0 < a < 216-28, then it is encoded as [a]16, i.e. two octets
      - If 216-28 < a < 232, then it is encoded as 0xff || 0xfe || [a]32, i.e. six octets
      - If 232 < a < 264, then it is encoded as 0xff || 0xff || [a]64, i.e. ten octets
   c. CTRx block : control blocks
      - Generation of CTR1 from first block B0 information : equal to B0 with first 5 bits zeroed and most significant bits storing octet length of P also zeroed, then incremented by one (see below Table 4)
      - Generation of CTR0: same as CTR1 with bit[0] set to zero.
2. Four phases are performed in CCM for CRYP1 IP:
   a. Init phase: IP prepares the GCM hash subkey (H) and do the IV processing
   b. Header phase: IP processes the Additional Authenticated Data (AAD), with hash computation only.
   c. Payload phase: IP processes the plaintext (P) with hash computation + keystream encryption + data XORing. It works in a similar way for ciphertext (C).
   d. Final phase: IP generates the authenticated tag (T) using the last block of data. HAL_CRYPEx_AESCCM_GenerateAuthTAG API used in this phase to generate 4 words which correspond to the Tag. user should consider only part of this 4 words, if Tag length is less than 128 bits

Callback registration

The compilation define USE_HAL_CRYP_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions @ref HAL_CRYP_RegisterCallback() or HAL_CRYP_RegisterXXXCallback() to register an interrupt callback.

Function @ref HAL_CRYP_RegisterCallback() allows to register following callbacks:

- InCpltCallback : Input FIFO transfer completed callback.
- OutCpltCallback : Output FIFO transfer completed callback.
- ErrorCallback : callback for error detection.
- MspInitCallback : CRYP MspInit.

- MspDeInitCallback : CRYP MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function @ref HAL_CRYP_UnRegisterCallback() to reset a callback to the default weak function. @ref HAL_CRYP_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- InCpltCallback : Input FIFO transfer completed callback.
- OutCpltCallback : Output FIFO transfer completed callback.
- ErrorCallback : callback for error detection.
- MspInitCallback : CRYP MspInit.
- MspDeInitCallback : CRYP MspDeInit.

By default, after the @ref HAL_CRYP_Init() and when the state is HAL_CRYP_STATE_RESET all callbacks are set to the corresponding weak functions : examples @ref HAL_CRYP_InCpltCallback() , @ref HAL_CRYP_OutCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak function in the @ref HAL_CRYP_Init()/ @ref HAL_CRYP_DeInit() only when these callbacks are null (not registered beforehand). if not, MspInit or MspDeInit are not null, the @ref HAL_CRYP_Init() / @ref HAL_CRYP_DeInit() keep and use the user MspInit/MspDeInit functions (registered beforehand)

Callbacks can be registered/unregistered in HAL_CRYP_STATE_READY state only. Exception done MspInit/MspDeInit callbacks that can be registered/unregistered in HAL_CRYP_STATE_READY or HAL_CRYP_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/ DeInit. In that case first register the MspInit/MspDeInit user callbacks using @ref HAL_CRYP_RegisterCallback() before calling @ref HAL_CRYP_DeInit() or @ref HAL_CRYP_Init() function.

When The compilation define USE_HAL_CRYP_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

16.2.2 Initialization, de-initialization and Set and Get configuration functions

This section provides functions allowing to:

- Initialize the CRYP
- DeInitialize the CRYP
- Initialize the CRYP MSP
- DeInitialize the CRYP MSP
- configure CRYP (HAL_CRYP_SetConfig) with the specified parameters in the CRYP_ConfigTypeDef Parameters which are configured in This section are :
  - Key size
  - Data Type : 32,16, 8 or 1bit
  - AlgoMode : for CRYP1 IP ECB and CBC in DES/TDES Standard ECB,CBC,CTR,GCM/GMAC and CCM in AES Standard.
- Get CRYP configuration (HAL_CRYP_GetConfig) from the specified parameters in the CRYP_HandleTypeDef

This section contains the following APIs:

- ***HAL_CRYP_Init()***
- ***HAL_CRYP_DeInit()***
- ***HAL_CRYP_SetConfig()***
- ***HAL_CRYP_GetConfig()***
- ***HAL_CRYP_MspInit()***
- ***HAL_CRYP_MspDeInit()***
- ***HAL_CRYP_RegisterCallback()***
- ***HAL_CRYP_UnRegisterCallback()***

16.2.3 Encrypt Decrypt functions

This section provides API allowing to Encrypt/Decrypt Data following Standard DES/TDES or AES, and Algorithm configured by the user:

- Standard DES/TDES only supported by CRYP1 IP, below list of Algorithm supported :
  - Electronic Code Book(ECB)
  - Cipher Block Chaining (CBC)
- Standard AES supported by CRYP1 IP , list of Algorithm supported:
  - Electronic Code Book(ECB)
  - Cipher Block Chaining (CBC)
  - Counter mode (CTR)
  - Cipher Block Chaining (CBC)
  - Counter mode (CTR)
  - Galois/counter mode (GCM)
  - Counter with Cipher Block Chaining-Message(CCM)

Three processing functions are available:

- Polling mode : HAL_CRYP_Encrypt & HAL_CRYP_Decrypt
- Interrupt mode : HAL_CRYP_Encrypt_IT & HAL_CRYP_Decrypt_IT
- DMA mode : HAL_CRYP_Encrypt_DMA & HAL_CRYP_Decrypt_DMA

This section contains the following APIs:

- ***HAL_CRYP_Encrypt()***
- ***HAL_CRYP_Decrypt()***
- ***HAL_CRYP_Encrypt_IT()***
- ***HAL_CRYP_Decrypt_IT()***
- ***HAL_CRYP_Encrypt_DMA()***
- ***HAL_CRYP_Decrypt_DMA()***

16.2.4 CRYP IRQ handler management

This section provides CRYP IRQ handler and callback functions.

- HAL_CRYP_IRQHandler CRYP interrupt request
- HAL_CRYP_InCpltCallback input data transfer complete callback
- HAL_CRYP_OutCpltCallback output data transfer complete callback
- HAL_CRYP_ErrorCallback CRYP error callback
- HAL_CRYP_GetState return the CRYP state
- HAL_CRYP_GetError return the CRYP error code

This section contains the following APIs:

- ***HAL_CRYP_IRQHandler()***
- ***HAL_CRYP_GetError()***
- ***HAL_CRYP_GetState()***
- ***HAL_CRYP_InCpltCallback()***
- ***HAL_CRYP_OutCpltCallback()***
- ***HAL_CRYP_ErrorCallback()***

16.2.5 Detailed description of functions

HAL_CRYP_Init

Function name

HAL_StatusTypeDef HAL_CRYP_Init (CRYP_HandleTypeDef * hcryp)

Function description

Initializes the CRYP according to the specified parameters in the CRYP_ConfigTypeDef and creates the associated handle.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **HAL:** status

HAL_CRYP_DeInit

Function name

HAL_StatusTypeDef HAL_CRYP_DeInit (CRYP_HandleTypeDef * hcryp)

Function description

De-Initializes the CRYP peripheral.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **HAL:** status

HAL_CRYP_MspInit

Function name

void HAL_CRYP_MspInit (CRYP_HandleTypeDef * hcryp)

Function description

Initializes the CRYP MSP.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **None:**

HAL_CRYP_MspDeInit

Function name

void HAL_CRYP_MspDeInit (CRYP_HandleTypeDef * hcryp)

Function description

DeInitializes CRYP MSP.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **None:**

HAL_CRYP_SetConfig

Function name

HAL_StatusTypeDef HAL_CRYP_SetConfig (CRYP_HandleTypeDef * hcryp, CRYP_ConfigTypeDef * pConf)

Function description

Configure the CRYP according to the specified parameters in the CRYP_ConfigTypeDef.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure
- **pConf:** pointer to a CRYP_ConfigTypeDef structure that contains the configuration information for CRYP module

Return values

- **HAL:** status

HAL_CRYP_GetConfig

Function name

HAL_StatusTypeDef HAL_CRYP_GetConfig (CRYP_HandleTypeDef * hcryp, CRYP_ConfigTypeDef * pConf)

Function description

Get CRYP Configuration parameters in associated handle.

Parameters

- **pConf:** pointer to a CRYP_ConfigTypeDef structure
- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **HAL:** status

HAL_CRYP_RegisterCallback

Function name

HAL_StatusTypeDef HAL_CRYP_RegisterCallback (CRYP_HandleTypeDef * hcryp, HAL_CRYP_CallbackIDTypeDef CallbackID, pCRYP_CallbackTypeDef pCallback)

Function description

Register a User CRYP Callback To be used instead of the weak predefined callback.

Parameters

- **hcryp:** cryp handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_CRYP_INPUT_COMPLETE_CB_ID Input FIFO transfer completed callback ID
  - HAL_CRYP_OUTPUT_COMPLETE_CB_ID Output FIFO transfer completed callback ID
  - HAL_CRYP_ERROR_CB_ID Rx Half Error callback ID
  - HAL_CRYP_MSPINIT_CB_ID MspInit callback ID
  - HAL_CRYP_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **status:**

HAL_CRYP_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_CRYP_UnRegisterCallback (CRYP_HandleTypeDef * hcryp, HAL_CRYP_CallbackIDTypeDef CallbackID)

Function description

Unregister an CRYP Callback CRYP callback is redirected to the weak predefined callback.

Parameters

- **hcryp:** cryp handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_CRYP_INPUT_COMPLETE_CB_ID Input FIFO transfer completed callback ID
  - HAL_CRYP_OUTPUT_COMPLETE_CB_ID Output FIFO transfer completed callback ID
  - HAL_CRYP_ERROR_CB_ID Rx Half Error callback ID
  - HAL_CRYP_MSPINIT_CB_ID MspInit callback ID
  - HAL_CRYP_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **status:**

HAL_CRYP_Encrypt

Function name

HAL_StatusTypeDef HAL_CRYP_Encrypt (CRYP_HandleTypeDef * hcryp, uint32_t * Input, uint16_t Size, uint32_t * Output, uint32_t Timeout)

Function description

Encryption mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **Input:** Pointer to the input buffer (plaintext)
- **Size:** Length of the plaintext buffer either in word or in byte, according to DataWidthUnit.
- **Output:** Pointer to the output buffer(ciphertext)
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_CRYP_Decrypt

Function name

HAL_StatusTypeDef HAL_CRYP_Decrypt (CRYP_HandleTypeDef * hcryp, uint32_t * Input, uint16_t Size, uint32_t * Output, uint32_t Timeout)

Function description

Decryption mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **Input:** Pointer to the input buffer (ciphertext)
- **Size:** Length of the plaintext buffer either in word or in byte, according to DataWidthUnit
- **Output:** Pointer to the output buffer(plaintext)
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_CRYP_Encrypt_IT

Function name

HAL_StatusTypeDef HAL_CRYP_Encrypt_IT (CRYP_HandleTypeDef * hcryp, uint32_t * Input, uint16_t Size, uint32_t * Output)

Function description

Encryption in interrupt mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **Input:** Pointer to the input buffer (plaintext)
- **Size:** Length of the plaintext buffer either in word or in byte, according to DataWidthUnit
- **Output:** Pointer to the output buffer(ciphertext)

Return values

- **HAL:** status

HAL_CRYP_Decrypt_IT

Function name

HAL_StatusTypeDef HAL_CRYP_Decrypt_IT (CRYP_HandleTypeDef * hcryp, uint32_t * Input, uint16_t Size, uint32_t * Output)

Function description

Decryption in itnterrupt mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **Input:** Pointer to the input buffer (ciphertext)
- **Size:** Length of the plaintext buffer either in word or in byte, according to DataWidthUnit
- **Output:** Pointer to the output buffer(plaintext)

Return values

- **HAL:** status

HAL_CRYP_Encrypt_DMA

Function name

HAL_StatusTypeDef HAL_CRYP_Encrypt_DMA (CRYP_HandleTypeDef * hcryp, uint32_t * Input, uint16_t Size, uint32_t * Output)

Function description

Encryption in DMA mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **Input:** Pointer to the input buffer (plaintext)
- **Size:** Length of the plaintext buffer either in word or in byte, according to DataWidthUnit
- **Output:** Pointer to the output buffer(ciphertext)

Return values

- **HAL:** status

HAL_CRYP_Decrypt_DMA

Function name

HAL_StatusTypeDef HAL_CRYP_Decrypt_DMA (CRYP_HandleTypeDef * hcryp, uint32_t * Input, uint16_t Size, uint32_t * Output)

Function description

Decryption in DMA mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **Input:** Pointer to the input buffer (ciphertext)
- **Size:** Length of the plaintext buffer either in word or in byte, according to DataWidthUnit
- **Output:** Pointer to the output buffer(plaintext)

Return values

- **HAL:** status

HAL_CRYP_IRQHandler

Function name

void HAL_CRYP_IRQHandler (CRYP_HandleTypeDef * hcryp)

Function description

This function handles cryptographic interrupt request.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **None:**

HAL_CRYP_GetState

Function name

HAL_CRYP_STATETypeDef HAL_CRYP_GetState (CRYP_HandleTypeDef * hcryp)

Function description

Returns the CRYP state.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module.

Return values

- **HAL:** state

HAL_CRYP_InCpltCallback

Function name

void HAL_CRYP_InCpltCallback (CRYP_HandleTypeDef * hcryp)

Function description

Input FIFO transfer completed callback.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module.

Return values

- **None:**

HAL_CRYP_OutCpltCallback

Function name

void HAL_CRYP_OutCpltCallback (CRYP_HandleTypeDef * hcryp)

Function description

Output FIFO transfer completed callback.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module.

Return values

- **None:**

HAL_CRYP_ErrorCallback

Function name

void HAL_CRYP_ErrorCallback (CRYP_HandleTypeDef * hcryp)

Function description

CRYP error callback.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module.

Return values

- **None:**

HAL_CRYP_GetError

Function name

uint32_t HAL_CRYP_GetError (CRYP_HandleTypeDef * hcryp)

Function description

Return the CRYP error code.

Parameters

- **hcryp:** : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for the CRYP IP

Return values

- **CRYP:** error code

16.3 CRYP Firmware driver defines

The following section lists the various define and macros of the module.

16.3.1 CRYP

CRYP

CRYP Algorithm Mode

CRYP_DES_ECB

CRYP_DES_CBC

CRYP_TDES_ECB

CRYP_TDES_CBC

CRYP_AES_ECB

CRYP_AES_CBC

CRYP_AES_CTR

CRYP_AES_GCM

CRYP_AES_CCM

CRYP Key and IV Configuration Skip Mode

CRYP_KEYIVCONFIG_ALWAYS

Peripheral Key and IV configuration to do systematically

CRYP_KEYIVCONFIG_ONCE

Peripheral Key and IV configuration to do only once

CRYP Data Type

CRYP_NO_SWAP

CRYP_HALFWORD_SWAP

CRYP_BYTE_SWAP

CRYP_BIT_SWAP

CRYP Data Width Unit

CRYP_DATAWIDTHUNIT_WORD

By default, size unit is word

CRYP_DATAWIDTHUNIT_BYTE

Size unit is byte, but all input will be loaded in HW CRYPT IP by block of 4 words

CRYP Error Definition

HAL_CRYP_ERROR_NONE

No error

HAL_CRYP_ERROR_WRITE

Write error

HAL_CRYP_ERROR_READ

Read error

HAL_CRYP_ERROR_DMA

DMA error

HAL_CRYP_ERROR_BUSY

Busy flag error

HAL_CRYP_ERROR_TIMEOUT

Timeout error

HAL_CRYP_ERROR_NOT_SUPPORTED

Not supported mode

HAL_CRYP_ERROR_AUTH_TAG_SEQUENCE

Sequence are not respected only for GCM or CCM

HAL_CRYP_ERROR_INVALID_CALLBACK

Invalid Callback error

CRYP Exported Macros

__HAL_CRYP_RESET_HANDLE_STATE

Description:

- Reset CRYP handle state.

Parameters:

- __HANDLE__: specifies the CRYP handle.

Return value:

- None

__HAL_CRYP_ENABLE

Description:

- Enable/Disable the CRYP peripheral.

Parameters:

- __HANDLE__: specifies the CRYP handle.

Return value:

- None

__HAL_CRYP_DISABLE

CRYP_FLAG_MASK

Description:

- Check whether the specified CRYP status flag is set or not.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values for CRYP:
  - CRYP_FLAG_BUSY: The CRYP core is currently processing a block of data or a key preparation (for AES decryption).
  - CRYP_FLAG_IFEM: Input FIFO is empty
  - CRYP_FLAG_IFNF: Input FIFO is not full
  - CRYP_FLAG_INRIS: Input FIFO service raw interrupt is pending
  - CRYP_FLAG_OFNE: Output FIFO is not empty
  - CRYP_FLAG_OFFU: Output FIFO is full
  - CRYP_FLAG_OUTRIS: Input FIFO service raw interrupt is pending

Return value:

- The: state of __FLAG__ (TRUE or FALSE).

__HAL_CRYP_GET_FLAG

__HAL_CRYP_GET_IT

Description:

- Check whether the specified CRYP interrupt is set or not.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __INTERRUPT__: specifies the interrupt to check. This parameter can be one of the following values for CRYP:
  - CRYP_IT_INI: Input FIFO service masked interrupt status
  - CRYP_IT_OUTI: Output FIFO service masked interrupt status

Return value:

- The: state of __INTERRUPT__ (TRUE or FALSE).

__HAL_CRYP_ENABLE_IT

Description:

- Enable the CRYP interrupt.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __INTERRUPT__: CRYP Interrupt. This parameter can be one of the following values for CRYP: @ CRYP_IT_INI : Input FIFO service interrupt mask. @ CRYP_IT_OUTI : Output FIFO service interrupt mask.CRYP interrupt.

Return value:

- None

__HAL_CRYP_DISABLE_IT

Description:

- Disable the CRYP interrupt.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __INTERRUPT__: CRYP Interrupt. This parameter can be one of the following values for CRYP: @ CRYP_IT_INI : Input FIFO service interrupt mask. @ CRYP_IT_OUTI : Output FIFO service interrupt mask.CRYP interrupt.

Return value:

- None

CRYP Flags

CRYP_FLAG_IFEM

Input FIFO is empty

CRYP_FLAG_IFNF

Input FIFO is not Full

CRYP_FLAG_OFNE

Output FIFO is not empty

CRYP_FLAG_OFFU

Output FIFO is Full

CRYP_FLAG_BUSY

The CRYP core is currently processing a block of data or a key preparation (for AES decryption).

CRYP_FLAG_OUTRIS

Output FIFO service raw interrupt status

CRYP_FLAG_INRIS

Input FIFO service raw interrupt status

CRYP Header Width Unit

CRYP_HEADERWIDTHUNIT_WORD

By default, header size unit is word

CRYP_HEADERWIDTHUNIT_BYTE

Size unit is byte, but all input will be loaded in HW CRYPT IP by block of 4 words

CRYP Interrupt

CRYP_IT_INI

Input FIFO Interrupt

CRYP_IT_OUTI

Output FIFO Interrupt

CRYP Private macros to check input parameters

IS_CRYP_ALGORITHM

IS_CRYP_KEYSIZE

IS_CRYP_DATATYPE

IS_CRYP_INIT

CRYP Key Size

CRYP_KEYSIZE_128B

CRYP_KEYSIZE_192B

CRYP_KEYSIZE_256B

17 HAL CRYP Extension Driver

17.1 CRYPEx Firmware driver API description

The following section lists the various functions of the CRYPEx library.

17.1.1 How to use this driver

The CRYP extension HAL driver can be used after AES-GCM or AES-CCM Encryption/Decryption to get the authentication messages.

17.1.2 Extended AES processing functions

This section provides functions allowing to generate the authentication TAG in Polling mode

- HAL_CRYPEx_AESGCM_GenerateAuthTAG
- HAL_CRYPEx_AESCCM_GenerateAuthTAG they should be used after Encrypt/Decrypt operation.

This section contains the following APIs:

- ***HAL_CRYPEx_AESGCM_GenerateAuthTAG()***
- ***HAL_CRYPEx_AESCCM_GenerateAuthTAG()***

17.1.3 Detailed description of functions

HAL_CRYPEx_AESGCM_GenerateAuthTAG

Function name

HAL_StatusTypeDef HAL_CRYPEx_AESGCM_GenerateAuthTAG (CRYP_HandleTypeDef * hcryp, uint32_t * AuthTag, uint32_t Timeout)

Function description

generate the GCM authentication TAG.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **AuthTag:** Pointer to the authentication buffer the AuthTag generated here is 128bits length, if the TAG length is less than 128bits, user should consider only the valid part of AuthTag buffer which correspond exactly to TAG length.
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_CRYPEx_AESCCM_GenerateAuthTAG

Function name

HAL_StatusTypeDef HAL_CRYPEx_AESCCM_GenerateAuthTAG (CRYP_HandleTypeDef * hcryp, uint32_t * AuthTag, uint32_t Timeout)

Function description

AES CCM Authentication TAG generation.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **AuthTag:** Pointer to the authentication buffer the AuthTag generated here is 128bits length, if the TAG length is less than 128bits, user should consider only the valid part of AuthTag buffer which correspond exactly to TAG length.
- **Timeout:** Timeout duration

Return values

- **HAL:** status

18 HAL DAC Generic Driver

18.1 DAC Firmware driver registers structures

18.1.1 __DAC_HandleTypeDef

__DAC_HandleTypeDef is defined in the stm32h7xx_hal_dac.h

Data Fields

- ***DAC_TypeDef * Instance***
- ***__IO HAL_DAC_StateTypeDef State***
- ***HAL_LockTypeDef Lock***
- ***DMA_HandleTypeDef * DMA_Handle1***
- ***DMA_HandleTypeDef * DMA_Handle2***
- ***__IO uint32_t ErrorCode***
- ***void(* ConvCpltCallbackCh1***
- ***void(* ConvHalfCpltCallbackCh1***
- ***void(* ErrorCallbackCh1***
- ***void(* DMAUnderrunCallbackCh1***
- ***void(* ConvCpltCallbackCh2***
- ***void(* ConvHalfCpltCallbackCh2***
- ***void(* ErrorCallbackCh2***
- ***void(* DMAUnderrunCallbackCh2***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***DAC_TypeDef* __DAC_HandleTypeDef::Instance***
  Register base address
- ***__IO HAL_DAC_StateTypeDef __DAC_HandleTypeDef::State***
  DAC communication state
- ***HAL_LockTypeDef __DAC_HandleTypeDef::Lock***
  DAC locking object
- ***DMA_HandleTypeDef* __DAC_HandleTypeDef::DMA_Handle1***
  Pointer DMA handler for channel 1
- ***DMA_HandleTypeDef* __DAC_HandleTypeDef::DMA_Handle2***
  Pointer DMA handler for channel 2
- ***__IO uint32_t __DAC_HandleTypeDef::ErrorCode***
  DAC Error code
- ***void(* __DAC_HandleTypeDef::ConvCpltCallbackCh1)(struct __DAC_HandleTypeDef *hdac)***
- ***void(* __DAC_HandleTypeDef::ConvHalfCpltCallbackCh1)(struct __DAC_HandleTypeDef *hdac)***
- ***void(* __DAC_HandleTypeDef::ErrorCallbackCh1)(struct __DAC_HandleTypeDef *hdac)***
- ***void(* __DAC_HandleTypeDef::DMAUnderrunCallbackCh1)(struct __DAC_HandleTypeDef *hdac)***
- ***void(* __DAC_HandleTypeDef::ConvCpltCallbackCh2)(struct __DAC_HandleTypeDef *hdac)***
- ***void(* __DAC_HandleTypeDef::ConvHalfCpltCallbackCh2)(struct __DAC_HandleTypeDef *hdac)***
- ***void(* __DAC_HandleTypeDef::ErrorCallbackCh2)(struct __DAC_HandleTypeDef *hdac)***
- ***void(* __DAC_HandleTypeDef::DMAUnderrunCallbackCh2)(struct __DAC_HandleTypeDef *hdac)***

- *void(* __DAC_HandleTypeDef::MspInitCallback)(struct __DAC_HandleTypeDef *hdac)*
- *void(* __DAC_HandleTypeDef::MspDeInitCallback)(struct __DAC_HandleTypeDef *hdac)*

18.1.2 DAC_SampleAndHoldConfTypeDef

DAC_SampleAndHoldConfTypeDef is defined in the stm32h7xx_hal_dac.h

Data Fields

- *uint32_t DAC_SampleTime*
- *uint32_t DAC_HoldTime*
- *uint32_t DAC_RefreshTime*

Field Documentation

- *uint32_t DAC_SampleAndHoldConfTypeDef::DAC_SampleTime*

  Specifies the Sample time for the selected channel. This parameter applies when DAC_SampleAndHold is DAC_SAMPLEANDHOLD_ENABLE. This parameter must be a number between Min_Data = 0 and Max_Data = 1023
- *uint32_t DAC_SampleAndHoldConfTypeDef::DAC_HoldTime*

  Specifies the hold time for the selected channel This parameter applies when DAC_SampleAndHold is DAC_SAMPLEANDHOLD_ENABLE. This parameter must be a number between Min_Data = 0 and Max_Data = 1023
- *uint32_t DAC_SampleAndHoldConfTypeDef::DAC_RefreshTime*

  Specifies the refresh time for the selected channel This parameter applies when DAC_SampleAndHold is DAC_SAMPLEANDHOLD_ENABLE. This parameter must be a number between Min_Data = 0 and Max_Data = 255

18.1.3 DAC_ChannelConfTypeDef

DAC_ChannelConfTypeDef is defined in the stm32h7xx_hal_dac.h

Data Fields

- *uint32_t DAC_SampleAndHold*
- *uint32_t DAC_Trigger*
- *uint32_t DAC_OutputBuffer*
- *uint32_t DAC_ConnectOnChipPeripheral*
- *uint32_t DAC_UserTrimming*
- *uint32_t DAC_TrimmingValue*
- *DAC_SampleAndHoldConfTypeDef DAC_SampleAndHoldConfig*

Field Documentation

- *uint32_t DAC_ChannelConfTypeDef::DAC_SampleAndHold*

  Specifies whether the DAC mode. This parameter can be a value of ***DAC_SampleAndHold***
- *uint32_t DAC_ChannelConfTypeDef::DAC_Trigger*

  Specifies the external trigger for the selected DAC channel. This parameter can be a value of ***DAC_trigger_selection***
- *uint32_t DAC_ChannelConfTypeDef::DAC_OutputBuffer*

  Specifies whether the DAC channel output buffer is enabled or disabled. This parameter can be a value of ***DAC_output_buffer***
- *uint32_t DAC_ChannelConfTypeDef::DAC_ConnectOnChipPeripheral*

  Specifies whether the DAC output is connected or not to on chip peripheral . This parameter can be a value of ***DAC_ConnectOnChipPeripheral***
- *uint32_t DAC_ChannelConfTypeDef::DAC_UserTrimming*

  Specifies the trimming mode This parameter must be a value of ***DAC_UserTrimming*** DAC_UserTrimming is either factory or user trimming
- *uint32_t DAC_ChannelConfTypeDef::DAC_TrimmingValue*

  Specifies the offset trimming value i.e. when DAC_SampleAndHold is DAC_TRIMMING_USER. This parameter must be a number between Min_Data = 1 and Max_Data = 31

- ***DAC_SampleAndHoldConfTypeDef DAC_ChannelConfTypeDef::DAC_SampleAndHoldConfig***
  Sample and Hold settings

18.2 DAC Firmware driver API description

The following section lists the various functions of the DAC library.

18.2.1 DAC Peripheral features

DAC Channels

STM32H7 devices integrate two 12-bit Digital Analog Converters The 2 converters (i.e. channel1 & channel2) can be used independently or simultaneously (dual mode):

1. DAC channel1 with DAC_OUT1 (PA4) as output or connected to on-chip peripherals (ex. OPAMPs, comparators).
2. DAC channel2 with DAC_OUT2 (PA5) as output or connected to on-chip peripherals (ex. OPAMPs, comparators).

DAC Triggers

Digital to Analog conversion can be non-triggered using DAC_TRIGGER_NONE and DAC_OUT1/DAC_OUT2 is available once writing to DHRx register.

Digital to Analog conversion can be triggered by:

1. External event: EXTI Line 9 (any GPIOx_PIN_9) using DAC_TRIGGER_EXT_IT9. The used pin (GPIOx_PIN_9) must be configured in input mode.
2. Timers TRGO: TIM1, TIM2, TIM4, TIM5, TIM6, TIM7, TIM8, TIM15, TIM23 and TIM24 (DAC_TRIGGER_T1_TRGO, DAC_TRIGGER_T2_TRGO...)
3. Low Power Timers TRGO: LPTIM1, LPTIM2 and LPTIM3 (DAC_TRIGGER_LPTIM1_OUT, DAC_TRIGGER_LPTIM2_OUT)
4. High Resolution Timer TRGO: HRTIM1 (DAC_TRIGGER_HR1_TRGO1, DAC_TRIGGER_HR1_TRGO2)
5. Software using DAC_TRIGGER_SOFTWARE

DAC Buffer mode feature

Each DAC channel integrates an output buffer that can be used to reduce the output impedance, and to drive external loads directly without having to add an external operational amplifier. To enable, the output buffer use sConfig.DAC_OutputBuffer = DAC_OUTPUTBUFFER_ENABLE;

Note: Refer to the device datasheet for more details about output impedance value with and without output buffer.

GPIO configurations guidelines

When a DAC channel is used (ex channel1 on PA4) and the other is not (ex channel2 on PA5 is configured in Analog and disabled). Channel1 may disturb channel2 as coupling effect. Note that there is no coupling on channel2 as soon as channel2 is turned on. Coupling on adjacent channel could be avoided as follows: when unused PA5 is configured as INPUT PULL-UP or DOWN. PA5 is configured in ANALOG just before it is turned on.

DAC Sample and Hold feature

For each converter, 2 modes are supported: normal mode and "sample and hold" mode (i.e. low power mode). In the sample and hold mode, the DAC core converts data, then holds the converted voltage on a capacitor. When not converting, the DAC cores and buffer are completely turned off between samples and the DAC output is tri-stated, therefore reducing the overall power consumption. A new stabilization period is needed before each new conversion. The sample and hold allow setting internal or external voltage @ low power consumption cost (output value can be at any given rate either by CPU or DMA). The Sample and hold block and registers uses either LSI & run in several power modes: run mode, sleep mode, low power run, low power sleep mode & stop1 mode. Low power stop1 mode allows only static conversion. To enable Sample and Hold mode Enable LSI using HAL_RCC_OscConfig with RCC_OSCILLATORTYPE_LSI & RCC_LSI_ON parameters. Use DAC_InitStructure.DAC_SampleAndHold = DAC_SAMPLEANDHOLD_ENABLE; & DAC_ChannelConfTypeDef.DAC_SampleAndHoldConfig.DAC_SampleTime, DAC_HoldTime & DAC_RefreshTime;

DAC calibration feature

1. The 2 converters (channel1 & channel2) provide calibration capabilities.
   - Calibration aims at correcting some offset of output buffer.
   - The DAC uses either factory calibration settings OR user defined calibration (trimming) settings (i.e. trimming mode).
   - The user defined settings can be figured out using self calibration handled by HAL_DACEx_SelfCalibrate.
   - HAL_DACEx_SelfCalibrate:
     - Runs automatically the calibration.
     - Enables the user trimming mode
     - Updates a structure with trimming values with fresh calibration results. The user may store the calibration results for larger (ex monitoring the trimming as a function of temperature for instance)

DAC wave generation feature

Both DAC channels can be used to generate

1. Noise wave
2. Triangle wave

DAC data format

The DAC data format can be:

1. 8-bit right alignment using DAC_ALIGN_8B_R
2. 12-bit left alignment using DAC_ALIGN_12B_L
3. 12-bit right alignment using DAC_ALIGN_12B_R

DAC data value to voltage correspondence

The analog output voltage on each DAC channel pin is determined by the following equation:

DAC_OUTx = VREF+ * DOR / 4095

- with DOR is the Data Output Register

VREF+ is the input voltage reference (refer to the device datasheet)

e.g. To set DAC_OUT1 to 0.7V, use

- Assuming that VREF+ = 3.3V, DAC_OUT1 = (3.3 * 868) / 4095 = 0.7V

DMA requests

A DMA request can be generated when an external trigger (but not a software trigger) occurs if DMA requests are enabled using HAL_DAC_Start_DMA(). DMA requests are mapped as following:

1. DAC channel1: mapped on DMA_REQUEST_DAC1_CH1
2. DAC channel2: mapped on DMA_REQUEST_DAC1_CH2

Note: For Dual mode and specific signal (Triangle and noise) generation please refer to Extended Features Driver description

18.2.2 How to use this driver

- DAC APB clock must be enabled to get write access to DAC registers using HAL_DAC_Init()
- Configure DAC_OUTx (DAC_OUT1: PA4, DAC_OUT2: PA5) in analog mode.
- Configure the DAC channel using HAL_DAC_ConfigChannel() function.
- Enable the DAC channel using HAL_DAC_Start() or HAL_DAC_Start_DMA() functions.

Calibration mode IO operation

- Retrieve the factory trimming (calibration settings) using HAL_DACEx_GetTrimOffset()
- Run the calibration using HAL_DACEx_SelfCalibrate()
- Update the trimming while DAC running using HAL_DACEx_SetUserTrimming()

Polling mode IO operation

- Start the DAC peripheral using HAL_DAC_Start()
- To read the DAC last data output value, use the HAL_DAC_GetValue() function.
- Stop the DAC peripheral using HAL_DAC_Stop()

DMA mode IO operation

- Start the DAC peripheral using HAL_DAC_Start_DMA(), at this stage the user specify the length of data to be transferred at each end of conversion First issued trigger will start the conversion of the value previously set by HAL_DAC_SetValue().
- At the middle of data transfer HAL_DAC_ConvHalfCpltCallbackCh1() or HAL_DACEx_ConvHalfCpltCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_ConvHalfCpltCallbackCh1() or HAL_DACEx_ConvHalfCpltCallbackCh2()
- At The end of data transfer HAL_DAC_ConvCpltCallbackCh1() or HAL_DACEx_ConvHalfCpltCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_ConvCpltCallbackCh1() or HAL_DACEx_ConvHalfCpltCallbackCh2()
- In case of transfer Error, HAL_DAC_ErrorCallbackCh1() function is executed and user can add his own code by customization of function pointer HAL_DAC_ErrorCallbackCh1
- In case of DMA underrun, DAC interruption triggers and execute internal function HAL_DAC_IRQHandler. HAL_DAC_DMAUnderrunCallbackCh1() or HAL_DACEx_DMAUnderrunCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_DMAUnderrunCallbackCh1() or HAL_DACEx_DMAUnderrunCallbackCh2() and add his own code by customization of function pointer HAL_DAC_ErrorCallbackCh1()
- Stop the DAC peripheral using HAL_DAC_Stop_DMA()

Callback registration

The compilation define USE_HAL_DAC_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_DAC_RegisterCallback() to register a user callback, it allows to register following callbacks:

- ConvCpltCallbackCh1 : callback when a half transfer is completed on Ch1.
- ConvHalfCpltCallbackCh1 : callback when a transfer is completed on Ch1.
- ErrorCallbackCh1 : callback when an error occurs on Ch1.
- DMAUnderrunCallbackCh1 : callback when an underrun error occurs on Ch1.
- ConvCpltCallbackCh2 : callback when a half transfer is completed on Ch2.
- ConvHalfCpltCallbackCh2 : callback when a transfer is completed on Ch2.
- ErrorCallbackCh2 : callback when an error occurs on Ch2.
- DMAUnderrunCallbackCh2 : callback when an underrun error occurs on Ch2.
- MspInitCallback : DAC MspInit.

- MspDeInitCallback : DAC MspdeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. Use function HAL_DAC_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. It allows to reset following callbacks:
- ConvCpltCallbackCh1 : callback when a half transfer is completed on Ch1.
- ConvHalfCpltCallbackCh1 : callback when a transfer is completed on Ch1.
- ErrorCallbackCh1 : callback when an error occurs on Ch1.
- DMAUnderrunCallbackCh1 : callback when an underrun error occurs on Ch1.
- ConvCpltCallbackCh2 : callback when a half transfer is completed on Ch2.
- ConvHalfCpltCallbackCh2 : callback when a transfer is completed on Ch2.
- ErrorCallbackCh2 : callback when an error occurs on Ch2.
- DMAUnderrunCallbackCh2 : callback when an underrun error occurs on Ch2.
- MspInitCallback : DAC MspInit.
- MspDeInitCallback : DAC MspdeInit.
- All Callbacks This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the HAL_DAC_Init and if the state is HAL_DAC_STATE_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the HAL_DAC_Init and HAL_DAC_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_DAC_Init and HAL_DAC_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_DAC_RegisterCallback before calling HAL_DAC_DeInit or HAL_DAC_Init function. When The compilation define USE_HAL_DAC_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

DAC HAL driver macros list

Below the list of most used macros in DAC HAL driver.

- __HAL_DAC_ENABLE : Enable the DAC peripheral
- __HAL_DAC_DISABLE : Disable the DAC peripheral
- __HAL_DAC_CLEAR_FLAG: Clear the DAC's pending flags
- __HAL_DAC_GET_FLAG: Get the selected DAC's flag status

Note: You can refer to the DAC HAL driver header file for more useful macros

18.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the DAC.
- De-initialize the DAC.

This section contains the following APIs:

- ***HAL_DAC_Init()***
- ***HAL_DAC_DeInit()***
- ***HAL_DAC_MspInit()***
- ***HAL_DAC_MspDeInit()***
- ***HAL_DAC_RegisterCallback()***
- ***HAL_DAC_UnRegisterCallback()***

18.2.4 IO operation functions

This section provides functions allowing to:

- Start conversion.
- Stop conversion.
- Start conversion and enable DMA transfer.

- Stop conversion and disable DMA transfer.
- Get result of conversion.

This section contains the following APIs:

- *HAL_DAC_Start()*
- *HAL_DAC_Stop()*
- *HAL_DAC_Start_DMA()*
- *HAL_DAC_Stop_DMA()*
- *HAL_DAC_IRQHandler()*
- *HAL_DAC_SetValue()*
- *HAL_DAC_ConvCpltCallbackCh1()*
- *HAL_DAC_ConvHalfCpltCallbackCh1()*
- *HAL_DAC_ErrorCallbackCh1()*
- *HAL_DAC_DMAUnderrunCallbackCh1()*
- *HAL_DAC_RegisterCallback()*
- *HAL_DAC_UnRegisterCallback()*

18.2.5 Peripheral Control functions

This section provides functions allowing to:

- Configure channels.
- Set the specified data holding register value for DAC channel.

This section contains the following APIs:

- *HAL_DAC_GetValue()*
- *HAL_DAC_ConfigChannel()*

18.2.6 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the DAC state.
- Check the DAC Errors.

This section contains the following APIs:

- *HAL_DAC_GetState()*
- *HAL_DAC_GetError()*

18.2.7 Detailed description of functions

HAL_DAC_Init

Function name

HAL_StatusTypeDef HAL_DAC_Init (DAC_HandleTypeDef * hdac)

Function description

Initialize the DAC peripheral according to the specified parameters in the DAC_InitStruct and initialize the associated handle.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **HAL:** status

HAL_DAC_DeInit

Function name

HAL_StatusTypeDef HAL_DAC_DeInit (DAC_HandleTypeDef * hdac)

Function description

Deinitialize the DAC peripheral registers to their default reset values.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **HAL:** status

HAL_DAC_MspInit

Function name

void HAL_DAC_MspInit (DAC_HandleTypeDef * hdac)

Function description

Initialize the DAC MSP.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_MspDeInit

Function name

void HAL_DAC_MspDeInit (DAC_HandleTypeDef * hdac)

Function description

DeInitialize the DAC MSP.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_Start

Function name

HAL_StatusTypeDef HAL_DAC_Start (DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description

Enables DAC and starts conversion of channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **HAL:** status

HAL_DAC_Stop

Function name

HAL_StatusTypeDef HAL_DAC_Stop (DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description

Disables DAC and stop conversion of channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **HAL:** status

HAL_DAC_Start_DMA

Function name

HAL_StatusTypeDef HAL_DAC_Start_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t * pData, uint32_t Length, uint32_t Alignment)

Function description

Enables DAC and starts conversion of channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to DAC peripheral
- **Alignment:** Specifies the data alignment for DAC channel. This parameter can be one of the following values:
  - DAC_ALIGN_8B_R: 8bit right data alignment selected
  - DAC_ALIGN_12B_L: 12bit left data alignment selected
  - DAC_ALIGN_12B_R: 12bit right data alignment selected

Return values

- **HAL:** status

HAL_DAC_Stop_DMA

Function name

HAL_StatusTypeDef HAL_DAC_Stop_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description

Disables DAC and stop conversion of channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **HAL:** status

HAL_DAC_IRQHandler

Function name

void HAL_DAC_IRQHandler (DAC_HandleTypeDef * hdac)

Function description

Handles DAC interrupt request This function uses the interruption of DMA underrun.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_SetValue

Function name

HAL_StatusTypeDef HAL_DAC_SetValue (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Alignment, uint32_t Data)

Function description

Set the specified data holding register value for DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected
- **Alignment:** Specifies the data alignment. This parameter can be one of the following values:
  - DAC_ALIGN_8B_R: 8bit right data alignment selected
  - DAC_ALIGN_12B_L: 12bit left data alignment selected
  - DAC_ALIGN_12B_R: 12bit right data alignment selected
- **Data:** Data to be loaded in the selected data holding register.

Return values

- **HAL:** status

HAL_DAC_ConvCpltCallbackCh1

Function name

void HAL_DAC_ConvCpltCallbackCh1 (DAC_HandleTypeDef * hdac)

Function description

Conversion complete callback in non-blocking mode for Channel1.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_ConvHalfCpltCallbackCh1

Function name

void HAL_DAC_ConvHalfCpltCallbackCh1 (DAC_HandleTypeDef * hdac)

Function description

Conversion half DMA transfer callback in non-blocking mode for Channel1.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_ErrorCallbackCh1

Function name

void HAL_DAC_ErrorCallbackCh1 (DAC_HandleTypeDef * hdac)

Function description

Error DAC callback for Channel1.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_DMAUnderrunCallbackCh1

Function name

void HAL_DAC_DMAUnderrunCallbackCh1 (DAC_HandleTypeDef * hdac)

Function description

DMA underrun DAC callback for channel1.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_DAC_RegisterCallback (DAC_HandleTypeDef * hdac, HAL_DAC_CallbackIDTypeDef CallbackID, pDAC_CallbackTypeDef pCallback)

Function description

Register a User DAC Callback To be used instead of the weak (surcharged) predefined callback.

Parameters

- **hdac:** DAC handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_DAC_ERROR_INVALID_CALLBACK DAC Error Callback ID
  - HAL_DAC_CH1_COMPLETE_CB_ID DAC CH1 Complete Callback ID
  - HAL_DAC_CH1_HALF_COMPLETE_CB_ID DAC CH1 Half Complete Callback ID
  - HAL_DAC_CH1_ERROR_ID DAC CH1 Error Callback ID
  - HAL_DAC_CH1_UNDERRUN_CB_ID DAC CH1 UnderRun Callback ID
  - HAL_DAC_CH2_COMPLETE_CB_ID DAC CH2 Complete Callback ID
  - HAL_DAC_CH2_HALF_COMPLETE_CB_ID DAC CH2 Half Complete Callback ID
  - HAL_DAC_CH2_ERROR_ID DAC CH2 Error Callback ID
  - HAL_DAC_CH2_UNDERRUN_CB_ID DAC CH2 UnderRun Callback ID
  - HAL_DAC_MSPINIT_CB_ID DAC MSP Init Callback ID
  - HAL_DAC_MSPDEINIT_CB_ID DAC MSP DeInit Callback ID
- **pCallback:** pointer to the Callback function

Return values

- **status:**

HAL_DAC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_DAC_UnRegisterCallback (DAC_HandleTypeDef * hdac, HAL_DAC_CallbackIDTypeDef CallbackID)

Function description

Unregister a User DAC Callback DAC Callback is redirected to the weak (surcharged) predefined callback.

Parameters

- **hdac:** DAC handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_DAC_CH1_COMPLETE_CB_ID DAC CH1 transfer Complete Callback ID
  - HAL_DAC_CH1_HALF_COMPLETE_CB_ID DAC CH1 Half Complete Callback ID
  - HAL_DAC_CH1_ERROR_ID DAC CH1 Error Callback ID
  - HAL_DAC_CH1_UNDERRUN_CB_ID DAC CH1 UnderRun Callback ID
  - HAL_DAC_CH2_COMPLETE_CB_ID DAC CH2 Complete Callback ID
  - HAL_DAC_CH2_HALF_COMPLETE_CB_ID DAC CH2 Half Complete Callback ID
  - HAL_DAC_CH2_ERROR_ID DAC CH2 Error Callback ID
  - HAL_DAC_CH2_UNDERRUN_CB_ID DAC CH2 UnderRun Callback ID
  - HAL_DAC_MSPINIT_CB_ID DAC MSP Init Callback ID
  - HAL_DAC_MSPDEINIT_CB_ID DAC MSP DeInit Callback ID
  - HAL_DAC_ALL_CB_ID DAC All callbacks

Return values

- **status:**

HAL_DAC_GetValue

Function name

uint32_t HAL_DAC_GetValue (DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description

Returns the last data output value of the selected DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **The:** selected DAC channel data output value.

HAL_DAC_ConfigChannel

Function name

HAL_StatusTypeDef HAL_DAC_ConfigChannel (DAC_HandleTypeDef * hdac, DAC_ChannelConfTypeDef * sConfig, uint32_t Channel)

Function description

Configures the selected DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **sConfig:** DAC configuration structure.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **HAL:** status

HAL_DAC_GetState

Function name

HAL_DAC_StateTypeDef HAL_DAC_GetState (DAC_HandleTypeDef * hdac)

Function description

return the DAC handle state

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **HAL:** state

HAL_DAC_GetError

Function name

uint32_t HAL_DAC_GetError (DAC_HandleTypeDef * hdac)

Function description

Return the DAC error code.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **DAC:** Error Code

DAC_DMAConvCpltCh1

Function name

void DAC_DMAConvCpltCh1 (DMA_HandleTypeDef * hdma)

Function description

DMA conversion complete callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

DAC_DMAErrorCh1

Function name

void DAC_DMAErrorCh1 (DMA_HandleTypeDef * hdma)

Function description

DMA error callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

DAC_DMAHalfConvCpltCh1

Function name

void DAC_DMAHalfConvCpltCh1 (DMA_HandleTypeDef * hdma)

Function description

DMA half transfer complete callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

18.3 DAC Firmware driver defines

The following section lists the various define and macros of the module.

18.3.1 DAC

DAC

DAC Channel selection

DAC_CHANNEL_1

DAC_CHANNEL_2

DAC ConnectOnChipPeripheral

DAC_CHIPCONNECT_EXTERNAL

DAC_CHIPCONNECT_INTERNAL

DAC_CHIPCONNECT_BOTH

DAC data alignment

DAC_ALIGN_12B_R

DAC_ALIGN_12B_L

DAC_ALIGN_8B_R

DAC Error Code

HAL_DAC_ERROR_NONE

No error

HAL_DAC_ERROR_DMAUNDERRUNCH1

DAC channel1 DMA underrun error

HAL_DAC_ERROR_DMAUNDERRUNCH2

DAC channel2 DMA underrun error

HAL_DAC_ERROR_DMA

DMA error

HAL_DAC_ERROR_TIMEOUT

Timeout error

HAL_DAC_ERROR_INVALID_CALLBACK

Invalid callback error

DAC Exported Macros

__HAL_DAC_RESET_HANDLE_STATE

Description:

- Reset DAC handle state.

Parameters:

- __HANDLE__: specifies the DAC handle.

Return value:

- None

__HAL_DAC_ENABLE

Description:

- Enable the DAC channel.

Parameters:

- __HANDLE__: specifies the DAC handle.
- __DAC_Channel__: specifies the DAC channel

Return value:

- None

__HAL_DAC_DISABLE

Description:

- Disable the DAC channel.

Parameters:

- __HANDLE__: specifies the DAC handle
- __DAC_Channel__: specifies the DAC channel.

Return value:

- None

DAC_DHR12R1_ALIGNMENT

Description:

- Set DHR12R1 alignment.

Parameters:

- __ALIGNMENT__: specifies the DAC alignment

Return value:

- None

DAC_DHR12R2_ALIGNMENT

Description:

- Set DHR12R2 alignment.

Parameters:

- __ALIGNMENT__: specifies the DAC alignment

Return value:

- None

DAC_DHR12RD_ALIGNMENT

Description:

- Set DHR12RD alignment.

Parameters:

- __ALIGNMENT__: specifies the DAC alignment

Return value:

- None

__HAL_DAC_ENABLE_IT

Description:

- Enable the DAC interrupt.

Parameters:

- __HANDLE__: specifies the DAC handle
- __INTERRUPT__: specifies the DAC interrupt. This parameter can be any combination of the following values:
  - DAC_IT_DMAUDR1 DAC channel 1 DMA underrun interrupt
  - DAC_IT_DMAUDR2 DAC channel 2 DMA underrun interrupt

Return value:

- None

__HAL_DAC_DISABLE_IT

Description:

- Disable the DAC interrupt.

Parameters:

- __HANDLE__: specifies the DAC handle
- __INTERRUPT__: specifies the DAC interrupt. This parameter can be any combination of the following values:
  - DAC_IT_DMAUDR1 DAC channel 1 DMA underrun interrupt
  - DAC_IT_DMAUDR2 DAC channel 2 DMA underrun interrupt

Return value:

- None

__HAL_DAC_GET_IT_SOURCE

Description:

- Check whether the specified DAC interrupt source is enabled or not.

Parameters:

- __HANDLE__: DAC handle
- __INTERRUPT__: DAC interrupt source to check This parameter can be any combination of the following values:
  - DAC_IT_DMAUDR1 DAC channel 1 DMA underrun interrupt
  - DAC_IT_DMAUDR2 DAC channel 2 DMA underrun interrupt

Return value:

- State: of interruption (SET or RESET)

__HAL_DAC_GET_FLAG

Description:

- Get the selected DAC's flag status.

Parameters:

- __HANDLE__: specifies the DAC handle.
- __FLAG__: specifies the DAC flag to get. This parameter can be any combination of the following values:
  - DAC_FLAG_DMAUDR1 DAC channel 1 DMA underrun flag
  - DAC_FLAG_DMAUDR2 DAC channel 2 DMA underrun flag

Return value:

- None

__HAL_DAC_CLEAR_FLAG

Description:

- Clear the DAC's flag.

Parameters:

- __HANDLE__: specifies the DAC handle.
- __FLAG__: specifies the DAC flag to clear. This parameter can be any combination of the following values:
  - DAC_FLAG_DMAUDR1 DAC channel 1 DMA underrun flag
  - DAC_FLAG_DMAUDR2 DAC channel 2 DMA underrun flag

Return value:

- None

DAC flags definition

DAC_FLAG_DMAUDR1

DAC_FLAG_DMAUDR2

DAC IT definition

DAC_IT_DMAUDR1

DAC_IT_DMAUDR2

DAC output buffer

DAC_OUTPUTBUFFER_ENABLE

DAC_OUTPUTBUFFER_DISABLE

DAC power mode

DAC_SAMPLEANDHOLD_DISABLE

DAC_SAMPLEANDHOLD_ENABLE

DAC trigger selection

DAC_TRIGGER_NONE

Conversion is automatic once the DAC_DHRxxxx register has been loaded, and not by external trigger

DAC_TRIGGER_SOFTWARE

Conversion started by software trigger for DAC channel

DAC_TRIGGER_T1_TRGO

TIM1 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T2_TRGO

TIM2 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T4_TRGO

TIM4 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T5_TRGO

TIM5 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T6_TRGO

TIM6 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T7_TRGO

TIM7 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T8_TRGO

TIM8 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T15_TRGO

TIM15 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_HR1_TRGO1

HR1 TRGO1 selected as external conversion trigger for DAC channel

DAC_TRIGGER_HR1_TRGO2

HR1 TRGO2 selected as external conversion trigger for DAC channel

DAC_TRIGGER_LPTIM1_OUT

LPTIM1 OUT TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_LPTIM2_OUT

LPTIM2 OUT TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_EXT_IT9

EXTI Line9 event selected as external conversion trigger for DAC channel

DAC User Trimming

DAC_TRIMMING_FACTORY

Factory trimming

DAC_TRIMMING_USER

User trimming

19 HAL DAC Extension Driver

19.1 DACEx Firmware driver API description

The following section lists the various functions of the DACEx library.

19.1.1 How to use this driver

Dual mode IO operation

- Use HAL_DACEx_DualStart() to enable both channel and start conversion for dual mode operation. If software trigger is selected, using HAL_DACEx_DualStart() will start the conversion of the value previously set by HAL_DACEx_DualSetValue().
- Use HAL_DACEx_DualStop() to disable both channel and stop conversion for dual mode operation.
- Use HAL_DACEx_DualStart_DMA() to enable both channel and start conversion for dual mode operation using DMA to feed DAC converters. First issued trigger will start the conversion of the value previously set by HAL_DACEx_DualSetValue(). The same callbacks that are used in single mode are called in dual mode to notify transfer completion (half complete or complete), errors or underrun.
- Use HAL_DACEx_DualStop_DMA() to disable both channel and stop conversion for dual mode operation using DMA to feed DAC converters.
- When Dual mode is enabled (i.e. DAC Channel1 and Channel2 are used simultaneously) : Use HAL_DACEx_DualGetValue() to get digital data to be converted and use HAL_DACEx_DualSetValue() to set digital value to converted simultaneously in Channel 1 and Channel 2.

Signal generation operation

- Use HAL_DACEx_TriangleWaveGenerate() to generate Triangle signal.
- Use HAL_DACEx_NoiseWaveGenerate() to generate Noise signal.
- HAL_DACEx_SelfCalibrate to calibrate one DAC channel.
- HAL_DACEx_SetUserTrimming to set user trimming value.
- HAL_DACEx_GetTrimOffset to retrieve trimming value (factory setting after reset, user setting if HAL_DACEx_SetUserTrimming have been used at least one time after reset).

19.1.2 Extended features functions

This section provides functions allowing to:

- Start conversion.
- Stop conversion.
- Start conversion and enable DMA transfer.
- Stop conversion and disable DMA transfer.
- Get result of conversion.
- Get result of dual mode conversion.

This section contains the following APIs:

- ***HAL_DACEx_DualStart()***
- ***HAL_DACEx_DualStop()***
- ***HAL_DACEx_DualStart_DMA()***
- ***HAL_DACEx_DualStop_DMA()***
- ***HAL_DACEx_TriangleWaveGenerate()***
- ***HAL_DACEx_NoiseWaveGenerate()***
- ***HAL_DACEx_DualSetValue()***
- ***HAL_DACEx_ConvCpltCallbackCh2()***

- *HAL_DACEx_ConvHalfCpltCallbackCh2()*
- *HAL_DACEx_ErrorCallbackCh2()*
- *HAL_DACEx_DMAUnderrunCallbackCh2()*
- *HAL_DACEx_SelfCalibrate()*
- *HAL_DACEx_SetUserTrimming()*
- *HAL_DACEx_GetTrimOffset()*
- *HAL_DACEx_DualGetValue()*

19.1.3 Peripheral Control functions

This section provides functions allowing to:

- Set the specified data holding register value for DAC channel.

This section contains the following APIs:

- *HAL_DACEx_DualGetValue()*
- *HAL_DACEx_SelfCalibrate()*
- *HAL_DACEx_SetUserTrimming()*
- *HAL_DACEx_GetTrimOffset()*

19.1.4 Detailed description of functions

HAL_DACEx_TriangleWaveGenerate

Function name

HAL_StatusTypeDef HAL_DACEx_TriangleWaveGenerate (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Amplitude)

Function description

Enable or disable the selected DAC channel wave generation.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected
- **Amplitude:** Select max triangle amplitude. This parameter can be one of the following values:
  - DAC_TRIANGLEAMPLITUDE_1: Select max triangle amplitude of 1
  - DAC_TRIANGLEAMPLITUDE_3: Select max triangle amplitude of 3
  - DAC_TRIANGLEAMPLITUDE_7: Select max triangle amplitude of 7
  - DAC_TRIANGLEAMPLITUDE_15: Select max triangle amplitude of 15
  - DAC_TRIANGLEAMPLITUDE_31: Select max triangle amplitude of 31
  - DAC_TRIANGLEAMPLITUDE_63: Select max triangle amplitude of 63
  - DAC_TRIANGLEAMPLITUDE_127: Select max triangle amplitude of 127
  - DAC_TRIANGLEAMPLITUDE_255: Select max triangle amplitude of 255
  - DAC_TRIANGLEAMPLITUDE_511: Select max triangle amplitude of 511
  - DAC_TRIANGLEAMPLITUDE_1023: Select max triangle amplitude of 1023
  - DAC_TRIANGLEAMPLITUDE_2047: Select max triangle amplitude of 2047
  - DAC_TRIANGLEAMPLITUDE_4095: Select max triangle amplitude of 4095

Return values

- **HAL:** status

HAL_DACEx_NoiseWaveGenerate

Function name

HAL_StatusTypeDef HAL_DACEx_NoiseWaveGenerate (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Amplitude)

Function description

Enable or disable the selected DAC channel wave generation.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected
- **Amplitude:** Unmask DAC channel LFSR for noise wave generation. This parameter can be one of the following values:
  - DAC_LFSRUNMASK_BIT0: Unmask DAC channel LFSR bit0 for noise wave generation
  - DAC_LFSRUNMASK_BITS1_0: Unmask DAC channel LFSR bit[1:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS2_0: Unmask DAC channel LFSR bit[2:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS3_0: Unmask DAC channel LFSR bit[3:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS4_0: Unmask DAC channel LFSR bit[4:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS5_0: Unmask DAC channel LFSR bit[5:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS6_0: Unmask DAC channel LFSR bit[6:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS7_0: Unmask DAC channel LFSR bit[7:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS8_0: Unmask DAC channel LFSR bit[8:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS9_0: Unmask DAC channel LFSR bit[9:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS10_0: Unmask DAC channel LFSR bit[10:0] for noise wave generation
  - DAC_LFSRUNMASK_BITS11_0: Unmask DAC channel LFSR bit[11:0] for noise wave generation

Return values

- **HAL:** status

HAL_DACEx_DualStart

Function name

HAL_StatusTypeDef HAL_DACEx_DualStart (DAC_HandleTypeDef * hdac)

Function description

Enables DAC and starts conversion of both channels.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **HAL:** status

HAL_DACEx_DualStop

Function name

HAL_StatusTypeDef HAL_DACEx_DualStop (DAC_HandleTypeDef * hdac)

Function description

Disables DAC and stop conversion of both channels.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **HAL:** status

HAL_DACEx_DualStart_DMA

Function name

HAL_StatusTypeDef HAL_DACEx_DualStart_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t * pData, uint32_t Length, uint32_t Alignment)

Function description

Enables DAC and starts conversion of both channel 1 and 2 of the same DAC.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The DAC channel that will request data from DMA. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected
- **pData:** The destination peripheral Buffer address.
- **Length:** The length of data to be transferred from memory to DAC peripheral
- **Alignment:** Specifies the data alignment for DAC channel. This parameter can be one of the following values:
  - DAC_ALIGN_8B_R: 8bit right data alignment selected
  - DAC_ALIGN_12B_L: 12bit left data alignment selected
  - DAC_ALIGN_12B_R: 12bit right data alignment selected

Return values

- **HAL:** status

HAL_DACEx_DualStop_DMA

Function name

HAL_StatusTypeDef HAL_DACEx_DualStop_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description

Disables DAC and stop conversion both channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The DAC channel that requests data from DMA. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **HAL:** status

HAL_DACEx_DualSetValue

Function name

HAL_StatusTypeDef HAL_DACEx_DualSetValue (DAC_HandleTypeDef * hdac, uint32_t Alignment, uint32_t Data1, uint32_t Data2)

Function description

Set the specified data holding register value for dual DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Alignment:** Specifies the data alignment for dual channel DAC. This parameter can be one of the following values: DAC_ALIGN_8B_R: 8bit right data alignment selected DAC_ALIGN_12B_L: 12bit left data alignment selected DAC_ALIGN_12B_R: 12bit right data alignment selected
- **Data1:** Data for DAC Channel1 to be loaded in the selected data holding register.
- **Data2:** Data for DAC Channel2 to be loaded in the selected data holding register.

Return values

- **HAL:** status

Notes

- In dual mode, a unique register access is required to write in both DAC channels at the same time.

HAL_DACEx_DualGetValue

Function name

uint32_t HAL_DACEx_DualGetValue (DAC_HandleTypeDef * hdac)

Function description

Return the last data output value of the selected DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **The:** selected DAC channel data output value.

HAL_DACEx_ConvCpltCallbackCh2

Function name

void HAL_DACEx_ConvCpltCallbackCh2 (DAC_HandleTypeDef * hdac)

Function description

Conversion complete callback in non-blocking mode for Channel2.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DACEx_ConvHalfCpltCallbackCh2

Function name

void HAL_DACEx_ConvHalfCpltCallbackCh2 (DAC_HandleTypeDef * hdac)

Function description

Conversion half DMA transfer callback in non-blocking mode for Channel2.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DACEx_ErrorCallbackCh2

Function name

void HAL_DACEx_ErrorCallbackCh2 (DAC_HandleTypeDef * hdac)

Function description

Error DAC callback for Channel2.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DACEx_DMAUnderrunCallbackCh2

Function name

void HAL_DACEx_DMAUnderrunCallbackCh2 (DAC_HandleTypeDef * hdac)

Function description

DMA underrun DAC callback for Channel2.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DACEx_SelfCalibrate

Function name

HAL_StatusTypeDef HAL_DACEx_SelfCalibrate (DAC_HandleTypeDef * hdac, DAC_ChannelConfTypeDef * sConfig, uint32_t Channel)

Function description

Run the self calibration of one DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **sConfig:** DAC channel configuration structure.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **Updates:** DAC_TrimmingValue. , DAC_UserTrimming set to DAC_UserTrimming
- **HAL:** status

Notes

- Calibration runs about 7 ms.

HAL_DACEx_SetUserTrimming

Function name

HAL_StatusTypeDef HAL_DACEx_SetUserTrimming (DAC_HandleTypeDef * hdac, DAC_ChannelConfTypeDef * sConfig, uint32_t Channel, uint32_t NewTrimmingValue)

Function description

Set the trimming mode and trimming value (user trimming mode applied).

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **sConfig:** DAC configuration structure updated with new DAC trimming value.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected
- **NewTrimmingValue:** DAC new trimming value

Return values

- **HAL:** status

HAL_DACEx_GetTrimOffset

Function name

uint32_t HAL_DACEx_GetTrimOffset (DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description

Return the DAC trimming value.

Parameters

- **hdac:** DAC handle
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **Trimming:** value : range: 0->31

DAC_DMAConvCpltCh2

Function name

void DAC_DMAConvCpltCh2 (DMA_HandleTypeDef * hdma)

Function description

DMA conversion complete callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

DAC_DMAErrorCh2

Function name

void DAC_DMAErrorCh2 (DMA_HandleTypeDef * hdma)

Function description

DMA error callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

DAC_DMAHalfConvCpltCh2

Function name

void DAC_DMAHalfConvCpltCh2 (DMA_HandleTypeDef * hdma)

Function description

DMA half transfer complete callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

19.2 DACEx Firmware driver defines

The following section lists the various define and macros of the module.

19.2.1 DACEx

DACEx

DACEx lfsrunmask triangle amplitude

DAC_LFSRUNMASK_BIT0

Unmask DAC channel LFSR bit0 for noise wave generation

DAC_LFSRUNMASK_BITS1_0

Unmask DAC channel LFSR bit[1:0] for noise wave generation

DAC_LFSRUNMASK_BITS2_0

Unmask DAC channel LFSR bit[2:0] for noise wave generation

DAC_LFSRUNMASK_BITS3_0

Unmask DAC channel LFSR bit[3:0] for noise wave generation

DAC_LFSRUNMASK_BITS4_0

Unmask DAC channel LFSR bit[4:0] for noise wave generation

DAC_LFSRUNMASK_BITS5_0

Unmask DAC channel LFSR bit[5:0] for noise wave generation

DAC_LFSRUNMASK_BITS6_0

Unmask DAC channel LFSR bit[6:0] for noise wave generation

DAC_LFSRUNMASK_BITS7_0

Unmask DAC channel LFSR bit[7:0] for noise wave generation

DAC_LFSRUNMASK_BITS8_0

Unmask DAC channel LFSR bit[8:0] for noise wave generation

DAC_LFSRUNMASK_BITS9_0

Unmask DAC channel LFSR bit[9:0] for noise wave generation

DAC_LFSRUNMASK_BITS10_0

Unmask DAC channel LFSR bit[10:0] for noise wave generation

DAC_LFSRUNMASK_BITS11_0

Unmask DAC channel LFSR bit[11:0] for noise wave generation

DAC_TRIANGLEAMPLITUDE_1

Select max triangle amplitude of 1

DAC_TRIANGLEAMPLITUDE_3

Select max triangle amplitude of 3

DAC_TRIANGLEAMPLITUDE_7

Select max triangle amplitude of 7

DAC_TRIANGLEAMPLITUDE_15

Select max triangle amplitude of 15

DAC_TRIANGLEAMPLITUDE_31

Select max triangle amplitude of 31

DAC_TRIANGLEAMPLITUDE_63

Select max triangle amplitude of 63

DAC_TRIANGLEAMPLITUDE_127

Select max triangle amplitude of 127

DAC_TRIANGLEAMPLITUDE_255

Select max triangle amplitude of 255

DAC_TRIANGLEAMPLITUDE_511

Select max triangle amplitude of 511

DAC_TRIANGLEAMPLITUDE_1023

Select max triangle amplitude of 1023

DAC_TRIANGLEAMPLITUDE_2047

Select max triangle amplitude of 2047

DAC_TRIANGLEAMPLITUDE_4095

Select max triangle amplitude of 4095

20 HAL DCMI Generic Driver

20.1 DCMI Firmware driver registers structures

20.1.1 DCMI_CodesInitTypeDef

DCMI_CodesInitTypeDef is defined in the stm32h7xx_hal_dcmi.h

Data Fields

- ***uint8_t FrameStartCode***
- ***uint8_t LineStartCode***
- ***uint8_t LineEndCode***
- ***uint8_t FrameEndCode***

Field Documentation

- ***uint8_t DCMI_CodesInitTypeDef::FrameStartCode***
  Specifies the code of the frame start delimiter.
- ***uint8_t DCMI_CodesInitTypeDef::LineStartCode***
  Specifies the code of the line start delimiter.
- ***uint8_t DCMI_CodesInitTypeDef::LineEndCode***
  Specifies the code of the line end delimiter.
- ***uint8_t DCMI_CodesInitTypeDef::FrameEndCode***
  Specifies the code of the frame end delimiter.

20.1.2 DCMI_SyncUnmaskTypeDef

DCMI_SyncUnmaskTypeDef is defined in the stm32h7xx_hal_dcmi.h

Data Fields

- ***uint8_t FrameStartUnmask***
- ***uint8_t LineStartUnmask***
- ***uint8_t LineEndUnmask***
- ***uint8_t FrameEndUnmask***

Field Documentation

- ***uint8_t DCMI_SyncUnmaskTypeDef::FrameStartUnmask***
  Specifies the frame start delimiter unmask.
- ***uint8_t DCMI_SyncUnmaskTypeDef::LineStartUnmask***
  Specifies the line start delimiter unmask.
- ***uint8_t DCMI_SyncUnmaskTypeDef::LineEndUnmask***
  Specifies the line end delimiter unmask.
- ***uint8_t DCMI_SyncUnmaskTypeDef::FrameEndUnmask***
  Specifies the frame end delimiter unmask.

20.1.3 DCMI_InitTypeDef

DCMI_InitTypeDef is defined in the stm32h7xx_hal_dcmi.h

Data Fields

- ***uint32_t SynchroMode***
- ***uint32_t PCKPolarity***
- ***uint32_t VSPolarity***
- ***uint32_t HSPolarity***
- ***uint32_t CaptureRate***

- ***uint32_t ExtendedDataMode***
- ***DCMI_CodesInitTypeDef SyncroCode***
- ***uint32_t JPEGMode***
- ***uint32_t ByteSelectMode***
- ***uint32_t ByteSelectStart***
- ***uint32_t LineSelectMode***
- ***uint32_t LineSelectStart***

Field Documentation

- ***uint32_t DCMI_InitTypeDef::SynchroMode***

  Specifies the Synchronization Mode: Hardware or Embedded. This parameter can be a value of ***DCMI_Synchronization_Mode***
- ***uint32_t DCMI_InitTypeDef::PCKPolarity***

  Specifies the Pixel clock polarity: Falling or Rising. This parameter can be a value of ***DCMI_PIXCK_Polarity***
- ***uint32_t DCMI_InitTypeDef::VSPolarity***

  Specifies the Vertical synchronization polarity: High or Low. This parameter can be a value of ***DCMI_VSYNC_Polarity***
- ***uint32_t DCMI_InitTypeDef::HSPolarity***

  Specifies the Horizontal synchronization polarity: High or Low. This parameter can be a value of ***DCMI_HSYNC_Polarity***
- ***uint32_t DCMI_InitTypeDef::CaptureRate***

  Specifies the frequency of frame capture: All, 1/2 or 1/4. This parameter can be a value of ***DCMI_Capture_Rate***
- ***uint32_t DCMI_InitTypeDef::ExtendedDataMode***

  Specifies the data width: 8-bit, 10-bit, 12-bit or 14-bit. This parameter can be a value of ***DCMI_Extended_Data_Mode***
- ***DCMI_CodesInitTypeDef DCMI_InitTypeDef::SyncroCode***

  Specifies the code of the line/frame start delimiter and the line/frame end delimiter
- ***uint32_t DCMI_InitTypeDef::JPEGMode***

  Enable or Disable the JPEG mode. This parameter can be a value of ***DCMI_MODE_JPEG***
- ***uint32_t DCMI_InitTypeDef::ByteSelectMode***

  Specifies the data to be captured by the interface This parameter can be a value of ***DCMI_Byte_Select_Mode***
- ***uint32_t DCMI_InitTypeDef::ByteSelectStart***

  Specifies if the data to be captured by the interface is even or odd This parameter can be a value of ***DCMI_Byte_Select_Start***
- ***uint32_t DCMI_InitTypeDef::LineSelectMode***

  Specifies the line of data to be captured by the interface This parameter can be a value of ***DCMI_Line_Select_Mode***
- ***uint32_t DCMI_InitTypeDef::LineSelectStart***

  Specifies if the line of data to be captured by the interface is even or odd This parameter can be a value of ***DCMI_Line_Select_Start***

20.1.4 __DCMI_HandleTypeDef

__DCMI_HandleTypeDef is defined in the stm32h7xx_hal_dcmi.h

Data Fields

- ***DCMI_TypeDef * Instance***
- ***DCMI_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_DCMI_StateTypeDef State***
- ***__IO uint32_t XferCount***
- ***__IO uint32_t XferSize***

- ***uint32_t XferTransferNumber***
- ***uint32_t pBuffPtr***
- ***DMA_HandleTypeDef * DMA_Handle***
- ***__IO uint32_t ErrorCode***
- ***void(* FrameEventCallback***
- ***void(* VsyncEventCallback***
- ***void(* LineEventCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***DCMI_TypeDef* __DCMI_HandleTypeDef::Instance***
  DCMI Register base address
- ***DCMI_InitTypeDef __DCMI_HandleTypeDef::Init***
  DCMI parameters
- ***HAL_LockTypeDef __DCMI_HandleTypeDef::Lock***
  DCMI locking object
- ***__IO HAL_DCMI_StateTypeDef __DCMI_HandleTypeDef::State***
  DCMI state
- ***__IO uint32_t __DCMI_HandleTypeDef::XferCount***
  DMA transfer counter
- ***__IO uint32_t __DCMI_HandleTypeDef::XferSize***
  DMA transfer size
- ***uint32_t __DCMI_HandleTypeDef::XferTransferNumber***
  DMA transfer number
- ***uint32_t __DCMI_HandleTypeDef::pBuffPtr***
  Pointer to DMA output buffer
- ***DMA_HandleTypeDef* __DCMI_HandleTypeDef::DMA_Handle***
  Pointer to the DMA handler
- ***__IO uint32_t __DCMI_HandleTypeDef::ErrorCode***
  DCMI Error code
- ***void(* __DCMI_HandleTypeDef::FrameEventCallback)(struct __DCMI_HandleTypeDef *hdcmi)***
  DCMI Frame Event Callback
- ***void(* __DCMI_HandleTypeDef::VsyncEventCallback)(struct __DCMI_HandleTypeDef *hdcmi)***
  DCMI Vsync Event Callback
- ***void(* __DCMI_HandleTypeDef::LineEventCallback)(struct __DCMI_HandleTypeDef *hdcmi)***
  DCMI Line Event Callback
- ***void(* __DCMI_HandleTypeDef::ErrorCallback)(struct __DCMI_HandleTypeDef *hdcmi)***
  DCMI Error Callback
- ***void(* __DCMI_HandleTypeDef::MspInitCallback)(struct __DCMI_HandleTypeDef *hdcmi)***
  DCMI Msp Init callback
- ***void(* __DCMI_HandleTypeDef::MspDeInitCallback)(struct __DCMI_HandleTypeDef *hdcmi)***
  DCMI Msp DeInit callback

20.2 DCMI Firmware driver API description

The following section lists the various functions of the DCMI library.

20.2.1 How to use this driver

The sequence below describes how to use this driver to capture image from a camera module connected to the DCMI Interface. This sequence does not take into account the configuration of the camera module, which should be made before to configure and enable the DCMI to capture images.

1. Program the required configuration through following parameters: horizontal and vertical polarity, pixel clock polarity, Capture Rate, Synchronization Mode, code of the frame delimiter and data width using HAL_DCMI_Init() function.
2. Configure the selected DMA stream to transfer Data from DCMI DR register to the destination memory buffer.
3. Program the required configuration through following parameters: DCMI mode, destination memory Buffer address and the data length and enable capture using HAL_DCMI_Start_DMA() function.
4. Optionally, configure and Enable the CROP feature to select a rectangular window from the received image using HAL_DCMI_ConfigCrop() and HAL_DCMI_EnableCrop() functions
5. The capture can be stopped using HAL_DCMI_Stop() function.
6. To control DCMI state you can use the function HAL_DCMI_GetState().

Callback registration

DCMI HAL driver macros list

Below the list of most used macros in DCMI HAL driver.

- __HAL_DCMI_ENABLE: Enable the DCMI peripheral.
- __HAL_DCMI_DISABLE: Disable the DCMI peripheral.
- __HAL_DCMI_GET_FLAG: Get the DCMI pending flags.
- __HAL_DCMI_CLEAR_FLAG: Clear the DCMI pending flags.
- __HAL_DCMI_ENABLE_IT: Enable the specified DCMI interrupts.
- __HAL_DCMI_DISABLE_IT: Disable the specified DCMI interrupts.
- __HAL_DCMI_GET_IT_SOURCE: Check whether the specified DCMI interrupt has occurred or not.

Note: You can refer to the DCMI HAL driver header file for more useful macros

20.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the DCMI
- De-initialize the DCMI

This section contains the following APIs:

- ***HAL_DCMI_Init()***
- ***HAL_DCMI_DeInit()***
- ***HAL_DCMI_MspInit()***
- ***HAL_DCMI_MspDeInit()***
- ***HAL_DCMI_RegisterCallback()***
- ***HAL_DCMI_UnRegisterCallback()***

20.2.3 IO operation functions

This section provides functions allowing to:

- Configure destination address and data length and Enables DCMI DMA request and enables DCMI capture
- Stop the DCMI capture.
- Handles DCMI interrupt request.

This section contains the following APIs:

- ***HAL_DCMI_Start_DMA()***
- ***HAL_DCMI_Stop()***
- ***HAL_DCMI_Suspend()***
- ***HAL_DCMI_Resume()***

- *HAL_DCMI_IRQHandler()*
- *HAL_DCMI_ErrorCallback()*
- *HAL_DCMI_LineEventCallback()*
- *HAL_DCMI_VsyncEventCallback()*
- *HAL_DCMI_FrameEventCallback()*

20.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure the CROP feature.
- Enable/Disable the CROP feature.
- Set embedded synchronization delimiters unmasks.

This section contains the following APIs:

- *HAL_DCMI_ConfigCrop()*
- *HAL_DCMI_DisableCrop()*
- *HAL_DCMI_EnableCrop()*
- *HAL_DCMI_ConfigSyncUnmask()*

20.2.5 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the DCMI state.
- Get the specific DCMI error flag.

This section contains the following APIs:

- *HAL_DCMI_GetState()*
- *HAL_DCMI_GetError()*
- *HAL_DCMI_RegisterCallback()*
- *HAL_DCMI_UnRegisterCallback()*

20.2.6 Detailed description of functions

HAL_DCMI_Init

Function name

HAL_StatusTypeDef HAL_DCMI_Init (DCMI_HandleTypeDef * hdcmi)

Function description

Initializes the DCMI according to the specified parameters in the DCMI_InitTypeDef and create the associated handle.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** status

HAL_DCMI_DeInit

Function name

HAL_StatusTypeDef HAL_DCMI_DeInit (DCMI_HandleTypeDef * hdcmi)

Function description

Deinitializes the DCMI peripheral registers to their default reset values.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** status

HAL_DCMI_MspInit

Function name

void HAL_DCMI_MspInit (DCMI_HandleTypeDef * hdcmi)

Function description

Initializes the DCMI MSP.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **None:**

HAL_DCMI_MspDeInit

Function name

void HAL_DCMI_MspDeInit (DCMI_HandleTypeDef * hdcmi)

Function description

DeInitializes the DCMI MSP.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **None:**

HAL_DCMI_RegisterCallback

Function name

HAL_StatusTypeDef HAL_DCMI_RegisterCallback (DCMI_HandleTypeDef * hdcmi, HAL_DCMI_CallbackIDTypeDef CallbackID, pDCMI_CallbackTypeDef pCallback)

Function description

Register a User DCMI Callback To be used instead of the weak predefined callback.

Parameters

- **hdcmi:** DCMI handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_DCMI_LINE_EVENT_CB_ID Line Event callback ID
  - HAL_DCMI_FRAME_EVENT_CB_ID Frame Event callback ID
  - HAL_DCMI_VSYNC_EVENT_CB_ID Vsync Event callback ID
  - HAL_DCMI_ERROR_CB_ID Error callback ID
  - HAL_DCMI_MSPINIT_CB_ID MspInit callback ID
  - HAL_DCMI_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_DCMI_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_DCMI_UnRegisterCallback (DCMI_HandleTypeDef * hdcmi, HAL_DCMI_CallbackIDTypeDef CallbackID)

Function description

Unregister a DCMI Callback DCMI callback is redirected to the weak predefined callback.

Parameters

- **hdcmi:** DCMI handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_DCMI_LINE_EVENT_CB_ID Line Event callback ID
  - HAL_DCMI_FRAME_EVENT_CB_ID Frame Event callback ID
  - HAL_DCMI_VSYNC_EVENT_CB_ID Vsync Event callback ID
  - HAL_DCMI_ERROR_CB_ID Error callback ID
  - HAL_DCMI_MSPINIT_CB_ID MspInit callback ID
  - HAL_DCMI_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_DCMI_Start_DMA

Function name

HAL_StatusTypeDef HAL_DCMI_Start_DMA (DCMI_HandleTypeDef * hdcmi, uint32_t DCMI_Mode, uint32_t pData, uint32_t Length)

Function description

Enables DCMI DMA request and enables DCMI capture.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
- **DCMI_Mode:** DCMI capture mode snapshot or continuous grab.
- **pData:** The destination memory Buffer address (LCD Frame buffer).
- **Length:** The length of capture to be transferred.

Return values

- **HAL:** status

HAL_DCMI_Stop

Function name

HAL_StatusTypeDef HAL_DCMI_Stop (DCMI_HandleTypeDef * hdcmi)

Function description

Disable DCMI DMA request and Disable DCMI capture.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** status

HAL_DCMI_Suspend

Function name

HAL_StatusTypeDef HAL_DCMI_Suspend (DCMI_HandleTypeDef * hdcmi)

Function description

Suspend DCMI capture.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** status

HAL_DCMI_Resume

Function name

HAL_StatusTypeDef HAL_DCMI_Resume (DCMI_HandleTypeDef * hdcmi)

Function description

Resume DCMI capture.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** status

HAL_DCMI_ErrorCallback

Function name

void HAL_DCMI_ErrorCallback (DCMI_HandleTypeDef * hdcmi)

Function description

Error DCMI callback.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **None:**

HAL_DCMI_LineEventCallback

Function name

void HAL_DCMI_LineEventCallback (DCMI_HandleTypeDef * hdcmi)

Function description

Line Event callback.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **None:**

HAL_DCMI_FrameEventCallback

Function name

void HAL_DCMI_FrameEventCallback (DCMI_HandleTypeDef * hdcmi)

Function description

Frame Event callback.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **None:**

HAL_DCMI_VsyncEventCallback

Function name

void HAL_DCMI_VsyncEventCallback (DCMI_HandleTypeDef * hdcmi)

Function description

VSYNC Event callback.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **None:**

HAL_DCMI_IRQHandler

Function name

void HAL_DCMI_IRQHandler (DCMI_HandleTypeDef * hdcmi)

Function description

Handles DCMI interrupt request.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for the DCMI.

Return values

- **None:**

HAL_DCMI_ConfigCrop

Function name

HAL_StatusTypeDef HAL_DCMI_ConfigCrop (DCMI_HandleTypeDef * hdcmi, uint32_t X0, uint32_t Y0, uint32_t XSize, uint32_t YSize)

Function description

Configure the DCMI CROP coordinate.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
- **YSize:** DCMI Line number
- **XSize:** DCMI Pixel per line
- **X0:** DCMI window X offset
- **Y0:** DCMI window Y offset

Return values

- **HAL:** status

HAL_DCMI_EnableCrop

Function name

HAL_StatusTypeDef HAL_DCMI_EnableCrop (DCMI_HandleTypeDef * hdcmi)

Function description

Enable the Crop feature.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** status

HAL_DCMI_DisableCrop

Function name

HAL_StatusTypeDef HAL_DCMI_DisableCrop (DCMI_HandleTypeDef * hdcmi)

Function description

Disable the Crop feature.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** status

HAL_DCMI_ConfigSyncUnmask

Function name

HAL_StatusTypeDef HAL_DCMI_ConfigSyncUnmask (DCMI_HandleTypeDef * hdcmi, DCMI_SyncUnmaskTypeDef * SyncUnmask)

Function description

Set embedded synchronization delimiters unmasks.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
- **SyncUnmask:** pointer to a DCMI_SyncUnmaskTypeDef structure that contains the embedded synchronization delimiters unmasks.

Return values

- **HAL:** status

HAL_DCMI_GetState

Function name

HAL_DCMI_StateTypeDef HAL_DCMI_GetState (DCMI_HandleTypeDef * hdcmi)

Function description

Return the DCMI state.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** state

HAL_DCMI_GetError

Function name

uint32_t HAL_DCMI_GetError (DCMI_HandleTypeDef * hdcmi)

Function description

Return the DCMI error code.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **DCMI:** Error Code

20.3 DCMI Firmware driver defines

The following section lists the various define and macros of the module.

20.3.1 DCMI

DCMI

DCMI Byte Select Mode

DCMI_BSM_ALL

Interface captures all received data

DCMI_BSM_OTHER

Interface captures every other byte from the received data

DCMI_BSM_ALTERNATE_4

Interface captures one byte out of four

DCMI_BSM_ALTERNATE_2

Interface captures two bytes out of four

DCMI Byte Select Start

DCMI_OEBS_ODD

Interface captures first data from the frame/line start, second one being dropped

DCMI_OEBS_EVEN

Interface captures second data from the frame/line start, first one being dropped

DCMI Capture Mode

DCMI_MODE_CONTINUOUS

The received data are transferred continuously into the destination memory through the DMA

DCMI_MODE_SNAPSHOT

Once activated, the interface waits for the start of frame and then transfers a single frame through the DMA

DCMI Capture Rate

DCMI_CR_ALL_FRAME

All frames are captured

DCMI_CR_ALTERNATE_2_FRAME

Every alternate frame captured

DCMI_CR_ALTERNATE_4_FRAME

One frame in 4 frames captured

DCMI Error Code

HAL_DCMI_ERROR_NONE

No error

HAL_DCMI_ERROR_OVR

Overrun error

HAL_DCMI_ERROR_SYNC

Synchronization error

HAL_DCMI_ERROR_TIMEOUT

Timeout error

HAL_DCMI_ERROR_DMA

DMA error

HAL_DCMI_ERROR_INVALID_CALLBACK

Invalid Callback error

DCMI Exported Macros

__HAL_DCMI_RESET_HANDLE_STATE

Description:

- Reset DCMI handle state.

Parameters:

- __HANDLE__: specifies the DCMI handle.

Return value:

- None

__HAL_DCMI_ENABLE

Description:

- Enable the DCMI.

Parameters:

- __HANDLE__: DCMI handle

Return value:

- None

__HAL_DCMI_DISABLE

Description:

- Disable the DCMI.

Parameters:

- __HANDLE__: DCMI handle

Return value:

- None

__HAL_DCMI_GET_FLAG

Description:

- Get the DCMI pending flag.

Parameters:

- __HANDLE__: DCMI handle
- __FLAG__: Get the specified flag. This parameter can be one of the following values (no combination allowed)
  - DCMI_FLAG_HSYNC: HSYNC pin state (active line / synchronization between lines)
  - DCMI_FLAG_VSYNC: VSYNC pin state (active frame / synchronization between frames)
  - DCMI_FLAG_FNE: FIFO empty flag
  - DCMI_FLAG_FRAMERI: Frame capture complete flag mask
  - DCMI_FLAG_OVRRI: Overrun flag mask
  - DCMI_FLAG_ERRRI: Synchronization error flag mask
  - DCMI_FLAG_VSYNCRI: VSYNC flag mask
  - DCMI_FLAG_LINERI: Line flag mask
  - DCMI_FLAG_FRAMEMI: DCMI Capture complete masked interrupt status
  - DCMI_FLAG_OVRMI: DCMI Overrun masked interrupt status
  - DCMI_FLAG_ERRMI: DCMI Synchronization error masked interrupt status
  - DCMI_FLAG_VSYNCMI: DCMI VSYNC masked interrupt status
  - DCMI_FLAG_LINEMI: DCMI Line masked interrupt status

Return value:

- The: state of FLAG.

__HAL_DCMI_CLEAR_FLAG

Description:

- Clear the DCMI pending flags.

Parameters:

- __HANDLE__: DCMI handle
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - DCMI_FLAG_FRAMERI: Frame capture complete flag mask
  - DCMI_FLAG_OVFRI: Overflow flag mask
  - DCMI_FLAG_ERRRI: Synchronization error flag mask
  - DCMI_FLAG_VSYNCRI: VSYNC flag mask
  - DCMI_FLAG_LINERI: Line flag mask

Return value:

- None

__HAL_DCMI_ENABLE_IT

Description:

- Enable the specified DCMI interrupts.

Parameters:

- __HANDLE__: DCMI handle
- __INTERRUPT__: specifies the DCMI interrupt sources to be enabled. This parameter can be any combination of the following values:
  - DCMI_IT_FRAME: Frame capture complete interrupt mask
  - DCMI_IT_OVF: Overflow interrupt mask
  - DCMI_IT_ERR: Synchronization error interrupt mask
  - DCMI_IT_VSYNC: VSYNC interrupt mask
  - DCMI_IT_LINE: Line interrupt mask

Return value:

- None

__HAL_DCMI_DISABLE_IT

Description:

- Disable the specified DCMI interrupts.

Parameters:

- __HANDLE__: DCMI handle
- __INTERRUPT__: specifies the DCMI interrupt sources to be enabled. This parameter can be any combination of the following values:
  - DCMI_IT_FRAME: Frame capture complete interrupt mask
  - DCMI_IT_OVF: Overflow interrupt mask
  - DCMI_IT_ERR: Synchronization error interrupt mask
  - DCMI_IT_VSYNC: VSYNC interrupt mask
  - DCMI_IT_LINE: Line interrupt mask

Return value:

- None

__HAL_DCMI_GET_IT_SOURCE

Description:

- Check whether the specified DCMI interrupt has occurred or not.

Parameters:

- __HANDLE__: DCMI handle
- __INTERRUPT__: specifies the DCMI interrupt source to check. This parameter can be one of the following values:
  - DCMI_IT_FRAME: Frame capture complete interrupt mask
  - DCMI_IT_OVF: Overflow interrupt mask
  - DCMI_IT_ERR: Synchronization error interrupt mask
  - DCMI_IT_VSYNC: VSYNC interrupt mask
  - DCMI_IT_LINE: Line interrupt mask

Return value:

- The: state of INTERRUPT.

DCMI Extended Data Mode

DCMI_EXTEND_DATA_8B

Interface captures 8-bit data on every pixel clock

DCMI_EXTEND_DATA_10B

Interface captures 10-bit data on every pixel clock

DCMI_EXTEND_DATA_12B

Interface captures 12-bit data on every pixel clock

DCMI_EXTEND_DATA_14B

Interface captures 14-bit data on every pixel clock

DCMI Flags

DCMI_FLAG_HSYNC

HSYNC pin state (active line / synchronization between lines)

DCMI_FLAG_VSYNC

VSYNC pin state (active frame / synchronization between frames)

DCMI_FLAG_FNE

FIFO not empty flag

DCMI_FLAG_FRAMERI

Frame capture complete interrupt flag

DCMI_FLAG_OVRRI

Overrun interrupt flag

DCMI_FLAG_ERRRI

Synchronization error interrupt flag

DCMI_FLAG_VSYNCRI

VSYNC interrupt flag

DCMI_FLAG_LINERI

Line interrupt flag

DCMI_FLAG_FRAMEMI

DCMI Frame capture complete masked interrupt status

DCMI_FLAG_OVRMI

DCMI Overrun masked interrupt status

DCMI_FLAG_ERRMI

DCMI Synchronization error masked interrupt status

DCMI_FLAG_VSYNCMI

DCMI VSYNC masked interrupt status

DCMI_FLAG_LINEMI

DCMI Line masked interrupt status

DCMI HSYNC Polarity

DCMI_HSPOLARITY_LOW

Horizontal synchronization active Low

DCMI_HSPOLARITY_HIGH

Horizontal synchronization active High

DCMI interrupt sources

DCMI_IT_FRAME

Capture complete interrupt

DCMI_IT_OVR

Overrun interrupt

DCMI_IT_ERR

Synchronization error interrupt

DCMI_IT_VSYNC

VSYNC interrupt

DCMI_IT_LINE

Line interrupt

DCMI Line Select Mode

DCMI_LSM_ALL

Interface captures all received lines

DCMI_LSM_ALTERNATE_2

Interface captures one line out of two

DCMI Line Select Start

DCMI_OELS_ODD

Interface captures first line from the frame start, second one being dropped

DCMI_OELS_EVEN

Interface captures second line from the frame start, first one being dropped

DCMI MODE JPEG

DCMI_JPEG_DISABLE

Mode JPEG Disabled

DCMI_JPEG_ENABLE

Mode JPEG Enabled

DCMI PIXCK Polarity

DCMI_PCKPOLARITY_FALLING

Pixel clock active on Falling edge

DCMI_PCKPOLARITY_RISING

Pixel clock active on Rising edge

DCMI Synchronization Mode

DCMI_SYNCHRO_HARDWARE

Hardware synchronization data capture (frame/line start/stop) is synchronized with the HSYNC/VSYNC signals

DCMI_SYNCHRO_EMBEDDED

Embedded synchronization data capture is synchronized with synchronization codes embedded in the data flow

DCMI VSYNC Polarity

DCMI_VSPOLARITY_LOW

Vertical synchronization active Low

DCMI_VSPOLARITY_HIGH

Vertical synchronization active High

DCMI Window Coordinate

DCMI_WINDOW_COORDINATE

Window coordinate

DCMI Window Height

DCMI_WINDOW_HEIGHT

Window Height

21 HAL DFSDM Generic Driver

21.1 DFSDM Firmware driver registers structures

21.1.1 DFSDM_Channel_OutputClockTypeDef

DFSDM_Channel_OutputClockTypeDef is defined in the stm32h7xx_hal_dfsdm.h

Data Fields

- ***FunctionalState Activation***
- ***uint32_t Selection***
- ***uint32_t Divider***

Field Documentation

- ***FunctionalState DFSDM_Channel_OutputClockTypeDef::Activation***
  Output clock enable/disable
- ***uint32_t DFSDM_Channel_OutputClockTypeDef::Selection***
  Output clock is system clock or audio clock. This parameter can be a value of ***DFSDM_Channel_OuputClock***
- ***uint32_t DFSDM_Channel_OutputClockTypeDef::Divider***
  Output clock divider. This parameter must be a number between Min_Data = 2 and Max_Data = 256

21.1.2 DFSDM_Channel_InputTypeDef

DFSDM_Channel_InputTypeDef is defined in the stm32h7xx_hal_dfsdm.h

Data Fields

- ***uint32_t Multiplexer***
- ***uint32_t DataPacking***
- ***uint32_t Pins***

Field Documentation

- ***uint32_t DFSDM_Channel_InputTypeDef::Multiplexer***
  Input is external serial inputs, internal register or ADC output. This parameter can be a value of ***DFSDM_Channel_InputMultiplexer***
- ***uint32_t DFSDM_Channel_InputTypeDef::DataPacking***
  Standard, interleaved or dual mode for internal register. This parameter can be a value of ***DFSDM_Channel_DataPacking***
- ***uint32_t DFSDM_Channel_InputTypeDef::Pins***
  Input pins are taken from same or following channel. This parameter can be a value of ***DFSDM_Channel_InputPins***

21.1.3 DFSDM_Channel_SerialInterfaceTypeDef

DFSDM_Channel_SerialInterfaceTypeDef is defined in the stm32h7xx_hal_dfsdm.h

Data Fields

- ***uint32_t Type***
- ***uint32_t SpiClock***

Field Documentation

- ***uint32_t DFSDM_Channel_SerialInterfaceTypeDef::Type***
  SPI or Manchester modes. This parameter can be a value of ***DFSDM_Channel_SerialInterfaceType***
- ***uint32_t DFSDM_Channel_SerialInterfaceTypeDef::SpiClock***
  SPI clock select (external or internal with different sampling point). This parameter can be a value of ***DFSDM_Channel_SpiClock***

21.1.4 DFSDM_Channel_AwdTypeDef

DFSDM_Channel_AwdTypeDef is defined in the stm32h7xx_hal_dfsdm.h

Data Fields

- ***uint32_t FilterOrder***
- ***uint32_t Oversampling***

Field Documentation

- ***uint32_t DFSDM_Channel_AwdTypeDef::FilterOrder***
  Analog watchdog Sinc filter order. This parameter can be a value of ***DFSDM_Channel_AwdFilterOrder***
- ***uint32_t DFSDM_Channel_AwdTypeDef::Oversampling***
  Analog watchdog filter oversampling ratio. This parameter must be a number between Min_Data = 1 and Max_Data = 32

21.1.5 DFSDM_Channel_InitTypeDef

DFSDM_Channel_InitTypeDef is defined in the stm32h7xx_hal_dfsdm.h

Data Fields

- ***DFSDM_Channel_OutputClockTypeDef OutputClock***
- ***DFSDM_Channel_InputTypeDef Input***
- ***DFSDM_Channel_SerialInterfaceTypeDef SerialInterface***
- ***DFSDM_Channel_AwdTypeDef Awd***
- ***int32_t Offset***
- ***uint32_t RightBitShift***

Field Documentation

- ***DFSDM_Channel_OutputClockTypeDef DFSDM_Channel_InitTypeDef::OutputClock***
  DFSDM channel output clock parameters
- ***DFSDM_Channel_InputTypeDef DFSDM_Channel_InitTypeDef::Input***
  DFSDM channel input parameters
- ***DFSDM_Channel_SerialInterfaceTypeDef DFSDM_Channel_InitTypeDef::SerialInterface***
  DFSDM channel serial interface parameters
- ***DFSDM_Channel_AwdTypeDef DFSDM_Channel_InitTypeDef::Awd***
  DFSDM channel analog watchdog parameters
- ***int32_t DFSDM_Channel_InitTypeDef::Offset***
  DFSDM channel offset. This parameter must be a number between Min_Data = -8388608 and Max_Data = 8388607
- ***uint32_t DFSDM_Channel_InitTypeDef::RightBitShift***
  DFSDM channel right bit shift. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x1F

21.1.6 __DFSDM_Channel_HandleTypeDef

__DFSDM_Channel_HandleTypeDef is defined in the stm32h7xx_hal_dfsdm.h

Data Fields

- ***DFSDM_Channel_TypeDef * Instance***
- ***DFSDM_Channel_InitTypeDef Init***
- ***HAL_DFSDM_Channel_StateTypeDef State***
- ***void(* CkabCallback***
- ***void(* ScdCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***DFSDM_Channel_TypeDef* __DFSDM_Channel_HandleTypeDef::Instance***
  DFSDM channel instance
- ***DFSDM_Channel_InitTypeDef __DFSDM_Channel_HandleTypeDef::Init***
  DFSDM channel init parameters
- ***HAL_DFSDM_Channel_StateTypeDef __DFSDM_Channel_HandleTypeDef::State***
  DFSDM channel state
- ***void(* __DFSDM_Channel_HandleTypeDef::CkabCallback)(struct __DFSDM_Channel_HandleTypeDef *hdfsdm_channel)***
  DFSDM channel clock absence detection callback
- ***void(* __DFSDM_Channel_HandleTypeDef::ScdCallback)(struct __DFSDM_Channel_HandleTypeDef *hdfsdm_channel)***
  DFSDM channel short circuit detection callback
- ***void(* __DFSDM_Channel_HandleTypeDef::MspInitCallback)(struct __DFSDM_Channel_HandleTypeDef *hdfsdm_channel)***
  DFSDM channel MSP init callback
- ***void(* __DFSDM_Channel_HandleTypeDef::MspDeInitCallback)(struct __DFSDM_Channel_HandleTypeDef *hdfsdm_channel)***
  DFSDM channel MSP de-init callback

21.1.7 DFSDM_Filter_RegularParamTypeDef

DFSDM_Filter_RegularParamTypeDef is defined in the stm32h7xx_hal_dfsdm.h

Data Fields

- ***uint32_t Trigger***
- ***FunctionalState FastMode***
- ***FunctionalState DmaMode***

Field Documentation

- ***uint32_t DFSDM_Filter_RegularParamTypeDef::Trigger***
  Trigger used to start regular conversion: software or synchronous. This parameter can be a value of ***DFSDM_Filter_Trigger***
- ***FunctionalState DFSDM_Filter_RegularParamTypeDef::FastMode***
  Enable/disable fast mode for regular conversion
- ***FunctionalState DFSDM_Filter_RegularParamTypeDef::DmaMode***
  Enable/disable DMA for regular conversion

21.1.8 DFSDM_Filter_InjectedParamTypeDef

DFSDM_Filter_InjectedParamTypeDef is defined in the stm32h7xx_hal_dfsdm.h

Data Fields

- ***uint32_t Trigger***
- ***FunctionalState ScanMode***
- ***FunctionalState DmaMode***
- ***uint32_t ExtTrigger***
- ***uint32_t ExtTriggerEdge***

Field Documentation

- ***uint32_t DFSDM_Filter_InjectedParamTypeDef::Trigger***
  Trigger used to start injected conversion: software, external or synchronous. This parameter can be a value of ***DFSDM_Filter_Trigger***
- ***FunctionalState DFSDM_Filter_InjectedParamTypeDef::ScanMode***
  Enable/disable scanning mode for injected conversion
- ***FunctionalState DFSDM_Filter_InjectedParamTypeDef::DmaMode***
  Enable/disable DMA for injected conversion

- ***uint32_t DFSDM_Filter_InjectedParamTypeDef::ExtTrigger***
  External trigger. This parameter can be a value of ***DFSDM_Filter_ExtTrigger***
- ***uint32_t DFSDM_Filter_InjectedParamTypeDef::ExtTriggerEdge***
  External trigger edge: rising, falling or both. This parameter can be a value of ***DFSDM_Filter_ExtTriggerEdge***

21.1.9 DFSDM_Filter_FilterParamTypeDef

DFSDM_Filter_FilterParamTypeDef is defined in the stm32h7xx_hal_dfsdm.h

Data Fields

- ***uint32_t SincOrder***
- ***uint32_t Oversampling***
- ***uint32_t IntOversampling***

Field Documentation

- ***uint32_t DFSDM_Filter_FilterParamTypeDef::SincOrder***
  Sinc filter order. This parameter can be a value of ***DFSDM_Filter_SincOrder***
- ***uint32_t DFSDM_Filter_FilterParamTypeDef::Oversampling***
  Filter oversampling ratio. This parameter must be a number between Min_Data = 1 and Max_Data = 1024
- ***uint32_t DFSDM_Filter_FilterParamTypeDef::IntOversampling***
  Integrator oversampling ratio. This parameter must be a number between Min_Data = 1 and Max_Data = 256

21.1.10 DFSDM_Filter_InitTypeDef

DFSDM_Filter_InitTypeDef is defined in the stm32h7xx_hal_dfsdm.h

Data Fields

- ***DFSDM_Filter_RegularParamTypeDef RegularParam***
- ***DFSDM_Filter_InjectedParamTypeDef InjectedParam***
- ***DFSDM_Filter_FilterParamTypeDef FilterParam***

Field Documentation

- ***DFSDM_Filter_RegularParamTypeDef DFSDM_Filter_InitTypeDef::RegularParam***
  DFSDM regular conversion parameters
- ***DFSDM_Filter_InjectedParamTypeDef DFSDM_Filter_InitTypeDef::InjectedParam***
  DFSDM injected conversion parameters
- ***DFSDM_Filter_FilterParamTypeDef DFSDM_Filter_InitTypeDef::FilterParam***
  DFSDM filter parameters

21.1.11 __DFSDM_Filter_HandleTypeDef

__DFSDM_Filter_HandleTypeDef is defined in the stm32h7xx_hal_dfsdm.h

Data Fields

- ***DFSDM_Filter_TypeDef * Instance***
- ***DFSDM_Filter_InitTypeDef Init***
- ***DMA_HandleTypeDef * hdmaReg***
- ***DMA_HandleTypeDef * hdmaInj***
- ***uint32_t RegularContMode***
- ***uint32_t RegularTrigger***
- ***uint32_t InjectedTrigger***
- ***uint32_t ExtTriggerEdge***
- ***FunctionalState InjectedScanMode***
- ***uint32_t InjectedChannelsNbr***
- ***uint32_t InjConvRemaining***

- ***HAL_DFSDM_Filter_StateTypeDef State***
- ***uint32_t ErrorCode***
- ***void(* AwdCallback***
- ***void(* RegConvCpltCallback***
- ***void(* RegConvHalfCpltCallback***
- ***void(* InjConvCpltCallback***
- ***void(* InjConvHalfCpltCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***DFSDM_Filter_TypeDef* __DFSDM_Filter_HandleTypeDef::Instance***
  DFSDM filter instance
- ***DFSDM_Filter_InitTypeDef __DFSDM_Filter_HandleTypeDef::Init***
  DFSDM filter init parameters
- ***DMA_HandleTypeDef* __DFSDM_Filter_HandleTypeDef::hdmaReg***
  Pointer on DMA handler for regular conversions
- ***DMA_HandleTypeDef* __DFSDM_Filter_HandleTypeDef::hdmaInj***
  Pointer on DMA handler for injected conversions
- ***uint32_t __DFSDM_Filter_HandleTypeDef::RegularContMode***
  Regular conversion continuous mode
- ***uint32_t __DFSDM_Filter_HandleTypeDef::RegularTrigger***
  Trigger used for regular conversion
- ***uint32_t __DFSDM_Filter_HandleTypeDef::InjectedTrigger***
  Trigger used for injected conversion
- ***uint32_t __DFSDM_Filter_HandleTypeDef::ExtTriggerEdge***
  Rising, falling or both edges selected
- ***FunctionalState __DFSDM_Filter_HandleTypeDef::InjectedScanMode***
  Injected scanning mode
- ***uint32_t __DFSDM_Filter_HandleTypeDef::InjectedChannelsNbr***
  Number of channels in injected sequence
- ***uint32_t __DFSDM_Filter_HandleTypeDef::InjConvRemaining***
  Injected conversions remaining
- ***HAL_DFSDM_Filter_StateTypeDef __DFSDM_Filter_HandleTypeDef::State***
  DFSDM filter state
- ***uint32_t __DFSDM_Filter_HandleTypeDef::ErrorCode***
  DFSDM filter error code
- ***void(* __DFSDM_Filter_HandleTypeDef::AwdCallback)(struct __DFSDM_Filter_HandleTypeDef *hdfsdm_filter, uint32_t Channel, uint32_t Threshold)***
  DFSDM filter analog watchdog callback
- ***void(* __DFSDM_Filter_HandleTypeDef::RegConvCpltCallback)(struct __DFSDM_Filter_HandleTypeDef *hdfsdm_filter)***
  DFSDM filter regular conversion complete callback
- ***void(* __DFSDM_Filter_HandleTypeDef::RegConvHalfCpltCallback)(struct __DFSDM_Filter_HandleTypeDef *hdfsdm_filter)***
  DFSDM filter half regular conversion complete callback
- ***void(* __DFSDM_Filter_HandleTypeDef::InjConvCpltCallback)(struct __DFSDM_Filter_HandleTypeDef *hdfsdm_filter)***
  DFSDM filter injected conversion complete callback

- ***void(* __DFSDM_Filter_HandleTypeDef::InjConvHalfCpltCallback)(struct __DFSDM_Filter_HandleTypeDef *hdfsdm_filter)***
  DFSDM filter half injected conversion complete callback
- ***void(* __DFSDM_Filter_HandleTypeDef::ErrorCallback)(struct __DFSDM_Filter_HandleTypeDef *hdfsdm_filter)***
  DFSDM filter error callback
- ***void(* __DFSDM_Filter_HandleTypeDef::MspInitCallback)(struct __DFSDM_Filter_HandleTypeDef *hdfsdm_filter)***
  DFSDM filter MSP init callback
- ***void(* __DFSDM_Filter_HandleTypeDef::MspDeInitCallback)(struct __DFSDM_Filter_HandleTypeDef *hdfsdm_filter)***
  DFSDM filter MSP de-init callback

21.1.12 DFSDM_Filter_AwdParamTypeDef

DFSDM_Filter_AwdParamTypeDef is defined in the stm32h7xx_hal_dfsdm.h

Data Fields

- ***uint32_t DataSource***
- ***uint32_t Channel***
- ***int32_t HighThreshold***
- ***int32_t LowThreshold***
- ***uint32_t HighBreakSignal***
- ***uint32_t LowBreakSignal***

Field Documentation

- ***uint32_t DFSDM_Filter_AwdParamTypeDef::DataSource***
  Values from digital filter or from channel watchdog filter. This parameter can be a value of ***DFSDM_Filter_AwdDataSource***
- ***uint32_t DFSDM_Filter_AwdParamTypeDef::Channel***
  Analog watchdog channel selection. This parameter can be a values combination of ***DFSDM_Channel_Selection***
- ***int32_t DFSDM_Filter_AwdParamTypeDef::HighThreshold***
  High threshold for the analog watchdog. This parameter must be a number between Min_Data = -8388608 and Max_Data = 8388607
- ***int32_t DFSDM_Filter_AwdParamTypeDef::LowThreshold***
  Low threshold for the analog watchdog. This parameter must be a number between Min_Data = -8388608 and Max_Data = 8388607
- ***uint32_t DFSDM_Filter_AwdParamTypeDef::HighBreakSignal***
  Break signal assigned to analog watchdog high threshold event. This parameter can be a values combination of ***DFSDM_BreakSignals***
- ***uint32_t DFSDM_Filter_AwdParamTypeDef::LowBreakSignal***
  Break signal assigned to analog watchdog low threshold event. This parameter can be a values combination of ***DFSDM_BreakSignals***

21.2 DFSDM Firmware driver API description

The following section lists the various functions of the DFSDM library.

21.2.1 How to use this driver

Channel initialization

1. User has first to initialize channels (before filters initialization).

2. As prerequisite, fill in the HAL_DFSDM_ChannelMspInit() :
   – Enable DFSDMz clock interface with __HAL_RCC_DFSDMz_CLK_ENABLE().
   – Enable the clocks for the DFSDMz GPIOS with __HAL_RCC_GPIOx_CLK_ENABLE().
   – Configure these DFSDMz pins in alternate mode using HAL_GPIO_Init().
   – If interrupt mode is used, enable and configure DFSDMz_FLT0 global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().
3. Configure the output clock, input, serial interface, analog watchdog, offset and data right bit shift parameters for this channel using the HAL_DFSDM_ChannelInit() function.

Channel clock absence detector

1. Start clock absence detector using HAL_DFSDM_ChannelCkabStart() or HAL_DFSDM_ChannelCkabStart_IT().
2. In polling mode, use HAL_DFSDM_ChannelPollForCkab() to detect the clock absence.
3. In interrupt mode, HAL_DFSDM_ChannelCkabCallback() will be called if clock absence is detected.
4. Stop clock absence detector using HAL_DFSDM_ChannelCkabStop() or HAL_DFSDM_ChannelCkabStop_IT().
5. Please note that the same mode (polling or interrupt) has to be used for all channels because the channels are sharing the same interrupt.
6. Please note also that in interrupt mode, if clock absence detector is stopped for one channel, interrupt will be disabled for all channels.

Channel short circuit detector

1. Start short circuit detector using HAL_DFSDM_ChannelScdStart() or or HAL_DFSDM_ChannelScdStart_IT().
2. In polling mode, use HAL_DFSDM_ChannelPollForScd() to detect short circuit.
3. In interrupt mode, HAL_DFSDM_ChannelScdCallback() will be called if short circuit is detected.
4. Stop short circuit detector using HAL_DFSDM_ChannelScdStop() or or HAL_DFSDM_ChannelScdStop_IT().
5. Please note that the same mode (polling or interrupt) has to be used for all channels because the channels are sharing the same interrupt.
6. Please note also that in interrupt mode, if short circuit detector is stopped for one channel, interrupt will be disabled for all channels.

Channel analog watchdog value

1. Get analog watchdog filter value of a channel using HAL_DFSDM_ChannelGetAwdValue().

Channel offset value

1. Modify offset value of a channel using HAL_DFSDM_ChannelModifyOffset().

Filter initialization

1. After channel initialization, user has to init filters.
2. As prerequisite, fill in the HAL_DFSDM_FilterMspInit() :
   – If interrupt mode is used , enable and configure DFSDMz_FLTx global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ(). Please note that DFSDMz_FLT0 global interrupt could be already enabled if interrupt is used for channel.
   – If DMA mode is used, configure DMA with HAL_DMA_Init() and link it with DFSDMz filter handle using __HAL_LINKDMA().
3. Configure the regular conversion, injected conversion and filter parameters for this filter using the HAL_DFSDM_FilterInit() function.

Filter regular channel conversion

1. Select regular channel and enable/disable continuous mode using HAL_DFSDM_FilterConfigRegChannel().

2. Start regular conversion using HAL_DFSDM_FilterRegularStart(), HAL_DFSDM_FilterRegularStart_IT(), HAL_DFSDM_FilterRegularStart_DMA() or HAL_DFSDM_FilterRegularMsbStart_DMA().
3. In polling mode, use HAL_DFSDM_FilterPollForRegConversion() to detect the end of regular conversion.
4. In interrupt mode, HAL_DFSDM_FilterRegConvCpltCallback() will be called at the end of regular conversion.
5. Get value of regular conversion and corresponding channel using HAL_DFSDM_FilterGetRegularValue().
6. In DMA mode, HAL_DFSDM_FilterRegConvHalfCpltCallback() and HAL_DFSDM_FilterRegConvCpltCallback() will be called respectively at the half transfer and at the transfer complete. Please note that HAL_DFSDM_FilterRegConvHalfCpltCallback() will be called only in DMA circular mode.
7. Stop regular conversion using HAL_DFSDM_FilterRegularStop(), HAL_DFSDM_FilterRegularStop_IT() or HAL_DFSDM_FilterRegularStop_DMA().

Filter injected channels conversion

1. Select injected channels using HAL_DFSDM_FilterConfigInjChannel().
2. Start injected conversion using HAL_DFSDM_FilterInjectedStart(), HAL_DFSDM_FilterInjectedStart_IT(), HAL_DFSDM_FilterInjectedStart_DMA() or HAL_DFSDM_FilterInjectedMsbStart_DMA().
3. In polling mode, use HAL_DFSDM_FilterPollForInjConversion() to detect the end of injected conversion.
4. In interrupt mode, HAL_DFSDM_FilterInjConvCpltCallback() will be called at the end of injected conversion.
5. Get value of injected conversion and corresponding channel using HAL_DFSDM_FilterGetInjectedValue().
6. In DMA mode, HAL_DFSDM_FilterInjConvHalfCpltCallback() and HAL_DFSDM_FilterInjConvCpltCallback() will be called respectively at the half transfer and at the transfer complete. Please note that HAL_DFSDM_FilterInjConvCpltCallback() will be called only in DMA circular mode.
7. Stop injected conversion using HAL_DFSDM_FilterInjectedStop(), HAL_DFSDM_FilterInjectedStop_IT() or HAL_DFSDM_FilterInjectedStop_DMA().

Filter analog watchdog

1. Start filter analog watchdog using HAL_DFSDM_FilterAwdStart_IT().
2. HAL_DFSDM_FilterAwdCallback() will be called if analog watchdog occurs.
3. Stop filter analog watchdog using HAL_DFSDM_FilterAwdStop_IT().

Filter extreme detector

1. Start filter extreme detector using HAL_DFSDM_FilterExdStart().
2. Get extreme detector maximum value using HAL_DFSDM_FilterGetExdMaxValue().
3. Get extreme detector minimum value using HAL_DFSDM_FilterGetExdMinValue().
4. Start filter extreme detector using HAL_DFSDM_FilterExdStop().

Filter conversion time

1. Get conversion time value using HAL_DFSDM_FilterGetConvTimeValue().

Callback registration

The compilation define USE_HAL_DFSDM_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use functions HAL_DFSDM_Channel_RegisterCallback(), HAL_DFSDM_Filter_RegisterCallback() or HAL_DFSDM_Filter_RegisterAwdCallback() to register a user callback.

Function HAL_DFSDM_Channel_RegisterCallback() allows to register following callbacks:

- CkabCallback : DFSDM channel clock absence detection callback.
- ScdCallback : DFSDM channel short circuit detection callback.
- MspInitCallback : DFSDM channel MSP init callback.
- MspDeInitCallback : DFSDM channel MSP de-init callback.

This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Function HAL_DFSDM_Filter_RegisterCallback() allows to register following callbacks:

- RegConvCpltCallback : DFSDM filter regular conversion complete callback.
- RegConvHalfCpltCallback : DFSDM filter half regular conversion complete callback.
- InjConvCpltCallback : DFSDM filter injected conversion complete callback.
- InjConvHalfCpltCallback : DFSDM filter half injected conversion complete callback.
- ErrorCallback : DFSDM filter error callback.
- MspInitCallback : DFSDM filter MSP init callback.
- MspDeInitCallback : DFSDM filter MSP de-init callback.

This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

For specific DFSDM filter analog watchdog callback use dedicated register callback: HAL_DFSDM_Filter_RegisterAwdCallback().

Use functions HAL_DFSDM_Channel_UnRegisterCallback() or HAL_DFSDM_Filter_UnRegisterCallback() to reset a callback to the default weak function.

HAL_DFSDM_Channel_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID.

This function allows to reset following callbacks:

- CkabCallback : DFSDM channel clock absence detection callback.
- ScdCallback : DFSDM channel short circuit detection callback.
- MspInitCallback : DFSDM channel MSP init callback.
- MspDeInitCallback : DFSDM channel MSP de-init callback.

HAL_DFSDM_Filter_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID.

This function allows to reset following callbacks:

- RegConvCpltCallback : DFSDM filter regular conversion complete callback.
- RegConvHalfCpltCallback : DFSDM filter half regular conversion complete callback.
- InjConvCpltCallback : DFSDM filter injected conversion complete callback.
- InjConvHalfCpltCallback : DFSDM filter half injected conversion complete callback.
- ErrorCallback : DFSDM filter error callback.
- MspInitCallback : DFSDM filter MSP init callback.
- MspDeInitCallback : DFSDM filter MSP de-init callback.

For specific DFSDM filter analog watchdog callback use dedicated unregister callback: HAL_DFSDM_Filter_UnRegisterAwdCallback().

By default, after the call of init function and if the state is RESET all callbacks are reset to the corresponding legacy weak functions: examples HAL_DFSDM_ChannelScdCallback(), HAL_DFSDM_FilterErrorCallback(). Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak functions in the init and de-init only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the init and de-init keep and use the user MspInit/MspDeInit callbacks (registered beforehand)

Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the init/de-init. In that case first register the MspInit/MspDeInit user callbacks using HAL_DFSDM_Channel_RegisterCallback() or HAL_DFSDM_Filter_RegisterCallback() before calling init or de-init function.

When The compilation define USE_HAL_DFSDM_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak callbacks are used.

21.2.2 Channel initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the DFSDM channel.
- De-initialize the DFSDM channel.

This section contains the following APIs:

- ***HAL_DFSDM_ChannelInit()***
- ***HAL_DFSDM_ChannelDeInit()***

- *HAL_DFSDM_ChannelMspInit()*
- *HAL_DFSDM_ChannelMspDeInit()*
- *HAL_DFSDM_Channel_RegisterCallback()*
- *HAL_DFSDM_Channel_UnRegisterCallback()*

21.2.3 Channel operation functions

This section provides functions allowing to:

- Manage clock absence detector feature.
- Manage short circuit detector feature.
- Get analog watchdog value.
- Modify offset value.

This section contains the following APIs:

- *HAL_DFSDM_ChannelCkabStart()*
- *HAL_DFSDM_ChannelPollForCkab()*
- *HAL_DFSDM_ChannelCkabStop()*
- *HAL_DFSDM_ChannelCkabStart_IT()*
- *HAL_DFSDM_ChannelCkabCallback()*
- *HAL_DFSDM_ChannelCkabStop_IT()*
- *HAL_DFSDM_ChannelScdStart()*
- *HAL_DFSDM_ChannelPollForScd()*
- *HAL_DFSDM_ChannelScdStop()*
- *HAL_DFSDM_ChannelScdStart_IT()*
- *HAL_DFSDM_ChannelScdCallback()*
- *HAL_DFSDM_ChannelScdStop_IT()*
- *HAL_DFSDM_ChannelGetAwdValue()*
- *HAL_DFSDM_ChannelModifyOffset()*

21.2.4 Channel state function

This section provides function allowing to:

- Get channel handle state.

This section contains the following APIs:

- *HAL_DFSDM_ChannelGetState()*

21.2.5 Filter initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the DFSDM filter.
- De-initialize the DFSDM filter.

This section contains the following APIs:

- *HAL_DFSDM_FilterInit()*
- *HAL_DFSDM_FilterDeInit()*
- *HAL_DFSDM_FilterMspInit()*
- *HAL_DFSDM_FilterMspDeInit()*
- *HAL_DFSDM_Filter_RegisterCallback()*
- *HAL_DFSDM_Filter_UnRegisterCallback()*
- *HAL_DFSDM_Filter_RegisterAwdCallback()*
- *HAL_DFSDM_Filter_UnRegisterAwdCallback()*

21.2.6 Filter control functions

This section provides functions allowing to:

- Select channel and enable/disable continuous mode for regular conversion.
- Select channels for injected conversion.

This section contains the following APIs:

- ***HAL_DFSDM_FilterConfigRegChannel()***
- ***HAL_DFSDM_FilterConfigInjChannel()***

21.2.7 Filter operation functions

This section provides functions allowing to:

- Start conversion of regular/injected channel.
- Poll for the end of regular/injected conversion.
- Stop conversion of regular/injected channel.
- Start conversion of regular/injected channel and enable interrupt.
- Call the callback functions at the end of regular/injected conversions.
- Stop conversion of regular/injected channel and disable interrupt.
- Start conversion of regular/injected channel and enable DMA transfer.
- Stop conversion of regular/injected channel and disable DMA transfer.
- Start analog watchdog and enable interrupt.
- Call the callback function when analog watchdog occurs.
- Stop analog watchdog and disable interrupt.
- Start extreme detector.
- Stop extreme detector.
- Get result of regular channel conversion.
- Get result of injected channel conversion.
- Get extreme detector maximum and minimum values.
- Get conversion time.
- Handle DFSDM interrupt request.

This section contains the following APIs:

- ***HAL_DFSDM_FilterRegularStart()***
- ***HAL_DFSDM_FilterPollForRegConversion()***
- ***HAL_DFSDM_FilterRegularStop()***
- ***HAL_DFSDM_FilterRegularStart_IT()***
- ***HAL_DFSDM_FilterRegularStop_IT()***
- ***HAL_DFSDM_FilterRegularStart_DMA()***
- ***HAL_DFSDM_FilterRegularMsbStart_DMA()***
- ***HAL_DFSDM_FilterRegularStop_DMA()***
- ***HAL_DFSDM_FilterGetRegularValue()***
- ***HAL_DFSDM_FilterInjectedStart()***
- ***HAL_DFSDM_FilterPollForInjConversion()***
- ***HAL_DFSDM_FilterInjectedStop()***
- ***HAL_DFSDM_FilterInjectedStart_IT()***
- ***HAL_DFSDM_FilterInjectedStop_IT()***
- ***HAL_DFSDM_FilterInjectedStart_DMA()***
- ***HAL_DFSDM_FilterInjectedMsbStart_DMA()***
- ***HAL_DFSDM_FilterInjectedStop_DMA()***
- ***HAL_DFSDM_FilterGetInjectedValue()***
- ***HAL_DFSDM_FilterAwdStart_IT()***
- ***HAL_DFSDM_FilterAwdStop_IT()***
- ***HAL_DFSDM_FilterExdStart()***

- *HAL_DFSDM_FilterExdStop()*
- *HAL_DFSDM_FilterGetExdMaxValue()*
- *HAL_DFSDM_FilterGetExdMinValue()*
- *HAL_DFSDM_FilterGetConvTimeValue()*
- *HAL_DFSDM_IRQHandler()*
- *HAL_DFSDM_FilterRegConvCpltCallback()*
- *HAL_DFSDM_FilterRegConvHalfCpltCallback()*
- *HAL_DFSDM_FilterInjConvCpltCallback()*
- *HAL_DFSDM_FilterInjConvHalfCpltCallback()*
- *HAL_DFSDM_FilterAwdCallback()*
- *HAL_DFSDM_FilterErrorCallback()*

21.2.8 Filter state functions

This section provides functions allowing to:

- Get the DFSDM filter state.
- Get the DFSDM filter error.

This section contains the following APIs:

- *HAL_DFSDM_FilterGetState()*
- *HAL_DFSDM_FilterGetError()*

21.2.9 Detailed description of functions

HAL_DFSDM_ChannelInit

Function name

HAL_StatusTypeDef HAL_DFSDM_ChannelInit (DFSDM_Channel_HandleTypeDef * hdfsdm_channel)

Function description

Initialize the DFSDM channel according to the specified parameters in the DFSDM_ChannelInitTypeDef structure and initialize the associated handle.

Parameters

- **hdfsdm_channel:** DFSDM channel handle.

Return values

- **HAL:** status.

HAL_DFSDM_ChannelDeInit

Function name

HAL_StatusTypeDef HAL_DFSDM_ChannelDeInit (DFSDM_Channel_HandleTypeDef * hdfsdm_channel)

Function description

De-initialize the DFSDM channel.

Parameters

- **hdfsdm_channel:** DFSDM channel handle.

Return values

- **HAL:** status.

HAL_DFSDM_ChannelMspInit

Function name

void HAL_DFSDM_ChannelMspInit (DFSDM_Channel_HandleTypeDef * hdfsdm_channel)

Function description

Initialize the DFSDM channel MSP.

Parameters

- **hdfsdm_channel:** DFSDM channel handle.

Return values

- **None:**

HAL_DFSDM_ChannelMspDeInit

Function name

void HAL_DFSDM_ChannelMspDeInit (DFSDM_Channel_HandleTypeDef * hdfsdm_channel)

Function description

De-initialize the DFSDM channel MSP.

Parameters

- **hdfsdm_channel:** DFSDM channel handle.

Return values

- **None:**

HAL_DFSDM_Channel_RegisterCallback

Function name

HAL_StatusTypeDef HAL_DFSDM_Channel_RegisterCallback (DFSDM_Channel_HandleTypeDef * hdfsdm_channel, HAL_DFSDM_Channel_CallbackIDTypeDef CallbackID, pDFSDM_Channel_CallbackTypeDef pCallback)

Function description

Register a user DFSDM channel callback to be used instead of the weak predefined callback.

Parameters

- **hdfsdm_channel:** DFSDM channel handle.
- **CallbackID:** ID of the callback to be registered. This parameter can be one of the following values:
  - HAL_DFSDM_CHANNEL_CKAB_CB_ID clock absence detection callback ID.
  - HAL_DFSDM_CHANNEL_SCD_CB_ID short circuit detection callback ID.
  - HAL_DFSDM_CHANNEL_MSPINIT_CB_ID MSP init callback ID.
  - HAL_DFSDM_CHANNEL_MSPDEINIT_CB_ID MSP de-init callback ID.
- **pCallback:** pointer to the callback function.

Return values

- **HAL:** status.

HAL_DFSDM_Channel_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_DFSDM_Channel_UnRegisterCallback (DFSDM_Channel_HandleTypeDef * hdfsdm_channel, HAL_DFSDM_Channel_CallbackIDTypeDef CallbackID)

Function description

Unregister a user DFSDM channel callback.

Parameters

- **hdfsdm_channel:** DFSDM channel handle.
- **CallbackID:** ID of the callback to be unregistered. This parameter can be one of the following values:
  - HAL_DFSDM_CHANNEL_CKAB_CB_ID clock absence detection callback ID.
  - HAL_DFSDM_CHANNEL_SCD_CB_ID short circuit detection callback ID.
  - HAL_DFSDM_CHANNEL_MSPINIT_CB_ID MSP init callback ID.
  - HAL_DFSDM_CHANNEL_MSPDEINIT_CB_ID MSP de-init callback ID.

Return values

- **HAL:** status.

HAL_DFSDM_ChannelCkabStart

Function name

HAL_StatusTypeDef HAL_DFSDM_ChannelCkabStart (DFSDM_Channel_HandleTypeDef * hdfsdm_channel)

Function description

This function allows to start clock absence detection in polling mode.

Parameters

- **hdfsdm_channel:** DFSDM channel handle.

Return values

- **HAL:** status

Notes

- Same mode has to be used for all channels.
- If clock is not available on this channel during 5 seconds, clock absence detection will not be activated and function will return HAL_TIMEOUT error.

HAL_DFSDM_ChannelCkabStart_IT

Function name

HAL_StatusTypeDef HAL_DFSDM_ChannelCkabStart_IT (DFSDM_Channel_HandleTypeDef * hdfsdm_channel)

Function description

This function allows to start clock absence detection in interrupt mode.

Parameters

- **hdfsdm_channel:** DFSDM channel handle.

Return values

- **HAL:** status

Notes

- Same mode has to be used for all channels.
- If clock is not available on this channel during 5 seconds, clock absence detection will not be activated and function will return HAL_TIMEOUT error.

HAL_DFSDM_ChannelCkabStop

Function name

HAL_StatusTypeDef HAL_DFSDM_ChannelCkabStop (DFSDM_Channel_HandleTypeDef * hdfsdm_channel)

Function description

This function allows to stop clock absence detection in polling mode.

Parameters

- **hdfsdm_channel:** DFSDM channel handle.

Return values

- **HAL:** status

HAL_DFSDM_ChannelCkabStop_IT

Function name

HAL_StatusTypeDef HAL_DFSDM_ChannelCkabStop_IT (DFSDM_Channel_HandleTypeDef * hdfsdm_channel)

Function description

This function allows to stop clock absence detection in interrupt mode.

Parameters

- **hdfsdm_channel:** DFSDM channel handle.

Return values

- **HAL:** status

Notes

- Interrupt will be disabled for all channels

HAL_DFSDM_ChannelScdStart

Function name

HAL_StatusTypeDef HAL_DFSDM_ChannelScdStart (DFSDM_Channel_HandleTypeDef * hdfsdm_channel, uint32_t Threshold, uint32_t BreakSignal)

Function description

This function allows to start short circuit detection in polling mode.

Parameters

- **hdfsdm_channel:** DFSDM channel handle.
- **Threshold:** Short circuit detector threshold. This parameter must be a number between Min_Data = 0 and Max_Data = 255.
- **BreakSignal:** Break signals assigned to short circuit event. This parameter can be a values combination of DFSDM break signals.

Return values

- **HAL:** status

Notes

- Same mode has to be used for all channels

HAL_DFSDM_ChannelScdStart_IT

Function name

HAL_StatusTypeDef HAL_DFSDM_ChannelScdStart_IT (DFSDM_Channel_HandleTypeDef * hdfsdm_channel, uint32_t Threshold, uint32_t BreakSignal)

Function description

This function allows to start short circuit detection in interrupt mode.

Parameters

- **hdfsdm_channel:** DFSDM channel handle.
- **Threshold:** Short circuit detector threshold. This parameter must be a number between Min_Data = 0 and Max_Data = 255.
- **BreakSignal:** Break signals assigned to short circuit event. This parameter can be a values combination of DFSDM break signals.

Return values

- **HAL:** status

Notes

- Same mode has to be used for all channels

HAL_DFSDM_ChannelScdStop

Function name

HAL_StatusTypeDef HAL_DFSDM_ChannelScdStop (DFSDM_Channel_HandleTypeDef * hdfsdm_channel)

Function description

This function allows to stop short circuit detection in polling mode.

Parameters

- **hdfsdm_channel:** DFSDM channel handle.

Return values

- **HAL:** status

HAL_DFSDM_ChannelScdStop_IT

Function name

HAL_StatusTypeDef HAL_DFSDM_ChannelScdStop_IT (DFSDM_Channel_HandleTypeDef * hdfsdm_channel)

Function description

This function allows to stop short circuit detection in interrupt mode.

Parameters

- **hdfsdm_channel:** DFSDM channel handle.

Return values

- **HAL:** status

Notes

- Interrupt will be disabled for all channels

HAL_DFSDM_ChannelGetAwdValue

Function name

int16_t HAL_DFSDM_ChannelGetAwdValue (const DFSDM_Channel_HandleTypeDef * hdfsdm_channel)

Function description

This function allows to get channel analog watchdog value.

Parameters

- **hdfsdm_channel:** DFSDM channel handle.

Return values

- **Channel:** analog watchdog value.

HAL_DFSDM_ChannelModifyOffset

Function name

HAL_StatusTypeDef HAL_DFSDM_ChannelModifyOffset (DFSDM_Channel_HandleTypeDef * hdfsdm_channel, int32_t Offset)

Function description

This function allows to modify channel offset value.

Parameters

- **hdfsdm_channel:** DFSDM channel handle.
- **Offset:** DFSDM channel offset. This parameter must be a number between Min_Data = -8388608 and Max_Data = 8388607.

Return values

- **HAL:** status.

HAL_DFSDM_ChannelPollForCkab

Function name

HAL_StatusTypeDef HAL_DFSDM_ChannelPollForCkab (const DFSDM_Channel_HandleTypeDef * hdfsdm_channel, uint32_t Timeout)

Function description

This function allows to poll for the clock absence detection.

Parameters

- **hdfsdm_channel:** DFSDM channel handle.
- **Timeout:** Timeout value in milliseconds.

Return values

- **HAL:** status

HAL_DFSDM_ChannelPollForScd

Function name

HAL_StatusTypeDef HAL_DFSDM_ChannelPollForScd (const DFSDM_Channel_HandleTypeDef * hdfsdm_channel, uint32_t Timeout)

Function description

This function allows to poll for the short circuit detection.

Parameters

- **hdfsdm_channel:** DFSDM channel handle.
- **Timeout:** Timeout value in milliseconds.

Return values

- **HAL:** status

HAL_DFSDM_ChannelCkabCallback

Function name

void HAL_DFSDM_ChannelCkabCallback (DFSDM_Channel_HandleTypeDef * hdfsdm_channel)

Function description

Clock absence detection callback.

Parameters

- **hdfsdm_channel:** DFSDM channel handle.

Return values

- **None:**

HAL_DFSDM_ChannelScdCallback

Function name

void HAL_DFSDM_ChannelScdCallback (DFSDM_Channel_HandleTypeDef * hdfsdm_channel)

Function description

Short circuit detection callback.

Parameters

- **hdfsdm_channel:** DFSDM channel handle.

Return values

- **None:**

HAL_DFSDM_ChannelGetState

Function name

HAL_DFSDM_Channel_StateTypeDef HAL_DFSDM_ChannelGetState (const DFSDM_Channel_HandleTypeDef * hdfsdm_channel)

Function description

This function allows to get the current DFSDM channel handle state.

Parameters

- **hdfsdm_channel:** DFSDM channel handle.

Return values

- **DFSDM:** channel state.

HAL_DFSDM_FilterInit

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterInit (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)

Function description

Initialize the DFSDM filter according to the specified parameters in the DFSDM_FilterInitTypeDef structure and initialize the associated handle.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.

Return values

- **HAL:** status.

HAL_DFSDM_FilterDeInit

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterDeInit (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)

Function description

De-initializes the DFSDM filter.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.

Return values

- **HAL:** status.

HAL_DFSDM_FilterMspInit

Function name

void HAL_DFSDM_FilterMspInit (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)

Function description

Initializes the DFSDM filter MSP.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.

Return values

- **None:**

HAL_DFSDM_FilterMspDeInit

Function name

void HAL_DFSDM_FilterMspDeInit (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)

Function description

De-initializes the DFSDM filter MSP.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.

Return values

- **None:**

HAL_DFSDM_Filter_RegisterCallback

Function name

HAL_StatusTypeDef HAL_DFSDM_Filter_RegisterCallback (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, HAL_DFSDM_Filter_CallbackIDTypeDef CallbackID, pDFSDM_Filter_CallbackTypeDef pCallback)

Function description

Register a user DFSDM filter callback to be used instead of the weak predefined callback.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.
- **CallbackID:** ID of the callback to be registered. This parameter can be one of the following values:
  - HAL_DFSDM_FILTER_REGCONV_COMPLETE_CB_ID regular conversion complete callback ID.
  - HAL_DFSDM_FILTER_REGCONV_HALFCOMPLETE_CB_ID half regular conversion complete callback ID.
  - HAL_DFSDM_FILTER_INJCONV_COMPLETE_CB_ID injected conversion complete callback ID.
  - HAL_DFSDM_FILTER_INJCONV_HALFCOMPLETE_CB_ID half injected conversion complete callback ID.
  - HAL_DFSDM_FILTER_ERROR_CB_ID error callback ID.
  - HAL_DFSDM_FILTER_MSPINIT_CB_ID MSP init callback ID.
  - HAL_DFSDM_FILTER_MSPDEINIT_CB_ID MSP de-init callback ID.
- **pCallback:** pointer to the callback function.

Return values

- **HAL:** status.

HAL_DFSDM_Filter_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_DFSDM_Filter_UnRegisterCallback (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, HAL_DFSDM_Filter_CallbackIDTypeDef CallbackID)

Function description

Unregister a user DFSDM filter callback.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.
- **CallbackID:** ID of the callback to be unregistered. This parameter can be one of the following values:
  - HAL_DFSDM_FILTER_REGCONV_COMPLETE_CB_ID regular conversion complete callback ID.
  - HAL_DFSDM_FILTER_REGCONV_HALFCOMPLETE_CB_ID half regular conversion complete callback ID.
  - HAL_DFSDM_FILTER_INJCONV_COMPLETE_CB_ID injected conversion complete callback ID.
  - HAL_DFSDM_FILTER_INJCONV_HALFCOMPLETE_CB_ID half injected conversion complete callback ID.
  - HAL_DFSDM_FILTER_ERROR_CB_ID error callback ID.
  - HAL_DFSDM_FILTER_MSPINIT_CB_ID MSP init callback ID.
  - HAL_DFSDM_FILTER_MSPDEINIT_CB_ID MSP de-init callback ID.

Return values

- **HAL:** status.

HAL_DFSDM_Filter_RegisterAwdCallback

Function name

HAL_StatusTypeDef HAL_DFSDM_Filter_RegisterAwdCallback (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, pDFSDM_Filter_AwdCallbackTypeDef pCallback)

Function description

Register a user DFSDM filter analog watchdog callback to be used instead of the weak predefined callback.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.
- **pCallback:** pointer to the DFSDM filter analog watchdog callback function.

Return values

- **HAL:** status.

HAL_DFSDM_Filter_UnRegisterAwdCallback

Function name

HAL_StatusTypeDef HAL_DFSDM_Filter_UnRegisterAwdCallback (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)

Function description

Unregister a user DFSDM filter analog watchdog callback.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.

Return values

- **HAL:** status.

HAL_DFSDM_FilterConfigRegChannel

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterConfigRegChannel (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, uint32_t Channel, uint32_t ContinuousMode)

Function description

This function allows to select channel and to enable/disable continuous mode for regular conversion.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.
- **Channel:** Channel for regular conversion. This parameter can be a value of DFSDM Channel Selection.
- **ContinuousMode:** Enable/disable continuous mode for regular conversion. This parameter can be a value of DFSDM Continuous Mode.

Return values

- **HAL:** status

HAL_DFSDM_FilterConfigInjChannel

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterConfigInjChannel (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, uint32_t Channel)

Function description

This function allows to select channels for injected conversion.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.
- **Channel:** Channels for injected conversion. This parameter can be a values combination of DFSDM Channel Selection.

Return values

- **HAL:** status

HAL_DFSDM_FilterRegularStart

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterRegularStart (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)

Function description

This function allows to start regular conversion in polling mode.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.

Return values

- **HAL:** status

Notes

- This function should be called only when DFSDM filter instance is in idle state or if injected conversion is ongoing.

HAL_DFSDM_FilterRegularStart_IT

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterRegularStart_IT (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)

Function description

This function allows to start regular conversion in interrupt mode.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.

Return values

- **HAL:** status

Notes

- This function should be called only when DFSDM filter instance is in idle state or if injected conversion is ongoing.

HAL_DFSDM_FilterRegularStart_DMA

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterRegularStart_DMA (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, int32_t * pData, uint32_t Length)

Function description

This function allows to start regular conversion in DMA mode.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.
- **pData:** The destination buffer address.
- **Length:** The length of data to be transferred from DFSDM filter to memory.

Return values

- **HAL:** status

Notes

- This function should be called only when DFSDM filter instance is in idle state or if injected conversion is ongoing. Please note that data on buffer will contain signed regular conversion value on 24 most significant bits and corresponding channel on 3 least significant bits.

HAL_DFSDM_FilterRegularMsbStart_DMA

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterRegularMsbStart_DMA (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, int16_t * pData, uint32_t Length)

Function description

This function allows to start regular conversion in DMA mode and to get only the 16 most significant bits of conversion.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.
- **pData:** The destination buffer address.
- **Length:** The length of data to be transferred from DFSDM filter to memory.

Return values

- **HAL:** status

Notes

- This function should be called only when DFSDM filter instance is in idle state or if injected conversion is ongoing. Please note that data on buffer will contain signed 16 most significant bits of regular conversion.

HAL_DFSDM_FilterRegularStop

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterRegularStop (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)

Function description

This function allows to stop regular conversion in polling mode.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.

Return values

- **HAL:** status

Notes

- This function should be called only if regular conversion is ongoing.

HAL_DFSDM_FilterRegularStop_IT

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterRegularStop_IT (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)

Function description

This function allows to stop regular conversion in interrupt mode.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.

Return values

- **HAL:** status

Notes

- This function should be called only if regular conversion is ongoing.

HAL_DFSDM_FilterRegularStop_DMA

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterRegularStop_DMA (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)

Function description

This function allows to stop regular conversion in DMA mode.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.

Return values

- **HAL:** status

Notes

- This function should be called only if regular conversion is ongoing.

HAL_DFSDM_FilterInjectedStart

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterInjectedStart (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)

Function description

This function allows to start injected conversion in polling mode.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.

Return values

- **HAL:** status

Notes

- This function should be called only when DFSDM filter instance is in idle state or if regular conversion is ongoing.

HAL_DFSDM_FilterInjectedStart_IT

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterInjectedStart_IT (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)

Function description

This function allows to start injected conversion in interrupt mode.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.

Return values

- **HAL:** status

Notes

- This function should be called only when DFSDM filter instance is in idle state or if regular conversion is ongoing.

HAL_DFSDM_FilterInjectedStart_DMA

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterInjectedStart_DMA (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, int32_t * pData, uint32_t Length)

Function description

This function allows to start injected conversion in DMA mode.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.
- **pData:** The destination buffer address.
- **Length:** The length of data to be transferred from DFSDM filter to memory.

Return values

- **HAL:** status

Notes

- This function should be called only when DFSDM filter instance is in idle state or if regular conversion is ongoing. Please note that data on buffer will contain signed injected conversion value on 24 most significant bits and corresponding channel on 3 least significant bits.

HAL_DFSDM_FilterInjectedMsbStart_DMA

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterInjectedMsbStart_DMA (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, int16_t * pData, uint32_t Length)

Function description

This function allows to start injected conversion in DMA mode and to get only the 16 most significant bits of conversion.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.
- **pData:** The destination buffer address.
- **Length:** The length of data to be transferred from DFSDM filter to memory.

Return values

- **HAL:** status

Notes

- This function should be called only when DFSDM filter instance is in idle state or if regular conversion is ongoing. Please note that data on buffer will contain signed 16 most significant bits of injected conversion.

HAL_DFSDM_FilterInjectedStop

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterInjectedStop (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)

Function description

This function allows to stop injected conversion in polling mode.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.

Return values

- **HAL:** status

Notes

- This function should be called only if injected conversion is ongoing.

HAL_DFSDM_FilterInjectedStop_IT

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterInjectedStop_IT (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)

Function description

This function allows to stop injected conversion in interrupt mode.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.

Return values

- **HAL:** status

Notes

- This function should be called only if injected conversion is ongoing.

HAL_DFSDM_FilterInjectedStop_DMA

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterInjectedStop_DMA (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)

Function description

This function allows to stop injected conversion in DMA mode.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.

Return values

- **HAL:** status

Notes

- This function should be called only if injected conversion is ongoing.

HAL_DFSDM_FilterAwdStart_IT

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterAwdStart_IT (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, const DFSDM_Filter_AwdParamTypeDef * awdParam)

Function description

This function allows to start filter analog watchdog in interrupt mode.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.
- **awdParam:** DFSDM filter analog watchdog parameters.

Return values

- **HAL:** status

HAL_DFSDM_FilterAwdStop_IT

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterAwdStop_IT (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)

Function description

This function allows to stop filter analog watchdog in interrupt mode.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.

Return values

- **HAL:** status

HAL_DFSDM_FilterExdStart

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterExdStart (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, uint32_t Channel)

Function description

This function allows to start extreme detector feature.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.
- **Channel:** Channels where extreme detector is enabled. This parameter can be a values combination of DFSDM Channel Selection.

Return values

- **HAL:** status

HAL_DFSDM_FilterExdStop

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterExdStop (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)

Function description

This function allows to stop extreme detector feature.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.

Return values

- **HAL:** status

HAL_DFSDM_FilterGetRegularValue

Function name

int32_t HAL_DFSDM_FilterGetRegularValue (const DFSDM_Filter_HandleTypeDef * hdfsdm_filter, uint32_t * Channel)

Function description

This function allows to get regular conversion value.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.
- **Channel:** Corresponding channel of regular conversion.

Return values

- **Regular:** conversion value

HAL_DFSDM_FilterGetInjectedValue

Function name

int32_t HAL_DFSDM_FilterGetInjectedValue (const DFSDM_Filter_HandleTypeDef * hdfsdm_filter, uint32_t * Channel)

Function description

This function allows to get injected conversion value.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.
- **Channel:** Corresponding channel of injected conversion.

Return values

- **Injected:** conversion value

HAL_DFSDM_FilterGetExdMaxValue

Function name

int32_t HAL_DFSDM_FilterGetExdMaxValue (const DFSDM_Filter_HandleTypeDef * hdfsdm_filter, uint32_t * Channel)

Function description

This function allows to get extreme detector maximum value.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.
- **Channel:** Corresponding channel.

Return values

- **Extreme:** detector maximum value This value is between Min_Data = -8388608 and Max_Data = 8388607.

HAL_DFSDM_FilterGetExdMinValue

Function name

int32_t HAL_DFSDM_FilterGetExdMinValue (const DFSDM_Filter_HandleTypeDef * hdfsdm_filter, uint32_t * Channel)

Function description

This function allows to get extreme detector minimum value.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.
- **Channel:** Corresponding channel.

Return values

- **Extreme:** detector minimum value This value is between Min_Data = -8388608 and Max_Data = 8388607.

HAL_DFSDM_FilterGetConvTimeValue

Function name

uint32_t HAL_DFSDM_FilterGetConvTimeValue (const DFSDM_Filter_HandleTypeDef * hdfsdm_filter)

Function description

This function allows to get conversion time value.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.

Return values

- **Conversion:** time value

Notes

- To get time in second, this value has to be divided by DFSDM clock frequency.

HAL_DFSDM_IRQHandler

Function name

void HAL_DFSDM_IRQHandler (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)

Function description

This function handles the DFSDM interrupts.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.

Return values

- **None:**

HAL_DFSDM_FilterPollForRegConversion

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterPollForRegConversion (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, uint32_t Timeout)

Function description

This function allows to poll for the end of regular conversion.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.
- **Timeout:** Timeout value in milliseconds.

Return values

- **HAL:** status

Notes

- This function should be called only if regular conversion is ongoing.

HAL_DFSDM_FilterPollForInjConversion

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterPollForInjConversion (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, uint32_t Timeout)

Function description

This function allows to poll for the end of injected conversion.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.
- **Timeout:** Timeout value in milliseconds.

Return values

- **HAL:** status

Notes

- This function should be called only if injected conversion is ongoing.

HAL_DFSDM_FilterRegConvCpltCallback

Function name

void HAL_DFSDM_FilterRegConvCpltCallback (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)

Function description

Regular conversion complete callback.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.

Return values

- **None:**

Notes

- In interrupt mode, user has to read conversion value in this function using HAL_DFSDM_FilterGetRegularValue.

HAL_DFSDM_FilterRegConvHalfCpltCallback

Function name

void HAL_DFSDM_FilterRegConvHalfCpltCallback (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)

Function description

Half regular conversion complete callback.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.

Return values

- **None:**

HAL_DFSDM_FilterInjConvCpltCallback

Function name

void HAL_DFSDM_FilterInjConvCpltCallback (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)

Function description

Injected conversion complete callback.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.

Return values

- **None:**

Notes

- In interrupt mode, user has to read conversion value in this function using HAL_DFSDM_FilterGetInjectedValue.

HAL_DFSDM_FilterInjConvHalfCpltCallback

Function name

void HAL_DFSDM_FilterInjConvHalfCpltCallback (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)

Function description

Half injected conversion complete callback.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.

Return values

- **None:**

HAL_DFSDM_FilterAwdCallback

Function name

void HAL_DFSDM_FilterAwdCallback (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, uint32_t Channel, uint32_t Threshold)

Function description

Filter analog watchdog callback.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.
- **Channel:** Corresponding channel.
- **Threshold:** Low or high threshold has been reached.

Return values

- **None:**

HAL_DFSDM_FilterErrorCallback

Function name

void HAL_DFSDM_FilterErrorCallback (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)

Function description

Error callback.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.

Return values

- **None:**

HAL_DFSDM_FilterGetState

Function name

HAL_DFSDM_Filter_StateTypeDef HAL_DFSDM_FilterGetState (const DFSDM_Filter_HandleTypeDef * hdfsdm_filter)

Function description

This function allows to get the current DFSDM filter handle state.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.

Return values

- **DFSDM:** filter state.

HAL_DFSDM_FilterGetError

Function name

uint32_t HAL_DFSDM_FilterGetError (const DFSDM_Filter_HandleTypeDef * hdfsdm_filter)

Function description

This function allows to get the current DFSDM filter error.

Parameters

- **hdfsdm_filter:** DFSDM filter handle.

Return values

- **DFSDM:** filter error code.

21.3 DFSDM Firmware driver defines

The following section lists the various define and macros of the module.

21.3.1 DFSDM

DFSDM

DFSDM analog watchdog threshold

DFSDM_AWD_HIGH_THRESHOLD

Analog watchdog high threshold

DFSDM_AWD_LOW_THRESHOLD

Analog watchdog low threshold

DFSDM break signals

DFSDM_NO_BREAK_SIGNAL

No break signal

DFSDM_BREAK_SIGNAL_0

Break signal 0

DFSDM_BREAK_SIGNAL_1

Break signal 1

DFSDM_BREAK_SIGNAL_2

Break signal 2

DFSDM_BREAK_SIGNAL_3

Break signal 3

DFSDM channel analog watchdog filter order

DFSDM_CHANNEL_FASTSINC_ORDER

FastSinc filter type

DFSDM_CHANNEL_SINC1_ORDER

Sinc 1 filter type

DFSDM_CHANNEL_SINC2_ORDER

Sinc 2 filter type

DFSDM_CHANNEL_SINC3_ORDER

Sinc 3 filter type

DFSDM channel input data packing

DFSDM_CHANNEL_STANDARD_MODE

Standard data packing mode

DFSDM_CHANNEL_INTERLEAVED_MODE

Interleaved data packing mode

DFSDM_CHANNEL_DUAL_MODE

Dual data packing mode

DFSDM channel input multiplexer

DFSDM_CHANNEL_EXTERNAL_INPUTS

Data are taken from external inputs

DFSDM_CHANNEL_ADC_OUTPUT

Data are taken from ADC output

DFSDM_CHANNEL_INTERNAL_REGISTER

Data are taken from internal register

DFSDM channel input pins

DFSDM_CHANNEL_SAME_CHANNEL_PINS

Input from pins on same channel

DFSDM_CHANNEL_FOLLOWING_CHANNEL_PINS

Input from pins on following channel

DFSDM channel output clock selection

DFSDM_CHANNEL_OUTPUT_CLOCK_SYSTEM

Source for output clock is system clock

DFSDM_CHANNEL_OUTPUT_CLOCK_AUDIO

Source for output clock is audio clock

DFSDM Channel Selection

DFSDM_CHANNEL_0

DFSDM_CHANNEL_1

DFSDM_CHANNEL_2

DFSDM_CHANNEL_3

DFSDM_CHANNEL_4

DFSDM_CHANNEL_5

DFSDM_CHANNEL_6

DFSDM_CHANNEL_7

DFSDM channel serial interface type

DFSDM_CHANNEL_SPI_RISING

SPI with rising edge

DFSDM_CHANNEL_SPI_FALLING

SPI with falling edge

DFSDM_CHANNEL_MANCHESTER_RISING

Manchester with rising edge

DFSDM_CHANNEL_MANCHESTER_FALLING

Manchester with falling edge

DFSDM channel SPI clock selection

DFSDM_CHANNEL_SPI_CLOCK_EXTERNAL

External SPI clock

DFSDM_CHANNEL_SPI_CLOCK_INTERNAL

Internal SPI clock

DFSDM_CHANNEL_SPI_CLOCK_INTERNAL_DIV2_FALLING

Internal SPI clock divided by 2, falling edge

DFSDM_CHANNEL_SPI_CLOCK_INTERNAL_DIV2_RISING

Internal SPI clock divided by 2, rising edge

DFSDM Continuous Mode

DFSDM_CONTINUOUS_CONV_OFF

Conversion are not continuous

DFSDM_CONTINUOUS_CONV_ON

Conversion are continuous

DFSDM Exported Macros

__HAL_DFSDM_CHANNEL_RESET_HANDLE_STATE

Description:

- Reset DFSDM channel handle state.

Parameters:

- __HANDLE__: DFSDM channel handle.

Return value:

- None

__HAL_DFSDM_FILTER_RESET_HANDLE_STATE

Description:

- Reset DFSDM filter handle state.

Parameters:

- __HANDLE__: DFSDM filter handle.

Return value:

- None

DFSDM filter analog watchdog data source

DFSDM_FILTER_AWD_FILTER_DATA

From digital filter

DFSDM_FILTER_AWD_CHANNEL_DATA

From analog watchdog channel

DFSDM filter error code

DFSDM_FILTER_ERROR_NONE

No error

DFSDM_FILTER_ERROR_REGULAR_OVERRUN

Overrun occurs during regular conversion

DFSDM_FILTER_ERROR_INJECTED_OVERRUN

Overrun occurs during injected conversion

DFSDM_FILTER_ERROR_DMA

DMA error occurs

DFSDM_FILTER_ERROR_INVALID_CALLBACK

Invalid callback error occurs

DFSDM filter external trigger

DFSDM_FILTER_EXT_TRIG_TIM1_TRGO

For all DFSDM filters

DFSDM_FILTER_EXT_TRIG_TIM1_TRGO2

For all DFSDM filters

DFSDM_FILTER_EXT_TRIG_TIM8_TRGO

For all DFSDM filters

DFSDM_FILTER_EXT_TRIG_TIM8_TRGO2

For all DFSDM filters

DFSDM_FILTER_EXT_TRIG_TIM3_TRGO

For all DFSDM filters

DFSDM_FILTER_EXT_TRIG_TIM4_TRGO

For all DFSDM filters

DFSDM_FILTER_EXT_TRIG_TIM16_OC1

For all DFSDM filters

DFSDM_FILTER_EXT_TRIG_TIM6_TRGO

For all DFSDM filters

DFSDM_FILTER_EXT_TRIG_TIM7_TRGO

For all DFSDM filters

DFSDM_FILTER_EXT_TRIG_HRTIM1_ADCTRG1

DFSDM_FILTER_EXT_TRIG_HRTIM1_ADCTRG3

DFSDM_FILTER_EXT_TRIG_EXTI11

For all DFSDM filters

DFSDM_FILTER_EXT_TRIG_EXTI15

For all DFSDM filters

DFSDM_FILTER_EXT_TRIG_LPTIM1_OUT

For all DFSDM filters

DFSDM_FILTER_EXT_TRIG_LPTIM2_OUT

For all DFSDM filters

DFSDM_FILTER_EXT_TRIG_LPTIM3_OUT

For all DFSDM filters

DFSDM filter external trigger edge

DFSDM_FILTER_EXT_TRIG_RISING_EDGE

External rising edge

DFSDM_FILTER_EXT_TRIG_FALLING_EDGE

External falling edge

DFSDM_FILTER_EXT_TRIG_BOTH_EDGES

External rising and falling edges

DFSDM filter sinc order

DFSDM_FILTER_FASTSINC_ORDER

FastSinc filter type

DFSDM_FILTER_SINC1_ORDER

Sinc 1 filter type

DFSDM_FILTER_SINC2_ORDER

Sinc 2 filter type

DFSDM_FILTER_SINC3_ORDER

Sinc 3 filter type

DFSDM_FILTER_SINC4_ORDER

Sinc 4 filter type

DFSDM_FILTER_SINC5_ORDER

Sinc 5 filter type

DFSDM filter conversion trigger

DFSDM_FILTER_SW_TRIGGER

Software trigger

DFSDM_FILTER_SYNC_TRIGGER

Synchronous with DFSDM_FLT0

DFSDM_FILTER_EXT_TRIGGER

External trigger (only for injected conversion)

22 HAL DMA2D Generic Driver

22.1 DMA2D Firmware driver registers structures

22.1.1 DMA2D_CLUTCfgTypeDef

DMA2D_CLUTCfgTypeDef is defined in the stm32h7xx_hal_dma2d.h

Data Fields

- ***uint32_t * pCLUT***
- ***uint32_t CLUTColorMode***
- ***uint32_t Size***

Field Documentation

- ***uint32_t* DMA2D_CLUTCfgTypeDef::pCLUT***
  Configures the DMA2D CLUT memory address.
- ***uint32_t DMA2D_CLUTCfgTypeDef::CLUTColorMode***
  Configures the DMA2D CLUT color mode. This parameter can be one value of ***DMA2D_CLUT_CM***.
- ***uint32_t DMA2D_CLUTCfgTypeDef::Size***
  Configures the DMA2D CLUT size. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.

22.1.2 DMA2D_InitTypeDef

DMA2D_InitTypeDef is defined in the stm32h7xx_hal_dma2d.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t ColorMode***
- ***uint32_t OutputOffset***
- ***uint32_t AlphaInverted***
- ***uint32_t RedBlueSwap***
- ***uint32_t BytesSwap***
- ***uint32_t LineOffsetMode***

Field Documentation

- ***uint32_t DMA2D_InitTypeDef::Mode***
  Configures the DMA2D transfer mode. This parameter can be one value of ***DMA2D_Mode***.
- ***uint32_t DMA2D_InitTypeDef::ColorMode***
  Configures the color format of the output image. This parameter can be one value of ***DMA2D_Output_Color_Mode***.
- ***uint32_t DMA2D_InitTypeDef::OutputOffset***
  Specifies the Offset value. This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x3FFF.
- ***uint32_t DMA2D_InitTypeDef::AlphaInverted***
  Select regular or inverted alpha value for the output pixel format converter. This parameter can be one value of ***DMA2D_Alpha_Inverted***.
- ***uint32_t DMA2D_InitTypeDef::RedBlueSwap***
  Select regular mode (RGB or ARGB) or swap mode (BGR or ABGR) for the output pixel format converter. This parameter can be one value of ***DMA2D_RB_Swap***.
- ***uint32_t DMA2D_InitTypeDef::BytesSwap***
  Select byte regular mode or bytes swap mode (two by two). This parameter can be one value of ***DMA2D_Bytes_Swap***.

- ***uint32_t DMA2D_InitTypeDef::LineOffsetMode***

  Configures how is expressed the line offset for the foreground, background and output. This parameter can be one value of ***DMA2D_Line_Offset_Mode***.

22.1.3 DMA2D_LayerCfgTypeDef

DMA2D_LayerCfgTypeDef is defined in the stm32h7xx_hal_dma2d.h

Data Fields

- ***uint32_t InputOffset***
- ***uint32_t InputColorMode***
- ***uint32_t AlphaMode***
- ***uint32_t InputAlpha***
- ***uint32_t AlphaInverted***
- ***uint32_t RedBlueSwap***
- ***uint32_t ChromaSubSampling***

Field Documentation

- ***uint32_t DMA2D_LayerCfgTypeDef::InputOffset***

  Configures the DMA2D foreground or background offset. This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x3FFF.
- ***uint32_t DMA2D_LayerCfgTypeDef::InputColorMode***

  Configures the DMA2D foreground or background color mode. This parameter can be one value of ***DMA2D_Input_Color_Mode***.
- ***uint32_t DMA2D_LayerCfgTypeDef::AlphaMode***

  Configures the DMA2D foreground or background alpha mode. This parameter can be one value of ***DMA2D_Alpha_Mode***.
- ***uint32_t DMA2D_LayerCfgTypeDef::InputAlpha***

  Specifies the DMA2D foreground or background alpha value and color value in case of A8 or A4 color mode. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF except for the color modes detailed below.

  Note:

  – In case of A8 or A4 color mode (ARGB), this parameter must be a number between Min_Data = 0x00000000 and Max_Data = 0xFFFFFFFF where
    - InputAlpha[24:31] is the alpha value ALPHA[0:7]
    - InputAlpha[16:23] is the red value RED[0:7]
    - InputAlpha[8:15] is the green value GREEN[0:7]
    - InputAlpha[0:7] is the blue value BLUE[0:7].
- ***uint32_t DMA2D_LayerCfgTypeDef::AlphaInverted***

  Select regular or inverted alpha value. This parameter can be one value of ***DMA2D_Alpha_Inverted***.
- ***uint32_t DMA2D_LayerCfgTypeDef::RedBlueSwap***

  Select regular mode (RGB or ARGB) or swap mode (BGR or ABGR). This parameter can be one value of ***DMA2D_RB_Swap***.
- ***uint32_t DMA2D_LayerCfgTypeDef::ChromaSubSampling***

  Configure the chroma sub-sampling mode for the YCbCr color mode This parameter can be one value of ***DMA2D_Chroma_Sub_Sampling***

22.1.4 __DMA2D_HandleTypeDef

__DMA2D_HandleTypeDef is defined in the stm32h7xx_hal_dma2d.h

Data Fields

- ***DMA2D_TypeDef * Instance***
- ***DMA2D_InitTypeDef Init***
- ***void(* XferCpltCallback***
- ***void(* XferErrorCallback***

- ***void(* LineEventCallback***
- ***void(* CLUTLoadingCpltCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***DMA2D_LayerCfgTypeDef LayerCfg***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_DMA2D_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***DMA2D_TypeDef* __DMA2D_HandleTypeDef::Instance***
  DMA2D register base address.
- ***DMA2D_InitTypeDef __DMA2D_HandleTypeDef::Init***
  DMA2D communication parameters.
- ***void(* __DMA2D_HandleTypeDef::XferCpltCallback)(struct __DMA2D_HandleTypeDef *hdma2d)***
  DMA2D transfer complete callback.
- ***void(* __DMA2D_HandleTypeDef::XferErrorCallback)(struct __DMA2D_HandleTypeDef *hdma2d)***
  DMA2D transfer error callback.
- ***void(* __DMA2D_HandleTypeDef::LineEventCallback)(struct __DMA2D_HandleTypeDef *hdma2d)***
  DMA2D line event callback.
- ***void(* __DMA2D_HandleTypeDef::CLUTLoadingCpltCallback)(struct __DMA2D_HandleTypeDef *hdma2d)***
  DMA2D CLUT loading completion callback
- ***void(* __DMA2D_HandleTypeDef::MspInitCallback)(struct __DMA2D_HandleTypeDef *hdma2d)***
  DMA2D Msp Init callback.
- ***void(* __DMA2D_HandleTypeDef::MspDeInitCallback)(struct __DMA2D_HandleTypeDef *hdma2d)***
  DMA2D Msp DeInit callback.
- ***DMA2D_LayerCfgTypeDef __DMA2D_HandleTypeDef::LayerCfg[MAX_DMA2D_LAYER]***
  DMA2D Layers parameters
- ***HAL_LockTypeDef __DMA2D_HandleTypeDef::Lock***
  DMA2D lock.
- ***__IO HAL_DMA2D_StateTypeDef __DMA2D_HandleTypeDef::State***
  DMA2D transfer state.
- ***__IO uint32_t __DMA2D_HandleTypeDef::ErrorCode***
  DMA2D error code.

22.2 DMA2D Firmware driver API description

The following section lists the various functions of the DMA2D library.

22.2.1 How to use this driver

1. Program the required configuration through the following parameters: the transfer mode, the output color mode and the output offset using HAL_DMA2D_Init() function.
2. Program the required configuration through the following parameters: the input color mode, the input color, the input alpha value, the alpha mode, the red/blue swap mode, the inverted alpha mode and the input offset using HAL_DMA2D_ConfigLayer() function for foreground or/and background layer.

Polling mode IO operation

1. Configure pdata parameter (explained hereafter), destination and data length and enable the transfer using HAL_DMA2D_Start().
2. Wait for end of transfer using HAL_DMA2D_PollForTransfer(), at this stage user can specify the value of timeout according to his end application.

Interrupt mode IO operation

1. Configure pdata parameter, destination and data length and enable the transfer using HAL_DMA2D_Start_IT().
2. Use HAL_DMA2D_IRQHandler() called under DMA2D_IRQHandler() interrupt subroutine.
3. At the end of data transfer HAL_DMA2D_IRQHandler() function is executed and user can add his own function by customization of function pointer XferCpltCallback (member of DMA2D handle structure).
4. In case of error, the HAL_DMA2D_IRQHandler() function calls the callback XferErrorCallback.

Note: In Register-to-Memory transfer mode, pdata parameter is the register color, in Memory-to-memory or Memory-to-Memory with pixel format conversion pdata is the source address.

Note: Configure the foreground source address, the background source address, the destination and data length then Enable the transfer using HAL_DMA2D_BlendingStart() in polling mode and HAL_DMA2D_BlendingStart_IT() in interrupt mode.

Note: HAL_DMA2D_BlendingStart() and HAL_DMA2D_BlendingStart_IT() functions are used if the memory to memory with blending transfer mode is selected.

5. Optionally, configure and enable the CLUT using HAL_DMA2D_CLUTLoad() in polling mode or HAL_DMA2D_CLUTLoad_IT() in interrupt mode.
6. Optionally, configure the line watermark in using the API HAL_DMA2D_ProgramLineEvent().
7. Optionally, configure the dead time value in the AHB clock cycle inserted between two consecutive accesses on the AHB master port in using the API HAL_DMA2D_ConfigDeadTime() and enable/disable the functionality with the APIs HAL_DMA2D_EnableDeadTime() or HAL_DMA2D_DisableDeadTime().
8. The transfer can be suspended, resumed and aborted using the following functions: HAL_DMA2D_Suspend(), HAL_DMA2D_Resume(), HAL_DMA2D_Abort().
9. The CLUT loading can be suspended, resumed and aborted using the following functions: HAL_DMA2D_CLUTLoading_Suspend(), HAL_DMA2D_CLUTLoading_Resume(), HAL_DMA2D_CLUTLoading_Abort().
10. To control the DMA2D state, use the following function: HAL_DMA2D_GetState().
11. To read the DMA2D error code, use the following function: HAL_DMA2D_GetError().

DMA2D HAL driver macros list

Below the list of most used macros in DMA2D HAL driver :

- __HAL_DMA2D_ENABLE: Enable the DMA2D peripheral.
- __HAL_DMA2D_GET_FLAG: Get the DMA2D pending flags.
- __HAL_DMA2D_CLEAR_FLAG: Clear the DMA2D pending flags.
- __HAL_DMA2D_ENABLE_IT: Enable the specified DMA2D interrupts.
- __HAL_DMA2D_DISABLE_IT: Disable the specified DMA2D interrupts.
- __HAL_DMA2D_GET_IT_SOURCE: Check whether the specified DMA2D interrupt is enabled or not.

Callback registration

1. The compilation define USE_HAL_DMA2D_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use function @ref HAL_DMA2D_RegisterCallback() to register a user callback.
2. Function @ref HAL_DMA2D_RegisterCallback() allows to register following callbacks: (+) XferCpltCallback : callback for transfer complete. (+) XferErrorCallback : callback for transfer error. (+) LineEventCallback : callback for line event. (+) CLUTLoadingCpltCallback : callback for CLUT loading completion. (+) MspInitCallback : DMA2D MspInit. (+) MspDeInitCallback : DMA2D MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.
3. Use function @ref HAL_DMA2D_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. @ref HAL_DMA2D_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks: (+) XferCpltCallback : callback for transfer complete. (+) XferErrorCallback : callback for transfer error. (+) LineEventCallback : callback for line event. (+) CLUTLoadingCpltCallback : callback for CLUT loading completion. (+) MspInitCallback : DMA2D MspInit. (+) MspDeInitCallback : DMA2D MspDeInit.

4. By default, after the @ref HAL_DMA2D_Init and if the state is HAL_DMA2D_STATE_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions: examples @ref HAL_DMA2D_LineEventCallback(), @ref HAL_DMA2D_CLUTLoadingCpltCallback() Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the @ref HAL_DMA2D_Init and @ref HAL_DMA2D_DeInit only when these callbacks are null (not registered beforehand) If not, MspInit or MspDeInit are not null, the @ref HAL_DMA2D_Init and @ref HAL_DMA2D_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand). Exception as well for Transfer Completion and Transfer Error callbacks that are not defined as weak (surcharged) functions. They must be defined by the user to be resorted to. Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using @ref HAL_DMA2D_RegisterCallback before calling @ref HAL_DMA2D_DeInit or @ref HAL_DMA2D_Init function. When The compilation define USE_HAL_DMA2D_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

(#) The compilation define USE_HAL_DMA2D_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use function @ref HAL_DMA2D_RegisterCallback() to register a user callback. (#) Function @ref HAL_DMA2D_RegisterCallback() allows to register following callbacks:

- XferCpltCallback : callback for transfer complete.
- XferErrorCallback : callback for transfer error.
- LineEventCallback : callback for line event.
- CLUTLoadingCpltCallback : callback for CLUT loading completion.
- MspInitCallback : DMA2D MspInit.
- MspDeInitCallback : DMA2D MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. (#) Use function @ref HAL_DMA2D_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. @ref HAL_DMA2D_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:
- XferCpltCallback : callback for transfer complete.
- XferErrorCallback : callback for transfer error.
- LineEventCallback : callback for line event.
- CLUTLoadingCpltCallback : callback for CLUT loading completion.
- MspInitCallback : DMA2D MspInit.
- MspDeInitCallback : DMA2D MspDeInit. (#) By default, after the @ref HAL_DMA2D_Init and if the state is HAL_DMA2D_STATE_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions: examples @ref HAL_DMA2D_LineEventCallback(), @ref HAL_DMA2D_CLUTLoadingCpltCallback() Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the @ref HAL_DMA2D_Init and @ref HAL_DMA2D_DeInit only when these callbacks are null (not registered beforehand) If not, MspInit or MspDeInit are not null, the @ref HAL_DMA2D_Init and @ref HAL_DMA2D_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand). Exception as well for Transfer Completion and Transfer Error callbacks that are not defined as weak (surcharged) functions. They must be defined by the user to be resorted to. Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using @ref HAL_DMA2D_RegisterCallback before calling @ref HAL_DMA2D_DeInit or @ref HAL_DMA2D_Init function. When The compilation define USE_HAL_DMA2D_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

Note: You can refer to the DMA2D HAL driver header file for more useful macros

22.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the DMA2D
- De-initialize the DMA2D

This section contains the following APIs:

- *HAL_DMA2D_Init()*
- *HAL_DMA2D_DeInit()*
- *HAL_DMA2D_MspInit()*
- *HAL_DMA2D_MspDeInit()*
- *HAL_DMA2D_RegisterCallback()*
- *HAL_DMA2D_UnRegisterCallback()*

22.2.3 IO operation functions

This section provides functions allowing to:

- Configure the pdata, destination address and data size then start the DMA2D transfer.
- Configure the source for foreground and background, destination address and data size then start a MultiBuffer DMA2D transfer.
- Configure the pdata, destination address and data size then start the DMA2D transfer with interrupt.
- Configure the source for foreground and background, destination address and data size then start a MultiBuffer DMA2D transfer with interrupt.
- Abort DMA2D transfer.
- Suspend DMA2D transfer.
- Resume DMA2D transfer.
- Enable CLUT transfer.
- Configure CLUT loading then start transfer in polling mode.
- Configure CLUT loading then start transfer in interrupt mode.
- Abort DMA2D CLUT loading.
- Suspend DMA2D CLUT loading.
- Resume DMA2D CLUT loading.
- Poll for transfer complete.
- handle DMA2D interrupt request.
- Transfer watermark callback.
- CLUT Transfer Complete callback.

This section contains the following APIs:

- *HAL_DMA2D_Start()*
- *HAL_DMA2D_Start_IT()*
- *HAL_DMA2D_BlendingStart()*
- *HAL_DMA2D_BlendingStart_IT()*
- *HAL_DMA2D_Abort()*
- *HAL_DMA2D_Suspend()*
- *HAL_DMA2D_Resume()*
- *HAL_DMA2D_EnableCLUT()*
- *HAL_DMA2D_CLUTStartLoad()*
- *HAL_DMA2D_CLUTStartLoad_IT()*
- *HAL_DMA2D_CLUTLoad()*
- *HAL_DMA2D_CLUTLoad_IT()*
- *HAL_DMA2D_CLUTLoading_Abort()*
- *HAL_DMA2D_CLUTLoading_Suspend()*
- *HAL_DMA2D_CLUTLoading_Resume()*
- *HAL_DMA2D_PollForTransfer()*
- *HAL_DMA2D_IRQHandler()*
- *HAL_DMA2D_LineEventCallback()*
- *HAL_DMA2D_CLUTLoadingCpltCallback()*

22.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure the DMA2D foreground or background layer parameters.
- Configure the DMA2D CLUT transfer.
- Configure the line watermark
- Configure the dead time value.
- Enable or disable the dead time value functionality.

This section contains the following APIs:

- ***HAL_DMA2D_ConfigLayer()***
- ***HAL_DMA2D_ConfigCLUT()***
- ***HAL_DMA2D_ProgramLineEvent()***
- ***HAL_DMA2D_EnableDeadTime()***
- ***HAL_DMA2D_DisableDeadTime()***
- ***HAL_DMA2D_ConfigDeadTime()***

22.2.5 Peripheral State and Errors functions

This subsection provides functions allowing to:

- Get the DMA2D state
- Get the DMA2D error code

This section contains the following APIs:

- ***HAL_DMA2D_GetState()***
- ***HAL_DMA2D_GetError()***

22.2.6 Detailed description of functions

HAL_DMA2D_Init

Function name

HAL_StatusTypeDef HAL_DMA2D_Init (DMA2D_HandleTypeDef * hdma2d)

Function description

Initialize the DMA2D according to the specified parameters in the DMA2D_InitTypeDef and create the associated handle.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **HAL:** status

HAL_DMA2D_DeInit

Function name

HAL_StatusTypeDef HAL_DMA2D_DeInit (DMA2D_HandleTypeDef * hdma2d)

Function description

Deinitializes the DMA2D peripheral registers to their default reset values.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **None:**

HAL_DMA2D_MspInit

Function name

void HAL_DMA2D_MspInit (DMA2D_HandleTypeDef * hdma2d)

Function description

Initializes the DMA2D MSP.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **None:**

HAL_DMA2D_MspDeInit

Function name

void HAL_DMA2D_MspDeInit (DMA2D_HandleTypeDef * hdma2d)

Function description

DeInitializes the DMA2D MSP.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **None:**

HAL_DMA2D_RegisterCallback

Function name

HAL_StatusTypeDef HAL_DMA2D_RegisterCallback (DMA2D_HandleTypeDef * hdma2d, HAL_DMA2D_CallbackIDTypeDef CallbackID, pDMA2D_CallbackTypeDef pCallback)

Function description

Register a User DMA2D Callback To be used instead of the weak (surcharged) predefined callback.

Parameters

- **hdma2d:** DMA2D handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_DMA2D_TRANSFERCOMPLETE_CB_ID DMA2D transfer complete Callback ID
  - HAL_DMA2D_TRANSFERERROR_CB_ID DMA2D transfer error Callback ID
  - HAL_DMA2D_LINEEVENT_CB_ID DMA2D line event Callback ID
  - HAL_DMA2D_CLUTLOADINGCPLT_CB_ID DMA2D CLUT loading completion Callback ID
  - HAL_DMA2D_MSPINIT_CB_ID DMA2D MspInit callback ID
  - HAL_DMA2D_MSPDEINIT_CB_ID DMA2D MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **status:**

Notes

- No weak predefined callbacks are defined for HAL_DMA2D_TRANSFERCOMPLETE_CB_ID or HAL_DMA2D_TRANSFERERROR_CB_ID

HAL_DMA2D_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_DMA2D_UnRegisterCallback (DMA2D_HandleTypeDef * hdma2d, HAL_DMA2D_CallbackIDTypeDef CallbackID)

Function description

Unregister a DMA2D Callback DMA2D Callback is redirected to the weak (surcharged) predefined callback.

Parameters

- **hdma2d:** DMA2D handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_DMA2D_TRANSFERCOMPLETE_CB_ID DMA2D transfer complete Callback ID
  - HAL_DMA2D_TRANSFERERROR_CB_ID DMA2D transfer error Callback ID
  - HAL_DMA2D_LINEEVENT_CB_ID DMA2D line event Callback ID
  - HAL_DMA2D_CLUTLOADINGCPLT_CB_ID DMA2D CLUT loading completion Callback ID
  - HAL_DMA2D_MSPINIT_CB_ID DMA2D MspInit callback ID
  - HAL_DMA2D_MSPDEINIT_CB_ID DMA2D MspDeInit callback ID

Return values

- **status:**

Notes

- No weak predefined callbacks are defined for HAL_DMA2D_TRANSFERCOMPLETE_CB_ID or HAL_DMA2D_TRANSFERERROR_CB_ID

HAL_DMA2D_Start

Function name

HAL_StatusTypeDef HAL_DMA2D_Start (DMA2D_HandleTypeDef * hdma2d, uint32_t pdata, uint32_t DstAddress, uint32_t Width, uint32_t Height)

Function description

Start the DMA2D Transfer.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **pdata:** Configure the source memory Buffer address if Memory-to-Memory or Memory-to-Memory with pixel format conversion mode is selected, or configure the color value if Register-to-Memory mode is selected.
- **DstAddress:** The destination memory Buffer address.
- **Width:** The width of data to be transferred from source to destination (expressed in number of pixels per line).
- **Height:** The height of data to be transferred from source to destination (expressed in number of lines).

Return values

- **HAL:** status

HAL_DMA2D_BlendingStart

Function name

HAL_StatusTypeDef HAL_DMA2D_BlendingStart (DMA2D_HandleTypeDef * hdma2d, uint32_t SrcAddress1, uint32_t SrcAddress2, uint32_t DstAddress, uint32_t Width, uint32_t Height)

Function description

Start the multi-source DMA2D Transfer.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **SrcAddress1:** The source memory Buffer address for the foreground layer.
- **SrcAddress2:** The source memory Buffer address for the background layer.
- **DstAddress:** The destination memory Buffer address.
- **Width:** The width of data to be transferred from source to destination (expressed in number of pixels per line).
- **Height:** The height of data to be transferred from source to destination (expressed in number of lines).

Return values

- **HAL:** status

HAL_DMA2D_Start_IT

Function name

HAL_StatusTypeDef HAL_DMA2D_Start_IT (DMA2D_HandleTypeDef * hdma2d, uint32_t pdata, uint32_t DstAddress, uint32_t Width, uint32_t Height)

Function description

Start the DMA2D Transfer with interrupt enabled.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **pdata:** Configure the source memory Buffer address if the Memory-to-Memory or Memory-to-Memory with pixel format conversion mode is selected, or configure the color value if Register-to-Memory mode is selected.
- **DstAddress:** The destination memory Buffer address.
- **Width:** The width of data to be transferred from source to destination (expressed in number of pixels per line).
- **Height:** The height of data to be transferred from source to destination (expressed in number of lines).

Return values

- **HAL:** status

HAL_DMA2D_BlendingStart_IT

Function name

HAL_StatusTypeDef HAL_DMA2D_BlendingStart_IT (DMA2D_HandleTypeDef * hdma2d, uint32_t SrcAddress1, uint32_t SrcAddress2, uint32_t DstAddress, uint32_t Width, uint32_t Height)

Function description

Start the multi-source DMA2D Transfer with interrupt enabled.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **SrcAddress1:** The source memory Buffer address for the foreground layer.
- **SrcAddress2:** The source memory Buffer address for the background layer.
- **DstAddress:** The destination memory Buffer address.
- **Width:** The width of data to be transferred from source to destination (expressed in number of pixels per line).
- **Height:** The height of data to be transferred from source to destination (expressed in number of lines).

Return values

- **HAL:** status

HAL_DMA2D_Suspend

Function name

HAL_StatusTypeDef HAL_DMA2D_Suspend (DMA2D_HandleTypeDef * hdma2d)

Function description

Suspend the DMA2D Transfer.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **HAL:** status

HAL_DMA2D_Resume

Function name

HAL_StatusTypeDef HAL_DMA2D_Resume (DMA2D_HandleTypeDef * hdma2d)

Function description

Resume the DMA2D Transfer.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **HAL:** status

HAL_DMA2D_Abort

Function name

HAL_StatusTypeDef HAL_DMA2D_Abort (DMA2D_HandleTypeDef * hdma2d)

Function description

Abort the DMA2D Transfer.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **HAL:** status

HAL_DMA2D_EnableCLUT

Function name

HAL_StatusTypeDef HAL_DMA2D_EnableCLUT (DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)

Function description

Enable the DMA2D CLUT Transfer.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: DMA2D_BACKGROUND_LAYER(0) / DMA2D_FOREGROUND_LAYER(1)

Return values

- **HAL:** status

HAL_DMA2D_CLUTStartLoad

Function name

HAL_StatusTypeDef HAL_DMA2D_CLUTStartLoad (DMA2D_HandleTypeDef * hdma2d, DMA2D_CLUTCfgTypeDef * CLUTCfg, uint32_t LayerIdx)

Function description

Start DMA2D CLUT Loading.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **CLUTCfg:** Pointer to a DMA2D_CLUTCfgTypeDef structure that contains the configuration information for the color look up table.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: DMA2D_BACKGROUND_LAYER(0) / DMA2D_FOREGROUND_LAYER(1)

Return values

- **HAL:** status

HAL_DMA2D_CLUTStartLoad_IT

Function name

HAL_StatusTypeDef HAL_DMA2D_CLUTStartLoad_IT (DMA2D_HandleTypeDef * hdma2d, DMA2D_CLUTCfgTypeDef * CLUTCfg, uint32_t LayerIdx)

Function description

Start DMA2D CLUT Loading with interrupt enabled.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **CLUTCfg:** Pointer to a DMA2D_CLUTCfgTypeDef structure that contains the configuration information for the color look up table.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: DMA2D_BACKGROUND_LAYER(0) / DMA2D_FOREGROUND_LAYER(1)

Return values

- **HAL:** status

HAL_DMA2D_CLUTLoad

Function name

HAL_StatusTypeDef HAL_DMA2D_CLUTLoad (DMA2D_HandleTypeDef * hdma2d, DMA2D_CLUTCfgTypeDef CLUTCfg, uint32_t LayerIdx)

Function description

Start DMA2D CLUT Loading.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **CLUTCfg:** Pointer to a DMA2D_CLUTCfgTypeDef structure that contains the configuration information for the color look up table.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: DMA2D_BACKGROUND_LAYER(0) / DMA2D_FOREGROUND_LAYER(1)

Return values

- **HAL:** status

Notes

- API obsolete and maintained for compatibility with legacy. User is invited to resort to HAL_DMA2D_CLUTStartLoad() instead to benefit from code compactness, code size and improved heap usage.

HAL_DMA2D_CLUTLoad_IT

Function name

HAL_StatusTypeDef HAL_DMA2D_CLUTLoad_IT (DMA2D_HandleTypeDef * hdma2d, DMA2D_CLUTCfgTypeDef CLUTCfg, uint32_t LayerIdx)

Function description

Start DMA2D CLUT Loading with interrupt enabled.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **CLUTCfg:** Pointer to a DMA2D_CLUTCfgTypeDef structure that contains the configuration information for the color look up table.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: DMA2D_BACKGROUND_LAYER(0) / DMA2D_FOREGROUND_LAYER(1)

Return values

- **HAL:** status

Notes

- API obsolete and maintained for compatibility with legacy. User is invited to resort to HAL_DMA2D_CLUTStartLoad_IT() instead to benefit from code compactness, code size and improved heap usage.

HAL_DMA2D_CLUTLoading_Abort

Function name

HAL_StatusTypeDef HAL_DMA2D_CLUTLoading_Abort (DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)

Function description

Abort the DMA2D CLUT loading.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: DMA2D_BACKGROUND_LAYER(0) / DMA2D_FOREGROUND_LAYER(1)

Return values

- **HAL:** status

HAL_DMA2D_CLUTLoading_Suspend

Function name

HAL_StatusTypeDef HAL_DMA2D_CLUTLoading_Suspend (DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)

Function description

Suspend the DMA2D CLUT loading.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: DMA2D_BACKGROUND_LAYER(0) / DMA2D_FOREGROUND_LAYER(1)

Return values

- **HAL:** status

HAL_DMA2D_CLUTLoading_Resume

Function name

HAL_StatusTypeDef HAL_DMA2D_CLUTLoading_Resume (DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)

Function description

Resume the DMA2D CLUT loading.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: DMA2D_BACKGROUND_LAYER(0) / DMA2D_FOREGROUND_LAYER(1)

Return values

- **HAL:** status

HAL_DMA2D_PollForTransfer

Function name

HAL_StatusTypeDef HAL_DMA2D_PollForTransfer (DMA2D_HandleTypeDef * hdma2d, uint32_t Timeout)

Function description

Polling for transfer complete or CLUT loading.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_DMA2D_IRQHandler

Function name

void HAL_DMA2D_IRQHandler (DMA2D_HandleTypeDef * hdma2d)

Function description

Handle DMA2D interrupt request.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **HAL:** status

HAL_DMA2D_LineEventCallback

Function name

void HAL_DMA2D_LineEventCallback (DMA2D_HandleTypeDef * hdma2d)

Function description

Transfer watermark callback.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **None:**

HAL_DMA2D_CLUTLoadingCpltCallback

Function name

void HAL_DMA2D_CLUTLoadingCpltCallback (DMA2D_HandleTypeDef * hdma2d)

Function description

CLUT Transfer Complete callback.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **None:**

HAL_DMA2D_ConfigLayer

Function name

HAL_StatusTypeDef HAL_DMA2D_ConfigLayer (DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)

Function description

Configure the DMA2D Layer according to the specified parameters in the DMA2D_HandleTypeDef.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: DMA2D_BACKGROUND_LAYER(0) / DMA2D_FOREGROUND_LAYER(1)

Return values

- **HAL:** status

HAL_DMA2D_ConfigCLUT

Function name

HAL_StatusTypeDef HAL_DMA2D_ConfigCLUT (DMA2D_HandleTypeDef * hdma2d, DMA2D_CLUTCfgTypeDef CLUTCfg, uint32_t LayerIdx)

Function description

Configure the DMA2D CLUT Transfer.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **CLUTCfg:** Pointer to a DMA2D_CLUTCfgTypeDef structure that contains the configuration information for the color look up table.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: DMA2D_BACKGROUND_LAYER(0) / DMA2D_FOREGROUND_LAYER(1)

Return values

- **HAL:** status

Notes

- API obsolete and maintained for compatibility with legacy. User is invited to resort to HAL_DMA2D_CLUTStartLoad() instead to benefit from code compactness, code size and improved heap usage.

HAL_DMA2D_ProgramLineEvent

Function name

HAL_StatusTypeDef HAL_DMA2D_ProgramLineEvent (DMA2D_HandleTypeDef * hdma2d, uint32_t Line)

Function description

Configure the line watermark.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **Line:** Line Watermark configuration (maximum 16-bit long value expected).

Return values

- **HAL:** status

Notes

- HAL_DMA2D_ProgramLineEvent() API enables the transfer watermark interrupt.
- The transfer watermark interrupt is disabled once it has occurred.

HAL_DMA2D_EnableDeadTime

Function name

HAL_StatusTypeDef HAL_DMA2D_EnableDeadTime (DMA2D_HandleTypeDef * hdma2d)

Function description

Enable DMA2D dead time feature.

Parameters

- **hdma2d:** DMA2D handle.

Return values

- **HAL:** status

HAL_DMA2D_DisableDeadTime

Function name

HAL_StatusTypeDef HAL_DMA2D_DisableDeadTime (DMA2D_HandleTypeDef * hdma2d)

Function description

Disable DMA2D dead time feature.

Parameters

- **hdma2d:** DMA2D handle.

Return values

- **HAL:** status

HAL_DMA2D_ConfigDeadTime

Function name

HAL_StatusTypeDef HAL_DMA2D_ConfigDeadTime (DMA2D_HandleTypeDef * hdma2d, uint8_t DeadTime)

Function description

Configure dead time.

Parameters

- **hdma2d:** DMA2D handle.
- **DeadTime:** dead time value.

Return values

- **HAL:** status

Notes

- The dead time value represents the guaranteed minimum number of cycles between two consecutive transactions on the AHB bus.

HAL_DMA2D_GetState

Function name

HAL_DMA2D_StateTypeDef HAL_DMA2D_GetState (DMA2D_HandleTypeDef * hdma2d)

Function description

Return the DMA2D state.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **HAL:** state

HAL_DMA2D_GetError

Function name

uint32_t HAL_DMA2D_GetError (DMA2D_HandleTypeDef * hdma2d)

Function description

Return the DMA2D error code.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for DMA2D.

Return values

- **DMA2D:** Error Code

22.3 DMA2D Firmware driver defines

The following section lists the various define and macros of the module.

22.3.1 DMA2D

DMA2D

DMA2D Alpha Inversion

DMA2D_REGULAR_ALPHA

No modification of the alpha channel value

DMA2D_INVERTED_ALPHA

Invert the alpha channel value

DMA2D Alpha Mode

DMA2D_NO_MODIF_ALPHA

No modification of the alpha channel value

DMA2D_REPLACE_ALPHA

Replace original alpha channel value by programmed alpha value

DMA2D_COMBINE_ALPHA

Replace original alpha channel value by programmed alpha value with original alpha channel value

DMA2D Bytes Swap

DMA2D_BYTES_REGULAR

Bytes in regular order in output FIFO

DMA2D_BYTES_SWAP

Bytes are swapped two by two in output FIFO

DMA2D Chroma Sub Sampling

DMA2D_NO_CSS

No chroma sub-sampling 4:4:4

DMA2D_CSS_422

chroma sub-sampling 4:2:2

DMA2D_CSS_420

chroma sub-sampling 4:2:0

DMA2D CLUT Color Mode

DMA2D_CCM_ARGB8888

ARGB8888 DMA2D CLUT color mode

DMA2D_CCM_RGB888

RGB888 DMA2D CLUT color mode

DMA2D CLUT Size

DMA2D_CLUT_SIZE

DMA2D maximum CLUT size

DMA2D Color Value

DMA2D_COLOR_VALUE

Color value mask

DMA2D Error Code

HAL_DMA2D_ERROR_NONE

No error

HAL_DMA2D_ERROR_TE

Transfer error

HAL_DMA2D_ERROR_CE

Configuration error

HAL_DMA2D_ERROR_CAE

CLUT access error

HAL_DMA2D_ERROR_TIMEOUT

Timeout error

HAL_DMA2D_ERROR_INVALID_CALLBACK

Invalid callback error

DMA2D Exported Macros

__HAL_DMA2D_RESET_HANDLE_STATE

Description:

- Reset DMA2D handle state.

Parameters:

- __HANDLE__: specifies the DMA2D handle.

Return value:

- None

__HAL_DMA2D_ENABLE

Description:

- Enable the DMA2D.

Parameters:

- __HANDLE__: DMA2D handle

Return value:

- None.

__HAL_DMA2D_GET_FLAG

Description:

- Get the DMA2D pending flags.

Parameters:

- __HANDLE__: DMA2D handle
- __FLAG__: flag to check. This parameter can be any combination of the following values:
  - DMA2D_FLAG_CE: Configuration error flag
  - DMA2D_FLAG_CTC: CLUT transfer complete flag
  - DMA2D_FLAG_CAE: CLUT access error flag
  - DMA2D_FLAG_TW: Transfer Watermark flag
  - DMA2D_FLAG_TC: Transfer complete flag
  - DMA2D_FLAG_TE: Transfer error flag

Return value:

- The: state of FLAG.

__HAL_DMA2D_CLEAR_FLAG

Description:

- Clear the DMA2D pending flags.

Parameters:

- __HANDLE__: DMA2D handle
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - DMA2D_FLAG_CE: Configuration error flag
  - DMA2D_FLAG_CTC: CLUT transfer complete flag
  - DMA2D_FLAG_CAE: CLUT access error flag
  - DMA2D_FLAG_TW: Transfer Watermark flag
  - DMA2D_FLAG_TC: Transfer complete flag
  - DMA2D_FLAG_TE: Transfer error flag

Return value:

- None

__HAL_DMA2D_ENABLE_IT

Description:

- Enable the specified DMA2D interrupts.

Parameters:

- __HANDLE__: DMA2D handle
- __INTERRUPT__: specifies the DMA2D interrupt sources to be enabled. This parameter can be any combination of the following values:
  - DMA2D_IT_CE: Configuration error interrupt mask
  - DMA2D_IT_CTC: CLUT transfer complete interrupt mask
  - DMA2D_IT_CAE: CLUT access error interrupt mask
  - DMA2D_IT_TW: Transfer Watermark interrupt mask
  - DMA2D_IT_TC: Transfer complete interrupt mask
  - DMA2D_IT_TE: Transfer error interrupt mask

Return value:

- None

__HAL_DMA2D_DISABLE_IT

Description:

- Disable the specified DMA2D interrupts.

Parameters:

- __HANDLE__: DMA2D handle
- __INTERRUPT__: specifies the DMA2D interrupt sources to be disabled. This parameter can be any combination of the following values:
  - DMA2D_IT_CE: Configuration error interrupt mask
  - DMA2D_IT_CTC: CLUT transfer complete interrupt mask
  - DMA2D_IT_CAE: CLUT access error interrupt mask
  - DMA2D_IT_TW: Transfer Watermark interrupt mask
  - DMA2D_IT_TC: Transfer complete interrupt mask
  - DMA2D_IT_TE: Transfer error interrupt mask

Return value:

- None

__HAL_DMA2D_GET_IT_SOURCE

Description:

- Check whether the specified DMA2D interrupt source is enabled or not.

Parameters:

- __HANDLE__: DMA2D handle
- __INTERRUPT__: specifies the DMA2D interrupt source to check. This parameter can be one of the following values:
  - DMA2D_IT_CE: Configuration error interrupt mask
  - DMA2D_IT_CTC: CLUT transfer complete interrupt mask
  - DMA2D_IT_CAE: CLUT access error interrupt mask
  - DMA2D_IT_TW: Transfer Watermark interrupt mask
  - DMA2D_IT_TC: Transfer complete interrupt mask
  - DMA2D_IT_TE: Transfer error interrupt mask

Return value:

- The: state of INTERRUPT source.

DMA2D Exported Types

MAX_DMA2D_LAYER

DMA2D maximum number of layers

DMA2D Flags

DMA2D_FLAG_CE

Configuration Error Interrupt Flag

DMA2D_FLAG_CTC

CLUT Transfer Complete Interrupt Flag

DMA2D_FLAG_CAE

CLUT Access Error Interrupt Flag

DMA2D_FLAG_TW

Transfer Watermark Interrupt Flag

DMA2D_FLAG_TC

Transfer Complete Interrupt Flag

DMA2D_FLAG_TE

Transfer Error Interrupt Flag

DMA2D Input Color Mode

DMA2D_INPUT_ARGB8888

ARGB8888 color mode

DMA2D_INPUT_RGB888

RGB888 color mode

DMA2D_INPUT_RGB565

RGB565 color mode

DMA2D_INPUT_ARGB1555

ARGB1555 color mode

DMA2D_INPUT_ARGB4444

ARGB4444 color mode

DMA2D_INPUT_L8

L8 color mode

DMA2D_INPUT_AL44

AL44 color mode

DMA2D_INPUT_AL88

AL88 color mode

DMA2D_INPUT_L4

L4 color mode

DMA2D_INPUT_A8

A8 color mode

DMA2D_INPUT_A4

A4 color mode

DMA2D_INPUT_YCBCR

YCbCr color mode

DMA2D Interrupts

DMA2D_IT_CE

Configuration Error Interrupt

DMA2D_IT_CTC

CLUT Transfer Complete Interrupt

DMA2D_IT_CAE

CLUT Access Error Interrupt

DMA2D_IT_TW

Transfer Watermark Interrupt

DMA2D_IT_TC

Transfer Complete Interrupt

DMA2D_IT_TE

Transfer Error Interrupt

DMA2D Layers

DMA2D_BACKGROUND_LAYER

DMA2D Background Layer (layer 0)

DMA2D_FOREGROUND_LAYER

DMA2D Foreground Layer (layer 1)

DMA2D Line Offset Mode

DMA2D_LOM_PIXELS

Line offsets expressed in pixels

DMA2D_LOM_BYTES

Line offsets expressed in bytes

DMA2D Maximum Line Watermark

DMA2D_LINE_WATERMARK_MAX

DMA2D maximum line watermark

DMA2D Maximum Number of Layers

DMA2D_MAX_LAYER

DMA2D maximum number of layers

DMA2D Mode

DMA2D_M2M

DMA2D memory to memory transfer mode

DMA2D_M2M_PFC

DMA2D memory to memory with pixel format conversion transfer mode

DMA2D_M2M_BLEND

DMA2D memory to memory with blending transfer mode

DMA2D_R2M

DMA2D register to memory transfer mode

DMA2D_M2M_BLEND_FG

DMA2D memory to memory with blending transfer mode and fixed color FG

DMA2D_M2M_BLEND_BG

DMA2D memory to memory with blending transfer mode and fixed color BG

DMA2D Offset

DMA2D_OFFSET

maximum Line Offset

DMA2D Output Color Mode

DMA2D_OUTPUT_ARGB8888

ARGB8888 DMA2D color mode

DMA2D_OUTPUT_RGB888

RGB888 DMA2D color mode

DMA2D_OUTPUT_RGB565

RGB565 DMA2D color mode

DMA2D_OUTPUT_ARGB1555

ARGB1555 DMA2D color mode

DMA2D_OUTPUT_ARGB4444

ARGB4444 DMA2D color mode

DMA2D Red and Blue Swap

DMA2D_RB_REGULAR

Select regular mode (RGB or ARGB)

DMA2D_RB_SWAP

Select swap mode (BGR or ABGR)

DMA2D Size

DMA2D_PIXEL

DMA2D maximum number of pixels per line

DMA2D_LINE

DMA2D maximum number of lines

DMA2D Time Out

DMA2D_TIMEOUT_ABORT

1s

DMA2D_TIMEOUT_SUSPEND

1s

23 HAL DMA Generic Driver

23.1 DMA Firmware driver registers structures

23.1.1 DMA_InitTypeDef

DMA_InitTypeDef is defined in the stm32h7xx_hal_dma.h

Data Fields

- ***uint32_t Request***
- ***uint32_t Direction***
- ***uint32_t PeriphInc***
- ***uint32_t MemInc***
- ***uint32_t PeriphDataAlignment***
- ***uint32_t MemDataAlignment***
- ***uint32_t Mode***
- ***uint32_t Priority***
- ***uint32_t FIFOMode***
- ***uint32_t FIFOThreshold***
- ***uint32_t MemBurst***
- ***uint32_t PeriphBurst***

Field Documentation

- ***uint32_t DMA_InitTypeDef::Request***

  Specifies the request selected for the specified stream. This parameter can be a value of ***DMA_Request_selection***
- ***uint32_t DMA_InitTypeDef::Direction***

  Specifies if the data will be transferred from memory to peripheral, from memory to memory or from peripheral to memory. This parameter can be a value of ***DMA_Data_transfer_direction***
- ***uint32_t DMA_InitTypeDef::PeriphInc***

  Specifies whether the Peripheral address register should be incremented or not. This parameter can be a value of ***DMA_Peripheral_incremented_mode***
- ***uint32_t DMA_InitTypeDef::MemInc***

  Specifies whether the memory address register should be incremented or not. This parameter can be a value of ***DMA_Memory_incremented_mode***
- ***uint32_t DMA_InitTypeDef::PeriphDataAlignment***

  Specifies the Peripheral data width. This parameter can be a value of ***DMA_Peripheral_data_size***
- ***uint32_t DMA_InitTypeDef::MemDataAlignment***

  Specifies the Memory data width. This parameter can be a value of ***DMA_Memory_data_size***
- ***uint32_t DMA_InitTypeDef::Mode***

  Specifies the operation mode of the DMAy Streamx. This parameter can be a value of ***DMA_mode***

  Note:

  – The circular buffer mode cannot be used if the memory-to-memory data transfer is configured on the selected Stream
- ***uint32_t DMA_InitTypeDef::Priority***

  Specifies the software priority for the DMAy Streamx. This parameter can be a value of ***DMA_Priority_level***

- ***uint32_t DMA_InitTypeDef::FIFOMode***

  Specifies if the FIFO mode or Direct mode will be used for the specified stream. This parameter can be a value of ***DMA_FIFO_direct_mode***

  Note:

  – The Direct mode (FIFO mode disabled) cannot be used if the memory-to-memory data transfer is configured on the selected stream
- ***uint32_t DMA_InitTypeDef::FIFOThreshold***

  Specifies the FIFO threshold level. This parameter can be a value of ***DMA_FIFO_threshold_level***
- ***uint32_t DMA_InitTypeDef::MemBurst***

  Specifies the Burst transfer configuration for the memory transfers. It specifies the amount of data to be transferred in a single non interruptible transaction. This parameter can be a value of ***DMA_Memory_burst***

  Note:

  – The burst mode is possible only if the address Increment mode is enabled.
- ***uint32_t DMA_InitTypeDef::PeriphBurst***

  Specifies the Burst transfer configuration for the peripheral transfers. It specifies the amount of data to be transferred in a single non interruptible transaction. This parameter can be a value of ***DMA_Peripheral_burst***

  Note:

  – The burst mode is possible only if the address Increment mode is enabled.

23.1.2 __DMA_HandleTypeDef

__DMA_HandleTypeDef is defined in the stm32h7xx_hal_dma.h

Data Fields

- ***void * Instance***
- ***DMA_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_DMA_StateTypeDef State***
- ***void * Parent***
- ***void(* XferCpltCallback***
- ***void(* XferHalfCpltCallback***
- ***void(* XferM1CpltCallback***
- ***void(* XferM1HalfCpltCallback***
- ***void(* XferErrorCallback***
- ***void(* XferAbortCallback***
- ***__IO uint32_t ErrorCode***
- ***uint32_t StreamBaseAddress***
- ***uint32_t StreamIndex***
- ***DMAMUX_Channel_TypeDef * DMAmuxChannel***
- ***DMAMUX_ChannelStatus_TypeDef * DMAmuxChannelStatus***
- ***uint32_t DMAmuxChannelStatusMask***
- ***DMAMUX_RequestGen_TypeDef * DMAmuxRequestGen***
- ***DMAMUX_RequestGenStatus_TypeDef * DMAmuxRequestGenStatus***
- ***uint32_t DMAmuxRequestGenStatusMask***

Field Documentation

- ***void* __DMA_HandleTypeDef::Instance***

  Register base address
- ***DMA_InitTypeDef __DMA_HandleTypeDef::Init***

  DMA communication parameters
- ***HAL_LockTypeDef __DMA_HandleTypeDef::Lock***

  DMA locking object

- ***__IO HAL_DMA_StateTypeDef __DMA_HandleTypeDef::State***
  DMA transfer state
- ***void* __DMA_HandleTypeDef::Parent***
  Parent object state
- ***void(* __DMA_HandleTypeDef::XferCpltCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA transfer complete callback
- ***void(* __DMA_HandleTypeDef::XferHalfCpltCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA Half transfer complete callback
- ***void(* __DMA_HandleTypeDef::XferM1CpltCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA transfer complete Memory1 callback
- ***void(* __DMA_HandleTypeDef::XferM1HalfCpltCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA transfer Half complete Memory1 callback
- ***void(* __DMA_HandleTypeDef::XferErrorCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA transfer error callback
- ***void(* __DMA_HandleTypeDef::XferAbortCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA transfer Abort callback
- ***__IO uint32_t __DMA_HandleTypeDef::ErrorCode***
  DMA Error code
- ***uint32_t __DMA_HandleTypeDef::StreamBaseAddress***
  DMA Stream Base Address
- ***uint32_t __DMA_HandleTypeDef::StreamIndex***
  DMA Stream Index
- ***DMAMUX_Channel_TypeDef* __DMA_HandleTypeDef::DMAmuxChannel***
  DMAMUX Channel Base Address
- ***DMAMUX_ChannelStatus_TypeDef* __DMA_HandleTypeDef::DMAmuxChannelStatus***
  DMAMUX Channels Status Base Address
- ***uint32_t __DMA_HandleTypeDef::DMAmuxChannelStatusMask***
  DMAMUX Channel Status Mask
- ***DMAMUX_RequestGen_TypeDef* __DMA_HandleTypeDef::DMAmuxRequestGen***
  DMAMUX request generator Base Address
- ***DMAMUX_RequestGenStatus_TypeDef* __DMA_HandleTypeDef::DMAmuxRequestGenStatus***
  DMAMUX request generator Status Address
- ***uint32_t __DMA_HandleTypeDef::DMAmuxRequestGenStatusMask***
  DMAMUX request generator Status mask

23.2 DMA Firmware driver API description

The following section lists the various functions of the DMA library.

23.2.1 How to use this driver

1. Enable and configure the peripheral to be connected to the DMA Stream (except for internal SRAM/ FLASH memories: no initialization is necessary) please refer to Reference manual for connection between peripherals and DMA requests .
2. For a given Stream, program the required configuration through the following parameters: Transfer Direction, Source and Destination data formats, Circular, Normal or peripheral flow control mode, Stream Priority level, Source and Destination Increment mode, FIFO mode and its Threshold (if needed), Burst mode for Source and/or Destination (if needed) using HAL_DMA_Init() function.

Polling mode IO operation

- Use HAL_DMA_Start() to start DMA transfer after the configuration of Source address and destination address and the Length of data to be transferred

- Use HAL_DMA_PollForTransfer() to poll for the end of current transfer, in this case a fixed Timeout can be configured by User depending from his application.

Interrupt mode IO operation

- Configure the DMA interrupt priority using HAL_NVIC_SetPriority()
- Enable the DMA IRQ handler using HAL_NVIC_EnableIRQ()
- Use HAL_DMA_Start_IT() to start DMA transfer after the configuration of Source address and destination address and the Length of data to be transferred. In this case the DMA interrupt is configured
- Use HAL_DMA_IRQHandler() called under DMA_IRQHandler() Interrupt subroutine
- At the end of data transfer HAL_DMA_IRQHandler() function is executed and user can add his own function by customization of function pointer XferCpltCallback and XferErrorCallback (i.e a member of DMA handle structure).

1. Use HAL_DMA_GetState() function to return the DMA state and HAL_DMA_GetError() in case of error detection.
2. Use HAL_DMA_Abort() function to abort the current transfer

Note: In Memory-to-Memory transfer mode, Circular mode is not allowed.

Note: The FIFO is used mainly to reduce bus usage and to allow data packing/unpacking: it is possible to set different Data Sizes for the Peripheral and the Memory (ie. you can set Half-Word data size for the peripheral to access its data register and set Word data size for the Memory to gain in access time. Each two half words will be packed and written in a single access to a Word in the Memory).

Note: When FIFO is disabled, it is not allowed to configure different Data Sizes for Source and Destination. In this case the Peripheral Data Size will be applied to both Source and Destination.

DMA HAL driver macros list

Below the list of most used macros in DMA HAL driver.

- __HAL_DMA_ENABLE: Enable the specified DMA Stream.
- __HAL_DMA_DISABLE: Disable the specified DMA Stream.
- __HAL_DMA_GET_FS: Return the current DMA Stream FIFO filled level.
- __HAL_DMA_ENABLE_IT: Enable the specified DMA Stream interrupts.
- __HAL_DMA_DISABLE_IT: Disable the specified DMA Stream interrupts.
- __HAL_DMA_GET_IT_SOURCE: Check whether the specified DMA Stream interrupt has occurred or not.

Note: You can refer to the DMA HAL driver header file for more useful macros.

23.2.2 Initialization and de-initialization functions

This section provides functions allowing to initialize the DMA Stream source and destination incrementation and data sizes, transfer direction, circular/normal mode selection, memory-to-memory mode selection and Stream priority value.

The HAL_DMA_Init() function follows the DMA configuration procedures as described in reference manual. The HAL_DMA_DeInit function allows to deinitialize the DMA stream.

This section contains the following APIs:

- ***HAL_DMA_Init()***
- ***HAL_DMA_DeInit()***

23.2.3 IO operation functions

This section provides functions allowing to:

- Configure the source, destination address and data length and Start DMA transfer
- Configure the source, destination address and data length and Start DMA transfer with interrupt
- Register and Unregister DMA callbacks
- Abort DMA transfer
- Poll for transfer complete

- Handle DMA interrupt request

This section contains the following APIs:

- *HAL_DMA_Start()*
- *HAL_DMA_Start_IT()*
- *HAL_DMA_Abort()*
- *HAL_DMA_Abort_IT()*
- *HAL_DMA_PollForTransfer()*
- *HAL_DMA_IRQHandler()*
- *HAL_DMA_RegisterCallback()*
- *HAL_DMA_UnRegisterCallback()*

23.2.4 State and Errors functions

This subsection provides functions allowing to

- Check the DMA state
- Get error code

This section contains the following APIs:

- *HAL_DMA_GetState()*
- *HAL_DMA_GetError()*

23.2.5 Detailed description of functions

HAL_DMA_Init

Function name

HAL_StatusTypeDef HAL_DMA_Init (DMA_HandleTypeDef * hdma)

Function description

Initialize the DMA according to the specified parameters in the DMA_InitTypeDef and create the associated handle.

Parameters

- **hdma:** Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.

Return values

- **HAL:** status

HAL_DMA_DeInit

Function name

HAL_StatusTypeDef HAL_DMA_DeInit (DMA_HandleTypeDef * hdma)

Function description

DeInitializes the DMA peripheral.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.

Return values

- **HAL:** status

HAL_DMA_Start

Function name

HAL_StatusTypeDef HAL_DMA_Start (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength)

Function description

Starts the DMA Transfer.

Parameters

- **hdma:** : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
- **SrcAddress:** The source memory Buffer address
- **DstAddress:** The destination memory Buffer address
- **DataLength:** The length of data to be transferred from source to destination

Return values

- **HAL:** status

HAL_DMA_Start_IT

Function name

HAL_StatusTypeDef HAL_DMA_Start_IT (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength)

Function description

Start the DMA Transfer with interrupt enabled.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
- **SrcAddress:** The source memory Buffer address
- **DstAddress:** The destination memory Buffer address
- **DataLength:** The length of data to be transferred from source to destination

Return values

- **HAL:** status

HAL_DMA_Abort

Function name

HAL_StatusTypeDef HAL_DMA_Abort (DMA_HandleTypeDef * hdma)

Function description

Aborts the DMA Transfer.

Parameters

- **hdma:** : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.

Return values

- **HAL:** status

Notes

- After disabling a DMA Stream, a check for wait until the DMA Stream is effectively disabled is added. If a Stream is disabled while a data transfer is ongoing, the current data will be transferred and the Stream will be effectively disabled only after the transfer of this single data is finished.

HAL_DMA_Abort_IT

Function name

HAL_StatusTypeDef HAL_DMA_Abort_IT (DMA_HandleTypeDef * hdma)

Function description

Aborts the DMA Transfer in Interrupt mode.

Parameters

- **hdma:** : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.

Return values

- **HAL:** status

HAL_DMA_PollForTransfer

Function name

HAL_StatusTypeDef HAL_DMA_PollForTransfer (DMA_HandleTypeDef * hdma, HAL_DMA_LevelCompleteTypeDef CompleteLevel, uint32_t Timeout)

Function description

Polling for transfer complete.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
- **CompleteLevel:** Specifies the DMA level complete.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- The polling mode is kept in this version for legacy. it is recommended to use the IT model instead. This model could be used for debug purpose.
- The HAL_DMA_PollForTransfer API cannot be used in circular and double buffering mode (automatic circular mode).

HAL_DMA_IRQHandler

Function name

void HAL_DMA_IRQHandler (DMA_HandleTypeDef * hdma)

Function description

Handles DMA interrupt request.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.

Return values

- **None:**

HAL_DMA_RegisterCallback

Function name

HAL_StatusTypeDef HAL_DMA_RegisterCallback (DMA_HandleTypeDef * hdma, HAL_DMA_CallbackIDTypeDef CallbackID, void(*)(DMA_HandleTypeDef *_hdma) pCallback)

Function description

Register callbacks.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
- **CallbackID:** User Callback identifier a DMA_HandleTypeDef structure as parameter.
- **pCallback:** pointer to private callback function which has pointer to a DMA_HandleTypeDef structure as parameter.

Return values

- **HAL:** status

HAL_DMA_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_DMA_UnRegisterCallback (DMA_HandleTypeDef * hdma, HAL_DMA_CallbackIDTypeDef CallbackID)

Function description

UnRegister callbacks.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
- **CallbackID:** User Callback identifier a HAL_DMA_CallbackIDTypeDef ENUM as parameter.

Return values

- **HAL:** status

HAL_DMA_GetState

Function name

HAL_DMA_StateTypeDef HAL_DMA_GetState (DMA_HandleTypeDef * hdma)

Function description

Returns the DMA state.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.

Return values

- **HAL:** state

HAL_DMA_GetError

Function name

uint32_t HAL_DMA_GetError (DMA_HandleTypeDef * hdma)

Function description

Return the DMA error code.

Parameters

- **hdma:** : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.

Return values

- **DMA:** Error Code

23.3 DMA Firmware driver defines

The following section lists the various define and macros of the module.

23.3.1 DMA

DMA

DMA Data transfer direction

DMA_PERIPH_TO_MEMORY

Peripheral to memory direction

DMA_MEMORY_TO_PERIPH

Memory to peripheral direction

DMA_MEMORY_TO_MEMORY

Memory to memory direction

DMA Error Code

HAL_DMA_ERROR_NONE

No error

HAL_DMA_ERROR_TE

Transfer error

HAL_DMA_ERROR_FE

FIFO error

HAL_DMA_ERROR_DME

Direct Mode error

HAL_DMA_ERROR_TIMEOUT

Timeout error

HAL_DMA_ERROR_PARAM

Parameter error

HAL_DMA_ERROR_NO_XFER

Abort requested with no Xfer ongoing

HAL_DMA_ERROR_NOT_SUPPORTED

Not supported mode

HAL_DMA_ERROR_SYNC

DMAMUX sync overrun error

HAL_DMA_ERROR_REQGEN

DMAMUX request generator overrun error

HAL_DMA_ERROR_BUSY

DMA Busy error

DMA Exported Macros

__HAL_DMA_RESET_HANDLE_STATE

Description:

- Reset DMA handle state.

Parameters:

- __HANDLE__: specifies the DMA handle.

Return value:

- None

__HAL_DMA_GET_FS

Description:

- Return the current DMA Stream FIFO filled level.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: FIFO filling state.
  - DMA_FIFOStatus_Less1QuarterFull: when FIFO is less than 1 quarter-full and not empty.
  - DMA_FIFOStatus_1QuarterFull: if more than 1 quarter-full.
  - DMA_FIFOStatus_HalfFull: if more than 1 half-full.
  - DMA_FIFOStatus_3QuartersFull: if more than 3 quarters-full.
  - DMA_FIFOStatus_Empty: when FIFO is empty
  - DMA_FIFOStatus_Full: when FIFO is full

__HAL_DMA_ENABLE

Description:

- Enable the specified DMA Stream.

Parameters:

- __HANDLE__: DMA handle

Return value:

- None

__HAL_DMA_DISABLE

Description:

- Disable the specified DMA Stream.

Parameters:

- __HANDLE__: DMA handle

Return value:

- None

__HAL_DMA_GET_TC_FLAG_INDEX

Description:

- Return the current DMA Stream transfer complete flag.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: specified transfer complete flag index.

__HAL_DMA_GET_HT_FLAG_INDEX

Description:

- Return the current DMA Stream half transfer complete flag.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: specified half transfer complete flag index.

__HAL_DMA_GET_TE_FLAG_INDEX

Description:

- Return the current DMA Stream transfer error flag.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: specified transfer error flag index.

__HAL_DMA_GET_FE_FLAG_INDEX

Description:

- Return the current DMA Stream FIFO error flag.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: specified FIFO error flag index.

__HAL_DMA_GET_DME_FLAG_INDEX

Description:

- Return the current DMA Stream direct mode error flag.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: specified direct mode error flag index.

__HAL_BDMA_GET_GI_FLAG_INDEX

Description:

- Returns the current BDMA Channel Global interrupt flag.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: specified transfer error flag index.

__HAL_DMA_GET_FLAG

Description:

- Get the DMA Stream pending flags.

Parameters:

- __HANDLE__: DMA handle
- __FLAG__: Get the specified flag. This parameter can be any combination of the following values:
  - DMA_FLAG_TCIFx: Transfer complete flag.
  - DMA_FLAG_HTIFx: Half transfer complete flag.
  - DMA_FLAG_TEIFx: Transfer error flag.
  - DMA_FLAG_DMEIFx: Direct mode error flag.
  - DMA_FLAG_FEIFx: FIFO error flag. Where x can be 0_4, 1_5, 2_6 or 3_7 to select the DMA Stream flag.

Return value:

- The: state of FLAG (SET or RESET).

__HAL_DMA_CLEAR_FLAG

Description:

- Clear the DMA Stream pending flags.

Parameters:

- __HANDLE__: DMA handle
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - DMA_FLAG_TCIFx: Transfer complete flag.
  - DMA_FLAG_HTIFx: Half transfer complete flag.
  - DMA_FLAG_TEIFx: Transfer error flag.
  - DMA_FLAG_DMEIFx: Direct mode error flag.
  - DMA_FLAG_FEIFx: FIFO error flag. Where x can be 0_4, 1_5, 2_6 or 3_7 to select the DMA Stream flag.

Return value:

- None

DMA_TO_BDMA_IT

__HAL_BDMA_CHANNEL_ENABLE_IT

__HAL_DMA_STREAM_ENABLE_IT

__HAL_DMA_ENABLE_IT

Description:

- Enable the specified DMA Stream interrupts.

Parameters:

- __HANDLE__: DMA handle
- __INTERRUPT__: specifies the DMA interrupt sources to be enabled or disabled. This parameter can be one of the following values:
  - DMA_IT_TC: Transfer complete interrupt mask.
  - DMA_IT_HT: Half transfer complete interrupt mask.
  - DMA_IT_TE: Transfer error interrupt mask.
  - DMA_IT_FE: FIFO error interrupt mask.
  - DMA_IT_DME: Direct mode error interrupt.

Return value:

- None

__HAL_BDMA_CHANNEL_DISABLE_IT

__HAL_DMA_STREAM_DISABLE_IT

__HAL_DMA_DISABLE_IT

Description:

- Disable the specified DMA Stream interrupts.

Parameters:

- __HANDLE__: DMA handle
- __INTERRUPT__: specifies the DMA interrupt sources to be enabled or disabled. This parameter can be one of the following values:
  - DMA_IT_TC: Transfer complete interrupt mask.
  - DMA_IT_HT: Half transfer complete interrupt mask.
  - DMA_IT_TE: Transfer error interrupt mask.
  - DMA_IT_FE: FIFO error interrupt mask.
  - DMA_IT_DME: Direct mode error interrupt.

Return value:

- None

__HAL_BDMA_CHANNEL_GET_IT_SOURCE

__HAL_DMA_STREAM_GET_IT_SOURCE

__HAL_DMA_GET_IT_SOURCE

Description:

- Check whether the specified DMA Stream interrupt is enabled or not.

Parameters:

- __HANDLE__: DMA handle
- __INTERRUPT__: specifies the DMA interrupt source to check. This parameter can be one of the following values:
  - DMA_IT_TC: Transfer complete interrupt mask.
  - DMA_IT_HT: Half transfer complete interrupt mask.
  - DMA_IT_TE: Transfer error interrupt mask.
  - DMA_IT_FE: FIFO error interrupt mask.
  - DMA_IT_DME: Direct mode error interrupt.

Return value:

- The: state of DMA_IT.

__HAL_DMA_SET_COUNTER

Description:

- Writes the number of data units to be transferred on the DMA Stream.

Parameters:

- __HANDLE__: DMA handle
- __COUNTER__: Number of data units to be transferred (from 0 to 65535) Number of data items depends only on the Peripheral data format.

Return value:

- The: number of remaining data units in the current DMAy Streamx transfer.

Notes:

- If Peripheral data format is Bytes: number of data units is equal to total number of bytes to be transferred.If Peripheral data format is Half-Word: number of data units is equal to total number of bytes to be transferred / 2.If Peripheral data format is Word: number of data units is equal to total number of bytes to be transferred / 4.

__HAL_DMA_GET_COUNTER

Description:

- Returns the number of remaining data units in the current DMAy Streamx transfer.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: number of remaining data units in the current DMA Stream transfer.

DMA FIFO direct mode

DMA_FIFOMODE_DISABLE

FIFO mode disable

DMA_FIFOMODE_ENABLE

FIFO mode enable

DMA FIFO threshold level

DMA_FIFO_THRESHOLD_1QUARTERFULL

FIFO threshold 1 quart full configuration

DMA_FIFO_THRESHOLD_HALFFULL

FIFO threshold half full configuration

DMA_FIFO_THRESHOLD_3QUARTERSFULL

FIFO threshold 3 quarts full configuration

DMA_FIFO_THRESHOLD_FULL

FIFO threshold full configuration

DMA flag definitions

DMA_FLAG_FEIF0_4

DMA_FLAG_DMEIF0_4

DMA_FLAG_TEIF0_4

DMA_FLAG_HTIF0_4

DMA_FLAG_TCIF0_4

DMA_FLAG_FEIF1_5

DMA_FLAG_DMEIF1_5

DMA_FLAG_TEIF1_5

DMA_FLAG_HTIF1_5

DMA_FLAG_TCIF1_5

DMA_FLAG_FEIF2_6

DMA_FLAG_DMEIF2_6

DMA_FLAG_TEIF2_6

DMA_FLAG_HTIF2_6

DMA_FLAG_TCIF2_6

DMA_FLAG_FEIF3_7

DMA_FLAG_DMEIF3_7

DMA_FLAG_TEIF3_7

DMA_FLAG_HTIF3_7

DMA_FLAG_TCIF3_7

TIM DMA Handle Index

TIM_DMA_ID_UPDATE

Index of the DMA handle used for Update DMA requests

TIM_DMA_ID_CC1

Index of the DMA handle used for Capture/Compare 1 DMA requests

TIM_DMA_ID_CC2

Index of the DMA handle used for Capture/Compare 2 DMA requests

TIM_DMA_ID_CC3

Index of the DMA handle used for Capture/Compare 3 DMA requests

TIM_DMA_ID_CC4

Index of the DMA handle used for Capture/Compare 4 DMA requests

TIM_DMA_ID_COMMUTATION

Index of the DMA handle used for Commutation DMA requests

TIM_DMA_ID_TRIGGER

Index of the DMA handle used for Trigger DMA requests

DMA interrupt enable definitions

DMA_IT_TC

DMA_IT_HT

DMA_IT_TE

DMA_IT_DME

DMA_IT_FE

DMA Memory burst

DMA_MBURST_SINGLE

DMA_MBURST_INC4

DMA_MBURST_INC8

DMA_MBURST_INC16

DMA Memory data size

DMA_MDATAALIGN_BYTE

Memory data alignment: Byte

DMA_MDATAALIGN_HALFWORD

Memory data alignment: HalfWord

DMA_MDATAALIGN_WORD

Memory data alignment: Word

DMA Memory incremented mode

DMA_MINC_ENABLE

Memory increment mode enable

DMA_MINC_DISABLE

Memory increment mode disable

DMA mode

DMA_NORMAL

Normal mode

DMA_CIRCULAR

Circular mode

DMA_PFCTRL

Peripheral flow control mode

DMA_DOUBLE_BUFFER_M0

Double buffer mode with first target memory M0

DMA_DOUBLE_BUFFER_M1

Double buffer mode with first target memory M1

DMA Peripheral burst

DMA_PBURST_SINGLE

DMA_PBURST_INC4

DMA_PBURST_INC8

DMA_PBURST_INC16

DMA Peripheral data size

DMA_PDATAALIGN_BYTE

Peripheral data alignment: Byte

DMA_PDATAALIGN_HALFWORD

Peripheral data alignment: HalfWord

DMA_PDATAALIGN_WORD

Peripheral data alignment: Word

DMA Peripheral incremented mode

DMA_PINC_ENABLE

Peripheral increment mode enable

DMA_PINC_DISABLE

Peripheral increment mode disable

DMA Priority level

DMA_PRIORITY_LOW

Priority level: Low

DMA_PRIORITY_MEDIUM

Priority level: Medium

DMA_PRIORITY_HIGH

Priority level: High

DMA_PRIORITY_VERY_HIGH

Priority level: Very High

DMA Request selection

DMA_REQUEST_MEM2MEM

memory to memory transfer

DMA_REQUEST_GENERATOR0

DMAMUX1 request generator 0

DMA_REQUEST_GENERATOR1

DMAMUX1 request generator 1

DMA_REQUEST_GENERATOR2

DMAMUX1 request generator 2

DMA_REQUEST_GENERATOR3

DMAMUX1 request generator 3

DMA_REQUEST_GENERATOR4

DMAMUX1 request generator 4

DMA_REQUEST_GENERATOR5

DMAMUX1 request generator 5

DMA_REQUEST_GENERATOR6

DMAMUX1 request generator 6

DMA_REQUEST_GENERATOR7

DMAMUX1 request generator 7

DMA_REQUEST_ADC1

DMAMUX1 ADC1 request

DMA_REQUEST_ADC2

DMAMUX1 ADC2 request

DMA_REQUEST_TIM1_CH1

DMAMUX1 TIM1 CH1 request

DMA_REQUEST_TIM1_CH2

DMAMUX1 TIM1 CH2 request

DMA_REQUEST_TIM1_CH3

DMAMUX1 TIM1 CH3 request

DMA_REQUEST_TIM1_CH4

DMAMUX1 TIM1 CH4 request

DMA_REQUEST_TIM1_UP

DMAMUX1 TIM1 UP request

DMA_REQUEST_TIM1_TRIG

DMAMUX1 TIM1 TRIG request

DMA_REQUEST_TIM1_COM

DMAMUX1 TIM1 COM request

DMA_REQUEST_TIM2_CH1

DMAMUX1 TIM2 CH1 request

DMA_REQUEST_TIM2_CH2

DMAMUX1 TIM2 CH2 request

DMA_REQUEST_TIM2_CH3

DMAMUX1 TIM2 CH3 request

DMA_REQUEST_TIM2_CH4

DMAMUX1 TIM2 CH4 request

DMA_REQUEST_TIM2_UP

DMAMUX1 TIM2 UP request

DMA_REQUEST_TIM3_CH1

DMAMUX1 TIM3 CH1 request

DMA_REQUEST_TIM3_CH2

DMAMUX1 TIM3 CH2 request

DMA_REQUEST_TIM3_CH3

DMAMUX1 TIM3 CH3 request

DMA_REQUEST_TIM3_CH4

DMAMUX1 TIM3 CH4 request

DMA_REQUEST_TIM3_UP

DMAMUX1 TIM3 UP request

DMA_REQUEST_TIM3_TRIG

DMAMUX1 TIM3 TRIG request

DMA_REQUEST_TIM4_CH1

DMAMUX1 TIM4 CH1 request

DMA_REQUEST_TIM4_CH2

DMAMUX1 TIM4 CH2 request

DMA_REQUEST_TIM4_CH3

DMAMUX1 TIM4 CH3 request

DMA_REQUEST_TIM4_UP

DMAMUX1 TIM4 UP request

DMA_REQUEST_I2C1_RX

DMAMUX1 I2C1 RX request

DMA_REQUEST_I2C1_TX

DMAMUX1 I2C1 TX request

DMA_REQUEST_I2C2_RX

DMAMUX1 I2C2 RX request

DMA_REQUEST_I2C2_TX

DMAMUX1 I2C2 TX request

DMA_REQUEST_SPI1_RX

DMAMUX1 SPI1 RX request

DMA_REQUEST_SPI1_TX

DMAMUX1 SPI1 TX request

DMA_REQUEST_SPI2_RX

DMAMUX1 SPI2 RX request

DMA_REQUEST_SPI2_TX

DMAMUX1 SPI2 TX request

DMA_REQUEST_USART1_RX

DMAMUX1 USART1 RX request

DMA_REQUEST_USART1_TX

DMAMUX1 USART1 TX request

DMA_REQUEST_USART2_RX

DMAMUX1 USART2 RX request

DMA_REQUEST_USART2_TX

DMAMUX1 USART2 TX request

DMA_REQUEST_USART3_RX

DMAMUX1 USART3 RX request

DMA_REQUEST_USART3_TX

DMAMUX1 USART3 TX request

DMA_REQUEST_TIM8_CH1

DMAMUX1 TIM8 CH1 request

DMA_REQUEST_TIM8_CH2

DMAMUX1 TIM8 CH2 request

DMA_REQUEST_TIM8_CH3

DMAMUX1 TIM8 CH3 request

DMA_REQUEST_TIM8_CH4

DMAMUX1 TIM8 CH4 request

DMA_REQUEST_TIM8_UP

DMAMUX1 TIM8 UP request

DMA_REQUEST_TIM8_TRIG

DMAMUX1 TIM8 TRIG request

DMA_REQUEST_TIM8_COM

DMAMUX1 TIM8 COM request

DMA_REQUEST_TIM5_CH1

DMAMUX1 TIM5 CH1 request

DMA_REQUEST_TIM5_CH2

DMAMUX1 TIM5 CH2 request

DMA_REQUEST_TIM5_CH3

DMAMUX1 TIM5 CH3 request

DMA_REQUEST_TIM5_CH4

DMAMUX1 TIM5 CH4 request

DMA_REQUEST_TIM5_UP

DMAMUX1 TIM5 UP request

DMA_REQUEST_TIM5_TRIG

DMAMUX1 TIM5 TRIG request

DMA_REQUEST_SPI3_RX

DMAMUX1 SPI3 RX request

DMA_REQUEST_SPI3_TX

DMAMUX1 SPI3 TX request

DMA_REQUEST_UART4_RX

DMAMUX1 UART4 RX request

DMA_REQUEST_UART4_TX

DMAMUX1 UART4 TX request

DMA_REQUEST_UART5_RX

DMAMUX1 UART5 RX request

DMA_REQUEST_UART5_TX

DMAMUX1 UART5 TX request

DMA_REQUEST_DAC1_CH1

DMAMUX1 DAC1 Channel 1 request

DMA_REQUEST_DAC1_CH2

DMAMUX1 DAC1 Channel 2 request

DMA_REQUEST_TIM6_UP

DMAMUX1 TIM6 UP request

DMA_REQUEST_TIM7_UP

DMAMUX1 TIM7 UP request

DMA_REQUEST_USART6_RX

DMAMUX1 USART6 RX request

DMA_REQUEST_USART6_TX

DMAMUX1 USART6 TX request

DMA_REQUEST_I2C3_RX

DMAMUX1 I2C3 RX request

DMA_REQUEST_I2C3_TX

DMAMUX1 I2C3 TX request

DMA_REQUEST_DCMI

DMAMUX1 DCMI request

DMA_REQUEST_CRYP_IN

DMAMUX1 CRYP IN request

DMA_REQUEST_CRYP_OUT

DMAMUX1 CRYP OUT request

DMA_REQUEST_HASH_IN

DMAMUX1 HASH IN request

DMA_REQUEST_UART7_RX

DMAMUX1 UART7 RX request

DMA_REQUEST_UART7_TX

DMAMUX1 UART7 TX request

DMA_REQUEST_UART8_RX

DMAMUX1 UART8 RX request

DMA_REQUEST_UART8_TX

DMAMUX1 UART8 TX request

DMA_REQUEST_SPI4_RX

DMAMUX1 SPI4 RX request

DMA_REQUEST_SPI4_TX

DMAMUX1 SPI4 TX request

DMA_REQUEST_SPI5_RX

DMAMUX1 SPI5 RX request

DMA_REQUEST_SPI5_TX

DMAMUX1 SPI5 TX request

DMA_REQUEST_SAI1_A

DMAMUX1 SAI1 A request

DMA_REQUEST_SAI1_B

DMAMUX1 SAI1 B request

DMA_REQUEST_SAI2_A

DMAMUX1 SAI2 A request

DMA_REQUEST_SAI2_B

DMAMUX1 SAI2 B request

DMA_REQUEST_SWPMI_RX

DMAMUX1 SWPMI RX request

DMA_REQUEST_SWPMI_TX

DMAMUX1 SWPMI TX request

DMA_REQUEST_SPDIF_RX_DT

DMAMUX1 SPDIF RXDT request

DMA_REQUEST_SPDIF_RX_CS

DMAMUX1 SPDIF RXCS request

DMA_REQUEST_HRTIM_MASTER

DMAMUX1 HRTIM1 Master request 1

DMA_REQUEST_HRTIM_TIMER_A

DMAMUX1 HRTIM1 Timer A request 2

DMA_REQUEST_HRTIM_TIMER_B

DMAMUX1 HRTIM1 Timer B request 3

DMA_REQUEST_HRTIM_TIMER_C

DMAMUX1 HRTIM1 Timer C request 4

DMA_REQUEST_HRTIM_TIMER_D

DMAMUX1 HRTIM1 Timer D request 5

DMA_REQUEST_HRTIM_TIMER_E

DMAMUX1 HRTIM1 Timer E request 6

DMA_REQUEST_DFSDM1_FLT0

DMAMUX1 DFSDM Filter0 request

DMA_REQUEST_DFSDM1_FLT1

DMAMUX1 DFSDM Filter1 request

DMA_REQUEST_DFSDM1_FLT2

DMAMUX1 DFSDM Filter2 request

DMA_REQUEST_DFSDM1_FLT3

DMAMUX1 DFSDM Filter3 request

DMA_REQUEST_TIM15_CH1

DMAMUX1 TIM15 CH1 request

DMA_REQUEST_TIM15_UP

DMAMUX1 TIM15 UP request

DMA_REQUEST_TIM15_TRIG

DMAMUX1 TIM15 TRIG request

DMA_REQUEST_TIM15_COM

DMAMUX1 TIM15 COM request

DMA_REQUEST_TIM16_CH1

DMAMUX1 TIM16 CH1 request

DMA_REQUEST_TIM16_UP

DMAMUX1 TIM16 UP request

DMA_REQUEST_TIM17_CH1

DMAMUX1 TIM17 CH1 request

DMA_REQUEST_TIM17_UP

DMAMUX1 TIM17 UP request

DMA_REQUEST_SAI3_A

DMAMUX1 SAI3 A request

DMA_REQUEST_SAI3_B

DMAMUX1 SAI3 B request

DMA_REQUEST_ADC3

DMAMUX1 ADC3 request

BDMA_REQUEST_MEM2MEM

memory to memory transfer

BDMA_REQUEST_GENERATOR0

DMAMUX2 request generator 0

BDMA_REQUEST_GENERATOR1

DMAMUX2 request generator 1

BDMA_REQUEST_GENERATOR2

DMAMUX2 request generator 2

BDMA_REQUEST_GENERATOR3

DMAMUX2 request generator 3

BDMA_REQUEST_GENERATOR4

DMAMUX2 request generator 4

BDMA_REQUEST_GENERATOR5

DMAMUX2 request generator 5

BDMA_REQUEST_GENERATOR6

DMAMUX2 request generator 6

BDMA_REQUEST_GENERATOR7

DMAMUX2 request generator 7

BDMA_REQUEST_LPUART1_RX

DMAMUX2 LP_UART1_RX request

BDMA_REQUEST_LPUART1_TX

DMAMUX2 LP_UART1_TX request

BDMA_REQUEST_SPI6_RX

DMAMUX2 SPI6 RX request

BDMA_REQUEST_SPI6_TX

DMAMUX2 SPI6 TX request

BDMA_REQUEST_I2C4_RX

DMAMUX2 I2C4 RX request

BDMA_REQUEST_I2C4_TX

DMAMUX2 I2C4 TX request

BDMA_REQUEST_SAI4_A

DMAMUX2 SAI4 A request

BDMA_REQUEST_SAI4_B

DMAMUX2 SAI4 B request

BDMA_REQUEST_ADC3

DMAMUX2 ADC3 request

24 HAL DMA Extension Driver

24.1 DMAEx Firmware driver registers structures

24.1.1 HAL_DMA_MuxSyncConfigTypeDef

HAL_DMA_MuxSyncConfigTypeDef is defined in the stm32h7xx_hal_dma_ex.h

Data Fields

- ***uint32_t SyncSignalID***
- ***uint32_t SyncPolarity***
- ***FunctionalState SyncEnable***
- ***FunctionalState EventEnable***
- ***uint32_t RequestNumber***

Field Documentation

- ***uint32_t HAL_DMA_MuxSyncConfigTypeDef::SyncSignalID***

  Specifies the synchronization signal gating the DMA request in periodic mode. This parameter can be a value of ***DMAEx_MUX_SyncSignalID_selection***
- ***uint32_t HAL_DMA_MuxSyncConfigTypeDef::SyncPolarity***

  Specifies the polarity of the signal on which the DMA request is synchronized. This parameter can be a value of ***DMAEx_MUX_SyncPolarity_selection***
- ***FunctionalState HAL_DMA_MuxSyncConfigTypeDef::SyncEnable***

  Specifies if the synchronization shall be enabled or disabled This parameter can take the value ENABLE or DISABLE
- ***FunctionalState HAL_DMA_MuxSyncConfigTypeDef::EventEnable***

  Specifies if an event shall be generated once the RequestNumber is reached. This parameter can take the value ENABLE or DISABLE
- ***uint32_t HAL_DMA_MuxSyncConfigTypeDef::RequestNumber***

  Specifies the number of DMA request that will be authorized after a sync event. This parameters can be in the range 1 to 32

24.1.2 HAL_DMA_MuxRequestGeneratorConfigTypeDef

HAL_DMA_MuxRequestGeneratorConfigTypeDef is defined in the stm32h7xx_hal_dma_ex.h

Data Fields

- ***uint32_t SignalID***
- ***uint32_t Polarity***
- ***uint32_t RequestNumber***

Field Documentation

- ***uint32_t HAL_DMA_MuxRequestGeneratorConfigTypeDef::SignalID***

  Specifies the ID of the signal used for DMAMUX request generator This parameter can be a value of ***DMAEx_MUX_SignalGeneratorID_selection***
- ***uint32_t HAL_DMA_MuxRequestGeneratorConfigTypeDef::Polarity***

  Specifies the polarity of the signal on which the request is generated. This parameter can be a value of ***DMAEx_MUX_RequestGeneneratorPolarity_selection***
- ***uint32_t HAL_DMA_MuxRequestGeneratorConfigTypeDef::RequestNumber***

  Specifies the number of DMA request that will be generated after a signal event. This parameters can be in the range 1 to 32

24.2 DMAEx Firmware driver API description

The following section lists the various functions of the DMAEx library.

24.2.1 How to use this driver

The DMA Extension HAL driver can be used as follows:

- Start a multi buffer transfer using the HAL_DMA_MultiBufferStart() function for polling mode or HAL_DMA_MultiBufferStart_IT() for interrupt mode.
- Configure the DMA_MUX Synchronization Block using HAL_DMAEx_ConfigMuxSync function.
- Configure the DMA_MUX Request Generator Block using HAL_DMAEx_ConfigMuxRequestGenerator function. Functions HAL_DMAEx_EnableMuxRequestGenerator and HAL_DMAEx_DisableMuxRequestGenerator can then be used to respectively enable/disable the request generator.
- To handle the DMAMUX Interrupts, the function HAL_DMAEx_MUX_IRQHandler should be called from the DMAMUX IRQ handler i.e DMAMUX1_OVR_IRQHandler or DMAMUX2_OVR_IRQHandler . As only one interrupt line is available for all DMAMUX channels and request generators , HAL_DMA_MUX_IRQHandler should be called with, as parameter, the appropriate DMA handle as many as used DMAs in the user project (exception done if a given DMA is not using the DMAMUX SYNC block neither a request generator)

Note: In Memory-to-Memory transfer mode, Multi (Double) Buffer mode is not allowed.

Note: When Multi (Double) Buffer mode is enabled, the transfer is circular by default.

Note: In Multi (Double) buffer mode, it is possible to update the base address for the AHB memory port on the fly (DMA_SxM0AR or DMA_SxM1AR) when the stream is enabled.

Note: Multi (Double) buffer mode is possible with DMA and BDMA instances.

24.2.2 Extended features functions

This section provides functions allowing to:

- Configure the source, destination address and data length and Start MultiBuffer DMA transfer
- Configure the source, destination address and data length and Start MultiBuffer DMA transfer with interrupt
- Change on the fly the memory0 or memory1 address.
- Configure the DMA_MUX Synchronization Block using HAL_DMAEx_ConfigMuxSync function.
- Configure the DMA_MUX Request Generator Block using HAL_DMAEx_ConfigMuxRequestGenerator function.
- Functions HAL_DMAEx_EnableMuxRequestGenerator and HAL_DMAEx_DisableMuxRequestGenerator can then be used to respectively enable/disable the request generator.
- Handle DMAMUX interrupts using HAL_DMAEx_MUX_IRQHandler : should be called from the DMAMUX IRQ handler i.e DMAMUX1_OVR_IRQHandler or DMAMUX2_OVR_IRQHandler

This section contains the following APIs:

- ***HAL_DMAEx_MultiBufferStart()***
- ***HAL_DMAEx_MultiBufferStart_IT()***
- ***HAL_DMAEx_ChangeMemory()***
- ***HAL_DMAEx_ConfigMuxSync()***
- ***HAL_DMAEx_ConfigMuxRequestGenerator()***
- ***HAL_DMAEx_EnableMuxRequestGenerator()***
- ***HAL_DMAEx_DisableMuxRequestGenerator()***
- ***HAL_DMAEx_MUX_IRQHandler()***

24.2.3 Detailed description of functions

HAL_DMAEx_MultiBufferStart

Function name

HAL_StatusTypeDef HAL_DMAEx_MultiBufferStart (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t SecondMemAddress, uint32_t DataLength)

Function description

Starts the multi_buffer DMA Transfer.

Parameters

- **hdma:** : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
- **SrcAddress:** The source memory Buffer address
- **DstAddress:** The destination memory Buffer address
- **SecondMemAddress:** The second memory Buffer address in case of multi buffer Transfer
- **DataLength:** The length of data to be transferred from source to destination

Return values

- **HAL:** status

HAL_DMAEx_MultiBufferStart_IT

Function name

HAL_StatusTypeDef HAL_DMAEx_MultiBufferStart_IT (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t SecondMemAddress, uint32_t DataLength)

Function description

Starts the multi_buffer DMA Transfer with interrupt enabled.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
- **SrcAddress:** The source memory Buffer address
- **DstAddress:** The destination memory Buffer address
- **SecondMemAddress:** The second memory Buffer address in case of multi buffer Transfer
- **DataLength:** The length of data to be transferred from source to destination

Return values

- **HAL:** status

HAL_DMAEx_ChangeMemory

Function name

HAL_StatusTypeDef HAL_DMAEx_ChangeMemory (DMA_HandleTypeDef * hdma, uint32_t Address, HAL_DMA_MemoryTypeDef memory)

Function description

Change the memory0 or memory1 address on the fly.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
- **Address:** The new address
- **memory:** the memory to be changed, This parameter can be one of the following values: MEMORY0 / MEMORY1

Return values

- **HAL:** status

Notes

- The MEMORY0 address can be changed only when the current transfer use MEMORY1 and the MEMORY1 address can be changed only when the current transfer use MEMORY0.

HAL_DMAEx_ConfigMuxSync

Function name

HAL_StatusTypeDef HAL_DMAEx_ConfigMuxSync (DMA_HandleTypeDef * hdma, HAL_DMA_MuxSyncConfigTypeDef * pSyncConfig)

Function description

Configure the DMAMUX synchronization parameters for a given DMA stream (instance).

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
- **pSyncConfig:** : pointer to HAL_DMA_MuxSyncConfigTypeDef : contains the DMAMUX synchronization parameters

Return values

- **HAL:** status

HAL_DMAEx_ConfigMuxRequestGenerator

Function name

HAL_StatusTypeDef HAL_DMAEx_ConfigMuxRequestGenerator (DMA_HandleTypeDef * hdma, HAL_DMA_MuxRequestGeneratorConfigTypeDef * pRequestGeneratorConfig)

Function description

Configure the DMAMUX request generator block used by the given DMA stream (instance).

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
- **pRequestGeneratorConfig:** : pointer to HAL_DMA_MuxRequestGeneratorConfigTypeDef : contains the request generator parameters.

Return values

- **HAL:** status

HAL_DMAEx_EnableMuxRequestGenerator

Function name

HAL_StatusTypeDef HAL_DMAEx_EnableMuxRequestGenerator (DMA_HandleTypeDef * hdma)

Function description

Enable the DMAMUX request generator block used by the given DMA stream (instance).

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.

Return values

- **HAL:** status

HAL_DMAEx_DisableMuxRequestGenerator

Function name

HAL_StatusTypeDef HAL_DMAEx_DisableMuxRequestGenerator (DMA_HandleTypeDef * hdma)

Function description

Disable the DMAMUX request generator block used by the given DMA stream (instance).

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.

Return values

- **HAL:** status

HAL_DMAEx_MUX_IRQHandler

Function name

void HAL_DMAEx_MUX_IRQHandler (DMA_HandleTypeDef * hdma)

Function description

Handles DMAMUX interrupt request.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.

Return values

- **None:**

24.3 DMAEx Firmware driver defines

The following section lists the various define and macros of the module.

24.3.1 DMAEx

DMAEx

DMAEx MUX RequestGeneneratorPolarity selection

HAL_DMAMUX_REQ_GEN_NO_EVENT

block request generator events

HAL_DMAMUX_REQ_GEN_RISING

generate request on rising edge events

HAL_DMAMUX_REQ_GEN_FALLING

generate request on falling edge events

HAL_DMAMUX_REQ_GEN_RISING_FALLING

generate request on rising and falling edge events

DMAEx MUX SignalGeneratorID selection

HAL_DMAMUX1_REQ_GEN_DMAMUX1_CH0_EVT

DMAMUX1 Request generator Signal is DMAMUX1 Channel0 Event

HAL_DMAMUX1_REQ_GEN_DMAMUX1_CH1_EVT

DMAMUX1 Request generator Signal is DMAMUX1 Channel1 Event

HAL_DMAMUX1_REQ_GEN_DMAMUX1_CH2_EVT

DMAMUX1 Request generator Signal is DMAMUX1 Channel2 Event

HAL_DMAMUX1_REQ_GEN_LPTIM1_OUT

DMAMUX1 Request generator Signal is LPTIM1 OUT

HAL_DMAMUX1_REQ_GEN_LPTIM2_OUT

DMAMUX1 Request generator Signal is LPTIM2 OUT

HAL_DMAMUX1_REQ_GEN_LPTIM3_OUT

DMAMUX1 Request generator Signal is LPTIM3 OUT

HAL_DMAMUX1_REQ_GEN_EXTI0

DMAMUX1 Request generator Signal is EXTI0 IT

HAL_DMAMUX1_REQ_GEN_TIM12_TRGO

DMAMUX1 Request generator Signal is TIM12 TRGO

HAL_DMAMUX2_REQ_GEN_DMAMUX2_CH0_EVT

DMAMUX2 Request generator Signal is DMAMUX2 Channel0 Event

HAL_DMAMUX2_REQ_GEN_DMAMUX2_CH1_EVT

DMAMUX2 Request generator Signal is DMAMUX2 Channel1 Event

HAL_DMAMUX2_REQ_GEN_DMAMUX2_CH2_EVT

DMAMUX2 Request generator Signal is DMAMUX2 Channel2 Event

HAL_DMAMUX2_REQ_GEN_DMAMUX2_CH3_EVT

DMAMUX2 Request generator Signal is DMAMUX2 Channel3 Event

HAL_DMAMUX2_REQ_GEN_DMAMUX2_CH4_EVT

DMAMUX2 Request generator Signal is DMAMUX2 Channel4 Event

HAL_DMAMUX2_REQ_GEN_DMAMUX2_CH5_EVT

DMAMUX2 Request generator Signal is DMAMUX2 Channel5 Event

HAL_DMAMUX2_REQ_GEN_DMAMUX2_CH6_EVT

DMAMUX2 Request generator Signal is DMAMUX2 Channel6 Event

HAL_DMAMUX2_REQ_GEN_LPUART1_RX_WKUP

DMAMUX2 Request generator Signal is LPUART1 RX Wakeup

HAL_DMAMUX2_REQ_GEN_LPUART1_TX_WKUP

DMAMUX2 Request generator Signal is LPUART1 TX Wakeup

HAL_DMAMUX2_REQ_GEN_LPTIM2_WKUP

DMAMUX2 Request generator Signal is LPTIM2 Wakeup

HAL_DMAMUX2_REQ_GEN_LPTIM2_OUT

DMAMUX2 Request generator Signal is LPTIM2 OUT

HAL_DMAMUX2_REQ_GEN_LPTIM3_WKUP

DMAMUX2 Request generator Signal is LPTIM3 Wakeup

HAL_DMAMUX2_REQ_GEN_LPTIM3_OUT

DMAMUX2 Request generator Signal is LPTIM3 OUT

HAL_DMAMUX2_REQ_GEN_LPTIM4_WKUP

DMAMUX2 Request generator Signal is LPTIM4 Wakeup

HAL_DMAMUX2_REQ_GEN_LPTIM5_WKUP

DMAMUX2 Request generator Signal is LPTIM5 Wakeup

HAL_DMAMUX2_REQ_GEN_I2C4_WKUP

DMAMUX2 Request generator Signal is I2C4 Wakeup

HAL_DMAMUX2_REQ_GEN_SPI6_WKUP

DMAMUX2 Request generator Signal is SPI6 Wakeup

HAL_DMAMUX2_REQ_GEN_COMP1_OUT

DMAMUX2 Request generator Signal is Comparator 1 output

HAL_DMAMUX2_REQ_GEN_COMP2_OUT

DMAMUX2 Request generator Signal is Comparator 2 output

HAL_DMAMUX2_REQ_GEN_RTC_WKUP

DMAMUX2 Request generator Signal is RTC Wakeup

HAL_DMAMUX2_REQ_GEN_EXTI0

DMAMUX2 Request generator Signal is EXTI0

HAL_DMAMUX2_REQ_GEN_EXTI2

DMAMUX2 Request generator Signal is EXTI2

HAL_DMAMUX2_REQ_GEN_I2C4_IT_EVT

DMAMUX2 Request generator Signal is I2C4 IT Event

HAL_DMAMUX2_REQ_GEN_SPI6_IT

DMAMUX2 Request generator Signal is SPI6 IT

HAL_DMAMUX2_REQ_GEN_LPUART1_TX_IT

DMAMUX2 Request generator Signal is LPUART1 Tx IT

HAL_DMAMUX2_REQ_GEN_LPUART1_RX_IT

DMAMUX2 Request generator Signal is LPUART1 Rx IT

HAL_DMAMUX2_REQ_GEN_ADC3_IT

DMAMUX2 Request generator Signal is ADC3 IT

HAL_DMAMUX2_REQ_GEN_ADC3_AWD1_OUT

DMAMUX2 Request generator Signal is ADC3 Analog Watchdog 1 output

HAL_DMAMUX2_REQ_GEN_BDMA_CH0_IT

DMAMUX2 Request generator Signal is BDMA Channel 0 IT

HAL_DMAMUX2_REQ_GEN_BDMA_CH1_IT

DMAMUX2 Request generator Signal is BDMA Channel 1 IT

DMAEx MUX SyncPolarity selection

HAL_DMAMUX_SYNC_NO_EVENT

block synchronization events

HAL_DMAMUX_SYNC_RISING

synchronize with rising edge events

HAL_DMAMUX_SYNC_FALLING

synchronize with falling edge events

HAL_DMAMUX_SYNC_RISING_FALLING

synchronize with rising and falling edge events

DMAEx MUX SyncSignalID selection

HAL_DMAMUX1_SYNC_DMAMUX1_CH0_EVT

DMAMUX1 synchronization Signal is DMAMUX1 Channel0 Event

HAL_DMAMUX1_SYNC_DMAMUX1_CH1_EVT

DMAMUX1 synchronization Signal is DMAMUX1 Channel1 Event

HAL_DMAMUX1_SYNC_DMAMUX1_CH2_EVT

DMAMUX1 synchronization Signal is DMAMUX1 Channel2 Event

HAL_DMAMUX1_SYNC_LPTIM1_OUT

DMAMUX1 synchronization Signal is LPTIM1 OUT

HAL_DMAMUX1_SYNC_LPTIM2_OUT

DMAMUX1 synchronization Signal is LPTIM2 OUT

HAL_DMAMUX1_SYNC_LPTIM3_OUT

DMAMUX1 synchronization Signal is LPTIM3 OUT

HAL_DMAMUX1_SYNC_EXTI0

DMAMUX1 synchronization Signal is EXTI0 IT

HAL_DMAMUX1_SYNC_TIM12_TRGO

DMAMUX1 synchronization Signal is TIM12 TRGO

HAL_DMAMUX2_SYNC_DMAMUX2_CH0_EVT

DMAMUX2 synchronization Signal is DMAMUX2 Channel0 Event

HAL_DMAMUX2_SYNC_DMAMUX2_CH1_EVT

DMAMUX2 synchronization Signal is DMAMUX2 Channel1 Event

HAL_DMAMUX2_SYNC_DMAMUX2_CH2_EVT

DMAMUX2 synchronization Signal is DMAMUX2 Channel2 Event

HAL_DMAMUX2_SYNC_DMAMUX2_CH3_EVT

DMAMUX2 synchronization Signal is DMAMUX2 Channel3 Event

HAL_DMAMUX2_SYNC_DMAMUX2_CH4_EVT

DMAMUX2 synchronization Signal is DMAMUX2 Channel4 Event

HAL_DMAMUX2_SYNC_DMAMUX2_CH5_EVT

DMAMUX2 synchronization Signal is DMAMUX2 Channel5 Event

HAL_DMAMUX2_SYNC_LPUART1_RX_WKUP

DMAMUX2 synchronization Signal is LPUART1 RX Wakeup

HAL_DMAMUX2_SYNC_LPUART1_TX_WKUP

DMAMUX2 synchronization Signal is LPUART1 TX Wakeup

HAL_DMAMUX2_SYNC_LPTIM2_OUT

DMAMUX2 synchronization Signal is LPTIM2 output

HAL_DMAMUX2_SYNC_LPTIM3_OUT

DMAMUX2 synchronization Signal is LPTIM3 output

HAL_DMAMUX2_SYNC_I2C4_WKUP

DMAMUX2 synchronization Signal is I2C4 Wakeup

HAL_DMAMUX2_SYNC_SPI6_WKUP

DMAMUX2 synchronization Signal is SPI6 Wakeup

HAL_DMAMUX2_SYNC_COMP1_OUT

DMAMUX2 synchronization Signal is Comparator 1 output

HAL_DMAMUX2_SYNC_RTC_WKUP

DMAMUX2 synchronization Signal is RTC Wakeup

HAL_DMAMUX2_SYNC_EXTI0

DMAMUX2 synchronization Signal is EXTI0 IT

HAL_DMAMUX2_SYNC_EXTI2

DMAMUX2 synchronization Signal is EXTI2 IT

25 HAL DSI Generic Driver

25.1 DSI Firmware driver registers structures

25.1.1 DSI_InitTypeDef

DSI_InitTypeDef is defined in the stm32h7xx_hal_dsi.h

Data Fields

- ***uint32_t AutomaticClockLaneControl***
- ***uint32_t TXEscapeCkdiv***
- ***uint32_t NumberOfLanes***

Field Documentation

- ***uint32_t DSI_InitTypeDef::AutomaticClockLaneControl***
  Automatic clock lane control This parameter can be any value of ***DSI_Automatic_Clk_Lane_Control***
- ***uint32_t DSI_InitTypeDef::TXEscapeCkdiv***
  TX Escape clock division The values 0 and 1 stop the TX_ESC clock generation
- ***uint32_t DSI_InitTypeDef::NumberOfLanes***
  Number of lanes This parameter can be any value of ***DSI_Number_Of_Lanes***

25.1.2 DSI_PLLInitTypeDef

DSI_PLLInitTypeDef is defined in the stm32h7xx_hal_dsi.h

Data Fields

- ***uint32_t PLLNDIV***
- ***uint32_t PLLIDF***
- ***uint32_t PLLODF***

Field Documentation

- ***uint32_t DSI_PLLInitTypeDef::PLLNDIV***
  PLL Loop Division Factor This parameter must be a value between 10 and 125
- ***uint32_t DSI_PLLInitTypeDef::PLLIDF***
  PLL Input Division Factor This parameter can be any value of ***DSI_PLL_IDF***
- ***uint32_t DSI_PLLInitTypeDef::PLLODF***
  PLL Output Division Factor This parameter can be any value of ***DSI_PLL_ODF***

25.1.3 DSI_VidCfgTypeDef

DSI_VidCfgTypeDef is defined in the stm32h7xx_hal_dsi.h

Data Fields

- ***uint32_t VirtualChannelID***
- ***uint32_t ColorCoding***
- ***uint32_t LooselyPacked***
- ***uint32_t Mode***
- ***uint32_t PacketSize***
- ***uint32_t NumberOfChunks***
- ***uint32_t NullPacketSize***
- ***uint32_t HSPolarity***
- ***uint32_t VSPolarity***
- ***uint32_t DEPolarity***
- ***uint32_t HorizontalSyncActive***

- ***uint32_t HorizontalBackPorch***
- ***uint32_t HorizontalLine***
- ***uint32_t VerticalSyncActive***
- ***uint32_t VerticalBackPorch***
- ***uint32_t VerticalFrontPorch***
- ***uint32_t VerticalActive***
- ***uint32_t LPCommandEnable***
- ***uint32_t LPLargestPacketSize***
- ***uint32_t LPVACTLargestPacketSize***
- ***uint32_t LPHorizontalFrontPorchEnable***
- ***uint32_t LPHorizontalBackPorchEnable***
- ***uint32_t LPVerticalActiveEnable***
- ***uint32_t LPVerticalFrontPorchEnable***
- ***uint32_t LPVerticalBackPorchEnable***
- ***uint32_t LPVerticalSyncActiveEnable***
- ***uint32_t FrameBTAAcknowledgeEnable***

Field Documentation

- ***uint32_t DSI_VidCfgTypeDef::VirtualChannelID***
  Virtual channel ID
- ***uint32_t DSI_VidCfgTypeDef::ColorCoding***
  Color coding for LTDC interface This parameter can be any value of ***DSI_Color_Coding***
- ***uint32_t DSI_VidCfgTypeDef::LooselyPacked***
  Enable or disable loosely packed stream (needed only when using 18-bit configuration). This parameter can be any value of ***DSI_LooselyPacked***
- ***uint32_t DSI_VidCfgTypeDef::Mode***
  Video mode type This parameter can be any value of ***DSI_Video_Mode_Type***
- ***uint32_t DSI_VidCfgTypeDef::PacketSize***
  Video packet size
- ***uint32_t DSI_VidCfgTypeDef::NumberOfChunks***
  Number of chunks
- ***uint32_t DSI_VidCfgTypeDef::NullPacketSize***
  Null packet size
- ***uint32_t DSI_VidCfgTypeDef::HSPolarity***
  HSYNC pin polarity This parameter can be any value of ***DSI_HSYNC_Polarity***
- ***uint32_t DSI_VidCfgTypeDef::VSPolarity***
  VSYNC pin polarity This parameter can be any value of ***DSI_VSYNC_Active_Polarity***
- ***uint32_t DSI_VidCfgTypeDef::DEPolarity***
  Data Enable pin polarity This parameter can be any value of ***DSI_DATA_ENABLE_Polarity***
- ***uint32_t DSI_VidCfgTypeDef::HorizontalSyncActive***
  Horizontal synchronism active duration (in lane byte clock cycles)
- ***uint32_t DSI_VidCfgTypeDef::HorizontalBackPorch***
  Horizontal back-porch duration (in lane byte clock cycles)
- ***uint32_t DSI_VidCfgTypeDef::HorizontalLine***
  Horizontal line duration (in lane byte clock cycles)
- ***uint32_t DSI_VidCfgTypeDef::VerticalSyncActive***
  Vertical synchronism active duration
- ***uint32_t DSI_VidCfgTypeDef::VerticalBackPorch***
  Vertical back-porch duration

- ***uint32_t DSI_VidCfgTypeDef::VerticalFrontPorch***
  Vertical front-porch duration
- ***uint32_t DSI_VidCfgTypeDef::VerticalActive***
  Vertical active duration
- ***uint32_t DSI_VidCfgTypeDef::LPCommandEnable***
  Low-power command enable This parameter can be any value of ***DSI_LP_Command***
- ***uint32_t DSI_VidCfgTypeDef::LPLargestPacketSize***
  The size, in bytes, of the low power largest packet that can fit in a line during VSA, VBP and VFP regions
- ***uint32_t DSI_VidCfgTypeDef::LPVACTLargestPacketSize***
  The size, in bytes, of the low power largest packet that can fit in a line during VACT region
- ***uint32_t DSI_VidCfgTypeDef::LPHorizontalFrontPorchEnable***
  Low-power horizontal front-porch enable This parameter can be any value of ***DSI_LP_HFP***
- ***uint32_t DSI_VidCfgTypeDef::LPHorizontalBackPorchEnable***
  Low-power horizontal back-porch enable This parameter can be any value of ***DSI_LP_HBP***
- ***uint32_t DSI_VidCfgTypeDef::LPVerticalActiveEnable***
  Low-power vertical active enable This parameter can be any value of ***DSI_LP_VACT***
- ***uint32_t DSI_VidCfgTypeDef::LPVerticalFrontPorchEnable***
  Low-power vertical front-porch enable This parameter can be any value of ***DSI_LP_VFP***
- ***uint32_t DSI_VidCfgTypeDef::LPVerticalBackPorchEnable***
  Low-power vertical back-porch enable This parameter can be any value of ***DSI_LP_VBP***
- ***uint32_t DSI_VidCfgTypeDef::LPVerticalSyncActiveEnable***
  Low-power vertical sync active enable This parameter can be any value of ***DSI_LP_VSYNC***
- ***uint32_t DSI_VidCfgTypeDef::FrameBTAAcknowledgeEnable***
  Frame bus-turn-around acknowledge enable This parameter can be any value of ***DSI_FBTA_acknowledge***

25.1.4 DSI_CmdCfgTypeDef

DSI_CmdCfgTypeDef is defined in the stm32h7xx_hal_dsi.h

Data Fields

- ***uint32_t VirtualChannelID***
- ***uint32_t ColorCoding***
- ***uint32_t CommandSize***
- ***uint32_t TearingEffectSource***
- ***uint32_t TearingEffectPolarity***
- ***uint32_t HSPolarity***
- ***uint32_t VSPolarity***
- ***uint32_t DEPolarity***
- ***uint32_t VSyncPol***
- ***uint32_t AutomaticRefresh***
- ***uint32_t TEAcknowledgeRequest***

Field Documentation

- ***uint32_t DSI_CmdCfgTypeDef::VirtualChannelID***
  Virtual channel ID
- ***uint32_t DSI_CmdCfgTypeDef::ColorCoding***
  Color coding for LTDC interface This parameter can be any value of ***DSI_Color_Coding***
- ***uint32_t DSI_CmdCfgTypeDef::CommandSize***
  Maximum allowed size for an LTDC write memory command, measured in pixels. This parameter can be any value between 0x00 and 0xFFFFU
- ***uint32_t DSI_CmdCfgTypeDef::TearingEffectSource***
  Tearing effect source This parameter can be any value of ***DSI_TearingEffectSource***

- ***uint32_t DSI_CmdCfgTypeDef::TearingEffectPolarity***
  Tearing effect pin polarity This parameter can be any value of ***DSI_TearingEffectPolarity***
- ***uint32_t DSI_CmdCfgTypeDef::HSPolarity***
  HSYNC pin polarity This parameter can be any value of ***DSI_HSYNC_Polarity***
- ***uint32_t DSI_CmdCfgTypeDef::VSPolarity***
  VSYNC pin polarity This parameter can be any value of ***DSI_VSYNC_Active_Polarity***
- ***uint32_t DSI_CmdCfgTypeDef::DEPolarity***
  Data Enable pin polarity This parameter can be any value of ***DSI_DATA_ENABLE_Polarity***
- ***uint32_t DSI_CmdCfgTypeDef::VSyncPol***
  VSync edge on which the LTDC is halted This parameter can be any value of ***DSI_Vsync_Polarity***
- ***uint32_t DSI_CmdCfgTypeDef::AutomaticRefresh***
  Automatic refresh mode This parameter can be any value of ***DSI_AutomaticRefresh***
- ***uint32_t DSI_CmdCfgTypeDef::TEAcknowledgeRequest***
  Tearing Effect Acknowledge Request Enable This parameter can be any value of ***DSI_TE_AcknowledgeRequest***

25.1.5 DSI_LPCmdTypeDef

DSI_LPCmdTypeDef is defined in the stm32h7xx_hal_dsi.h

Data Fields

- ***uint32_t LPGenShortWriteNoP***
- ***uint32_t LPGenShortWriteOneP***
- ***uint32_t LPGenShortWriteTwoP***
- ***uint32_t LPGenShortReadNoP***
- ***uint32_t LPGenShortReadOneP***
- ***uint32_t LPGenShortReadTwoP***
- ***uint32_t LPGenLongWrite***
- ***uint32_t LPDcsShortWriteNoP***
- ***uint32_t LPDcsShortWriteOneP***
- ***uint32_t LPDcsShortReadNoP***
- ***uint32_t LPDcsLongWrite***
- ***uint32_t LPMaxReadPacket***
- ***uint32_t AcknowledgeRequest***

Field Documentation

- ***uint32_t DSI_LPCmdTypeDef::LPGenShortWriteNoP***
  Generic Short Write Zero parameters Transmission This parameter can be any value of ***DSI_LP_LPGenShortWriteNoP***
- ***uint32_t DSI_LPCmdTypeDef::LPGenShortWriteOneP***
  Generic Short Write One parameter Transmission This parameter can be any value of ***DSI_LP_LPGenShortWriteOneP***
- ***uint32_t DSI_LPCmdTypeDef::LPGenShortWriteTwoP***
  Generic Short Write Two parameters Transmission This parameter can be any value of ***DSI_LP_LPGenShortWriteTwoP***
- ***uint32_t DSI_LPCmdTypeDef::LPGenShortReadNoP***
  Generic Short Read Zero parameters Transmission This parameter can be any value of ***DSI_LP_LPGenShortReadNoP***
- ***uint32_t DSI_LPCmdTypeDef::LPGenShortReadOneP***
  Generic Short Read One parameter Transmission This parameter can be any value of ***DSI_LP_LPGenShortReadOneP***
- ***uint32_t DSI_LPCmdTypeDef::LPGenShortReadTwoP***
  Generic Short Read Two parameters Transmission This parameter can be any value of ***DSI_LP_LPGenShortReadTwoP***

- ***uint32_t DSI_LPCmdTypeDef::LPGenLongWrite***
  Generic Long Write Transmission This parameter can be any value of ***DSI_LP_LPGenLongWrite***
- ***uint32_t DSI_LPCmdTypeDef::LPDcsShortWriteNoP***
  DCS Short Write Zero parameters Transmission This parameter can be any value of ***DSI_LP_LPDcsShortWriteNoP***
- ***uint32_t DSI_LPCmdTypeDef::LPDcsShortWriteOneP***
  DCS Short Write One parameter Transmission This parameter can be any value of ***DSI_LP_LPDcsShortWriteOneP***
- ***uint32_t DSI_LPCmdTypeDef::LPDcsShortReadNoP***
  DCS Short Read Zero parameters Transmission This parameter can be any value of ***DSI_LP_LPDcsShortReadNoP***
- ***uint32_t DSI_LPCmdTypeDef::LPDcsLongWrite***
  DCS Long Write Transmission This parameter can be any value of ***DSI_LP_LPDcsLongWrite***
- ***uint32_t DSI_LPCmdTypeDef::LPMaxReadPacket***
  Maximum Read Packet Size Transmission This parameter can be any value of ***DSI_LP_LPMaxReadPacket***
- ***uint32_t DSI_LPCmdTypeDef::AcknowledgeRequest***
  Acknowledge Request Enable This parameter can be any value of ***DSI_AcknowledgeRequest***

25.1.6 DSI_PHY_TimerTypeDef

DSI_PHY_TimerTypeDef is defined in the stm32h7xx_hal_dsi.h

Data Fields

- ***uint32_t ClockLaneHS2LPTime***
- ***uint32_t ClockLaneLP2HSTime***
- ***uint32_t DataLaneHS2LPTime***
- ***uint32_t DataLaneLP2HSTime***
- ***uint32_t DataLaneMaxReadTime***
- ***uint32_t StopWaitTime***

Field Documentation

- ***uint32_t DSI_PHY_TimerTypeDef::ClockLaneHS2LPTime***
  The maximum time that the D-PHY clock lane takes to go from high-speed to low-power transmission
- ***uint32_t DSI_PHY_TimerTypeDef::ClockLaneLP2HSTime***
  The maximum time that the D-PHY clock lane takes to go from low-power to high-speed transmission
- ***uint32_t DSI_PHY_TimerTypeDef::DataLaneHS2LPTime***
  The maximum time that the D-PHY data lanes takes to go from high-speed to low-power transmission
- ***uint32_t DSI_PHY_TimerTypeDef::DataLaneLP2HSTime***
  The maximum time that the D-PHY data lanes takes to go from low-power to high-speed transmission
- ***uint32_t DSI_PHY_TimerTypeDef::DataLaneMaxReadTime***
  The maximum time required to perform a read command
- ***uint32_t DSI_PHY_TimerTypeDef::StopWaitTime***
  The minimum wait period to request a High-Speed transmission after the Stop state

25.1.7 DSI_HOST_TimeoutTypeDef

DSI_HOST_TimeoutTypeDef is defined in the stm32h7xx_hal_dsi.h

Data Fields

- ***uint32_t TimeoutCkdiv***
- ***uint32_t HighSpeedTransmissionTimeout***
- ***uint32_t LowPowerReceptionTimeout***
- ***uint32_t HighSpeedReadTimeout***
- ***uint32_t LowPowerReadTimeout***
- ***uint32_t HighSpeedWriteTimeout***

- ***uint32_t HighSpeedWritePrespMode***
- ***uint32_t LowPowerWriteTimeout***
- ***uint32_t BTATimeout***

Field Documentation

- ***uint32_t DSI_HOST_TimeoutTypeDef::TimeoutCkdiv***
  Time-out clock division
- ***uint32_t DSI_HOST_TimeoutTypeDef::HighSpeedTransmissionTimeout***
  High-speed transmission time-out
- ***uint32_t DSI_HOST_TimeoutTypeDef::LowPowerReceptionTimeout***
  Low-power reception time-out
- ***uint32_t DSI_HOST_TimeoutTypeDef::HighSpeedReadTimeout***
  High-speed read time-out
- ***uint32_t DSI_HOST_TimeoutTypeDef::LowPowerReadTimeout***
  Low-power read time-out
- ***uint32_t DSI_HOST_TimeoutTypeDef::HighSpeedWriteTimeout***
  High-speed write time-out
- ***uint32_t DSI_HOST_TimeoutTypeDef::HighSpeedWritePrespMode***
  High-speed write presp mode This parameter can be any value of ***DSI_HS_PrespMode***
- ***uint32_t DSI_HOST_TimeoutTypeDef::LowPowerWriteTimeout***
  Low-speed write time-out
- ***uint32_t DSI_HOST_TimeoutTypeDef::BTATimeout***
  BTA time-out

25.1.8 __DSI_HandleTypeDef

__DSI_HandleTypeDef is defined in the stm32h7xx_hal_dsi.h

Data Fields

- ***DSI_TypeDef * Instance***
- ***DSI_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_DSI_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***uint32_t ErrorMsk***
- ***void(* TearingEffectCallback***
- ***void(* EndOfRefreshCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***DSI_TypeDef* __DSI_HandleTypeDef::Instance***
  Register base address
- ***DSI_InitTypeDef __DSI_HandleTypeDef::Init***
  DSI required parameters
- ***HAL_LockTypeDef __DSI_HandleTypeDef::Lock***
  DSI peripheral status
- ***__IO HAL_DSI_StateTypeDef __DSI_HandleTypeDef::State***
  DSI communication state
- ***__IO uint32_t __DSI_HandleTypeDef::ErrorCode***
  DSI Error code

- ***uint32_t __DSI_HandleTypeDef::ErrorMsk***
  DSI Error monitoring mask
- ***void(* __DSI_HandleTypeDef::TearingEffectCallback)(struct __DSI_HandleTypeDef *hdsi)***
  DSI Tearing Effect Callback
- ***void(* __DSI_HandleTypeDef::EndOfRefreshCallback)(struct __DSI_HandleTypeDef *hdsi)***
  DSI End Of Refresh Callback
- ***void(* __DSI_HandleTypeDef::ErrorCallback)(struct __DSI_HandleTypeDef *hdsi)***
  DSI Error Callback
- ***void(* __DSI_HandleTypeDef::MspInitCallback)(struct __DSI_HandleTypeDef *hdsi)***
  DSI Msp Init callback
- ***void(* __DSI_HandleTypeDef::MspDeInitCallback)(struct __DSI_HandleTypeDef *hdsi)***
  DSI Msp DeInit callback

25.2 DSI Firmware driver API description

The following section lists the various functions of the DSI library.

25.2.1 How to use this driver

The DSI HAL driver can be used as follows:

1. Declare a DSI_HandleTypeDef handle structure, for example: DSI_HandleTypeDef hdsi;
2. Initialize the DSI low level resources by implementing the HAL_DSI_MspInit() API:
   a. Enable the DSI interface clock
   b. NVIC configuration if you need to use interrupt process
      - Configure the DSI interrupt priority
      - Enable the NVIC DSI IRQ Channel
3. Initialize the DSI Host peripheral, the required PLL parameters, number of lances and TX Escape clock divider by calling the HAL_DSI_Init() API which calls HAL_DSI_MspInit().

Configuration

1. Use HAL_DSI_ConfigAdaptedCommandMode() function to configure the DSI host in adapted command mode.
2. When operating in video mode , use HAL_DSI_ConfigVideoMode() to configure the DSI host.
3. Function HAL_DSI_ConfigCommand() is used to configure the DSI commands behavior in low power mode.
4. To configure the DSI PHY timings parameters, use function HAL_DSI_ConfigPhyTimer().
5. The DSI Host can be started/stopped using respectively functions HAL_DSI_Start() and HAL_DSI_Stop(). Functions HAL_DSI_ShortWrite(), HAL_DSI_LongWrite() and HAL_DSI_Read() allows respectively to write DSI short packets, long packets and to read DSI packets.
6. The DSI Host Offers two Low power modes :
   – Low Power Mode on data lanes only: Only DSI data lanes are shut down. It is possible to enter/exit from this mode using respectively functions HAL_DSI_EnterULPMData() and HAL_DSI_ExitULPMData()
   – Low Power Mode on data and clock lanes : All DSI lanes are shut down including data and clock lanes. It is possible to enter/exit from this mode using respectively functions HAL_DSI_EnterULPM() and HAL_DSI_ExitULPM()
7. To control DSI state you can use the following function: HAL_DSI_GetState()

Error management

1. User can select the DSI errors to be reported/monitored using function HAL_DSI_ConfigErrorMonitor() When an error occurs, the callback HAL_DSI_ErrorCallback() is asserted and then user can retrieve the error code by calling function HAL_DSI_GetError()

DSI HAL driver macros list

Below the list of most used macros in DSI HAL driver.

- __HAL_DSI_ENABLE: Enable the DSI Host.
- __HAL_DSI_DISABLE: Disable the DSI Host.
- __HAL_DSI_WRAPPER_ENABLE: Enables the DSI wrapper.
- __HAL_DSI_WRAPPER_DISABLE: Disable the DSI wrapper.
- __HAL_DSI_PLL_ENABLE: Enables the DSI PLL.
- __HAL_DSI_PLL_DISABLE: Disables the DSI PLL.
- __HAL_DSI_REG_ENABLE: Enables the DSI regulator.
- __HAL_DSI_REG_DISABLE: Disables the DSI regulator.
- __HAL_DSI_GET_FLAG: Get the DSI pending flags.
- __HAL_DSI_CLEAR_FLAG: Clears the DSI pending flags.
- __HAL_DSI_ENABLE_IT: Enables the specified DSI interrupts.
- __HAL_DSI_DISABLE_IT: Disables the specified DSI interrupts.
- __HAL_DSI_GET_IT_SOURCE: Checks whether the specified DSI interrupt source is enabled or not.

Note: You can refer to the DSI HAL driver header file for more useful macros

Callback registration

The compilation define USE_HAL_DSI_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Function HAL_DSI_RegisterCallback() to register a callback.

Function HAL_DSI_RegisterCallback() allows to register following callbacks:

- TearingEffectCallback : DSI Tearing Effect Callback.
- EndOfRefreshCallback : DSI End Of Refresh Callback.
- ErrorCallback : DSI Error Callback
- MspInitCallback : DSI MspInit.
- MspDeInitCallback : DSI MspDeInit.

This function takes as parameters the HAL peripheral handle, the callback ID and a pointer to the user callback function.

Use function HAL_DSI_UnRegisterCallback() to reset a callback to the default weak function. HAL_DSI_UnRegisterCallback takes as parameters the HAL peripheral handle, and the callback ID.

This function allows to reset following callbacks:

- TearingEffectCallback : DSI Tearing Effect Callback.
- EndOfRefreshCallback : DSI End Of Refresh Callback.
- ErrorCallback : DSI Error Callback
- MspInitCallback : DSI MspInit.
- MspDeInitCallback : DSI MspDeInit.

By default, after the HAL_DSI_Init and when the state is HAL_DSI_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_DSI_TearingEffectCallback(), HAL_DSI_EndOfRefreshCallback(). Exception done for MspInit and MspDeInit functions that are respectively reset to the legacy weak (surcharged) functions in the HAL_DSI_Init() and HAL_DSI_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_DSI_Init() and HAL_DSI_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_DSI_STATE_READY state only. Exception done MspInit/ MspDeInit that can be registered/unregistered in HAL_DSI_STATE_READY or HAL_DSI_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_DSI_RegisterCallback() before calling HAL_DSI_DeInit() or HAL_DSI_Init() function.

When The compilation define USE_HAL_DSI_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

25.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the DSI
- De-initialize the DSI

This section contains the following APIs:

- ***HAL_DSI_Init()***
- ***HAL_DSI_DeInit()***
- ***HAL_DSI_ConfigErrorMonitor()***
- ***HAL_DSI_MspInit()***
- ***HAL_DSI_MspDeInit()***
- ***HAL_DSI_RegisterCallback()***
- ***HAL_DSI_UnRegisterCallback()***

25.2.3 IO operation functions

This section provides function allowing to:

- Handle DSI interrupt request

This section contains the following APIs:

- ***HAL_DSI_IRQHandler()***
- ***HAL_DSI_TearingEffectCallback()***
- ***HAL_DSI_EndOfRefreshCallback()***
- ***HAL_DSI_ErrorCallback()***

25.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure the Generic interface read-back Virtual Channel ID
- Select video mode and configure the corresponding parameters
- Configure command transmission mode: High-speed or Low-power
- Configure the flow control
- Configure the DSI PHY timer
- Configure the DSI HOST timeout
- Configure the DSI HOST timeout
- Start/Stop the DSI module
- Refresh the display in command mode
- Controls the display color mode in Video mode
- Control the display shutdown in Video mode
- write short DCS or short Generic command
- write long DCS or long Generic command
- Read command (DCS or generic)
- Enter/Exit the Ultra Low Power Mode on data only (D-PHY PLL running)
- Enter/Exit the Ultra Low Power Mode on data only and clock (D-PHY PLL turned off)
- Start/Stop test pattern generation
- Slew-Rate And Delay Tuning
- Low-Power Reception Filter Tuning
- Activate an additional current path on all lanes to meet the SDDTx parameter
- Custom lane pins configuration
- Set custom timing for the PHY
- Force the Clock/Data Lane in TX Stop Mode
- Force LP Receiver in Low-Power Mode

- Force Data Lanes in RX Mode after a BTA
- Enable a pull-down on the lanes to prevent from floating states when unused
- Switch off the contention detection on data lanes

This section contains the following APIs:

- *HAL_DSI_SetGenericVCID()*
- *HAL_DSI_ConfigVideoMode()*
- *HAL_DSI_ConfigAdaptedCommandMode()*
- *HAL_DSI_ConfigCommand()*
- *HAL_DSI_ConfigFlowControl()*
- *HAL_DSI_ConfigPhyTimer()*
- *HAL_DSI_ConfigHostTimeouts()*
- *HAL_DSI_Start()*
- *HAL_DSI_Stop()*
- *HAL_DSI_Refresh()*
- *HAL_DSI_ColorMode()*
- *HAL_DSI_Shutdown()*
- *HAL_DSI_ShortWrite()*
- *HAL_DSI_LongWrite()*
- *HAL_DSI_Read()*
- *HAL_DSI_EnterULPMData()*
- *HAL_DSI_ExitULPMData()*
- *HAL_DSI_EnterULPM()*
- *HAL_DSI_ExitULPM()*
- *HAL_DSI_PatternGeneratorStart()*
- *HAL_DSI_PatternGeneratorStop()*
- *HAL_DSI_SetSlewRateAndDelayTuning()*
- *HAL_DSI_SetLowPowerRXFilter()*
- *HAL_DSI_SetSDD()*
- *HAL_DSI_SetLanePinsConfiguration()*
- *HAL_DSI_SetPHYTimings()*
- *HAL_DSI_ForceTXStopMode()*
- *HAL_DSI_ForceRXLowPower()*
- *HAL_DSI_ForceDataLanesInRX()*
- *HAL_DSI_SetPullDown()*
- *HAL_DSI_SetContentionDetectionOff()*

25.2.5 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the DSI state.
- Get error code.

This section contains the following APIs:

- *HAL_DSI_GetState()*
- *HAL_DSI_GetError()*

25.2.6 Detailed description of functions

HAL_DSI_Init

Function name

HAL_StatusTypeDef HAL_DSI_Init (DSI_HandleTypeDef * hdsi, DSI_PLLInitTypeDef * PLLInit)

Function description

Initializes the DSI according to the specified parameters in the DSI_InitTypeDef and create the associated handle.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **PLLInit:** pointer to a DSI_PLLInitTypeDef structure that contains the PLL Clock structure definition for the DSI.

Return values

- **HAL:** status

HAL_DSI_DeInit

Function name

HAL_StatusTypeDef HAL_DSI_DeInit (DSI_HandleTypeDef * hdsi)

Function description

De-initializes the DSI peripheral registers to their default reset values.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **HAL:** status

HAL_DSI_MspInit

Function name

void HAL_DSI_MspInit (DSI_HandleTypeDef * hdsi)

Function description

Initializes the DSI MSP.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **None:**

HAL_DSI_MspDeInit

Function name

void HAL_DSI_MspDeInit (DSI_HandleTypeDef * hdsi)

Function description

De-initializes the DSI MSP.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **None:**

HAL_DSI_ConfigErrorMonitor

Function name

HAL_StatusTypeDef HAL_DSI_ConfigErrorMonitor (DSI_HandleTypeDef * hdsi, uint32_t ActiveErrors)

Function description

Enable the error monitor flags.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **ActiveErrors:** indicates which error interrupts will be enabled. This parameter can be any combination of
  - DSI_Error_Data_Type.

Return values

- **HAL:** status

HAL_DSI_RegisterCallback

Function name

HAL_StatusTypeDef HAL_DSI_RegisterCallback (DSI_HandleTypeDef * hdsi, HAL_DSI_CallbackIDTypeDef CallbackID, pDSI_CallbackTypeDef pCallback)

Function description

Register a User DSI Callback To be used instead of the weak predefined callback.

Parameters

- **hdsi:** dsi handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_DSI_TEARING_EFFECT_CB_ID Tearing Effect Callback ID
  - HAL_DSI_ENDOF_REFRESH_CB_ID End Of Refresh Callback ID
  - HAL_DSI_ERROR_CB_ID Error Callback ID
  - HAL_DSI_MSPINIT_CB_ID MspInit callback ID
  - HAL_DSI_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **status:**

HAL_DSI_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_DSI_UnRegisterCallback (DSI_HandleTypeDef * hdsi, HAL_DSI_CallbackIDTypeDef CallbackID)

Function description

Unregister a DSI Callback DSI callback is redirected to the weak predefined callback.

Parameters

- **hdsi:** dsi handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_DSI_TEARING_EFFECT_CB_ID Tearing Effect Callback ID
  - HAL_DSI_ENDOF_REFRESH_CB_ID End Of Refresh Callback ID
  - HAL_DSI_ERROR_CB_ID Error Callback ID
  - HAL_DSI_MSPINIT_CB_ID MspInit callback ID
  - HAL_DSI_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **status:**

HAL_DSI_IRQHandler

Function name

void HAL_DSI_IRQHandler (DSI_HandleTypeDef * hdsi)

Function description

Handles DSI interrupt request.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **HAL:** status

HAL_DSI_TearingEffectCallback

Function name

void HAL_DSI_TearingEffectCallback (DSI_HandleTypeDef * hdsi)

Function description

Tearing Effect DSI callback.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **None:**

HAL_DSI_EndOfRefreshCallback

Function name

void HAL_DSI_EndOfRefreshCallback (DSI_HandleTypeDef * hdsi)

Function description

End of Refresh DSI callback.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **None:**

HAL_DSI_ErrorCallback

Function name

void HAL_DSI_ErrorCallback (DSI_HandleTypeDef * hdsi)

Function description

Operation Error DSI callback.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **None:**

HAL_DSI_SetGenericVCID

Function name

HAL_StatusTypeDef HAL_DSI_SetGenericVCID (DSI_HandleTypeDef * hdsi, uint32_t VirtualChannelID)

Function description

Configure the Generic interface read-back Virtual Channel ID.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **VirtualChannelID:** Virtual channel ID

Return values

- **HAL:** status

HAL_DSI_ConfigVideoMode

Function name

HAL_StatusTypeDef HAL_DSI_ConfigVideoMode (DSI_HandleTypeDef * hdsi, DSI_VidCfgTypeDef * VidCfg)

Function description

Select video mode and configure the corresponding parameters.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **VidCfg:** pointer to a DSI_VidCfgTypeDef structure that contains the DSI video mode configuration parameters

Return values

- **HAL:** status

HAL_DSI_ConfigAdaptedCommandMode

Function name

HAL_StatusTypeDef HAL_DSI_ConfigAdaptedCommandMode (DSI_HandleTypeDef * hdsi, DSI_CmdCfgTypeDef * CmdCfg)

Function description

Select adapted command mode and configure the corresponding parameters.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **CmdCfg:** pointer to a DSI_CmdCfgTypeDef structure that contains the DSI command mode configuration parameters

Return values

- **HAL:** status

HAL_DSI_ConfigCommand

Function name

HAL_StatusTypeDef HAL_DSI_ConfigCommand (DSI_HandleTypeDef * hdsi, DSI_LPCmdTypeDef * LPCmd)

Function description

Configure command transmission mode: High-speed or Low-power and enable/disable acknowledge request after packet transmission.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **LPCmd:** pointer to a DSI_LPCmdTypeDef structure that contains the DSI command transmission mode configuration parameters

Return values

- **HAL:** status

HAL_DSI_ConfigFlowControl

Function name

HAL_StatusTypeDef HAL_DSI_ConfigFlowControl (DSI_HandleTypeDef * hdsi, uint32_t FlowControl)

Function description

Configure the flow control parameters.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **FlowControl:** flow control feature(s) to be enabled. This parameter can be any combination of
  - DSI_FlowControl.

Return values

- **HAL:** status

HAL_DSI_ConfigPhyTimer

Function name

HAL_StatusTypeDef HAL_DSI_ConfigPhyTimer (DSI_HandleTypeDef * hdsi, DSI_PHY_TimerTypeDef * PhyTimers)

Function description

Configure the DSI PHY timer parameters.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **PhyTimers:** DSI_PHY_TimerTypeDef structure that contains the DSI PHY timing parameters

Return values

- **HAL:** status

HAL_DSI_ConfigHostTimeouts

Function name

HAL_StatusTypeDef HAL_DSI_ConfigHostTimeouts (DSI_HandleTypeDef * hdsi, DSI_HOST_TimeoutTypeDef * HostTimeouts)

Function description

Configure the DSI HOST timeout parameters.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **HostTimeouts:** DSI_HOST_TimeoutTypeDef structure that contains the DSI host timeout parameters

Return values

- **HAL:** status

HAL_DSI_Start

Function name

HAL_StatusTypeDef HAL_DSI_Start (DSI_HandleTypeDef * hdsi)

Function description

Start the DSI module.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **HAL:** status

HAL_DSI_Stop

Function name

HAL_StatusTypeDef HAL_DSI_Stop (DSI_HandleTypeDef * hdsi)

Function description

Stop the DSI module.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **HAL:** status

HAL_DSI_Refresh

Function name

HAL_StatusTypeDef HAL_DSI_Refresh (DSI_HandleTypeDef * hdsi)

Function description

Refresh the display in command mode.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **HAL:** status

HAL_DSI_ColorMode

Function name

HAL_StatusTypeDef HAL_DSI_ColorMode (DSI_HandleTypeDef * hdsi, uint32_t ColorMode)

Function description

Controls the display color mode in Video mode.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **ColorMode:** Color mode (full or 8-colors). This parameter can be any value of
  - DSI_Color_Mode

Return values

- **HAL:** status

HAL_DSI_Shutdown

Function name

HAL_StatusTypeDef HAL_DSI_Shutdown (DSI_HandleTypeDef * hdsi, uint32_t Shutdown)

Function description

Control the display shutdown in Video mode.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **Shutdown:** Shut-down (Display-ON or Display-OFF). This parameter can be any value of
  - DSI_ShutDown

Return values

- **HAL:** status

HAL_DSI_ShortWrite

Function name

HAL_StatusTypeDef HAL_DSI_ShortWrite (DSI_HandleTypeDef * hdsi, uint32_t ChannelID, uint32_t Mode, uint32_t Param1, uint32_t Param2)

Function description

write short DCS or short Generic command

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **ChannelID:** Virtual channel ID.
- **Mode:** DSI short packet data type. This parameter can be any value of
  - DSI_SHORT_WRITE_PKT_Data_Type.
- **Param1:** DSC command or first generic parameter. This parameter can be any value of
  - DSI_DCS_Command or a generic command code.
- **Param2:** DSC parameter or second generic parameter.

Return values

- **HAL:** status

HAL_DSI_LongWrite

Function name

HAL_StatusTypeDef HAL_DSI_LongWrite (DSI_HandleTypeDef * hdsi, uint32_t ChannelID, uint32_t Mode, uint32_t NbParams, uint32_t Param1, uint8_t * ParametersTable)

Function description

write long DCS or long Generic command

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **ChannelID:** Virtual channel ID.
- **Mode:** DSI long packet data type. This parameter can be any value of
  - DSI_LONG_WRITE_PKT_Data_Type.
- **NbParams:** Number of parameters.
- **Param1:** DSC command or first generic parameter. This parameter can be any value of
  - DSI_DCS_Command or a generic command code
- **ParametersTable:** Pointer to parameter values table.

Return values

- **HAL:** status

HAL_DSI_Read

Function name

HAL_StatusTypeDef HAL_DSI_Read (DSI_HandleTypeDef * hdsi, uint32_t ChannelNbr, uint8_t * Array, uint32_t Size, uint32_t Mode, uint32_t DCSCmd, uint8_t * ParametersTable)

Function description

Read command (DCS or generic)

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **ChannelNbr:** Virtual channel ID
- **Array:** pointer to a buffer to store the payload of a read back operation.
- **Size:** Data size to be read (in byte).
- **Mode:** DSI read packet data type. This parameter can be any value of
  - DSI_SHORT_READ_PKT_Data_Type.
- **DCSCmd:** DCS get/read command.
- **ParametersTable:** Pointer to parameter values table.

Return values

- **HAL:** status

HAL_DSI_EnterULPMData

Function name

HAL_StatusTypeDef HAL_DSI_EnterULPMData (DSI_HandleTypeDef * hdsi)

Function description

Enter the ULPM (Ultra Low Power Mode) with the D-PHY PLL running (only data lanes are in ULPM)

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **HAL:** status

HAL_DSI_ExitULPMData

Function name

HAL_StatusTypeDef HAL_DSI_ExitULPMData (DSI_HandleTypeDef * hdsi)

Function description

Exit the ULPM (Ultra Low Power Mode) with the D-PHY PLL running (only data lanes are in ULPM)

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **HAL:** status

HAL_DSI_EnterULPM

Function name

HAL_StatusTypeDef HAL_DSI_EnterULPM (DSI_HandleTypeDef * hdsi)

Function description

Enter the ULPM (Ultra Low Power Mode) with the D-PHY PLL turned off (both data and clock lanes are in ULPM)

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **HAL:** status

HAL_DSI_ExitULPM

Function name

HAL_StatusTypeDef HAL_DSI_ExitULPM (DSI_HandleTypeDef * hdsi)

Function description

Exit the ULPM (Ultra Low Power Mode) with the D-PHY PLL turned off (both data and clock lanes are in ULPM)

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **HAL:** status

HAL_DSI_PatternGeneratorStart

Function name

HAL_StatusTypeDef HAL_DSI_PatternGeneratorStart (DSI_HandleTypeDef * hdsi, uint32_t Mode, uint32_t Orientation)

Function description

Start test pattern generation.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **Mode:** Pattern generator mode This parameter can be one of the following values: 0 : Color bars (horizontal or vertical) 1 : BER pattern (vertical only)
- **Orientation:** Pattern generator orientation This parameter can be one of the following values: 0 : Vertical color bars 1 : Horizontal color bars

Return values

- **HAL:** status

HAL_DSI_PatternGeneratorStop

Function name

HAL_StatusTypeDef HAL_DSI_PatternGeneratorStop (DSI_HandleTypeDef * hdsi)

Function description

Stop test pattern generation.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **HAL:** status

HAL_DSI_SetSlewRateAndDelayTuning

Function name

HAL_StatusTypeDef HAL_DSI_SetSlewRateAndDelayTuning (DSI_HandleTypeDef * hdsi, uint32_t CommDelay, uint32_t Lane, uint32_t Value)

Function description

Set Slew-Rate And Delay Tuning.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **CommDelay:** Communication delay to be adjusted. This parameter can be any value of
  - DSI_Communication_Delay
- **Lane:** select between clock or data lanes. This parameter can be any value of
  - DSI_Lane_Group
- **Value:** Custom value of the slew-rate or delay

Return values

- **HAL:** status

HAL_DSI_SetLowPowerRXFilter

Function name

HAL_StatusTypeDef HAL_DSI_SetLowPowerRXFilter (DSI_HandleTypeDef * hdsi, uint32_t Frequency)

Function description

Low-Power Reception Filter Tuning.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **Frequency:** cutoff frequency of low-pass filter at the input of LPRX

Return values

- **HAL:** status

HAL_DSI_SetSDD

Function name

HAL_StatusTypeDef HAL_DSI_SetSDD (DSI_HandleTypeDef * hdsi, FunctionalState State)

Function description

Activate an additional current path on all lanes to meet the SDDTx parameter defined in the MIPI D-PHY specification.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **State:** ENABLE or DISABLE

Return values

- **HAL:** status

HAL_DSI_SetLanePinsConfiguration

Function name

HAL_StatusTypeDef HAL_DSI_SetLanePinsConfiguration (DSI_HandleTypeDef * hdsi, uint32_t CustomLane, uint32_t Lane, FunctionalState State)

Function description

Custom lane pins configuration.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **CustomLane:** Function to be applied on selected lane. This parameter can be any value of
  - DSI_CustomLane
- **Lane:** select between clock or data lane 0 or data lane 1. This parameter can be any value of
  - DSI_Lane_Select
- **State:** ENABLE or DISABLE

Return values

- **HAL:** status

HAL_DSI_SetPHYTimings

Function name

HAL_StatusTypeDef HAL_DSI_SetPHYTimings (DSI_HandleTypeDef * hdsi, uint32_t Timing, FunctionalState State, uint32_t Value)

Function description

Set custom timing for the PHY.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **Timing:** PHY timing to be adjusted. This parameter can be any value of
  - DSI_PHY_Timing
- **State:** ENABLE or DISABLE
- **Value:** Custom value of the timing

Return values

- **HAL:** status

HAL_DSI_ForceTXStopMode

Function name

HAL_StatusTypeDef HAL_DSI_ForceTXStopMode (DSI_HandleTypeDef * hdsi, uint32_t Lane, FunctionalState State)

Function description

Force the Clock/Data Lane in TX Stop Mode.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **Lane:** select between clock or data lanes. This parameter can be any value of
  – DSI_Lane_Group
- **State:** ENABLE or DISABLE

Return values

- **HAL:** status

HAL_DSI_ForceRXLowPower

Function name

HAL_StatusTypeDef HAL_DSI_ForceRXLowPower (DSI_HandleTypeDef * hdsi, FunctionalState State)

Function description

Force LP Receiver in Low-Power Mode.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **State:** ENABLE or DISABLE

Return values

- **HAL:** status

HAL_DSI_ForceDataLanesInRX

Function name

HAL_StatusTypeDef HAL_DSI_ForceDataLanesInRX (DSI_HandleTypeDef * hdsi, FunctionalState State)

Function description

Force Data Lanes in RX Mode after a BTA.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **State:** ENABLE or DISABLE

Return values

- **HAL:** status

HAL_DSI_SetPullDown

Function name

HAL_StatusTypeDef HAL_DSI_SetPullDown (DSI_HandleTypeDef * hdsi, FunctionalState State)

Function description

Enable a pull-down on the lanes to prevent from floating states when unused.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **State:** ENABLE or DISABLE

Return values

- **HAL:** status

HAL_DSI_SetContentionDetectionOff

Function name

HAL_StatusTypeDef HAL_DSI_SetContentionDetectionOff (DSI_HandleTypeDef * hdsi, FunctionalState State)

Function description

Switch off the contention detection on data lanes.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **State:** ENABLE or DISABLE

Return values

- **HAL:** status

HAL_DSI_GetError

Function name

uint32_t HAL_DSI_GetError (DSI_HandleTypeDef * hdsi)

Function description

Return the DSI error code.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **DSI:** Error Code

HAL_DSI_GetState

Function name

HAL_DSI_StateTypeDef HAL_DSI_GetState (DSI_HandleTypeDef * hdsi)

Function description

Return the DSI state.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **HAL:** state

25.3 DSI Firmware driver defines

The following section lists the various define and macros of the module.

25.3.1 DSI

DSI

DSI Acknowledge Request

DSI_ACKNOWLEDGE_DISABLE

DSI_ACKNOWLEDGE_ENABLE

DSI Automatic Refresh

DSI_AR_DISABLE

DSI_AR_ENABLE

DSI Automatic Clk Lane Control

DSI_AUTO_CLK_LANE_CTRL_DISABLE

DSI_AUTO_CLK_LANE_CTRL_ENABLE

DSI Color Coding

DSI_RGB565

The values 0x00000001 and 0x00000002 can also be used for the RGB565 color mode configuration

DSI_RGB666

The value 0x00000004 can also be used for the RGB666 color mode configuration

DSI_RGB888

DSI Color Mode

DSI_COLOR_MODE_FULL

DSI_COLOR_MODE_EIGHT

DSI Communication Delay

DSI_SLEW_RATE_HSTX

DSI_SLEW_RATE_LPTX

DSI_HS_DELAY

DSI CustomLane

DSI_SWAP_LANE_PINS

DSI_INVERT_HS_SIGNAL

DSI DATA ENABLE Polarity

DSI_DATA_ENABLE_ACTIVE_HIGH

DSI_DATA_ENABLE_ACTIVE_LOW

DSI DCS Command

DSI_ENTER_IDLE_MODE

DSI_ENTER_INVERT_MODE

DSI_ENTER_NORMAL_MODE

DSI_ENTER_PARTIAL_MODE

DSI_ENTER_SLEEP_MODE

DSI_EXIT_IDLE_MODE

DSI_EXIT_INVERT_MODE

DSI_EXIT_SLEEP_MODE

DSI_GET_3D_CONTROL

DSI_GET_ADDRESS_MODE

DSI_GET_BLUE_CHANNEL

DSI_GET_DIAGNOSTIC_RESULT

DSI_GET_DISPLAY_MODE

DSI_GET_GREEN_CHANNEL

DSI_GET_PIXEL_FORMAT

DSI_GET_POWER_MODE

DSI_GET_RED_CHANNEL

DSI_GET_SCANLINE

DSI_GET_SIGNAL_MODE

DSI_NOP

DSI_READ_DDB_CONTINUE

DSI_READ_DDB_START

DSI_READ_MEMORY_CONTINUE

DSI_READ_MEMORY_START

DSI_SET_3D_CONTROL

DSI_SET_ADDRESS_MODE

DSI_SET_COLUMN_ADDRESS

DSI_SET_DISPLAY_OFF

DSI_SET_DISPLAY_ON

DSI_SET_GAMMA_CURVE

DSI_SET_PAGE_ADDRESS

DSI_SET_PARTIAL_COLUMNS

DSI_SET_PARTIAL_ROWS

DSI_SET_PIXEL_FORMAT

DSI_SET_SCROLL_AREA

DSI_SET_SCROLL_START

DSI_SET_TEAR_OFF

DSI_SET_TEAR_ON

DSI_SET_TEAR_SCANLINE

DSI_SET_VSYNC_TIMING

DSI_SOFT_RESET

DSI_WRITE_LUT

DSI_WRITE_MEMORY_CONTINUE

DSI_WRITE_MEMORY_START

DSI Error Data Type

HAL_DSI_ERROR_NONE

HAL_DSI_ERROR_ACK

Acknowledge errors

HAL_DSI_ERROR_PHY

PHY related errors

HAL_DSI_ERROR_TX

Transmission error

HAL_DSI_ERROR_RX

Reception error

HAL_DSI_ERROR_ECC

ECC errors

HAL_DSI_ERROR_CRC

CRC error

HAL_DSI_ERROR_PSE

Packet Size error

HAL_DSI_ERROR_EOT

End Of Transmission error

HAL_DSI_ERROR_OVF

FIFO overflow error

HAL_DSI_ERROR_GEN

Generic FIFO related errors

HAL_DSI_ERROR_INVALID_CALLBACK

DSI Invalid Callback error

DSI Exported Macros

__HAL_DSI_RESET_HANDLE_STATE

Description:

- Reset DSI handle state.

Parameters:

- __HANDLE__: DSI handle

Return value:

- None

__HAL_DSI_ENABLE

Description:

- Enables the DSI host.

Parameters:

- __HANDLE__: DSI handle

Return value:

- None.

__HAL_DSI_DISABLE

Description:

- Disables the DSI host.

Parameters:

- __HANDLE__: DSI handle

Return value:

- None.

__HAL_DSI_WRAPPER_ENABLE

Description:

- Enables the DSI wrapper.

Parameters:

- __HANDLE__: DSI handle

Return value:

- None.

__HAL_DSI_WRAPPER_DISABLE

Description:

- Disable the DSI wrapper.

Parameters:

- __HANDLE__: DSI handle

Return value:

- None.

__HAL_DSI_PLL_ENABLE

Description:

- Enables the DSI PLL.

Parameters:

- __HANDLE__: DSI handle

Return value:

- None.

__HAL_DSI_PLL_DISABLE

Description:

- Disables the DSI PLL.

Parameters:

- __HANDLE__: DSI handle

Return value:

- None.

__HAL_DSI_REG_ENABLE

Description:

- Enables the DSI regulator.

Parameters:

- __HANDLE__: DSI handle

Return value:

- None.

__HAL_DSI_REG_DISABLE

Description:

- Disables the DSI regulator.

Parameters:

- __HANDLE__: DSI handle

Return value:

- None.

__HAL_DSI_GET_FLAG

Description:

- Get the DSI pending flags.

Parameters:

- __HANDLE__: DSI handle.
- __FLAG__: Get the specified flag. This parameter can be any combination of the following values:
  - DSI_FLAG_TE : Tearing Effect Interrupt Flag
  - DSI_FLAG_ER : End of Refresh Interrupt Flag
  - DSI_FLAG_BUSY : Busy Flag
  - DSI_FLAG_PLLLS: PLL Lock Status
  - DSI_FLAG_PLLL : PLL Lock Interrupt Flag
  - DSI_FLAG_PLLU : PLL Unlock Interrupt Flag
  - DSI_FLAG_RRS : Regulator Ready Flag
  - DSI_FLAG_RR : Regulator Ready Interrupt Flag

Return value:

- The: state of FLAG (SET or RESET).

__HAL_DSI_CLEAR_FLAG

Description:

- Clears the DSI pending flags.

Parameters:

- __HANDLE__: DSI handle.
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - DSI_FLAG_TE : Tearing Effect Interrupt Flag
  - DSI_FLAG_ER : End of Refresh Interrupt Flag
  - DSI_FLAG_PLLL : PLL Lock Interrupt Flag
  - DSI_FLAG_PLLU : PLL Unlock Interrupt Flag
  - DSI_FLAG_RR : Regulator Ready Interrupt Flag

Return value:

- None

__HAL_DSI_ENABLE_IT

Description:

- Enables the specified DSI interrupts.

Parameters:

- __HANDLE__: DSI handle.
- __INTERRUPT__: specifies the DSI interrupt sources to be enabled. This parameter can be any combination of the following values:
  - DSI_IT_TE : Tearing Effect Interrupt
  - DSI_IT_ER : End of Refresh Interrupt
  - DSI_IT_PLLL: PLL Lock Interrupt
  - DSI_IT_PLLU: PLL Unlock Interrupt
  - DSI_IT_RR : Regulator Ready Interrupt

Return value:

- None

__HAL_DSI_DISABLE_IT

Description:

- Disables the specified DSI interrupts.

Parameters:

- __HANDLE__: DSI handle
- __INTERRUPT__: specifies the DSI interrupt sources to be disabled. This parameter can be any combination of the following values:
  - DSI_IT_TE : Tearing Effect Interrupt
  - DSI_IT_ER : End of Refresh Interrupt
  - DSI_IT_PLLL: PLL Lock Interrupt
  - DSI_IT_PLLU: PLL Unlock Interrupt
  - DSI_IT_RR : Regulator Ready Interrupt

Return value:

- None

__HAL_DSI_GET_IT_SOURCE

Description:

- Checks whether the specified DSI interrupt source is enabled or not.

Parameters:

- __HANDLE__: DSI handle
- __INTERRUPT__: specifies the DSI interrupt source to check. This parameter can be one of the following values:
  - DSI_IT_TE : Tearing Effect Interrupt
  - DSI_IT_ER : End of Refresh Interrupt
  - DSI_IT_PLLL: PLL Lock Interrupt
  - DSI_IT_PLLU: PLL Unlock Interrupt
  - DSI_IT_RR : Regulator Ready Interrupt

Return value:

- The: state of INTERRUPT (SET or RESET).

DSI FBTA Acknowledge

DSI_FBTAA_DISABLE

DSI_FBTAA_ENABLE

DSI Flags

DSI_FLAG_TE

DSI_FLAG_ER

DSI_FLAG_BUSY

DSI_FLAG_PLLLS

DSI_FLAG_PLLL

DSI_FLAG_PLLU

DSI_FLAG_RRS

DSI_FLAG_RR

DSI Flow Control

DSI_FLOW_CONTROL_CRC_RX

DSI_FLOW_CONTROL_ECC_RX

DSI_FLOW_CONTROL_BTA

DSI_FLOW_CONTROL_EOTP_RX

DSI_FLOW_CONTROL_EOTP_TX

DSI_FLOW_CONTROL_ALL

DSI HSYNC Polarity

DSI_HSYNC_ACTIVE_HIGH

DSI_HSYNC_ACTIVE_LOW

DSI HS Presp Mode

DSI_HS_PM_DISABLE

DSI_HS_PM_ENABLE

DSI Interrupts

DSI_IT_TE

DSI_IT_ER

DSI_IT_PLLL

DSI_IT_PLLU

DSI_IT_RR

DSI Lane Group

DSI_CLOCK_LANE

DSI_DATA_LANES

DSI Lane Select

DSI_CLK_LANE

DSI_DATA_LANE0

DSI_DATA_LANE1

DSI LONG WRITE PKT Data Type

DSI_DCS_LONG_PKT_WRITE

DCS long write

DSI_GEN_LONG_PKT_WRITE

Generic long write

DSI Loosely Packed

DSI_LOOSELY_PACKED_ENABLE

DSI_LOOSELY_PACKED_DISABLE

DSI LP Command

DSI_LP_COMMAND_DISABLE

DSI_LP_COMMAND_ENABLE

DSI LP HBP

DSI_LP_HBP_DISABLE

DSI_LP_HBP_ENABLE

DSI LP HFP

DSI_LP_HFP_DISABLE

DSI_LP_HFP_ENABLE

DSI LP LPDcs Long Write

DSI_LP_DLW_DISABLE

DSI_LP_DLW_ENABLE

DSI LP LPDcs Short Read NoP

DSI_LP_DSR0P_DISABLE

DSI_LP_DSR0P_ENABLE

DSI LP LPDcs Short Write NoP

DSI_LP_DSW0P_DISABLE

DSI_LP_DSW0P_ENABLE

DSI LP LPDcs Short Write OneP

DSI_LP_DSW1P_DISABLE

DSI_LP_DSW1P_ENABLE

DSI LP LPGen LongWrite

DSI_LP_GLW_DISABLE

DSI_LP_GLW_ENABLE

DSI LP LPGen Short Read NoP

DSI_LP_GSR0P_DISABLE

DSI_LP_GSR0P_ENABLE

DSI LP LPGen Short Read OneP

DSI_LP_GSR1P_DISABLE

DSI_LP_GSR1P_ENABLE

DSI LP LPGen Short Read TwoP

DSI_LP_GSR2P_DISABLE

DSI_LP_GSR2P_ENABLE

DSI LP LPGen Short Write NoP

DSI_LP_GSW0P_DISABLE

DSI_LP_GSW0P_ENABLE

DSI LP LPGen Short Write OneP

DSI_LP_GSW1P_DISABLE

DSI_LP_GSW1P_ENABLE

DSI LP LPGen Short Write TwoP

DSI_LP_GSW2P_DISABLE

DSI_LP_GSW2P_ENABLE

DSI LP LPMax Read Packet

DSI_LP_MRDP_DISABLE

DSI_LP_MRDP_ENABLE

DSI LP VACT

DSI_LP_VACT_DISABLE

DSI_LP_VACT_ENABLE

DSI LP VBP

DSI_LP_VBP_DISABLE

DSI_LP_VBP_ENABLE

DSI LP VFP

DSI_LP_VFP_DISABLE

DSI_LP_VFP_ENABLE

DSI LP VSYNC

DSI_LP_VSYNC_DISABLE

DSI_LP_VSYNC_ENABLE

DSI Number Of Lanes

DSI_ONE_DATA_LANE

DSI_TWO_DATA_LANES

DSI PHY Timing

DSI_TCLK_POST

DSI_TLPX_CLK

DSI_THS_EXIT

DSI_TLPX_DATA

DSI_THS_ZERO

DSI_THS_TRAIL

DSI_THS_PREPARE

DSI_TCLK_ZERO

DSI_TCLK_PREPARE

DSI PLL IDF

DSI_PLL_IN_DIV1

DSI_PLL_IN_DIV2

DSI_PLL_IN_DIV3

DSI_PLL_IN_DIV4

DSI_PLL_IN_DIV5

DSI_PLL_IN_DIV6

DSI_PLL_IN_DIV7

DSI PLL ODF

DSI_PLL_OUT_DIV1

DSI_PLL_OUT_DIV2

DSI_PLL_OUT_DIV4

DSI_PLL_OUT_DIV8

DSI SHORT READ PKT Data Type

DSI_DCS_SHORT_PKT_READ

DCS short read

DSI_GEN_SHORT_PKT_READ_P0

Generic short read, no parameters

DSI_GEN_SHORT_PKT_READ_P1

Generic short read, one parameter

DSI_GEN_SHORT_PKT_READ_P2

Generic short read, two parameters

DSI SHORT WRITE PKT Data Type

DSI_DCS_SHORT_PKT_WRITE_P0

DCS short write, no parameters

DSI_DCS_SHORT_PKT_WRITE_P1

DCS short write, one parameter

DSI_GEN_SHORT_PKT_WRITE_P0

Generic short write, no parameters

DSI_GEN_SHORT_PKT_WRITE_P1

Generic short write, one parameter

DSI_GEN_SHORT_PKT_WRITE_P2

Generic short write, two parameters

DSI ShutDown

DSI_DISPLAY_ON

DSI_DISPLAY_OFF

DSI Tearing Effect Polarity

DSI_TE_RISING_EDGE

DSI_TE_FALLING_EDGE

DSI Tearing Effect Source

DSI_TE_DSILINK

DSI_TE_EXTERNAL

DSI TE Acknowledge Request

DSI_TE_ACKNOWLEDGE_DISABLE

DSI_TE_ACKNOWLEDGE_ENABLE

DSI Video Mode Type

DSI_VID_MODE_NB_PULSES

DSI_VID_MODE_NB_EVENTS

DSI_VID_MODE_BURST

DSI VSYNC Active Polarity

DSI_VSYNC_ACTIVE_HIGH

DSI_VSYNC_ACTIVE_LOW

DSI Vsync Polarity

DSI_VSYNC_FALLING

DSI_VSYNC_RISING

26 HAL DTS Generic Driver

26.1 DTS Firmware driver registers structures

26.1.1 DTS_InitTypeDef

DTS_InitTypeDef is defined in the stm32h7xx_hal_dts.h

Data Fields

- ***uint32_t QuickMeasure***
- ***uint32_t RefClock***
- ***uint32_t TriggerInput***
- ***uint32_t SamplingTime***
- ***uint32_t Divider***
- ***uint32_t HighThreshold***
- ***uint32_t LowThreshold***

Field Documentation

- ***uint32_t DTS_InitTypeDef::QuickMeasure***
  Specifies the quick measure option selection of the DTS sensor. This parameter can be a value of ***DTS_Quick_Measurement***
- ***uint32_t DTS_InitTypeDef::RefClock***
  Specifies the reference clock selection of the DTS sensor. This parameter can be a value of ***DTS_Reference_Clock_Selection***
- ***uint32_t DTS_InitTypeDef::TriggerInput***
  Specifies the trigger input of the DTS sensor. This parameter can be a value of ***DTS_TriggerConfig***
- ***uint32_t DTS_InitTypeDef::SamplingTime***
  Specifies the sampling time configuration. This parameter can be a value of ***DTS_Sampling_Time***
- ***uint32_t DTS_InitTypeDef::Divider***
  Specifies the high speed clock divider ratio. This parameter can be a value from 0 to 127
- ***uint32_t DTS_InitTypeDef::HighThreshold***
  Specifies the high threshold of the DTS sensor
- ***uint32_t DTS_InitTypeDef::LowThreshold***
  Specifies the low threshold of the DTS sensor

26.1.2 __DTS_HandleTypeDef

__DTS_HandleTypeDef is defined in the stm32h7xx_hal_dts.h

Data Fields

- ***DTS_TypeDef * Instance***
- ***DTS_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_DTS_StateTypeDef State***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***void(* EndCallback***
- ***void(* LowCallback***
- ***void(* HighCallback***
- ***void(* AsyncEndCallback***
- ***void(* AsyncLowCallback***
- ***void(* AsyncHighCallback***

Field Documentation

- ***DTS_TypeDef* __DTS_HandleTypeDef::Instance***
  Register base address
- ***DTS_InitTypeDef __DTS_HandleTypeDef::Init***
  DTS required parameters
- ***HAL_LockTypeDef __DTS_HandleTypeDef::Lock***
  DTS Locking object
- ***__IO HAL_DTS_StateTypeDef __DTS_HandleTypeDef::State***
  DTS peripheral state
- ***void(* __DTS_HandleTypeDef::MspInitCallback)(struct __DTS_HandleTypeDef *hdts)***
  DTS Base Msp Init Callback
- ***void(* __DTS_HandleTypeDef::MspDeInitCallback)(struct __DTS_HandleTypeDef *hdts)***
  DTS Base Msp DeInit Callback
- ***void(* __DTS_HandleTypeDef::EndCallback)(struct __DTS_HandleTypeDef *hdts)***
  End measure Callback
- ***void(* __DTS_HandleTypeDef::LowCallback)(struct __DTS_HandleTypeDef *hdts)***
  low threshold Callback
- ***void(* __DTS_HandleTypeDef::HighCallback)(struct __DTS_HandleTypeDef *hdts)***
  high threshold Callback
- ***void(* __DTS_HandleTypeDef::AsyncEndCallback)(struct __DTS_HandleTypeDef *hdts)***
  Asynchronous end of measure Callback
- ***void(* __DTS_HandleTypeDef::AsyncLowCallback)(struct __DTS_HandleTypeDef *hdts)***
  Asynchronous low threshold Callback
- ***void(* __DTS_HandleTypeDef::AsyncHighCallback)(struct __DTS_HandleTypeDef *hdts)***
  Asynchronous high threshold Callback

26.2 DTS Firmware driver API description

The following section lists the various functions of the DTS library.

26.2.1 DTS Peripheral features

The STM32h7xx device family integrate one DTS sensor interface :

26.2.2 How to use this driver

26.2.3 Initialization and de-initialization functions

This section provides functions to initialize and de-initialize comparators
This section contains the following APIs:

- ***HAL_DTS_Init()***
- ***HAL_DTS_DeInit()***
- ***HAL_DTS_MspInit()***
- ***HAL_DTS_MspDeInit()***
- ***HAL_DTS_RegisterCallback()***
- ***HAL_DTS_UnRegisterCallback()***

26.2.4 DTS Start Stop operation functions

This section provides functions allowing to:

- Start a DTS Sensor without interrupt.
- Stop a DTS Sensor without interrupt.

- Start a DTS Sensor with interrupt generation.
- Stop a DTS Sensor with interrupt generation.

This section contains the following APIs:

- *HAL_DTS_Start()*
- *HAL_DTS_Stop()*
- *HAL_DTS_Start_IT()*
- *HAL_DTS_Stop_IT()*
- *HAL_DTS_GetTemperature()*
- *HAL_DTS_IRQHandler()*
- *HAL_DTS_EndCallback()*
- *HAL_DTS_LowCallback()*
- *HAL_DTS_HighCallback()*
- *HAL_DTS_AsyncEndCallback()*
- *HAL_DTS_AsyncLowCallback()*
- *HAL_DTS_AsyncHighCallback()*
- *HAL_DTS_GetState()*

26.2.5 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral.

This section contains the following APIs:

- *HAL_DTS_GetState()*

26.2.6 Detailed description of functions

HAL_DTS_Init

Function name

HAL_StatusTypeDef HAL_DTS_Init (DTS_HandleTypeDef * hdts)

Function description

Initialize the DTS according to the specified parameters in the DTS_InitTypeDef and initialize the associated handle.

Parameters

- **hdts:** DTS handle

Return values

- **HAL:** status

HAL_DTS_DeInit

Function name

HAL_StatusTypeDef HAL_DTS_DeInit (DTS_HandleTypeDef * hdts)

Function description

DeInitialize the DTS peripheral.

Parameters

- **hdts:** DTS handle

Return values

- **HAL:** status

Notes

- Deinitialization cannot be performed if the DTS configuration is locked. To unlock the configuration, perform a system reset.

HAL_DTS_MspInit

Function name

void HAL_DTS_MspInit (DTS_HandleTypeDef * hdts)

Function description

Initialize the DTS MSP.

Parameters

- **hdts:** DTS handle

Return values

- **None:**

HAL_DTS_MspDeInit

Function name

void HAL_DTS_MspDeInit (DTS_HandleTypeDef * hdts)

Function description

DeInitialize the DTS MSP.

Parameters

- **hdts:** DTS handle

Return values

- **None:**

HAL_DTS_RegisterCallback

Function name

HAL_StatusTypeDef HAL_DTS_RegisterCallback (DTS_HandleTypeDef * hdts, HAL_DTS_CallbackIDTypeDef CallbackID, pDTS_CallbackTypeDef pCallback)

Function description

Register a user DTS callback to be used instead of the weak predefined callback.

Parameters

- **hdts:** DTS handle.
- **CallbackID:** ID of the callback to be registered. This parameter can be one of the following values:
  - HAL_DTS_MEAS_COMPLETE_CB_ID measure complete callback ID.
  - HAL_DTS_ASYNC_MEAS_COMPLETE_CB_ID asynchronous measure complete callback ID.
  - HAL_DTS_LOW_THRESHOLD_CB_ID low threshold detection callback ID.
  - HAL_DTS_ASYNC_LOW_THRESHOLD_CB_ID asynchronous low threshold detection callback ID.
  - HAL_DTS_HIGH_THRESHOLD_CB_ID high threshold detection callback ID.
  - HAL_DTS_ASYNC_HIGH_THRESHOLD_CB_ID asynchronous high threshold detection callback ID.
  - HAL_DTS_MSPINIT_CB_ID MSP init callback ID.
  - HAL_DTS_MSPDEINIT_CB_ID MSP de-init callback ID.
- **pCallback:** pointer to the callback function.

Return values

- **HAL:** status.

HAL_DTS_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_DTS_UnRegisterCallback (DTS_HandleTypeDef * hdts, HAL_DTS_CallbackIDTypeDef CallbackID)

Function description

Unregister a user DTS callback.

Parameters

- **hdts:** DTS handle.
- **CallbackID:** ID of the callback to be unregistered. This parameter can be one of the following values:
  - HAL_DTS_MEAS_COMPLETE_CB_ID measure complete callback ID.
  - HAL_DTS_ASYNC_MEAS_COMPLETE_CB_ID asynchronous measure complete callback ID.
  - HAL_DTS_LOW_THRESHOLD_CB_ID low threshold detection callback ID.
  - HAL_DTS_ASYNC_LOW_THRESHOLD_CB_ID asynchronous low threshold detection callback ID.
  - HAL_DTS_HIGH_THRESHOLD_CB_ID high threshold detection callback ID.
  - HAL_DTS_ASYNC_HIGH_THRESHOLD_CB_ID asynchronous high threshold detection callback ID.
  - HAL_DTS_MSPINIT_CB_ID MSP init callback ID.
  - HAL_DTS_MSPDEINIT_CB_ID MSP de-init callback ID.

Return values

- **HAL:** status.

HAL_DTS_Start

Function name

HAL_StatusTypeDef HAL_DTS_Start (DTS_HandleTypeDef * hdts)

Function description

Start the DTS sensor.

Parameters

- **hdts:** DTS handle

Return values

- **HAL:** status

HAL_DTS_Stop

Function name

HAL_StatusTypeDef HAL_DTS_Stop (DTS_HandleTypeDef * hdts)

Function description

Stop the DTS Sensor.

Parameters

- **hdts:** DTS handle

Return values

- **HAL:** status

HAL_DTS_GetTemperature

Function name

HAL_StatusTypeDef HAL_DTS_GetTemperature (DTS_HandleTypeDef * hdts, int32_t * Temperature)

Function description

Get temperature from DTS.

Parameters

- **hdts:** DTS handle
- **Temperature:** Temperature in deg C

Return values

- **HAL:** status

Notes

- This function retrieves latest available measure

HAL_DTS_Start_IT

Function name

HAL_StatusTypeDef HAL_DTS_Start_IT (DTS_HandleTypeDef * hdts)

Function description

Enable the interrupt(s) and start the DTS sensor.

Parameters

- **hdts:** DTS handle

Return values

- **HAL:** status

HAL_DTS_Stop_IT

Function name

HAL_StatusTypeDef HAL_DTS_Stop_IT (DTS_HandleTypeDef * hdts)

Function description

Disable the interrupt(s) and stop the DTS sensor.

Parameters

- **hdts:** DTS handle

Return values

- **HAL:** status

HAL_DTS_IRQHandler

Function name

void HAL_DTS_IRQHandler (DTS_HandleTypeDef * hdts)

Function description

DTS sensor IRQ Handler.

Parameters

- **hdts:** DTS handle

Return values

- **None:**

HAL_DTS_GetState

Function name

HAL_DTS_StateTypeDef HAL_DTS_GetState (DTS_HandleTypeDef * hdts)

Function description

Return the DTS handle state.

Parameters

- **hdts:** DTS handle

Return values

- **HAL:** state

HAL_DTS_EndCallback

Function name

void HAL_DTS_EndCallback (DTS_HandleTypeDef * hdts)

Function description

DTS Sensor End measure callback.

Parameters

- **hdts:** DTS handle

Return values

- **None:**

HAL_DTS_LowCallback

Function name

void HAL_DTS_LowCallback (DTS_HandleTypeDef * hdts)

Function description

DTS Sensor low threshold measure callback.

Parameters

- **hdts:** DTS handle

Return values

- **None:**

HAL_DTS_HighCallback

Function name

void HAL_DTS_HighCallback (DTS_HandleTypeDef * hdts)

Function description

DTS Sensor high threshold measure callback.

Parameters

- **hdts:** DTS handle

Return values

- **None:**

HAL_DTS_AsyncEndCallback

Function name

void HAL_DTS_AsyncEndCallback (DTS_HandleTypeDef * hdts)

Function description

DTS Sensor asynchronous end measure callback.

Parameters

- **hdts:** DTS handle

Return values

- **None:**

HAL_DTS_AsyncLowCallback

Function name

void HAL_DTS_AsyncLowCallback (DTS_HandleTypeDef * hdts)

Function description

DTS Sensor asynchronous low threshold measure callback.

Parameters

- **hdts:** DTS handle

Return values

- **None:**

HAL_DTS_AsyncHighCallback

Function name

void HAL_DTS_AsyncHighCallback (DTS_HandleTypeDef * hdts)

Function description

DTS Sensor asynchronous high threshold measure callback.

Parameters

- **hdts:** DTS handle

Return values

- **None:**

26.3 DTS Firmware driver defines

The following section lists the various define and macros of the module.

26.3.1 DTS

DTS

DTS Exported Macros

__HAL_DTS_RESET_HANDLE_STATE

Description:

- Reset DTS handle state.

Parameters:

- __HANDLE__: DTS handle.

Return value:

- None

__HAL_DTS_ENABLE

Description:

- Enable the specified DTS sensor.

Parameters:

- __HANDLE__: DTS handle.

Return value:

- None

__HAL_DTS_DISABLE

Description:

- Disable the specified DTS sensor.

Parameters:

- __HANDLE__: DTS handle.

Return value:

- None

__HAL_DTS_EXTI_WAKEUP_ENABLE_IT

Description:

- Enable the DTS EXTI line in interrupt mode.

Return value:

- None

__HAL_DTS_EXTI_WAKEUP_DISABLE_IT

Description:

- Disable the DTS EXTI line in interrupt mode.

Return value:

- None

__HAL_DTS_EXTI_WAKEUP_ENABLE_EVENT

Description:

- Enable the DTS EXTI Line in event mode.

Return value:

- None

__HAL_DTS_EXTI_WAKEUP_DISABLE_EVENT

Description:

- Disable the DTS EXTI Line in event mode.

Return value:

- None

__HAL_DTS_GET_FLAG

Description:

- Checks whether the specified DTS flag is set or not.

Parameters:

- __HANDLE__: specifies the DTS Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - DTS_FLAG_TS1_ITE : interrupt flag for end of measure for DTS1
  - DTS_FLAG_TS1_ITL : interrupt flag for low threshold for DTS1
  - DTS_FLAG_TS1_ITH : interrupt flag for high threshold for DTS1
  - DTS_FLAG_TS1_AITE: asynchronous interrupt flag for end of measure for DTS1
  - DTS_FLAG_TS1_AITL: asynchronous interrupt flag for low threshold for DTS1
  - DTS_FLAG_TS1_AITH: asynchronous interrupt flag for high threshold for DTS1
  - DTS_FLAG_TS1_RDY : Ready flag for DTS1

Return value:

- The: new state of __FLAG__ (SET or RESET).

__HAL_DTS_CLEAR_FLAG

Description:

- Clears the specified DTS pending flag.

Parameters:

- __HANDLE__: specifies the DTS Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - DTS_FLAG_TS1_ITE : interrupt flag for end of measure for DTS1
  - DTS_FLAG_TS1_ITL : interrupt flag for low threshold for DTS1
  - DTS_FLAG_TS1_ITH : interrupt flag for high threshold for DTS1
  - DTS_FLAG_TS1_AITE: asynchronous interrupt flag for end of measure for DTS1
  - DTS_FLAG_TS1_AITL: asynchronous interrupt flag for low threshold for DTS1
  - DTS_FLAG_TS1_AITH: asynchronous interrupt flag for high threshold for DTS1

Return value:

- None

__HAL_DTS_ENABLE_IT

Description:

- Enable the specified DTS interrupt.

Parameters:

- __HANDLE__: specifies the DTS Handle.
- __INTERRUPT__: specifies the DTS interrupt source to enable. This parameter can be one of the following values:
  - DTS_IT_TS1_ITE : interrupt flag for end of measure for DTS1
  - DTS_IT_TS1_ITL : interrupt flag for low of measure for DTS1
  - DTS_IT_TS1_ITH : interrupt flag for high of measure for DTS1
  - DTS_IT_TS1_AITE : asynchronous interrupt flag for end of measure for DTS1
  - DTS_IT_TS1_AITL : asynchronous interrupt flag for low of measure for DTS1
  - DTS_IT_TS1_AITH : asynchronous interrupt flag for high of measure for DTS1

Return value:

- None

__HAL_DTS_DISABLE_IT

Description:

- Disable the specified DTS interrupt.

Parameters:

- __HANDLE__: specifies the DTS Handle.
- __INTERRUPT__: specifies the DTS interrupt source to enable. This parameter can be one of the following values:
  - DTS_IT_TS1_ITE : interrupt flag for end of measure for DTS1
  - DTS_IT_TS1_ITL : interrupt flag for low of measure for DTS1
  - DTS_IT_TS1_ITH : interrupt flag for high of measure for DTS1
  - DTS_IT_TS1_AITE : asynchronous interrupt flag for end of measure for DTS1
  - DTS_IT_TS1_AITL : asynchronous interrupt flag for low of measure for DTS1
  - DTS_IT_TS1_AITH : asynchronous interrupt flag for high of measure for DTS1

Return value:

- None

__HAL_DTS_GET_IT_SOURCE

Description:

- Check whether the specified DTS interrupt source is enabled or not.

Parameters:

- __HANDLE__: DTS handle.
- __INTERRUPT__: DTS interrupt source to check This parameter can be one of the following values:
  - DTS_IT_TS1_ITE : interrupt flag for end of measure for DTS1
  - DTS_IT_TS1_ITL : interrupt flag for low of measure for DTS1
  - DTS_IT_TS1_ITH : interrupt flag for high of measure for DTS1
  - DTS_IT_TS1_AITE : asynchronous interrupt flag for end of measure for DTS1
  - DTS_IT_TS1_AITL : asynchronous interrupt flag for low of measure for DTS1
  - DTS_IT_TS1_AITH : asynchronous interrupt flag for high of measure for DTS1

Return value:

- State: of interruption (SET or RESET)

__HAL_DTS_GET_REFCLK

Description:

- Check whether the specified DTS REFCLK is selected.

Parameters:

- __HANDLE__: DTS handle.
- __REFCLK__: DTS reference clock to check This parameter can be one of the following values:
  - DTS_REFCLKSEL_LSE: Low speed REF clock
  - DTS_REFCLKSEL_PCLK: High speed REF clock

Return value:

- State: of the REF clock tested (SET or RESET)

__HAL_DTS_GET_TRIGGER

Description:

- Get Trigger.

Parameters:

- __HANDLE__: DTS handle.

Return value:

- One: of the following trigger DTS_TRIGGER_HW_NONE : No HW trigger (SW trigger) DTS_TRIGGER_LPTIMER1: LPTIMER1 trigger DTS_TRIGGER_LPTIMER2: LPTIMER2 trigger DTS_TRIGGER_LPTIMER3: LPTIMER3 trigger DTS_TRIGGER_EXTI13 : EXTI13 trigger

DTS EXTI Lines

DTS_EXTI_LINE_DTS1

EXTI line 88 connected to DTS1 output

DTS Flag Definitions

DTS_FLAG_TS1_ITE

Interrupt flag for end of measure for DTS1

DTS_FLAG_TS1_ITL

Interrupt flag for low threshold for DTS1

DTS_FLAG_TS1_ITH

Interrupt flag for high threshold for DTS1

DTS_FLAG_TS1_AITE

Asynchronous Interrupt flag for end of measure for DTS1

DTS_FLAG_TS1_AITL

Asynchronous Interrupt flag for low threshold for DTS1

DTS_FLAG_TS1_AITH

Asynchronous Interrupt flag for high threshold for DTS1

DTS_FLAG_TS1_RDY

Ready flag for DTS1

DTS Interrupts Definitions

DTS_IT_TS1_ITE

Enable interrupt flag for end of measure for DTS1

DTS_IT_TS1_ITL

Enable interrupt flag for low threshold for DTS1

DTS_IT_TS1_ITH

Enable interrupt flag for high threshold for DTS1

DTS_IT_TS1_AITE

Enable asynchronous interrupt flag for end of measure for DTS1

DTS_IT_TS1_AITL

Enable asynchronous interrupt flag for low threshold for DTS1

DTS_IT_TS1_AITH

Enable asynchronous interrupt flag for high threshold for DTS1

DTS Private macros to check input parameters

IS_DTS_QUICKMEAS

IS_DTS_REFCLK

IS_DTS_TRIGGERINPUT

IS_DTS_THRESHOLD

IS_DTS_DIVIDER_RATIO_NUMBER

IS_DTS_SAMPLINGTIME

DTS Quick Measurement

DTS_QUICKMEAS_ENABLE

Enable the Quick Measure (Measure without calibration)

DTS_QUICKMEAS_DISABLE

Disable the Quick Measure (Measure with calibration)

DTS Reference Clock Selection

DTS_REFCLKSEL_LSE

Low speed REF clock (LSE)

DTS_REFCLKSEL_PCLK

High speed REF clock (PCLK)

DTS Sampling Time

DTS_SMP_TIME_1_CYCLE

1 clock cycle for the sampling time

DTS_SMP_TIME_2_CYCLE

2 clock cycle for the sampling time

DTS_SMP_TIME_3_CYCLE

3 clock cycle for the sampling time

DTS_SMP_TIME_4_CYCLE

4 clock cycle for the sampling time

DTS_SMP_TIME_5_CYCLE

5 clock cycle for the sampling time

DTS_SMP_TIME_6_CYCLE

6 clock cycle for the sampling time

DTS_SMP_TIME_7_CYCLE

7 clock cycle for the sampling time

DTS_SMP_TIME_8_CYCLE

8 clock cycle for the sampling time

DTS_SMP_TIME_9_CYCLE

9 clock cycle for the sampling time

DTS_SMP_TIME_10_CYCLE

10 clock cycle for the sampling time

DTS_SMP_TIME_11_CYCLE

11 clock cycle for the sampling time

DTS_SMP_TIME_12_CYCLE

12 clock cycle for the sampling time

DTS_SMP_TIME_13_CYCLE

13 clock cycle for the sampling time

DTS_SMP_TIME_14_CYCLE

14 clock cycle for the sampling time

DTS_SMP_TIME_15_CYCLE

15 clock cycle for the sampling time

DTS Trigger Configuration

DTS_TRIGGER_HW_NONE

DTS_TRIGGER_LPTIMER1

DTS_TRIGGER_LPTIMER2

DTS_TRIGGER_LPTIMER3

DTS_TRIGGER_EXTI13

27 HAL ETH Generic Driver

27.1 ETH Firmware driver registers structures

27.1.1 ETH_DMADescTypeDef

ETH_DMADescTypeDef is defined in the stm32h7xx_hal_eth.h

Data Fields

- ***__IO uint32_t DESC0***
- ***__IO uint32_t DESC1***
- ***__IO uint32_t DESC2***
- ***__IO uint32_t DESC3***
- ***uint32_t BackupAddr0***
- ***uint32_t BackupAddr1***

Field Documentation

- ***__IO uint32_t ETH_DMADescTypeDef::DESC0***
- ***__IO uint32_t ETH_DMADescTypeDef::DESC1***
- ***__IO uint32_t ETH_DMADescTypeDef::DESC2***
- ***__IO uint32_t ETH_DMADescTypeDef::DESC3***
- ***uint32_t ETH_DMADescTypeDef::BackupAddr0***
- ***uint32_t ETH_DMADescTypeDef::BackupAddr1***

27.1.2 __ETH_BufferTypeDef

__ETH_BufferTypeDef is defined in the stm32h7xx_hal_eth.h

Data Fields

- ***uint8_t * buffer***
- ***uint32_t len***
- ***struct __ETH_BufferTypeDef * next***

Field Documentation

- ***uint8_t* __ETH_BufferTypeDef::buffer***
- ***uint32_t __ETH_BufferTypeDef::len***
- ***struct __ETH_BufferTypeDef* __ETH_BufferTypeDef::next***

27.1.3 ETH_TxDescListTypeDef

ETH_TxDescListTypeDef is defined in the stm32h7xx_hal_eth.h

Data Fields

- ***uint32_t TxDesc***
- ***uint32_t CurTxDesc***
- ***uint32_t * PacketAddress***
- ***uint32_t * CurrentPacketAddress***
- ***uint32_t BuffersInUse***
- ***uint32_t releaseIndex***

Field Documentation

- ***uint32_t ETH_TxDescListTypeDef::TxDesc[ETH_TX_DESC_CNT]***

- ***uint32_t ETH_TxDescListTypeDef::CurTxDesc***
- ***uint32_t* ETH_TxDescListTypeDef::PacketAddress[ETH_TX_DESC_CNT]***
- ***uint32_t* ETH_TxDescListTypeDef::CurrentPacketAddress***
- ***uint32_t ETH_TxDescListTypeDef::BuffersInUse***
- ***uint32_t ETH_TxDescListTypeDef::releaseIndex***

27.1.4 ETH_TxPacketConfig

ETH_TxPacketConfig is defined in the stm32h7xx_hal_eth.h

Data Fields

- ***uint32_t Attributes***
- ***uint32_t Length***
- ***ETH_BufferTypeDef * TxBuffer***
- ***uint32_t SrcAddrCtrl***
- ***uint32_t CRCPadCtrl***
- ***uint32_t ChecksumCtrl***
- ***uint32_t MaxSegmentSize***
- ***uint32_t PayloadLen***
- ***uint32_t TCPHeaderLen***
- ***uint32_t VlanTag***
- ***uint32_t VlanCtrl***
- ***uint32_t InnerVlanTag***
- ***uint32_t InnerVlanCtrl***
- ***void * pData***

Field Documentation

- ***uint32_t ETH_TxPacketConfig::Attributes***
  Tx packet HW features capabilities. This parameter can be a combination of ***ETH_Tx_Packet_Attributes***
- ***uint32_t ETH_TxPacketConfig::Length***
  Total packet length
- ***ETH_BufferTypeDef* ETH_TxPacketConfig::TxBuffer***
  Tx buffers pointers
- ***uint32_t ETH_TxPacketConfig::SrcAddrCtrl***
  Specifies the source address insertion control. This parameter can be a value of ***ETH_Tx_Packet_Source_Addr_Control***
- ***uint32_t ETH_TxPacketConfig::CRCPadCtrl***
  Specifies the CRC and Pad insertion and replacement control. This parameter can be a value of ***ETH_Tx_Packet_CRC_Pad_Control***
- ***uint32_t ETH_TxPacketConfig::ChecksumCtrl***
  Specifies the checksum insertion control. This parameter can be a value of ***ETH_Tx_Packet_Checksum_Control***
- ***uint32_t ETH_TxPacketConfig::MaxSegmentSize***
  Sets TCP maximum segment size only when TCP segmentation is enabled. This parameter can be a value from 0x0 to 0x3FFF
- ***uint32_t ETH_TxPacketConfig::PayloadLen***
  Sets Total payload length only when TCP segmentation is enabled. This parameter can be a value from 0x0 to 0x3FFFF
- ***uint32_t ETH_TxPacketConfig::TCPHeaderLen***
  Sets TCP header length only when TCP segmentation is enabled. This parameter can be a value from 0x5 to 0xF

- ***uint32_t ETH_TxPacketConfig::VlanTag***

  Sets VLAN Tag only when VLAN is enabled. This parameter can be a value from 0x0 to 0xFFFF
- ***uint32_t ETH_TxPacketConfig::VlanCtrl***

  Specifies VLAN Tag insertion control only when VLAN is enabled. This parameter can be a value of ***ETH_Tx_Packet_VLAN_Control***
- ***uint32_t ETH_TxPacketConfig::InnerVlanTag***

  Sets Inner VLAN Tag only when Inner VLAN is enabled. This parameter can be a value from 0x0 to 0x3FFFF
- ***uint32_t ETH_TxPacketConfig::InnerVlanCtrl***

  Specifies Inner VLAN Tag insertion control only when Inner VLAN is enabled. This parameter can be a value of ***ETH_Tx_Packet_Inner_VLAN_Control***
- ***void* ETH_TxPacketConfig::pData***

  Specifies Application packet pointer to save

27.1.5 ETH_TimeStampTypeDef

ETH_TimeStampTypeDef is defined in the stm32h7xx_hal_eth.h

Data Fields

- ***uint32_t TimeStampLow***
- ***uint32_t TimeStampHigh***

Field Documentation

- ***uint32_t ETH_TimeStampTypeDef::TimeStampLow***
- ***uint32_t ETH_TimeStampTypeDef::TimeStampHigh***

27.1.6 ETH_TimeTypeDef

ETH_TimeTypeDef is defined in the stm32h7xx_hal_eth.h

Data Fields

- ***uint32_t Seconds***
- ***uint32_t NanoSeconds***

Field Documentation

- ***uint32_t ETH_TimeTypeDef::Seconds***
- ***uint32_t ETH_TimeTypeDef::NanoSeconds***

27.1.7 ETH_RxDescListTypeDef

ETH_RxDescListTypeDef is defined in the stm32h7xx_hal_eth.h

Data Fields

- ***uint32_t RxDesc***
- ***uint32_t ItMode***
- ***uint32_t RxDescIdx***
- ***uint32_t RxDescCnt***
- ***uint32_t RxDataLength***
- ***uint32_t RxBuildDescIdx***
- ***uint32_t RxBuildDescCnt***
- ***uint32_t pRxLastRxDesc***
- ***ETH_TimeStampTypeDef TimeStamp***
- ***void * pRxStart***
- ***void * pRxEnd***

Field Documentation

- ***uint32_t ETH_RxDescListTypeDef::RxDesc[ETH_RX_DESC_CNT]***

- *uint32_t ETH_RxDescListTypeDef::ItMode*
- *uint32_t ETH_RxDescListTypeDef::RxDescIdx*
- *uint32_t ETH_RxDescListTypeDef::RxDescCnt*
- *uint32_t ETH_RxDescListTypeDef::RxDataLength*
- *uint32_t ETH_RxDescListTypeDef::RxBuildDescIdx*
- *uint32_t ETH_RxDescListTypeDef::RxBuildDescCnt*
- *uint32_t ETH_RxDescListTypeDef::pRxLastRxDesc*
- *ETH_TimeStampTypeDef ETH_RxDescListTypeDef::TimeStamp*
- *void* ETH_RxDescListTypeDef::pRxStart*
- *void* ETH_RxDescListTypeDef::pRxEnd*

27.1.8 ETH_MACConfigTypeDef

ETH_MACConfigTypeDef is defined in the stm32h7xx_hal_eth.h

Data Fields

- *uint32_t SourceAddrControl*
- *FunctionalState ChecksumOffload*
- *uint32_t InterPacketGapVal*
- *FunctionalState GiantPacketSizeLimitControl*
- *FunctionalState Support2KPacket*
- *FunctionalState CRCStripTypePacket*
- *FunctionalState AutomaticPadCRCStrip*
- *FunctionalState Watchdog*
- *FunctionalState Jabber*
- *FunctionalState JumboPacket*
- *uint32_t Speed*
- *uint32_t DuplexMode*
- *FunctionalState LoopbackMode*
- *FunctionalState CarrierSenseBeforeTransmit*
- *FunctionalState ReceiveOwn*
- *FunctionalState CarrierSenseDuringTransmit*
- *FunctionalState RetryTransmission*
- *uint32_t BackOffLimit*
- *FunctionalState DeferralCheck*
- *uint32_t PreambleLength*
- *FunctionalState UnicastSlowProtocolPacketDetect*
- *FunctionalState SlowProtocolDetect*
- *FunctionalState CRCCheckingRxPackets*
- *uint32_t GiantPacketSizeLimit*
- *FunctionalState ExtendedInterPacketGap*
- *uint32_t ExtendedInterPacketGapVal*
- *FunctionalState ProgrammableWatchdog*
- *uint32_t WatchdogTimeout*
- *uint32_t PauseTime*
- *FunctionalState ZeroQuantaPause*
- *uint32_t PauseLowThreshold*
- *FunctionalState TransmitFlowControl*
- *FunctionalState UnicastPausePacketDetect*
- *FunctionalState ReceiveFlowControl*

- ***uint32_t TransmitQueueMode***
- ***uint32_t ReceiveQueueMode***
- ***FunctionalState DropTCPIPChecksumErrorPacket***
- ***FunctionalState ForwardRxErrorPacket***
- ***FunctionalState ForwardRxUndersizedGoodPacket***

Field Documentation

- ***uint32_t ETH_MACConfigTypeDef::SourceAddrControl***
  Selects the Source Address Insertion or Replacement Control. This parameter can be a value of ***ETH_Source_Addr_Control***
- ***FunctionalState ETH_MACConfigTypeDef::ChecksumOffload***
  Enables or Disable the checksum checking for received packet payloads TCP, UDP or ICMP headers
- ***uint32_t ETH_MACConfigTypeDef::InterPacketGapVal***
  Sets the minimum IPG between Packet during transmission. This parameter can be a value of ***ETH_Inter_Packet_Gap***
- ***FunctionalState ETH_MACConfigTypeDef::GiantPacketSizeLimitControl***
  Enables or disables the Giant Packet Size Limit Control.
- ***FunctionalState ETH_MACConfigTypeDef::Support2KPacket***
  Enables or disables the IEEE 802.3as Support for 2K length Packets
- ***FunctionalState ETH_MACConfigTypeDef::CRCStripTypePacket***
  Enables or disables the CRC stripping for Type packets.
- ***FunctionalState ETH_MACConfigTypeDef::AutomaticPadCRCStrip***
  Enables or disables the Automatic MAC Pad/CRC Stripping.
- ***FunctionalState ETH_MACConfigTypeDef::Watchdog***
  Enables or disables the Watchdog timer on Rx path.
- ***FunctionalState ETH_MACConfigTypeDef::Jabber***
  Enables or disables Jabber timer on Tx path.
- ***FunctionalState ETH_MACConfigTypeDef::JumboPacket***
  Enables or disables receiving Jumbo Packet When enabled, the MAC allows jumbo packets of 9,018 bytes without reporting a giant packet error
- ***uint32_t ETH_MACConfigTypeDef::Speed***
  Sets the Ethernet speed: 10/100 Mbps. This parameter can be a value of ***ETH_Speed***
- ***uint32_t ETH_MACConfigTypeDef::DuplexMode***
  Selects the MAC duplex mode: Half-Duplex or Full-Duplex mode This parameter can be a value of ***ETH_Duplex_Mode***
- ***FunctionalState ETH_MACConfigTypeDef::LoopbackMode***
  Enables or disables the loopback mode
- ***FunctionalState ETH_MACConfigTypeDef::CarrierSenseBeforeTransmit***
  Enables or disables the Carrier Sense Before Transmission in Full Duplex Mode.
- ***FunctionalState ETH_MACConfigTypeDef::ReceiveOwn***
  Enables or disables the Receive Own in Half Duplex mode.
- ***FunctionalState ETH_MACConfigTypeDef::CarrierSenseDuringTransmit***
  Enables or disables the Carrier Sense During Transmission in the Half Duplex mode
- ***FunctionalState ETH_MACConfigTypeDef::RetryTransmission***
  Enables or disables the MAC retry transmission, when a collision occurs in Half Duplex mode.
- ***uint32_t ETH_MACConfigTypeDef::BackOffLimit***
  Selects the BackOff limit value. This parameter can be a value of ***ETH_Back_Off_Limit***
- ***FunctionalState ETH_MACConfigTypeDef::DeferralCheck***
  Enables or disables the deferral check function in Half Duplex mode.

- ***uint32_t ETH_MACConfigTypeDef::PreambleLength***
  Selects or not the Preamble Length for Transmit packets (Full Duplex mode). This parameter can be a value of ***ETH_Preamble_Length***
- ***FunctionalState ETH_MACConfigTypeDef::UnicastSlowProtocolPacketDetect***
  Enable or disables the Detection of Slow Protocol Packets with unicast address.
- ***FunctionalState ETH_MACConfigTypeDef::SlowProtocolDetect***
  Enable or disables the Slow Protocol Detection.
- ***FunctionalState ETH_MACConfigTypeDef::CRCCheckingRxPackets***
  Enable or disables the CRC Checking for Received Packets.
- ***uint32_t ETH_MACConfigTypeDef::GiantPacketSizeLimit***
  Specifies the packet size that the MAC will declare it as Giant, If it's size is greater than the value programmed in this field in units of bytes This parameter must be a number between Min_Data = 0x618 (1518 byte) and Max_Data = 0x3FFF (32 Kbyte).
- ***FunctionalState ETH_MACConfigTypeDef::ExtendedInterPacketGap***
  Enable or disables the extended inter packet gap.
- ***uint32_t ETH_MACConfigTypeDef::ExtendedInterPacketGapVal***
  Sets the Extended IPG between Packet during transmission. This parameter can be a value from 0x0 to 0xFF
- ***FunctionalState ETH_MACConfigTypeDef::ProgrammableWatchdog***
  Enable or disables the Programmable Watchdog.
- ***uint32_t ETH_MACConfigTypeDef::WatchdogTimeout***
  This field is used as watchdog timeout for a received packet This parameter can be a value of ***ETH_Watchdog_Timeout***
- ***uint32_t ETH_MACConfigTypeDef::PauseTime***
  This field holds the value to be used in the Pause Time field in the transmit control packet. This parameter must be a number between Min_Data = 0x0 and Max_Data = 0xFFFF.
- ***FunctionalState ETH_MACConfigTypeDef::ZeroQuantaPause***
  Enable or disables the automatic generation of Zero Quanta Pause Control packets.
- ***uint32_t ETH_MACConfigTypeDef::PauseLowThreshold***
  This field configures the threshold of the PAUSE to be checked for automatic retransmission of PAUSE Packet. This parameter can be a value of ***ETH_Pause_Low_Threshold***
- ***FunctionalState ETH_MACConfigTypeDef::TransmitFlowControl***
  Enables or disables the MAC to transmit Pause packets in Full Duplex mode or the MAC back pressure operation in Half Duplex mode
- ***FunctionalState ETH_MACConfigTypeDef::UnicastPausePacketDetect***
  Enables or disables the MAC to detect Pause packets with unicast address of the station
- ***FunctionalState ETH_MACConfigTypeDef::ReceiveFlowControl***
  Enables or disables the MAC to decodes the received Pause packet and disables its transmitter for a specified (Pause) time
- ***uint32_t ETH_MACConfigTypeDef::TransmitQueueMode***
  Specifies the Transmit Queue operating mode. This parameter can be a value of ***ETH_Transmit_Mode***
- ***uint32_t ETH_MACConfigTypeDef::ReceiveQueueMode***
  Specifies the Receive Queue operating mode. This parameter can be a value of ***ETH_Receive_Mode***
- ***FunctionalState ETH_MACConfigTypeDef::DropTCPIPChecksumErrorPacket***
  Enables or disables Dropping of TCPIP Checksum Error Packets.
- ***FunctionalState ETH_MACConfigTypeDef::ForwardRxErrorPacket***
  Enables or disables forwarding Error Packets.
- ***FunctionalState ETH_MACConfigTypeDef::ForwardRxUndersizedGoodPacket***
  Enables or disables forwarding Undersized Good Packets.

27.1.9 ETH_DMAConfigTypeDef

ETH_DMAConfigTypeDef is defined in the stm32h7xx_hal_eth.h

Data Fields

- ***uint32_t DMAArbitration***
- ***FunctionalState AddressAlignedBeats***
- ***uint32_t BurstMode***
- ***FunctionalState RebuildINCRxBurst***
- ***FunctionalState PBLx8Mode***
- ***uint32_t TxDMABurstLength***
- ***FunctionalState SecondPacketOperate***
- ***uint32_t RxDMABurstLength***
- ***FunctionalState FlushRxPacket***
- ***FunctionalState TCPSegmentation***
- ***uint32_t MaximumSegmentSize***

Field Documentation

- ***uint32_t ETH_DMAConfigTypeDef::DMAArbitration***

  Sets the arbitration scheme between DMA Tx and Rx This parameter can be a value of ***ETH_DMA_Arbitration***
- ***FunctionalState ETH_DMAConfigTypeDef::AddressAlignedBeats***

  Enables or disables the AHB Master interface address aligned burst transfers on Read and Write channels
- ***uint32_t ETH_DMAConfigTypeDef::BurstMode***

  Sets the AHB Master interface burst transfers. This parameter can be a value of ***ETH_Burst_Mode***
- ***FunctionalState ETH_DMAConfigTypeDef::RebuildINCRxBurst***

  Enables or disables the AHB Master to rebuild the pending beats of any initiated burst transfer with INCRx and SINGLE transfers.
- ***FunctionalState ETH_DMAConfigTypeDef::PBLx8Mode***

  Enables or disables the PBL multiplication by eight.
- ***uint32_t ETH_DMAConfigTypeDef::TxDMABurstLength***

  Indicates the maximum number of beats to be transferred in one Tx DMA transaction. This parameter can be a value of ***ETH_Tx_DMA_Burst_Length***
- ***FunctionalState ETH_DMAConfigTypeDef::SecondPacketOperate***

  Enables or disables the Operate on second Packet mode, which allows the DMA to process a second Packet of Transmit data even before obtaining the status for the first one.
- ***uint32_t ETH_DMAConfigTypeDef::RxDMABurstLength***

  Indicates the maximum number of beats to be transferred in one Rx DMA transaction. This parameter can be a value of ***ETH_Rx_DMA_Burst_Length***
- ***FunctionalState ETH_DMAConfigTypeDef::FlushRxPacket***

  Enables or disables the Rx Packet Flush
- ***FunctionalState ETH_DMAConfigTypeDef::TCPSegmentation***

  Enables or disables the TCP Segmentation
- ***uint32_t ETH_DMAConfigTypeDef::MaximumSegmentSize***

  Sets the maximum segment size that should be used while segmenting the packet This parameter can be a value from 0x40 to 0x3FFF

27.1.10 ETH_InitTypeDef

ETH_InitTypeDef is defined in the stm32h7xx_hal_eth.h

Data Fields

- ***uint8_t * MACAddr***
- ***ETH_MediaInterfaceTypeDef MediaInterface***
- ***ETH_DMADescTypeDef * TxDesc***

- ***ETH_DMADescTypeDef * RxDesc***
- ***uint32_t RxBuffLen***

Field Documentation

- ***uint8_t* ETH_InitTypeDef::MACAddr***
  MAC Address of used Hardware: must be pointer on an array of 6 bytes
- ***ETH_MediaInterfaceTypeDef ETH_InitTypeDef::MediaInterface***
  Selects the MII interface or the RMII interface.
- ***ETH_DMADescTypeDef* ETH_InitTypeDef::TxDesc***
  Provides the address of the first DMA Tx descriptor in the list
- ***ETH_DMADescTypeDef* ETH_InitTypeDef::RxDesc***
  Provides the address of the first DMA Rx descriptor in the list
- ***uint32_t ETH_InitTypeDef::RxBuffLen***
  Provides the length of Rx buffers size

27.1.11 ETH_PTP_ConfigTypeDef

ETH_PTP_ConfigTypeDef is defined in the stm32h7xx_hal_eth.h

Data Fields

- ***uint32_t Timestamp***
- ***uint32_t TimestampUpdateMode***
- ***uint32_t TimestampInitialize***
- ***uint32_t TimestampUpdate***
- ***uint32_t TimestampAddendUpdate***
- ***uint32_t TimestampAll***
- ***uint32_t TimestampRolloverMode***
- ***uint32_t TimestampV2***
- ***uint32_t TimestampEthernet***
- ***uint32_t TimestampIPv6***
- ***uint32_t TimestampIPv4***
- ***uint32_t TimestampEvent***
- ***uint32_t TimestampMaster***
- ***uint32_t TimestampSnapshots***
- ***uint32_t TimestampFilter***
- ***uint32_t TimestampChecksumCorrection***
- ***uint32_t TimestampStatusMode***
- ***uint32_t TimestampAddend***
- ***uint32_t TimestampSubsecondInc***

Field Documentation

- ***uint32_t ETH_PTP_ConfigTypeDef::Timestamp***
  Enable Timestamp
- ***uint32_t ETH_PTP_ConfigTypeDef::TimestampUpdateMode***
  Fine or Coarse Timestamp Update
- ***uint32_t ETH_PTP_ConfigTypeDef::TimestampInitialize***
  Initialize Timestamp
- ***uint32_t ETH_PTP_ConfigTypeDef::TimestampUpdate***
  Timestamp Update
- ***uint32_t ETH_PTP_ConfigTypeDef::TimestampAddendUpdate***
  Timestamp Addend Update
- ***uint32_t ETH_PTP_ConfigTypeDef::TimestampAll***
  Enable Timestamp for All Packets

- ***uint32_t ETH_PTP_ConfigTypeDef::TimestampRolloverMode***
  Timestamp Digital or Binary Rollover Control
- ***uint32_t ETH_PTP_ConfigTypeDef::TimestampV2***
  Enable PTP Packet Processing for Version 2 Format
- ***uint32_t ETH_PTP_ConfigTypeDef::TimestampEthernet***
  Enable Processing of PTP over Ethernet Packets
- ***uint32_t ETH_PTP_ConfigTypeDef::TimestampIPv6***
  Enable Processing of PTP Packets Sent over IPv6-UDP
- ***uint32_t ETH_PTP_ConfigTypeDef::TimestampIPv4***
  Enable Processing of PTP Packets Sent over IPv4-UDP
- ***uint32_t ETH_PTP_ConfigTypeDef::TimestampEvent***
  Enable Timestamp Snapshot for Event Messages
- ***uint32_t ETH_PTP_ConfigTypeDef::TimestampMaster***
  Enable Timestamp Snapshot for Event Messages
- ***uint32_t ETH_PTP_ConfigTypeDef::TimestampSnapshots***
  Select PTP packets for Taking Snapshots
- ***uint32_t ETH_PTP_ConfigTypeDef::TimestampFilter***
  Enable MAC Address for PTP Packet Filtering
- ***uint32_t ETH_PTP_ConfigTypeDef::TimestampChecksumCorrection***
  Enable checksum correction during OST for PTP over UDP/IPv4 packets
- ***uint32_t ETH_PTP_ConfigTypeDef::TimestampStatusMode***
  Transmit Timestamp Status Mode
- ***uint32_t ETH_PTP_ConfigTypeDef::TimestampAddend***
  Timestamp addend value
- ***uint32_t ETH_PTP_ConfigTypeDef::TimestampSubsecondInc***
  Subsecond Increment

27.1.12 __ETH_HandleTypeDef

__ETH_HandleTypeDef is defined in the stm32h7xx_hal_eth.h

Data Fields

- ***ETH_TypeDef * Instance***
- ***ETH_InitTypeDef Init***
- ***ETH_TxDescListTypeDef TxDescList***
- ***ETH_RxDescListTypeDef RxDescList***
- ***__IO HAL_ETH_StateTypeDef gState***
- ***__IO uint32_t ErrorCode***
- ***__IO uint32_t DMAErrorCode***
- ***__IO uint32_t MACErrorCode***
- ***__IO uint32_t MACWakeUpEvent***
- ***__IO uint32_t MACLPIEvent***
- ***__IO uint32_t IsPtpConfigured***
- ***void(* TxCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* ErrorCallback***
- ***void(* PMTCallback***
- ***void(* EEECallback***
- ***void(* WakeUpCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

- ***pETH_rxAllocateCallbackTypeDef rxAllocateCallback***
- ***pETH_rxLinkCallbackTypeDef rxLinkCallback***
- ***pETH_txFreeCallbackTypeDef txFreeCallback***
- ***pETH_txPtpCallbackTypeDef txPtpCallback***

Field Documentation

- ***ETH_TypeDef* __ETH_HandleTypeDef::Instance***
  Register base address
- ***ETH_InitTypeDef __ETH_HandleTypeDef::Init***
  Ethernet Init Configuration
- ***ETH_TxDescListTypeDef __ETH_HandleTypeDef::TxDescList***
  Tx descriptor wrapper: holds all Tx descriptors list addresses and current descriptor index
- ***ETH_RxDescListTypeDef __ETH_HandleTypeDef::RxDescList***
  Rx descriptor wrapper: holds all Rx descriptors list addresses and current descriptor index
- ***__IO HAL_ETH_StateTypeDef __ETH_HandleTypeDef::gState***
  ETH state information related to global Handle management and also related to Tx operations. This parameter can be a value of ***HAL_ETH_StateTypeDef***
- ***__IO uint32_t __ETH_HandleTypeDef::ErrorCode***
  Holds the global Error code of the ETH HAL status machine This parameter can be a value of ***ETH_Error_Code***.
- ***__IO uint32_t __ETH_HandleTypeDef::DMAErrorCode***
  Holds the DMA Rx Tx Error code when a DMA AIS interrupt occurs This parameter can be a combination of ***ETH_DMA_Status_Flags***
- ***__IO uint32_t __ETH_HandleTypeDef::MACErrorCode***
  Holds the MAC Rx Tx Error code when a MAC Rx or Tx status interrupt occurs This parameter can be a combination of ***ETH_MAC_Rx_Tx_Status***
- ***__IO uint32_t __ETH_HandleTypeDef::MACWakeUpEvent***
  Holds the Wake Up event when the MAC exit the power down mode This parameter can be a value of ***ETH_MAC_Wake_Up_Event***
- ***__IO uint32_t __ETH_HandleTypeDef::MACLPIEvent***
  Holds the LPI event when the an LPI status interrupt occurs. This parameter can be a value of ***ETHEx_LPI_Event***
- ***__IO uint32_t __ETH_HandleTypeDef::IsPtpConfigured***
  Holds the PTP configuration status. This parameter can be a value of ***ETH_PTP_Config_Status***
- ***void(* __ETH_HandleTypeDef::TxCpltCallback)(struct __ETH_HandleTypeDef *heth)***
  ETH Tx Complete Callback
- ***void(* __ETH_HandleTypeDef::RxCpltCallback)(struct __ETH_HandleTypeDef *heth)***
  ETH Rx Complete Callback
- ***void(* __ETH_HandleTypeDef::ErrorCallback)(struct __ETH_HandleTypeDef *heth)***
  ETH Error Callback
- ***void(* __ETH_HandleTypeDef::PMTCallback)(struct __ETH_HandleTypeDef *heth)***
  ETH Power Management Callback
- ***void(* __ETH_HandleTypeDef::EEECallback)(struct __ETH_HandleTypeDef *heth)***
  ETH EEE Callback
- ***void(* __ETH_HandleTypeDef::WakeUpCallback)(struct __ETH_HandleTypeDef *heth)***
  ETH Wake UP Callback
- ***void(* __ETH_HandleTypeDef::MspInitCallback)(struct __ETH_HandleTypeDef *heth)***
  ETH Msp Init callback
- ***void(* __ETH_HandleTypeDef::MspDeInitCallback)(struct __ETH_HandleTypeDef *heth)***
  ETH Msp DeInit callback

- ***pETH_rxAllocateCallbackTypeDef __ETH_HandleTypeDef::rxAllocateCallback***
  ETH Rx Get Buffer Function
- ***pETH_rxLinkCallbackTypeDef __ETH_HandleTypeDef::rxLinkCallback***
  ETH Rx Set App Data Function
- ***pETH_txFreeCallbackTypeDef __ETH_HandleTypeDef::txFreeCallback***
  ETH Tx Free Function
- ***pETH_txPtpCallbackTypeDef __ETH_HandleTypeDef::txPtpCallback***
  ETH Tx Handle Ptp Function

27.1.13 ETH_MACFilterConfigTypeDef

ETH_MACFilterConfigTypeDef is defined in the stm32h7xx_hal_eth.h

Data Fields

- ***FunctionalState PromiscuousMode***
- ***FunctionalState ReceiveAllMode***
- ***FunctionalState HachOrPerfectFilter***
- ***FunctionalState HashUnicast***
- ***FunctionalState HashMulticast***
- ***FunctionalState PassAllMulticast***
- ***FunctionalState SrcAddrFiltering***
- ***FunctionalState SrcAddrInverseFiltering***
- ***FunctionalState DestAddrInverseFiltering***
- ***FunctionalState BroadcastFilter***
- ***uint32_t ControlPacketsFilter***

Field Documentation

- ***FunctionalState ETH_MACFilterConfigTypeDef::PromiscuousMode***
  Enable or Disable Promiscuous Mode
- ***FunctionalState ETH_MACFilterConfigTypeDef::ReceiveAllMode***
  Enable or Disable Receive All Mode
- ***FunctionalState ETH_MACFilterConfigTypeDef::HachOrPerfectFilter***
  Enable or Disable Perfect filtering in addition to Hash filtering
- ***FunctionalState ETH_MACFilterConfigTypeDef::HashUnicast***
  Enable or Disable Hash filtering on unicast packets
- ***FunctionalState ETH_MACFilterConfigTypeDef::HashMulticast***
  Enable or Disable Hash filtering on multicast packets
- ***FunctionalState ETH_MACFilterConfigTypeDef::PassAllMulticast***
  Enable or Disable passing all multicast packets
- ***FunctionalState ETH_MACFilterConfigTypeDef::SrcAddrFiltering***
  Enable or Disable source address filtering module
- ***FunctionalState ETH_MACFilterConfigTypeDef::SrcAddrInverseFiltering***
  Enable or Disable source address inverse filtering
- ***FunctionalState ETH_MACFilterConfigTypeDef::DestAddrInverseFiltering***
  Enable or Disable destination address inverse filtering
- ***FunctionalState ETH_MACFilterConfigTypeDef::BroadcastFilter***
  Enable or Disable broadcast filter
- ***uint32_t ETH_MACFilterConfigTypeDef::ControlPacketsFilter***
  Set the control packets filter This parameter can be a value of ***ETH_Control_Packets_Filter***

27.1.14 ETH_PowerDownConfigTypeDef

ETH_PowerDownConfigTypeDef is defined in the stm32h7xx_hal_eth.h

Data Fields

- ***FunctionalState WakeUpPacket***
- ***FunctionalState MagicPacket***
- ***FunctionalState GlobalUnicast***
- ***FunctionalState WakeUpForward***

Field Documentation

- ***FunctionalState ETH_PowerDownConfigTypeDef::WakeUpPacket***
  Enable or Disable Wake up packet detection in power down mode
- ***FunctionalState ETH_PowerDownConfigTypeDef::MagicPacket***
  Enable or Disable Magic packet detection in power down mode
- ***FunctionalState ETH_PowerDownConfigTypeDef::GlobalUnicast***
  Enable or Disable Global unicast packet detection in power down mode
- ***FunctionalState ETH_PowerDownConfigTypeDef::WakeUpForward***
  Enable or Disable Forwarding Wake up packets

27.2 ETH Firmware driver API description

The following section lists the various functions of the ETH library.

27.2.1 How to use this driver

The ETH HAL driver can be used as follows:

1. Declare a ETH_HandleTypeDef handle structure, for example: ETH_HandleTypeDef heth;
2. Fill parameters of Init structure in heth handle
3. Call HAL_ETH_Init() API to initialize the Ethernet peripheral (MAC, DMA, ...)
4. Initialize the ETH low level resources through the HAL_ETH_MspInit() API:
   a. Enable the Ethernet interface clock using
      - __HAL_RCC_ETH1MAC_CLK_ENABLE()
      - __HAL_RCC_ETH1TX_CLK_ENABLE()
      - __HAL_RCC_ETH1RX_CLK_ENABLE()
   b. Initialize the related GPIO clocks
   c. Configure Ethernet pinout
   d. Configure Ethernet NVIC interrupt (in Interrupt mode)
5. Ethernet data reception is asynchronous, so call the following API to start the listening mode:
   a. HAL_ETH_Start(): This API starts the MAC and DMA transmission and reception process, without enabling end of transfer interrupts, in this mode user has to poll for data reception by calling HAL_ETH_ReadData()
   b. HAL_ETH_Start_IT(): This API starts the MAC and DMA transmission and reception process, end of transfer interrupts are enabled in this mode, HAL_ETH_RxCpltCallback() will be executed when an Ethernet packet is received
6. When data is received user can call the following API to get received data:
   a. HAL_ETH_ReadData(): Read a received packet
7. For transmission path, two APIs are available:
   a. HAL_ETH_Transmit(): Transmit an ETH frame in blocking mode
   b. HAL_ETH_Transmit_IT(): Transmit an ETH frame in interrupt mode, HAL_ETH_TxCpltCallback() will be executed when end of transfer occur
8. Communication with an external PHY device:
   a. HAL_ETH_ReadPHYRegister(): Read a register from an external PHY
   b. HAL_ETH_WritePHYRegister(): Write data to an external RHY register

9. Configure the Ethernet MAC after ETH peripheral initialization
   a. HAL_ETH_GetMACConfig(): Get MAC actual configuration into ETH_MACConfigTypeDef
   b. HAL_ETH_SetMACConfig(): Set MAC configuration based on ETH_MACConfigTypeDef
10. Configure the Ethernet DMA after ETH peripheral initialization
   a. HAL_ETH_GetDMAConfig(): Get DMA actual configuration into ETH_DMAConfigTypeDef
   b. HAL_ETH_SetDMAConfig(): Set DMA configuration based on ETH_DMAConfigTypeDef
11. Configure the Ethernet PTP after ETH peripheral initialization
   a. Define HAL_ETH_USE_PTP to use PTP APIs.
   b. HAL_ETH_PTP_GetConfig(): Get PTP actual configuration into ETH_PTP_ConfigTypeDef
   c. HAL_ETH_PTP_SetConfig(): Set PTP configuration based on ETH_PTP_ConfigTypeDef
   d. HAL_ETH_PTP_GetTime(): Get Seconds and Nanoseconds for the Ethernet PTP registers
   e. HAL_ETH_PTP_SetTime(): Set Seconds and Nanoseconds for the Ethernet PTP registers
   f. HAL_ETH_PTP_AddTimeOffset(): Add Seconds and Nanoseconds offset for the Ethernet PTP registers
   g. HAL_ETH_PTP_InsertTxTimestamp(): Insert Timestamp in transmission
   h. HAL_ETH_PTP_GetTxTimestamp(): Get transmission timestamp
   i. HAL_ETH_PTP_GetRxTimestamp(): Get reception timestamp -@- The ARP offload feature is not supported in this driver. -@- The PTP offload feature is not supported in this driver.

Callback registration

27.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize and deinitialize the ETH peripheral:

- User must Implement HAL_ETH_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO and NVIC).
- Call the function HAL_ETH_Init() to configure the selected device with the selected configuration:
  - MAC address
  - Media interface (MII or RMII)
  - Rx DMA Descriptors Tab
  - Tx DMA Descriptors Tab
  - Length of Rx Buffers
- Call the function HAL_ETH_DeInit() to restore the default configuration of the selected ETH peripheral.

This section contains the following APIs:

- ***HAL_ETH_Init()***
- ***HAL_ETH_DeInit()***
- ***HAL_ETH_MspInit()***
- ***HAL_ETH_MspDeInit()***
- ***HAL_ETH_RegisterCallback()***
- ***HAL_ETH_UnRegisterCallback()***

27.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the ETH data transfer.

This section contains the following APIs:

- ***HAL_ETH_Start()***
- ***HAL_ETH_Start_IT()***
- ***HAL_ETH_Stop()***
- ***HAL_ETH_Stop_IT()***
- ***HAL_ETH_Transmit()***
- ***HAL_ETH_Transmit_IT()***
- ***HAL_ETH_ReadData()***

- *HAL_ETH_RegisterRxAllocateCallback()*
- *HAL_ETH_UnRegisterRxAllocateCallback()*
- *HAL_ETH_RxAllocateCallback()*
- *HAL_ETH_RxLinkCallback()*
- *HAL_ETH_RegisterRxLinkCallback()*
- *HAL_ETH_UnRegisterRxLinkCallback()*
- *HAL_ETH_GetRxDataErrorCode()*
- *HAL_ETH_RegisterTxFreeCallback()*
- *HAL_ETH_UnRegisterTxFreeCallback()*
- *HAL_ETH_TxFreeCallback()*
- *HAL_ETH_ReleaseTxPacket()*
- *HAL_ETH_IRQHandler()*
- *HAL_ETH_TxCpltCallback()*
- *HAL_ETH_RxCpltCallback()*
- *HAL_ETH_ErrorCallback()*
- *HAL_ETH_PMTCallback()*
- *HAL_ETH_EEECallback()*
- *HAL_ETH_WakeUpCallback()*
- *HAL_ETH_ReadPHYRegister()*
- *HAL_ETH_WritePHYRegister()*
- *HAL_ETH_TxPtpCallback()*

27.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the ETH peripheral.

This section contains the following APIs:

- *HAL_ETH_GetMACConfig()*
- *HAL_ETH_GetDMAConfig()*
- *HAL_ETH_SetMACConfig()*
- *HAL_ETH_SetDMAConfig()*
- *HAL_ETH_SetMDIOClockRange()*
- *HAL_ETH_SetMACFilterConfig()*
- *HAL_ETH_GetMACFilterConfig()*
- *HAL_ETH_SetSourceMACAddrMatch()*
- *HAL_ETH_SetHashTable()*
- *HAL_ETH_SetRxVLANIdentifier()*
- *HAL_ETH_EnterPowerDownMode()*
- *HAL_ETH_ExitPowerDownMode()*
- *HAL_ETH_SetWakeUpFilter()*

27.2.5 Peripheral State and Errors functions

This subsection provides a set of functions allowing to return the State of ETH communication process, return Peripheral Errors occurred during communication process

This section contains the following APIs:

- *HAL_ETH_GetState()*
- *HAL_ETH_GetError()*
- *HAL_ETH_GetDMAError()*
- *HAL_ETH_GetMACError()*
- *HAL_ETH_GetMACWakeUpSource()*

27.2.6 Detailed description of functions

HAL_ETH_Init

Function name

HAL_StatusTypeDef HAL_ETH_Init (ETH_HandleTypeDef * heth)

Function description

Initialize the Ethernet peripheral registers.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** status

HAL_ETH_DeInit

Function name

HAL_StatusTypeDef HAL_ETH_DeInit (ETH_HandleTypeDef * heth)

Function description

DeInitializes the ETH peripheral.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** status

HAL_ETH_MspInit

Function name

void HAL_ETH_MspInit (ETH_HandleTypeDef * heth)

Function description

Initializes the ETH MSP.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETH_MspDeInit

Function name

void HAL_ETH_MspDeInit (ETH_HandleTypeDef * heth)

Function description

DeInitializes ETH MSP.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETH_RegisterCallback

Function name

HAL_StatusTypeDef HAL_ETH_RegisterCallback (ETH_HandleTypeDef * heth, HAL_ETH_CallbackIDTypeDef CallbackID, pETH_CallbackTypeDef pCallback)

Function description

Register a User ETH Callback To be used instead of the weak predefined callback.

Parameters

- **heth:** eth handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_ETH_TX_COMPLETE_CB_ID Tx Complete Callback ID
  - HAL_ETH_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_ETH_ERROR_CB_ID Error Callback ID
  - HAL_ETH_PMT_CB_ID Power Management Callback ID
  - HAL_ETH_EEE_CB_ID EEE Callback ID
  - HAL_ETH_WAKEUP_CB_ID Wake UP Callback ID
  - HAL_ETH_MSPINIT_CB_ID MspInit callback ID
  - HAL_ETH_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **status:**

HAL_ETH_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_ETH_UnRegisterCallback (ETH_HandleTypeDef * heth, HAL_ETH_CallbackIDTypeDef CallbackID)

Function description

Unregister an ETH Callback ETH callabck is redirected to the weak predefined callback.

Parameters

- **heth:** eth handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_ETH_TX_COMPLETE_CB_ID Tx Complete Callback ID
  - HAL_ETH_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_ETH_ERROR_CB_ID Error Callback ID
  - HAL_ETH_PMT_CB_ID Power Management Callback ID
  - HAL_ETH_EEE_CB_ID EEE Callback ID
  - HAL_ETH_WAKEUP_CB_ID Wake UP Callback ID
  - HAL_ETH_MSPINIT_CB_ID MspInit callback ID
  - HAL_ETH_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **status:**

HAL_ETH_Start

Function name

HAL_StatusTypeDef HAL_ETH_Start (ETH_HandleTypeDef * heth)

Function description

Enables Ethernet MAC and DMA reception and transmission.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** status

HAL_ETH_Start_IT

Function name

HAL_StatusTypeDef HAL_ETH_Start_IT (ETH_HandleTypeDef * heth)

Function description

Enables Ethernet MAC and DMA reception/transmission in Interrupt mode.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** status

HAL_ETH_Stop

Function name

HAL_StatusTypeDef HAL_ETH_Stop (ETH_HandleTypeDef * heth)

Function description

Stop Ethernet MAC and DMA reception/transmission.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** status

HAL_ETH_Stop_IT

Function name

HAL_StatusTypeDef HAL_ETH_Stop_IT (ETH_HandleTypeDef * heth)

Function description

Stop Ethernet MAC and DMA reception/transmission in Interrupt mode.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** status

HAL_ETH_ReadData

Function name

HAL_StatusTypeDef HAL_ETH_ReadData (ETH_HandleTypeDef * heth, void ** pAppBuff)

Function description

Read a received packet.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pAppBuff:** Pointer to an application buffer to receive the packet.

Return values

- **HAL:** status

HAL_ETH_RegisterRxAllocateCallback

Function name

HAL_StatusTypeDef HAL_ETH_RegisterRxAllocateCallback (ETH_HandleTypeDef * heth, pETH_rxAllocateCallbackTypeDef rxAllocateCallback)

Function description

Register the Rx alloc callback.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **rxAllocateCallback:** pointer to function to alloc buffer

Return values

- **HAL:** status

HAL_ETH_UnRegisterRxAllocateCallback

Function name

HAL_StatusTypeDef HAL_ETH_UnRegisterRxAllocateCallback (ETH_HandleTypeDef * heth)

Function description

Unregister the Rx alloc callback.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** status

HAL_ETH_RegisterRxLinkCallback

Function name

HAL_StatusTypeDef HAL_ETH_RegisterRxLinkCallback (ETH_HandleTypeDef * heth, pETH_rxLinkCallbackTypeDef rxLinkCallback)

Function description

Set the Rx link data function.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **rxLinkCallback:** pointer to function to link data

Return values

- **HAL:** status

HAL_ETH_UnRegisterRxLinkCallback

Function name

HAL_StatusTypeDef HAL_ETH_UnRegisterRxLinkCallback (ETH_HandleTypeDef * heth)

Function description

Unregister the Rx link callback.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** status

HAL_ETH_GetRxDataErrorCode

Function name

HAL_StatusTypeDef HAL_ETH_GetRxDataErrorCode (ETH_HandleTypeDef * heth, uint32_t * pErrorCode)

Function description

Get the error state of the last received packet.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pErrorCode:** pointer to uint32_t to hold the error code

Return values

- **HAL:** status

HAL_ETH_RegisterTxFreeCallback

Function name

HAL_StatusTypeDef HAL_ETH_RegisterTxFreeCallback (ETH_HandleTypeDef * heth, pETH_txFreeCallbackTypeDef txFreeCallback)

Function description

Set the Tx free function.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **txFreeCallback:** pointer to function to release the packet

Return values

- **HAL:** status

HAL_ETH_UnRegisterTxFreeCallback

Function name

HAL_StatusTypeDef HAL_ETH_UnRegisterTxFreeCallback (ETH_HandleTypeDef * heth)

Function description

Unregister the Tx free callback.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** status

HAL_ETH_ReleaseTxPacket

Function name

HAL_StatusTypeDef HAL_ETH_ReleaseTxPacket (ETH_HandleTypeDef * heth)

Function description

Release transmitted Tx packets.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** status

HAL_ETH_Transmit

Function name

HAL_StatusTypeDef HAL_ETH_Transmit (ETH_HandleTypeDef * heth, ETH_TxPacketConfig * pTxConfig, uint32_t Timeout)

Function description

Sends an Ethernet Packet in polling mode.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pTxConfig:** Hold the configuration of packet to be transmitted
- **Timeout:** timeout value

Return values

- **HAL:** status

HAL_ETH_Transmit_IT

Function name

HAL_StatusTypeDef HAL_ETH_Transmit_IT (ETH_HandleTypeDef * heth, ETH_TxPacketConfig * pTxConfig)

Function description

Sends an Ethernet Packet in interrupt mode.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pTxConfig:** Hold the configuration of packet to be transmitted

Return values

- **HAL:** status

HAL_ETH_WritePHYRegister

Function name

HAL_StatusTypeDef HAL_ETH_WritePHYRegister (ETH_HandleTypeDef * heth, uint32_t PHYAddr, uint32_t PHYReg, uint32_t RegValue)

Function description

Writes to a PHY register.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **PHYAddr:** PHY port address, must be a value from 0 to 31
- **PHYReg:** PHY register address, must be a value from 0 to 31
- **RegValue:** the value to write

Return values

- **HAL:** status

HAL_ETH_ReadPHYRegister

Function name

HAL_StatusTypeDef HAL_ETH_ReadPHYRegister (ETH_HandleTypeDef * heth, uint32_t PHYAddr, uint32_t PHYReg, uint32_t * pRegValue)

Function description

Read a PHY register.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **PHYAddr:** PHY port address, must be a value from 0 to 31
- **PHYReg:** PHY register address, must be a value from 0 to 31
- **pRegValue:** parameter to hold read value

Return values

- **HAL:** status

HAL_ETH_IRQHandler

Function name

void HAL_ETH_IRQHandler (ETH_HandleTypeDef * heth)

Function description

This function handles ETH interrupt request.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** status

HAL_ETH_TxCpltCallback

Function name

void HAL_ETH_TxCpltCallback (ETH_HandleTypeDef * heth)

Function description

Tx Transfer completed callbacks.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETH_RxCpltCallback

Function name

void HAL_ETH_RxCpltCallback (ETH_HandleTypeDef * heth)

Function description

Rx Transfer completed callbacks.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETH_ErrorCallback

Function name

void HAL_ETH_ErrorCallback (ETH_HandleTypeDef * heth)

Function description

Ethernet transfer error callbacks.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETH_PMTCallback

Function name

void HAL_ETH_PMTCallback (ETH_HandleTypeDef * heth)

Function description

Ethernet Power Management module IT callback.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETH_EEECallback

Function name

void HAL_ETH_EEECallback (ETH_HandleTypeDef * heth)

Function description

Energy Efficient Etherent IT callback.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETH_WakeUpCallback

Function name

void HAL_ETH_WakeUpCallback (ETH_HandleTypeDef * heth)

Function description

ETH WAKEUP interrupt callback.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETH_RxAllocateCallback

Function name

void HAL_ETH_RxAllocateCallback (uint8_t ** buff)

Function description

Rx Allocate callback.

Parameters

- **buff:** pointer to allocated buffer

Return values

- **None:**

HAL_ETH_RxLinkCallback

Function name

void HAL_ETH_RxLinkCallback (void ** pStart, void ** pEnd, uint8_t * buff, uint16_t Length)

Function description

Rx Link callback.

Parameters

- **pStart:** pointer to packet start
- **pStart:** pointer to packet end
- **buff:** pointer to received data
- **Length:** received data length

Return values

- **None:**

HAL_ETH_TxFreeCallback

Function name

void HAL_ETH_TxFreeCallback (uint32_t * buff)

Function description

Tx Free callback.

Parameters

- **buff:** pointer to buffer to free

Return values

- **None:**

HAL_ETH_TxPtpCallback

Function name

void HAL_ETH_TxPtpCallback (uint32_t * buff, ETH_TimeStampTypeDef * timestamp)

Function description

HAL_ETH_GetMACConfig

Function name

HAL_StatusTypeDef HAL_ETH_GetMACConfig (ETH_HandleTypeDef * heth, ETH_MACConfigTypeDef * macconf)

Function description

Get the configuration of the MAC and MTL subsystems.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **macconf:** pointer to a ETH_MACConfigTypeDef structure that will hold the configuration of the MAC.

Return values

- **HAL:** Status

HAL_ETH_GetDMAConfig

Function name

HAL_StatusTypeDef HAL_ETH_GetDMAConfig (ETH_HandleTypeDef * heth, ETH_DMAConfigTypeDef * dmaconf)

Function description

Get the configuration of the DMA.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **dmaconf:** pointer to a ETH_DMAConfigTypeDef structure that will hold the configuration of the ETH DMA.

Return values

- **HAL:** Status

HAL_ETH_SetMACConfig

Function name

HAL_StatusTypeDef HAL_ETH_SetMACConfig (ETH_HandleTypeDef * heth, ETH_MACConfigTypeDef * macconf)

Function description

Set the MAC configuration.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **macconf:** pointer to a ETH_MACConfigTypeDef structure that contains the configuration of the MAC.

Return values

- **HAL:** status

HAL_ETH_SetDMAConfig

Function name

HAL_StatusTypeDef HAL_ETH_SetDMAConfig (ETH_HandleTypeDef * heth, ETH_DMAConfigTypeDef * dmaconf)

Function description

Set the ETH DMA configuration.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **dmaconf:** pointer to a ETH_DMAConfigTypeDef structure that will hold the configuration of the ETH DMA.

Return values

- **HAL:** status

HAL_ETH_SetMDIOClockRange

Function name

void HAL_ETH_SetMDIOClockRange (ETH_HandleTypeDef * heth)

Function description

Configures the Clock range of ETH MDIO interface.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETH_SetRxVLANIdentifier

Function name

void HAL_ETH_SetRxVLANIdentifier (ETH_HandleTypeDef * heth, uint32_t ComparisonBits, uint32_t VLANIdentifier)

Function description

Set the VLAN Identifier for Rx packets.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **ComparisonBits:** 12 or 16 bit comparison mode must be a value of ETH VLAN Tag Comparison
- **VLANIdentifier:** VLAN Identifier value

Return values

- **None:**

HAL_ETH_GetMACFilterConfig

Function name

HAL_StatusTypeDef HAL_ETH_GetMACFilterConfig (ETH_HandleTypeDef * heth, ETH_MACFilterConfigTypeDef * pFilterConfig)

Function description

Get the ETH MAC (L2) Filters configuration.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pFilterConfig:** pointer to a ETH_MACFilterConfigTypeDef structure that will hold the configuration of the ETH MAC filters.

Return values

- **HAL:** status

HAL_ETH_SetMACFilterConfig

Function name

HAL_StatusTypeDef HAL_ETH_SetMACFilterConfig (ETH_HandleTypeDef * heth, ETH_MACFilterConfigTypeDef * pFilterConfig)

Function description

Set the ETH MAC (L2) Filters configuration.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pFilterConfig:** pointer to a ETH_MACFilterConfigTypeDef structure that contains the configuration of the ETH MAC filters.

Return values

- **HAL:** status

HAL_ETH_SetHashTable

Function name

HAL_StatusTypeDef HAL_ETH_SetHashTable (ETH_HandleTypeDef * heth, uint32_t * pHashTable)

Function description

Set the ETH Hash Table Value.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pHashTable:** pointer to a table of two 32 bit values, that contains the 64 bits of the hash table.

Return values

- **HAL:** status

HAL_ETH_SetSourceMACAddrMatch

Function name

HAL_StatusTypeDef HAL_ETH_SetSourceMACAddrMatch (ETH_HandleTypeDef * heth, uint32_t AddrNbr, uint8_t * pMACAddr)

Function description

Set the source MAC Address to be matched.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **AddrNbr:** The MAC address to configure This parameter must be a value of the following: ETH_MAC_ADDRESS1 ETH_MAC_ADDRESS2 ETH_MAC_ADDRESS3
- **pMACAddr:** Pointer to MAC address buffer data (6 bytes)

Return values

- **HAL:** status

HAL_ETH_EnterPowerDownMode

Function name

void HAL_ETH_EnterPowerDownMode (ETH_HandleTypeDef * heth, ETH_PowerDownConfigTypeDef * pPowerDownConfig)

Function description

Enters the Power down mode.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pPowerDownConfig:** a pointer to ETH_PowerDownConfigTypeDef structure that contains the Power Down configuration

Return values

- **None.:**

HAL_ETH_ExitPowerDownMode

Function name

void HAL_ETH_ExitPowerDownMode (ETH_HandleTypeDef * heth)

Function description

Exits from the Power down mode.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None.:**

HAL_ETH_SetWakeUpFilter

Function name

HAL_StatusTypeDef HAL_ETH_SetWakeUpFilter (ETH_HandleTypeDef * heth, uint32_t * pFilter, uint32_t Count)

Function description

Set the WakeUp filter.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pFilter:** pointer to filter registers values
- **Count:** number of filter registers, must be from 1 to 8.

Return values

- **None.:**

HAL_ETH_GetState

Function name

HAL_ETH_StateTypeDef HAL_ETH_GetState (ETH_HandleTypeDef * heth)

Function description

Returns the ETH state.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** state

HAL_ETH_GetError

Function name

uint32_t HAL_ETH_GetError (ETH_HandleTypeDef * heth)

Function description

Returns the ETH error code.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **ETH:** Error Code

HAL_ETH_GetDMAError

Function name

uint32_t HAL_ETH_GetDMAError (ETH_HandleTypeDef * heth)

Function description

Returns the ETH DMA error code.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **ETH:** DMA Error Code

HAL_ETH_GetMACError

Function name

uint32_t HAL_ETH_GetMACError (ETH_HandleTypeDef * heth)

Function description

Returns the ETH MAC error code.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **ETH:** MAC Error Code

HAL_ETH_GetMACWakeUpSource

Function name

uint32_t HAL_ETH_GetMACWakeUpSource (ETH_HandleTypeDef * heth)

Function description

Returns the ETH MAC WakeUp event source.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **ETH:** MAC WakeUp event source

27.3 ETH Firmware driver defines

The following section lists the various define and macros of the module.

27.3.1 ETH

ETH

ETH Back Off Limit

ETH_BACKOFFLIMIT_10

ETH_BACKOFFLIMIT_8

ETH_BACKOFFLIMIT_4

ETH_BACKOFFLIMIT_1

ETH Burst Mode

ETH_BURSTLENGTH_FIXED

ETH_BURSTLENGTH_MIXED

ETH_BURSTLENGTH_UNSPECIFIED

ETH Control Packets Filter

ETH_CTRLPACKETS_BLOCK_ALL

ETH_CTRLPACKETS_FORWARD_ALL_EXCEPT_PA

ETH_CTRLPACKETS_FORWARD_ALL

ETH_CTRLPACKETS_FORWARD_PASSED_ADDR_FILTER

ETH DMA Arbitration

ETH_DMAARBITRATION_RX

ETH_DMAARBITRATION_RX1_TX1

ETH_DMAARBITRATION_RX2_TX1

ETH_DMAARBITRATION_RX3_TX1

ETH_DMAARBITRATION_RX4_TX1

ETH_DMAARBITRATION_RX5_TX1

ETH_DMAARBITRATION_RX6_TX1

ETH_DMAARBITRATION_RX7_TX1

ETH_DMAARBITRATION_RX8_TX1

ETH_DMAARBITRATION_TX

ETH_DMAARBITRATION_TX1_RX1

ETH_DMAARBITRATION_TX2_RX1

ETH_DMAARBITRATION_TX3_RX1

ETH_DMAARBITRATION_TX4_RX1

ETH_DMAARBITRATION_TX5_RX1

ETH_DMAARBITRATION_TX6_RX1

ETH_DMAARBITRATION_TX7_RX1

ETH_DMAARBITRATION_TX8_RX1

ETH DMA Interrupts

ETH_DMA_NORMAL_IT

ETH_DMA_ABNORMAL_IT

ETH_DMA_CONTEXT_DESC_ERROR_IT

ETH_DMA_FATAL_BUS_ERROR_IT

ETH_DMA_EARLY_RX_IT

ETH_DMA_EARLY_TX_IT

ETH_DMA_RX_WATCHDOG_TIMEOUT_IT

ETH_DMA_RX_PROCESS_STOPPED_IT

ETH_DMA_RX_BUFFER_UNAVAILABLE_IT

ETH_DMA_RX_IT

ETH_DMA_TX_BUFFER_UNAVAILABLE_IT

ETH_DMA_TX_PROCESS_STOPPED_IT

ETH_DMA_TX_IT

ETH DMA Rx Descriptor Bit Definition

ETH_DMARXNDESCRF_BUF1AP

Header or Buffer 1 Address Pointer

ETH_DMARXNDESCRF_BUF2AP

Buffer 2 Address Pointer

ETH_DMARXNDESCRF_OWN

OWN bit: descriptor is owned by DMA engine

ETH_DMARXNDESCRF_IOC

Interrupt Enabled on Completion

ETH_DMARXNDESCRF_BUF2V

Buffer 2 Address Valid

ETH_DMARXNDESCRF_BUF1V

Buffer 1 Address Valid

ETH_DMARXNDESCWBF_IVT

Inner VLAN Tag

ETH_DMARXNDESCWBF_OVT

Outer VLAN Tag

ETH_DMARXNDESCWBF_OPC

OAM Sub-Type Code, or MAC Control Packet opcode

ETH_DMARXNDESCWBF_TD

Timestamp Dropped

ETH_DMARXNDESCWBF_TSA

Timestamp Available

ETH_DMARXNDESCWBF_PV

PTP Version

ETH_DMARXNDESCWBF_PFT

PTP Packet Type

ETH_DMARXNDESCWBF_PMT_NO

PTP Message Type: No PTP message received

ETH_DMARXNDESCWBF_PMT_SYNC

PTP Message Type: SYNC (all clock types)

ETH_DMARXNDESCWBF_PMT_FUP

PTP Message Type: Follow_Up (all clock types)

ETH_DMARXNDESCWBF_PMT_DREQ

PTP Message Type: Delay_Req (all clock types)

ETH_DMARXNDESCWBF_PMT_DRESP

PTP Message Type: Delay_Resp (all clock types)

ETH_DMARXNDESCWBF_PMT_PDREQ

PTP Message Type: Pdelay_Req (in peer-to-peer transparent clock)

ETH_DMARXNDESCWBF_PMT_PDRESP

PTP Message Type: Pdelay_Resp (in peer-to-peer transparent clock)

ETH_DMARXNDESCWBF_PMT_PDRESPFUP

PTP Message Type: Pdelay_Resp_Follow_Up (in peer-to-peer transparent clock)

ETH_DMARXNDESCWBF_PMT_ANNOUNCE

PTP Message Type: Announce

ETH_DMARXNDESCWBF_PMT_MANAG

PTP Message Type: Management

ETH_DMARXNDESCWBF_PMT_SIGN

PTP Message Type: Signaling

ETH_DMARXNDESCWBF_PMT_RESERVED

PTP Message Type: PTP packet with Reserved message type

ETH_DMARXNDESCWBF_IPCE

IP Payload Error

ETH_DMARXNDESCWBF_IPCB

IP Checksum Bypassed

ETH_DMARXNDESCWBF_IPV6

IPv6 header Present

ETH_DMARXNDESCWBF_IPV4

IPv4 header Present

ETH_DMARXNDESCWBF_IPHE

IP Header Error

ETH_DMARXNDESCWBF_PT

Payload Type mask

ETH_DMARXNDESCWBF_PT_UNKNOWN

Payload Type: Unknown type or IP/AV payload not processed

ETH_DMARXNDESCWBF_PT_UDP

Payload Type: UDP

ETH_DMARXNDESCWBF_PT_TCP

Payload Type: TCP

ETH_DMARXNDESCWBF_PT_ICMP

Payload Type: ICMP

ETH_DMARXNDESCWBF_L3L4FM

L3 and L4 Filter Number Matched: if reset filter 0 is matched , if set filter 1 is matched

ETH_DMARXNDESCWBF_L4FM

Layer 4 Filter Match

ETH_DMARXNDESCWBF_L3FM

Layer 3 Filter Match

ETH_DMARXNDESCWBF_MADRM

MAC Address Match or Hash Value

ETH_DMARXNDESCWBF_HF

Hash Filter Status

ETH_DMARXNDESCWBF_DAF

Destination Address Filter Fail

ETH_DMARXNDESCWBF_SAF

SA Address Filter Fail

ETH_DMARXNDESCWBF_VF

VLAN Filter Status

ETH_DMARXNDESCWBF_ARPNR

ARP Reply Not Generated

ETH_DMARXNDESCWBF_OWN

Own Bit

ETH_DMARXNDESCWBF_CTXT

Receive Context Descriptor

ETH_DMARXNDESCWBF_FD

First Descriptor

ETH_DMARXNDESCWBF_LD

Last Descriptor

ETH_DMARXNDESCWBF_RS2V

Receive Status RDES2 Valid

ETH_DMARXNDESCWBF_RS1V

Receive Status RDES1 Valid

ETH_DMARXNDESCWBF_RS0V

Receive Status RDES0 Valid

ETH_DMARXNDESCWBF_CE

CRC Error

ETH_DMARXNDESCWBF_GP

Giant Packet

ETH_DMARXNDESCWBF_RWT

Receive Watchdog Timeout

ETH_DMARXNDESCWBF_OE

Overflow Error

ETH_DMARXNDESCWBF_RE

Receive Error

ETH_DMARXNDESCWBF_DE

Dribble Bit Error

ETH_DMARXNDESCWBF_LT

Length/Type Field

ETH_DMARXNDESCWBF_LT_LP

The packet is a length packet

ETH_DMARXNDESCWBF_LT_TP

The packet is a type packet

ETH_DMARXNDESCWBF_LT_ARP

The packet is a ARP Request packet type

ETH_DMARXNDESCWBF_LT_VLAN

The packet is a type packet with VLAN Tag

ETH_DMARXNDESCWBF_LT_DVLAN

The packet is a type packet with Double VLAN Tag

ETH_DMARXNDESCWBF_LT_MAC

The packet is a MAC Control packet type

ETH_DMARXNDESCWBF_LT_OAM

The packet is a OAM packet type

ETH_DMARXNDESCWBF_ES

Error Summary

ETH_DMARXNDESCWBF_PL

Packet Length

ETH_DMARXCDESC_RTSL

Receive Packet Timestamp Low

ETH_DMARXCDESC_RTSH

Receive Packet Timestamp High

ETH_DMARXCDESC_OWN

Own Bit

ETH_DMARXCDESC_CTXT

Receive Context Descriptor

ETH DMA Status Flags

ETH_DMA_RX_NO_ERROR_FLAG

ETH_DMA_RX_DESC_READ_ERROR_FLAG

ETH_DMA_RX_DESC_WRITE_ERROR_FLAG

ETH_DMA_RX_BUFFER_READ_ERROR_FLAG

ETH_DMA_RX_BUFFER_WRITE_ERROR_FLAG

ETH_DMA_TX_NO_ERROR_FLAG

ETH_DMA_TX_DESC_READ_ERROR_FLAG

ETH_DMA_TX_DESC_WRITE_ERROR_FLAG

ETH_DMA_TX_BUFFER_READ_ERROR_FLAG

ETH_DMA_TX_BUFFER_WRITE_ERROR_FLAG

ETH_DMA_CONTEXT_DESC_ERROR_FLAG

ETH_DMA_FATAL_BUS_ERROR_FLAG

ETH_DMA_EARLY_TX_IT_FLAG

ETH_DMA_RX_WATCHDOG_TIMEOUT_FLAG

ETH_DMA_RX_PROCESS_STOPPED_FLAG

ETH_DMA_RX_BUFFER_UNAVAILABLE_FLAG

ETH_DMA_TX_PROCESS_STOPPED_FLAG

ETH DMA Tx Descriptor Bit Definition

ETH_DMATXNDESCRF_B1AP

Transmit Packet Timestamp Low

ETH_DMATXNDESCRF_B2AP

Transmit Packet Timestamp High

ETH_DMATXNDESCRF_IOC

Interrupt on Completion

ETH_DMATXNDESCRF_TTSE

Transmit Timestamp Enable

ETH_DMATXNDESCRF_B2L

Buffer 2 Length

ETH_DMATXNDESCRF_VTIR

VLAN Tag Insertion or Replacement mask

ETH_DMATXNDESCRF_VTIR_DISABLE

Do not add a VLAN tag.

ETH_DMATXNDESCRF_VTIR_REMOVE

Remove the VLAN tag from the packets before transmission.

ETH_DMATXNDESCRF_VTIR_INSERT

Insert a VLAN tag.

ETH_DMATXNDESCRF_VTIR_REPLACE

Replace the VLAN tag.

ETH_DMATXNDESCRF_B1L

Buffer 1 Length

ETH_DMATXNDESCRF_HL

Header Length

ETH_DMATXNDESCRF_OWN

OWN bit: descriptor is owned by DMA engine

ETH_DMATXNDESCRF_CTXT

Context Type

ETH_DMATXNDESCRF_FD

First Descriptor

ETH_DMATXNDESCRF_LD

Last Descriptor

ETH_DMATXNDESCRF_CPC

CRC Pad Control mask

ETH_DMATXNDESCRF_CPC_CRCPAD_INSERT

CRC Pad Control: CRC and Pad Insertion

ETH_DMATXNDESCRF_CPC_CRC_INSERT

CRC Pad Control: CRC Insertion (Disable Pad Insertion)

ETH_DMATXNDESCRF_CPC_DISABLE

CRC Pad Control: Disable CRC Insertion

ETH_DMATXNDESCRF_CPC_CRC_REPLACE

CRC Pad Control: CRC Replacement

ETH_DMATXNDESCRF_SAIC

SA Insertion Control mask

ETH_DMATXNDESCRF_SAIC_DISABLE

SA Insertion Control: Do not include the source address

ETH_DMATXNDESCRF_SAIC_INSERT

SA Insertion Control: Include or insert the source address

ETH_DMATXNDESCRF_SAIC_REPLACE

SA Insertion Control: Replace the source address

ETH_DMATXNDESCRF_THL

TCP Header Length

ETH_DMATXNDESCRF_TSE

TCP segmentation enable

ETH_DMATXNDESCRF_CIC

Checksum Insertion Control: 4 cases

ETH_DMATXNDESCRF_CIC_DISABLE

Do Nothing: Checksum Engine is disabled

ETH_DMATXNDESCRF_CIC_IPHDR_INSERT

Only IP header checksum calculation and insertion are enabled.

ETH_DMATXNDESCRF_CIC_IPHDR_PAYLOAD_INSERT

IP header checksum and payload checksum calculation and insertion are enabled, but pseudo header checksum is not calculated in hardware

ETH_DMATXNDESCRF_CIC_IPHDR_PAYLOAD_INSERT_PHDR_CALC

IP Header checksum and payload checksum calculation and insertion are enabled, and pseudo header checksum is calculated in hardware.

ETH_DMATXNDESCRF_TPL

TCP Payload Length

ETH_DMATXNDESCRF_FL

Transmit End of Ring

ETH_DMATXNDESCWBF_TTSL

Buffer1 Address Pointer or TSO Header Address Pointer

ETH_DMATXNDESCWBF_TTSH

Buffer2 Address Pointer

ETH_DMATXNDESCWBF_OWN

OWN bit: descriptor is owned by DMA engine

ETH_DMATXNDESCWBF_CTXT

Context Type

ETH_DMATXNDESCWBF_FD

First Descriptor

ETH_DMATXNDESCWBF_LD

Last Descriptor

ETH_DMATXNDESCWBF_TTSS

Tx Timestamp Status

ETH_DMATXNDESCWBF_DP

Disable Padding

ETH_DMATXNDESCWBF_TTSE

Transmit Timestamp Enable

ETH_DMATXNDESCWBF_ES

Error summary: OR of the following bits: IHE || UF || ED || EC || LCO || PCE || NC || LCA || FF || JT

ETH_DMATXNDESCWBF_JT

Jabber Timeout

ETH_DMATXNDESCWBF_FF

Packet Flushed: DMA/MTL flushed the packet due to SW flush

ETH_DMATXNDESCWBF_PCE

Payload Checksum Error

ETH_DMATXNDESCWBF_LCA

Loss of Carrier: carrier lost during transmission

ETH_DMATXNDESCWBF_NC

No Carrier: no carrier signal from the transceiver

ETH_DMATXNDESCWBF_LCO

Late Collision: transmission aborted due to collision

ETH_DMATXNDESCWBF_EC

Excessive Collision: transmission aborted after 16 collisions

ETH_DMATXNDESCWBF_CC

Collision Count

ETH_DMATXNDESCWBF_ED

Excessive Deferral

ETH_DMATXNDESCWBF_UF

Underflow Error: late data arrival from the memory

ETH_DMATXNDESCWBF_DB

Deferred Bit

ETH_DMATXNDESCWBF_IHE

IP Header Error

ETH_DMATXCDESC_TTSL

Transmit Packet Timestamp Low

ETH_DMATXCDESC_TTSH

Transmit Packet Timestamp High

ETH_DMATXCDESC_IVT

Inner VLAN Tag

ETH_DMATXCDESC_MSS

Maximum Segment Size

ETH_DMATXCDESC_OWN

OWN bit: descriptor is owned by DMA engine

ETH_DMATXCDESC_CTXT

Context Type

ETH_DMATXCDESC_OSTC

One-Step Timestamp Correction Enable

ETH_DMATXCDESC_TCMSSV

One-Step Timestamp Correction Input or MSS Valid

ETH_DMATXCDESC_CDE

Context Descriptor Error

ETH_DMATXCDESC_IVTIR

Inner VLAN Tag Insert or Replace Mask

ETH_DMATXCDESC_IVTIR_DISABLE

Do not add the inner VLAN tag.

ETH_DMATXCDESC_IVTIR_REMOVE

Remove the inner VLAN tag from the packets before transmission.

ETH_DMATXCDESC_IVTIR_INSERT

Insert the inner VLAN tag.

ETH_DMATXCDESC_IVTIR_REPLACE

Replace the inner VLAN tag.

ETH_DMATXCDESC_IVLTV

Inner VLAN Tag Valid

ETH_DMATXCDESC_VLTV

VLAN Tag Valid

ETH_DMATXCDESC_VT

VLAN Tag

ETH Duplex Mode

ETH_FULLDUPLEX_MODE

ETH_HALFDUPLEX_MODE

ETH Error Code

HAL_ETH_ERROR_NONE

No error

HAL_ETH_ERROR_PARAM

Busy error

HAL_ETH_ERROR_BUSY

Parameter error

HAL_ETH_ERROR_TIMEOUT

Timeout error

HAL_ETH_ERROR_DMA

DMA transfer error

HAL_ETH_ERROR_MAC

MAC transfer error

HAL_ETH_ERROR_INVALID_CALLBACK

Invalid Callback error

ETH Exported Macros

__HAL_ETH_RESET_HANDLE_STATE

Description:

- Reset ETH handle state.

Parameters:

- __HANDLE__: specifies the ETH handle.

Return value:

- None

__HAL_ETH_DMA_ENABLE_IT

Description:

- Enables the specified ETHERNET DMA interrupts.

Parameters:

- __HANDLE__: : ETH Handle
- __INTERRUPT__: specifies the ETHERNET DMA interrupt sources to be enabled

Return value:

- None

__HAL_ETH_DMA_DISABLE_IT

Description:

- Disables the specified ETHERNET DMA interrupts.

Parameters:

- __HANDLE__: : ETH Handle
- __INTERRUPT__: specifies the ETHERNET DMA interrupt sources to be disabled.

Return value:

- None

__HAL_ETH_DMA_GET_IT_SOURCE

Description:

- Gets the ETHERNET DMA IT source enabled or disabled.

Parameters:

- __HANDLE__: : ETH Handle
- __INTERRUPT__: specifies the interrupt source to get .

Return value:

- The: ETH DMA IT Source enabled or disabled

__HAL_ETH_DMA_GET_IT

Description:

- Gets the ETHERNET DMA IT pending bit.

Parameters:

- __HANDLE__: : ETH Handle
- __INTERRUPT__: specifies the interrupt source to get .

Return value:

- The: state of ETH DMA IT (SET or RESET)

__HAL_ETH_DMA_CLEAR_IT

Description:

- Clears the ETHERNET DMA IT pending bit.

Parameters:

- __HANDLE__: : ETH Handle
- __INTERRUPT__: specifies the interrupt pending bit to clear.

Return value:

- None

__HAL_ETH_DMA_GET_FLAG

Description:

- Checks whether the specified ETHERNET DMA flag is set or not.

Parameters:

- __HANDLE__: ETH Handle
- __FLAG__: specifies the flag to check.

Return value:

- The: state of ETH DMA FLAG (SET or RESET).

__HAL_ETH_DMA_CLEAR_FLAG

Description:

- Clears the specified ETHERNET DMA flag.

Parameters:

- __HANDLE__: ETH Handle
- __FLAG__: specifies the flag to check.

Return value:

- The: state of ETH DMA FLAG (SET or RESET).

__HAL_ETH_MAC_ENABLE_IT

Description:

- Enables the specified ETHERNET MAC interrupts.

Parameters:

- __HANDLE__: : ETH Handle
- __INTERRUPT__: specifies the ETHERNET MAC interrupt sources to be enabled

Return value:

- None

__HAL_ETH_MAC_DISABLE_IT

Description:

- Disables the specified ETHERNET MAC interrupts.

Parameters:

- __HANDLE__: : ETH Handle
- __INTERRUPT__: specifies the ETHERNET MAC interrupt sources to be enabled

Return value:

- None

__HAL_ETH_MAC_GET_IT

Description:

- Checks whether the specified ETHERNET MAC flag is set or not.

Parameters:

- __HANDLE__: ETH Handle
- __INTERRUPT__: specifies the flag to check.

Return value:

- The: state of ETH MAC IT (SET or RESET). External interrupt line 86 Connected to the ETH wakeup EXTI Line

ETH_WAKEUP_EXTI_LINE

__HAL_ETH_WAKEUP_EXTI_ENABLE_IT

Description:

- Enable the ETH WAKEUP Exti Line.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP Exti sources to be enabled.
  - ETH_WAKEUP_EXTI_LINE

Return value:

- None.

__HAL_ETH_WAKEUP_EXTI_GET_FLAG

Description:

- checks whether the specified ETH WAKEUP Exti interrupt flag is set or not.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP Exti sources to be cleared.
  - ETH_WAKEUP_EXTI_LINE

Return value:

- EXTI: ETH WAKEUP Line Status.

__HAL_ETH_WAKEUP_EXTI_CLEAR_FLAG

Description:

- Clear the ETH WAKEUP Exti flag.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP Exti sources to be cleared.
  - ETH_WAKEUP_EXTI_LINE

Return value:

- None.

__HAL_ETH_WAKEUP_EXTID2_ENABLE_IT

Description:

- Enable the ETH WAKEUP Exti Line by Core2.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP Exti sources to be enabled.
  - ETH_WAKEUP_EXTI_LINE

Return value:

- None.

__HAL_ETH_WAKEUP_EXTID2_GET_FLAG

Description:

- checks whether the specified ETH WAKEUP Exti interrupt flag is set or not.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP Exti sources to be cleared.
  - ETH_WAKEUP_EXTI_LINE

Return value:

- EXTI: ETH WAKEUP Line Status.

__HAL_ETH_WAKEUP_EXTID2_CLEAR_FLAG

Description:

- Clear the ETH WAKEUP Exti flag.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP Exti sources to be cleared.
  - ETH_WAKEUP_EXTI_LINE

Return value:

- None.

__HAL_ETH_WAKEUP_EXTI_ENABLE_RISING_EDGE

Description:

- enable rising edge interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP EXTI sources to be disabled.
  - ETH_WAKEUP_EXTI_LINE

Return value:

- None

__HAL_ETH_WAKEUP_EXTI_ENABLE_FALLING_EDGE

Description:

- enable falling edge interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP EXTI sources to be disabled.
  - ETH_WAKEUP_EXTI_LINE

Return value:

- None

__HAL_ETH_WAKEUP_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- enable falling edge interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP EXTI sources to be disabled.
  - ETH_WAKEUP_EXTI_LINE

Return value:

- None

__HAL_ETH_WAKEUP_EXTI_GENERATE_SWIT

Description:

- Generates a Software interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP EXTI sources to be disabled.
  - ETH_WAKEUP_EXTI_LINE

Return value:

- None

__HAL_ETH_GET_PTP_CONTROL

__HAL_ETH_SET_PTP_CONTROL

ETH frame settings

ETH_MAX_PACKET_SIZE

ETH_HEADER + 2*VLAN_TAG + MAX_ETH_PAYLOAD + ETH_CRC

ETH_HEADER

6 byte Dest addr, 6 byte Src addr, 2 byte length/type

ETH_CRC

Ethernet CRC

ETH_VLAN_TAG

optional 802.1q VLAN Tag

ETH_MIN_PAYLOAD

Minimum Ethernet payload size

ETH_MAX_PAYLOAD

Maximum Ethernet payload size

ETH_JUMBO_FRAME_PAYLOAD

Jumbo frame payload size

ETH Inter Packet Gap

ETH_INTERPACKETGAP_96BIT

ETH_INTERPACKETGAP_88BIT

ETH_INTERPACKETGAP_80BIT

ETH_INTERPACKETGAP_72BIT

ETH_INTERPACKETGAP_64BIT

ETH_INTERPACKETGAP_56BIT

ETH_INTERPACKETGAP_48BIT

ETH_INTERPACKETGAP_40BIT

ETH MAC addresses

ETH_MAC_ADDRESS0

ETH_MAC_ADDRESS1

ETH_MAC_ADDRESS2

ETH_MAC_ADDRESS3

ETH MAC Interrupts

ETH_MAC_RX_STATUS_IT

ETH_MAC_TX_STATUS_IT

ETH_MAC_TIMESTAMP_IT

ETH_MAC_LPI_IT

ETH_MAC_PMT_IT

ETH_MAC_PHY_IT

ETH MAC Rx Tx Status

ETH_RECEIVE_WATCHDOG_TIMEOUT

ETH_EXECESSIVE_COLLISIONS

ETH_LATE_COLLISIONS

ETH_EXECESSIVE_DEFERRAL

ETH_LOSS_OF_CARRIER

ETH_NO_CARRIER

ETH_TRANSMIT_JABBR_TIMEOUT

ETH MAC Wake Up Event

ETH_WAKEUP_PACKET_RECIEVED

ETH_MAGIC_PACKET_RECIEVED

ETH Pause Low Threshold

ETH_PAUSELOWTHRESHOLD_MINUS_4

ETH_PAUSELOWTHRESHOLD_MINUS_28

ETH_PAUSELOWTHRESHOLD_MINUS_36

ETH_PAUSELOWTHRESHOLD_MINUS_144

ETH_PAUSELOWTHRESHOLD_MINUS_256

ETH_PAUSELOWTHRESHOLD_MINUS_512

ETH Preamble Length

ETH_PREAMBLELENGTH_7

ETH_PREAMBLELENGTH_5

ETH_PREAMBLELENGTH_3

ETH PTP Config Status

HAL_ETH_PTP_NOT_CONFIGURATED

ETH PTP Configuration not done

HAL_ETH_PTP_CONFIGURATED

ETH PTP Configuration done

ETH Receive Mode

ETH_RECEIVESTOREFORWARD

ETH_RECEIVETHRESHOLD8_64

ETH_RECEIVETHRESHOLD8_32

ETH_RECEIVETHRESHOLD8_96

ETH_RECEIVETHRESHOLD8_128

ETH Rx Checksum Status

ETH_CHECKSUM_BYPASSED

ETH_CHECKSUM_IP_HEADER_ERROR

ETH_CHECKSUM_IP_PAYLOAD_ERROR

ETH Rx DMA Burst Length

ETH_RXDMABURSTLENGTH_1BEAT

ETH_RXDMABURSTLENGTH_2BEAT

ETH_RXDMABURSTLENGTH_4BEAT

ETH_RXDMABURSTLENGTH_8BEAT

ETH_RXDMABURSTLENGTH_16BEAT

ETH_RXDMABURSTLENGTH_32BEAT

ETH Rx Error Code

ETH_DRIBBLE_BIT_ERROR

ETH_RECEIVE_ERROR

ETH_RECEIVE_OVERFLOW

ETH_WATCHDOG_TIMEOUT

ETH_GIANT_PACKET

ETH_CRC_ERROR

ETH Rx IP Header Type

ETH_IP_HEADER_IPV4

ETH_IP_HEADER_IPV6

ETH Rx L3 Filter Status

ETH_L3_FILTER0_MATCH

ETH_L3_FILTER1_MATCH

ETH Rx L4 Filter Status

ETH_L4_FILTER0_MATCH

ETH_L4_FILTER1_MATCH

ETH Rx MAC Filter Status

ETH_HASH_FILTER_PASS

ETH_VLAN_FILTER_PASS

ETH_DEST_ADDRESS_FAIL

ETH_SOURCE_ADDRESS_FAIL

ETH Rx Payload Type

ETH_IP_PAYLOAD_UNKNOWN

ETH_IP_PAYLOAD_UDP

ETH_IP_PAYLOAD_TCP

ETH_IP_PAYLOAD_ICMPN

ETH Source Addr Control

ETH_SOURCEADDRESS_DISABLE

ETH_SOURCEADDRESS_INSERT_ADDR0

ETH_SOURCEADDRESS_INSERT_ADDR1

ETH_SOURCEADDRESS_REPLACE_ADDR0

ETH_SOURCEADDRESS_REPLACE_ADDR1

ETH Speed

ETH_SPEED_10M

ETH_SPEED_100M

ETH Transmit Mode

ETH_TRANSMITSTOREFORWARD

ETH_TRANSMITTHRESHOLD_32

ETH_TRANSMITTHRESHOLD_64

ETH_TRANSMITTHRESHOLD_96

ETH_TRANSMITTHRESHOLD_128

ETH_TRANSMITTHRESHOLD_192

ETH_TRANSMITTHRESHOLD_256

ETH_TRANSMITTHRESHOLD_384

ETH_TRANSMITTHRESHOLD_512

ETH Tx DMA Burst Length

ETH_TXDMABURSTLENGTH_1BEAT

ETH_TXDMABURSTLENGTH_2BEAT

ETH_TXDMABURSTLENGTH_4BEAT

ETH_TXDMABURSTLENGTH_8BEAT

ETH_TXDMABURSTLENGTH_16BEAT

ETH_TXDMABURSTLENGTH_32BEAT

ETH Tx Packet Attributes

ETH_TX_PACKETS_FEATURES_CSUM

ETH_TX_PACKETS_FEATURES_SAIC

ETH_TX_PACKETS_FEATURES_VLANTAG

ETH_TX_PACKETS_FEATURES_INNERVLANTAG

ETH_TX_PACKETS_FEATURES_TSO

ETH_TX_PACKETS_FEATURES_CRCPAD

ETH Tx Packet Checksum Control

ETH_CHECKSUM_DISABLE

ETH_CHECKSUM_IPHDR_INSERT

ETH_CHECKSUM_IPHDR_PAYLOAD_INSERT

ETH_CHECKSUM_IPHDR_PAYLOAD_INSERT_PHDR_CALC

ETH Tx Packet CRC Pad Control

ETH_CRC_PAD_DISABLE

ETH_CRC_PAD_INSERT

ETH_CRC_INSERT

ETH_CRC_REPLACE

ETH Tx Packet Inner VLAN Control

ETH_INNER_VLAN_DISABLE

ETH_INNER_VLAN_REMOVE

ETH_INNER_VLAN_INSERT

ETH_INNER_VLAN_REPLACE

ETH Tx Packet Source Addr Control

ETH_SRC_ADDR_CONTROL_DISABLE

ETH_SRC_ADDR_INSERT

ETH_SRC_ADDR_REPLACE

ETH Tx Packet VLAN Control

ETH_VLAN_DISABLE

ETH_VLAN_REMOVE

ETH_VLAN_INSERT

ETH_VLAN_REPLACE

ETH VLAN Tag Comparison

ETH_VLANTAGCOMPARISON_16BIT

ETH_VLANTAGCOMPARISON_12BIT

ETH Watchdog Timeout

ETH_WATCHDOGTIMEOUT_2KB

ETH_WATCHDOGTIMEOUT_3KB

ETH_WATCHDOGTIMEOUT_4KB

ETH_WATCHDOGTIMEOUT_5KB

ETH_WATCHDOGTIMEOUT_6KB

ETH_WATCHDOGTIMEOUT_7KB

ETH_WATCHDOGTIMEOUT_8KB

ETH_WATCHDOGTIMEOUT_9KB

ETH_WATCHDOGTIMEOUT_10KB

ETH_WATCHDOGTIMEOUT_11KB

ETH_WATCHDOGTIMEOUT_12KB

ETH_WATCHDOGTIMEOUT_13KB

ETH_WATCHDOGTIMEOUT_14KB

ETH_WATCHDOGTIMEOUT_15KB

ETH_WATCHDOGTIMEOUT_16KB

28 HAL ETH Extension Driver

28.1 ETHEx Firmware driver registers structures

28.1.1 ETH_RxVLANConfigTypeDef

ETH_RxVLANConfigTypeDef is defined in the stm32h7xx_hal_eth_ex.h

Data Fields

- ***FunctionalState InnerVLANTagInStatus***
- ***uint32_t StripInnerVLANTag***
- ***FunctionalState InnerVLANTag***
- ***FunctionalState DoubleVLANProcessing***
- ***FunctionalState VLANTagHashTableMatch***
- ***FunctionalState VLANTagInStatus***
- ***uint32_t StripVLANTag***
- ***uint32_t VLANTypeCheck***
- ***FunctionalState VLANTagInverceMatch***

Field Documentation

- ***FunctionalState ETH_RxVLANConfigTypeDef::InnerVLANTagInStatus***
  Enables or disables Inner VLAN Tag in Rx Status
- ***uint32_t ETH_RxVLANConfigTypeDef::StripInnerVLANTag***
  Sets the Inner VLAN Tag Stripping on Receive This parameter can be a value of ***ETHEx_Rx_Inner_VLAN_Tag_Stripping***
- ***FunctionalState ETH_RxVLANConfigTypeDef::InnerVLANTag***
  Enables or disables Inner VLAN Tag
- ***FunctionalState ETH_RxVLANConfigTypeDef::DoubleVLANProcessing***
  Enable or Disable double VLAN processing
- ***FunctionalState ETH_RxVLANConfigTypeDef::VLANTagHashTableMatch***
  Enable or Disable VLAN Tag Hash Table Match
- ***FunctionalState ETH_RxVLANConfigTypeDef::VLANTagInStatus***
  Enable or Disable VLAN Tag in Rx status
- ***uint32_t ETH_RxVLANConfigTypeDef::StripVLANTag***
  Set the VLAN Tag Stripping on Receive This parameter can be a value of ***ETHEx_Rx_VLAN_Tag_Stripping***
- ***uint32_t ETH_RxVLANConfigTypeDef::VLANTypeCheck***
  Enable or Disable VLAN Type Check This parameter can be a value of ***ETHEx_VLAN_Type_Check***
- ***FunctionalState ETH_RxVLANConfigTypeDef::VLANTagInverceMatch***
  Enable or disable VLAN Tag Inverse Match

28.1.2 ETH_TxVLANConfigTypeDef

ETH_TxVLANConfigTypeDef is defined in the stm32h7xx_hal_eth_ex.h

Data Fields

- ***FunctionalState SourceTxDesc***
- ***FunctionalState SVLANType***
- ***uint32_t VLANTagControl***

Field Documentation

- ***FunctionalState ETH_TxVLANConfigTypeDef::SourceTxDesc***
  Enable or Disable VLAN tag source from DMA tx descriptors
- ***FunctionalState ETH_TxVLANConfigTypeDef::SVLANType***
  Enable or Disable insertion of SVLAN type
- ***uint32_t ETH_TxVLANConfigTypeDef::VLANTagControl***
  Sets the VLAN tag control in tx packets This parameter can be a value of ***ETHEx_VLAN_Tag_Control***

28.1.3 ETH_L3FilterConfigTypeDef

ETH_L3FilterConfigTypeDef is defined in the stm32h7xx_hal_eth_ex.h

Data Fields

- ***uint32_t Protocol***
- ***uint32_t SrcAddrFilterMatch***
- ***uint32_t DestAddrFilterMatch***
- ***uint32_t SrcAddrHigherBitsMatch***
- ***uint32_t DestAddrHigherBitsMatch***
- ***uint32_t Ip4SrcAddr***
- ***uint32_t Ip4DestAddr***
- ***uint32_t Ip6Addr***

Field Documentation

- ***uint32_t ETH_L3FilterConfigTypeDef::Protocol***
  Sets the L3 filter protocol to IPv4 or IPv6 This parameter can be a value of ***ETHEx_L3_Protocol***
- ***uint32_t ETH_L3FilterConfigTypeDef::SrcAddrFilterMatch***
  Sets the L3 filter source address match This parameter can be a value of ***ETHEx_L3_Source_Match***
- ***uint32_t ETH_L3FilterConfigTypeDef::DestAddrFilterMatch***
  Sets the L3 filter destination address match This parameter can be a value of ***ETHEx_L3_Destination_Match***
- ***uint32_t ETH_L3FilterConfigTypeDef::SrcAddrHigherBitsMatch***
  Sets the L3 filter source address higher bits match This parameter can be a value from 0 to 31
- ***uint32_t ETH_L3FilterConfigTypeDef::DestAddrHigherBitsMatch***
  Sets the L3 filter destination address higher bits match This parameter can be a value from 0 to 31
- ***uint32_t ETH_L3FilterConfigTypeDef::Ip4SrcAddr***
  Sets the L3 filter IPv4 source address if IPv4 protocol is used This parameter can be a value from 0x0 to 0xFFFFFFFF
- ***uint32_t ETH_L3FilterConfigTypeDef::Ip4DestAddr***
  Sets the L3 filter IPv4 destination address if IPv4 protocol is used This parameter can be a value from 0 to 0xFFFFFFFF
- ***uint32_t ETH_L3FilterConfigTypeDef::Ip6Addr[4]***
  Sets the L3 filter IPv6 address if IPv6 protocol is used This parameter must be a table of 4 words (4* 32 bits)

28.1.4 ETH_L4FilterConfigTypeDef

ETH_L4FilterConfigTypeDef is defined in the stm32h7xx_hal_eth_ex.h

Data Fields

- ***uint32_t Protocol***
- ***uint32_t SrcPortFilterMatch***
- ***uint32_t DestPortFilterMatch***
- ***uint32_t SourcePort***
- ***uint32_t DestinationPort***

Field Documentation

- ***uint32_t ETH_L4FilterConfigTypeDef::Protocol***
  Sets the L4 filter protocol to TCP or UDP This parameter can be a value of ***ETHEx_L4_Protocol***

- ***uint32_t ETH_L4FilterConfigTypeDef::SrcPortFilterMatch***
  Sets the L4 filter source port match This parameter can be a value of ***ETHEx_L4_Source_Match***
- ***uint32_t ETH_L4FilterConfigTypeDef::DestPortFilterMatch***
  Sets the L4 filter destination port match This parameter can be a value of ***ETHEx_L4_Destination_Match***
- ***uint32_t ETH_L4FilterConfigTypeDef::SourcePort***
  Sets the L4 filter source port This parameter must be a value from 0x0 to 0xFFFF
- ***uint32_t ETH_L4FilterConfigTypeDef::DestinationPort***
  Sets the L4 filter destination port This parameter must be a value from 0x0 to 0xFFFF

28.2 ETHEx Firmware driver API description

The following section lists the various functions of the ETHEx library.

28.2.1 Extended features functions

This section provides functions allowing to:

- Configure ARP offload module
- Configure L3 and L4 filters
- Configure Extended VLAN features
- Configure Energy Efficient Ethernet module

This section contains the following APIs:

- ***HAL_ETHEx_EnableARPOffload()***
- ***HAL_ETHEx_DisableARPOffload()***
- ***HAL_ETHEx_SetARPAddressMatch()***
- ***HAL_ETHEx_SetL4FilterConfig()***
- ***HAL_ETHEx_GetL4FilterConfig()***
- ***HAL_ETHEx_SetL3FilterConfig()***
- ***HAL_ETHEx_GetL3FilterConfig()***
- ***HAL_ETHEx_EnableL3L4Filtering()***
- ***HAL_ETHEx_DisableL3L4Filtering()***
- ***HAL_ETHEx_GetRxVLANConfig()***
- ***HAL_ETHEx_SetRxVLANConfig()***
- ***HAL_ETHEx_SetVLANHashTable()***
- ***HAL_ETHEx_GetTxVLANConfig()***
- ***HAL_ETHEx_SetTxVLANConfig()***
- ***HAL_ETHEx_SetTxVLANIdentifier()***
- ***HAL_ETHEx_EnableVLANProcessing()***
- ***HAL_ETHEx_DisableVLANProcessing()***
- ***HAL_ETHEx_EnterLPIMode()***
- ***HAL_ETHEx_ExitLPIMode()***
- ***HAL_ETHEx_GetMACLPIEvent()***

28.2.2 Detailed description of functions

HAL_ETHEx_EnableARPOffload

Function name

void HAL_ETHEx_EnableARPOffload (ETH_HandleTypeDef * heth)

Function description

Enables ARP Offload.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETHEx_DisableARPOffload

Function name

void HAL_ETHEx_DisableARPOffload (ETH_HandleTypeDef * heth)

Function description

Disables ARP Offload.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETHEx_SetARPAddressMatch

Function name

void HAL_ETHEx_SetARPAddressMatch (ETH_HandleTypeDef * heth, uint32_t IpAddress)

Function description

Set the ARP Match IP address.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **IpAddress:** IP Address to be matched for incoming ARP requests

Return values

- **None:**

HAL_ETHEx_EnableL3L4Filtering

Function name

void HAL_ETHEx_EnableL3L4Filtering (ETH_HandleTypeDef * heth)

Function description

Enables L3 and L4 filtering process.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None.:**

HAL_ETHEx_DisableL3L4Filtering

Function name

void HAL_ETHEx_DisableL3L4Filtering (ETH_HandleTypeDef * heth)

Function description

Disables L3 and L4 filtering process.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None.:**

HAL_ETHEx_GetL3FilterConfig

Function name

HAL_StatusTypeDef HAL_ETHEx_GetL3FilterConfig (ETH_HandleTypeDef * heth, uint32_t Filter, ETH_L3FilterConfigTypeDef * pL3FilterConfig)

Function description

Configures the L3 Filter, this function allow to: set the layer 3 protocol to be matched (IPv4 or IPv6) enable/ disable L3 source/destination port perfect/inverse match.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **Filter:** L3 filter to configured, this parameter must be one of the following ETH_L3_FILTER_0 ETH_L3_FILTER_1
- **pL3FilterConfig:** pointer to a ETH_L3FilterConfigTypeDef structure that will contain the L3 filter configuration.

Return values

- **HAL:** status

HAL_ETHEx_GetL4FilterConfig

Function name

HAL_StatusTypeDef HAL_ETHEx_GetL4FilterConfig (ETH_HandleTypeDef * heth, uint32_t Filter, ETH_L4FilterConfigTypeDef * pL4FilterConfig)

Function description

Configures the L4 Filter, this function allow to: set the layer 4 protocol to be matched (TCP or UDP) enable/ disable L4 source/destination port perfect/inverse match.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **Filter:** L4 filter to configured, this parameter must be one of the following ETH_L4_FILTER_0 ETH_L4_FILTER_1
- **pL4FilterConfig:** pointer to a ETH_L4FilterConfigTypeDef structure that contains L4 filter configuration.

Return values

- **HAL:** status

HAL_ETHEx_SetL3FilterConfig

Function name

HAL_StatusTypeDef HAL_ETHEx_SetL3FilterConfig (ETH_HandleTypeDef * heth, uint32_t Filter, ETH_L3FilterConfigTypeDef * pL3FilterConfig)

Function description

Configures the L3 Filter, this function allow to: set the layer 3 protocol to be matched (IPv4 or IPv6) enable/ disable L3 source/destination port perfect/inverse match.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **Filter:** L3 filter to configured, this parameter must be one of the following ETH_L3_FILTER_0 ETH_L3_FILTER_1
- **pL3FilterConfig:** pointer to a ETH_L3FilterConfigTypeDef structure that contains L3 filter configuration.

Return values

- **HAL:** status

HAL_ETHEx_SetL4FilterConfig

Function name

HAL_StatusTypeDef HAL_ETHEx_SetL4FilterConfig (ETH_HandleTypeDef * heth, uint32_t Filter, ETH_L4FilterConfigTypeDef * pL4FilterConfig)

Function description

Configures the L4 Filter, this function allow to: set the layer 4 protocol to be matched (TCP or UDP) enable/ disable L4 source/destination port perfect/inverse match.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **Filter:** L4 filter to configured, this parameter must be one of the following ETH_L4_FILTER_0 ETH_L4_FILTER_1
- **pL4FilterConfig:** pointer to a ETH_L4FilterConfigTypeDef structure that contains L4 filter configuration.

Return values

- **HAL:** status

HAL_ETHEx_EnableVLANProcessing

Function name

void HAL_ETHEx_EnableVLANProcessing (ETH_HandleTypeDef * heth)

Function description

Enables the VLAN Tag Filtering process.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None.:**

HAL_ETHEx_DisableVLANProcessing

Function name

void HAL_ETHEx_DisableVLANProcessing (ETH_HandleTypeDef * heth)

Function description

Disables the VLAN Tag Filtering process.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None.:**

HAL_ETHEx_GetRxVLANConfig

Function name

HAL_StatusTypeDef HAL_ETHEx_GetRxVLANConfig (ETH_HandleTypeDef * heth, ETH_RxVLANConfigTypeDef * pVlanConfig)

Function description

Get the VLAN Configuration for Receive Packets.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pVlanConfig:** pointer to a ETH_RxVLANConfigTypeDef structure that will contain the VLAN filter configuration.

Return values

- **HAL:** status

HAL_ETHEx_SetRxVLANConfig

Function name

HAL_StatusTypeDef HAL_ETHEx_SetRxVLANConfig (ETH_HandleTypeDef * heth, ETH_RxVLANConfigTypeDef * pVlanConfig)

Function description

Set the VLAN Configuration for Receive Packets.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pVlanConfig:** pointer to a ETH_RxVLANConfigTypeDef structure that contains VLAN filter configuration.

Return values

- **HAL:** status

HAL_ETHEx_SetVLANHashTable

Function name

void HAL_ETHEx_SetVLANHashTable (ETH_HandleTypeDef * heth, uint32_t VLANHashTable)

Function description

Set the VLAN Hash Table.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **VLANHashTable:** VLAN hash table 16 bit value

Return values

- **None:**

HAL_ETHEx_GetTxVLANConfig

Function name

HAL_StatusTypeDef HAL_ETHEx_GetTxVLANConfig (ETH_HandleTypeDef * heth, uint32_t VLANTag, ETH_TxVLANConfigTypeDef * pVlanConfig)

Function description

Get the VLAN Configuration for Transmit Packets.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **VLANTag:** Selects the vlan tag, this parameter must be one of the following ETH_OUTER_TX_VLANTAG ETH_INNER_TX_VLANTAG
- **pVlanConfig:** pointer to a ETH_TxVLANConfigTypeDef structure that will contain the Tx VLAN filter configuration.

Return values

- **HAL:** Status.

HAL_ETHEx_SetTxVLANConfig

Function name

HAL_StatusTypeDef HAL_ETHEx_SetTxVLANConfig (ETH_HandleTypeDef * heth, uint32_t VLANTag, ETH_TxVLANConfigTypeDef * pVlanConfig)

Function description

Set the VLAN Configuration for Transmit Packets.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **VLANTag:** Selects the vlan tag, this parameter must be one of the following ETH_OUTER_TX_VLANTAG ETH_INNER_TX_VLANTAG
- **pVlanConfig:** pointer to a ETH_TxVLANConfigTypeDef structure that contains Tx VLAN filter configuration.

Return values

- **HAL:** Status

HAL_ETHEx_SetTxVLANIdentifier

Function name

void HAL_ETHEx_SetTxVLANIdentifier (ETH_HandleTypeDef * heth, uint32_t VLANTag, uint32_t VLANIdentifier)

Function description

Set the VLAN Tag Identifier for Transmit Packets.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **VLANTag:** Selects the vlan tag, this parameter must be one of the following ETH_OUTER_TX_VLANTAG ETH_INNER_TX_VLANTAG
- **VLANIdentifier:** VLAN Identifier 16 bit value

Return values

- **None:**

HAL_ETHEx_EnterLPIMode

Function name

void HAL_ETHEx_EnterLPIMode (ETH_HandleTypeDef * heth, FunctionalState TxAutomate, FunctionalState TxClockStop)

Function description

Enters the Low Power Idle (LPI) mode.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **TxAutomate:** Enable/Disable automate enter/exit LPI mode.
- **TxClockStop:** Enable/Disable Tx clock stop in LPI mode.

Return values

- **None:**

HAL_ETHEx_ExitLPIMode

Function name

void HAL_ETHEx_ExitLPIMode (ETH_HandleTypeDef * heth)

Function description

Exits the Low Power Idle (LPI) mode.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETHEx_GetMACLPIEvent

Function name

uint32_t HAL_ETHEx_GetMACLPIEvent (ETH_HandleTypeDef * heth)

Function description

Returns the ETH MAC LPI event.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **ETH:** MAC WakeUp event

28.3 ETHEx Firmware driver defines

The following section lists the various define and macros of the module.

28.3.1 ETHEx

ETHEx

ETHEx L3 Destination Match

ETH_L3_DEST_ADDR_PERFECT_MATCH_ENABLE

ETH_L3_DEST_ADDR_INVERSE_MATCH_ENABLE

ETH_L3_DEST_ADDR_MATCH_DISABLE

ETHEx L3 Filter

ETH_L3_FILTER_0

ETH_L3_FILTER_1

ETHEx L3 Protocol

ETH_L3_IPV6_MATCH

ETH_L3_IPV4_MATCH

ETHEx L3 Source Match

ETH_L3_SRC_ADDR_PERFECT_MATCH_ENABLE

ETH_L3_SRC_ADDR_INVERSE_MATCH_ENABLE

ETH_L3_SRC_ADDR_MATCH_DISABLE

ETHEx L4 Destination Match

ETH_L4_DEST_PORT_PERFECT_MATCH_ENABLE

ETH_L4_DEST_PORT_INVERSE_MATCH_ENABLE

ETH_L4_DEST_PORT_MATCH_DISABLE

ETHEx L4 Filter

ETH_L4_FILTER_0

ETH_L4_FILTER_1

ETHEx L4 Protocol

ETH_L4_UDP_MATCH

ETH_L4_TCP_MATCH

ETHEx L4 Source Match

ETH_L4_SRC_PORT_PERFECT_MATCH_ENABLE

ETH_L4_SRC_PORT_INVERSE_MATCH_ENABLE

ETH_L4_SRC_PORT_MATCH_DISABLE

ETHEx LPI Event

ETH_TX_LPI_ENTRY

ETH_TX_LPI_EXIT

ETH_RX_LPI_ENTRY

ETH_RX_LPI_EXIT

ETHEx Rx Inner VLAN Tag Stripping

ETH_INNERVLANTAGRXSTRIPPING_NONE

ETH_INNERVLANTAGRXSTRIPPING_IFPASS

ETH_INNERVLANTAGRXSTRIPPING_IFFAILS

ETH_INNERVLANTAGRXSTRIPPING_ALWAYS

ETHEx Rx VLAN Tag Stripping

ETH_VLANTAGRXSTRIPPING_NONE

ETH_VLANTAGRXSTRIPPING_IFPASS

ETH_VLANTAGRXSTRIPPING_IFFAILS

ETH_VLANTAGRXSTRIPPING_ALWAYS

ETHEx Tx VLAN Tag

ETH_INNER_TX_VLANTAG

ETH_OUTER_TX_VLANTAG

ETHEx_VLAN_Tag_Control

ETH_VLANTAGCONTROL_NONE

ETH_VLANTAGCONTROL_DELETE

ETH_VLANTAGCONTROL_INSERT

ETH_VLANTAGCONTROL_REPLACE

ETHEx VLAN Type Check

ETH_VLANTYPECHECK_DISABLE

ETH_VLANTYPECHECK_SVLAN

ETH_VLANTYPECHECK_CVLAN

29 HAL EXTI Generic Driver

29.1 EXTI Firmware driver registers structures

29.1.1 EXTI_HandleTypeDef

EXTI_HandleTypeDef is defined in the stm32h7xx_hal_exti.h

Data Fields

- ***uint32_t Line***
- ***void(* PendingCallback***

Field Documentation

- ***uint32_t EXTI_HandleTypeDef::Line***
  Exti line number
- ***void(* EXTI_HandleTypeDef::PendingCallback)(void)***
  Exti pending callback

29.1.2 EXTI_ConfigTypeDef

EXTI_ConfigTypeDef is defined in the stm32h7xx_hal_exti.h

Data Fields

- ***uint32_t Line***
- ***uint32_t Mode***
- ***uint32_t Trigger***
- ***uint32_t GPIOSel***
- ***uint32_t PendClearSource***

Field Documentation

- ***uint32_t EXTI_ConfigTypeDef::Line***
  The Exti line to be configured. This parameter can be a value of ***EXTI_Line***
- ***uint32_t EXTI_ConfigTypeDef::Mode***
  The Exit Mode to be configured for a core. This parameter can be a combination of ***EXTI_Mode***
- ***uint32_t EXTI_ConfigTypeDef::Trigger***
  The Exti Trigger to be configured. This parameter can be a value of ***EXTI_Trigger***
- ***uint32_t EXTI_ConfigTypeDef::GPIOSel***
  The Exti GPIO multiplexer selection to be configured. This parameter is only possible for line 0 to 15. It can be a value of ***EXTI_GPIOSel***
- ***uint32_t EXTI_ConfigTypeDef::PendClearSource***
  Specifies the event pending clear source for D3/SRD domain. This parameter can be a value of ***EXTI_PendClear_Source***

29.2 EXTI Firmware driver API description

The following section lists the various functions of the EXTI library.

29.2.1 EXTI Peripheral features

- Each Exti line can be configured within this driver.

- Exti line can be configured in 3 different modes
  - Interrupt (CORE1 or CORE2 in case of dual core line)
  - Event (CORE1 or CORE2 in case of dual core line)
  - a combination of the previous
- Configurable Exti lines can be configured with 3 different triggers
  - Rising
  - Falling
  - Both of them
- When set in interrupt mode, configurable Exti lines have two diffenrents interrupt pending registers which allow to distinguish which transition occurs:
  - Rising edge pending interrupt
  - Falling
- Exti lines 0 to 15 are linked to gpio pin number 0 to 15. Gpio port can be selected through multiplexer.
- PendClearSource used to set the D3 Smart Run Domain autoamtic pend clear source. It is applicable for line with wkaeup target is Any (CPU1 , CPU2 and D3 smart run domain). Value can be one of the following:
  - EXTI_D3_PENDCLR_SRC_NONE : no pend clear source is selected : In this case corresponding bit of D2PMRx register is set to 0
    - On a configurable Line : the D3 domain wakeup signal is automatically cleared after after the Delay + Rising Edge detect
    - On a direct Line : the D3 domain wakeup signal is cleared after the direct event input signal is cleared
  - EXTI_D3_PENDCLR_SRC_DMACH6 : no pend clear source is selected : In this case corresponding bit of D2PMRx register is set to 1 and corresponding bits(2) of D3PCRxL/H is set to b00 : DMA ch6 event selected as D3 domain pendclear source
  - EXTI_D3_PENDCLR_SRC_DMACH7 : no pend clear source is selected : In this case corresponding bit of D2PMRx register is set to 1 and corresponding bits(2) of D3PCRxL/H is set to b01 : DMA ch7 event selected as D3 domain pendclear source
  - EXTI_D3_PENDCLR_SRC_LPTIM4 : no pend clear source is selected : In this case corresponding bit of D2PMRx register is set to 1 and corresponding bits(2) of D3PCRxL/H is set to b10 : LPTIM4 out selected as D3 domain pendclear source
  - EXTI_D3_PENDCLR_SRC_LPTIM5 : no pend clear source is selected : In this case corresponding bit of D2PMRx register is set to 1 and corresponding bits(2) of D3PCRxL/H is set to b11 : LPTIM5 out selected as D3 domain pendclear source

29.2.2 How to use this driver

1. Configure the EXTI line using HAL_EXTI_SetConfigLine().
   - Choose the interrupt line number by setting "Line" member from EXTI_ConfigTypeDef structure.
   - Configure the interrupt and/or event mode using "Mode" member from EXTI_ConfigTypeDef structure.
   - For configurable lines, configure rising and/or falling trigger "Trigger" member from EXTI_ConfigTypeDef structure.
   - For Exti lines linked to gpio, choose gpio port using "GPIOSel" member from GPIO_InitTypeDef structure.
   - For Exti lines with wkaeup target is Any (CPU1 , CPU2 and D3 smart run domain), choose gpio D3 PendClearSource using PendClearSource member from EXTI_PendClear_Source structure.
2. Get current Exti configuration of a dedicated line using HAL_EXTI_GetConfigLine().
   - Provide exiting handle as parameter.
   - Provide pointer on EXTI_ConfigTypeDef structure as second parameter.
3. Clear Exti configuration of a dedicated line using HAL_EXTI_GetConfigLine().
   - Provide exiting handle as parameter.

4. Register callback to treat Exti interrupts using HAL_EXTI_RegisterCallback().
   – Provide exiting handle as first parameter.
   – Provide which callback will be registered using one value from EXTI_CallbackIDTypeDef.
   – Provide callback function pointer.
5. Get interrupt pending bit using HAL_EXTI_GetPending().
6. Clear interrupt pending bit using HAL_EXTI_GetPending().
7. Generate software interrupt using HAL_EXTI_GenerateSWI().

29.2.3 Configuration functions

This section contains the following APIs:

- *HAL_EXTI_SetConfigLine()*
- *HAL_EXTI_GetConfigLine()*
- *HAL_EXTI_ClearConfigLine()*
- *HAL_EXTI_RegisterCallback()*
- *HAL_EXTI_GetHandle()*

29.2.4 Detailed description of functions

HAL_EXTI_SetConfigLine

Function name

HAL_StatusTypeDef HAL_EXTI_SetConfigLine (EXTI_HandleTypeDef * hexti, EXTI_ConfigTypeDef * pExtiConfig)

Function description

Set configuration of a dedicated Exti line.

Parameters

- **hexti:** Exti handle.
- **pExtiConfig:** Pointer on EXTI configuration to be set.

Return values

- **HAL:** Status.

HAL_EXTI_GetConfigLine

Function name

HAL_StatusTypeDef HAL_EXTI_GetConfigLine (EXTI_HandleTypeDef * hexti, EXTI_ConfigTypeDef * pExtiConfig)

Function description

Get configuration of a dedicated Exti line.

Parameters

- **hexti:** Exti handle.
- **pExtiConfig:** Pointer on structure to store Exti configuration.

Return values

- **HAL:** Status.

HAL_EXTI_ClearConfigLine

Function name

HAL_StatusTypeDef HAL_EXTI_ClearConfigLine (EXTI_HandleTypeDef * hexti)

Function description

Clear whole configuration of a dedicated Exti line.

Parameters

- **hexti:** Exti handle.

Return values

- **HAL:** Status.

HAL_EXTI_RegisterCallback

Function name

HAL_StatusTypeDef HAL_EXTI_RegisterCallback (EXTI_HandleTypeDef * hexti, EXTI_CallbackIDTypeDef CallbackID, void(*)(void) pPendingCbfn)

Function description

Register callback for a dedicated Exti line.

Parameters

- **hexti:** Exti handle.
- **CallbackID:** User callback identifier. This parameter can be one of
  - EXTI_CallbackIDTypeDef values.
- **pPendingCbfn:** function pointer to be stored as callback.

Return values

- **HAL:** Status.

HAL_EXTI_GetHandle

Function name

HAL_StatusTypeDef HAL_EXTI_GetHandle (EXTI_HandleTypeDef * hexti, uint32_t ExtiLine)

Function description

Store line number as handle private field.

Parameters

- **hexti:** Exti handle.
- **ExtiLine:** Exti line number. This parameter can be from 0 to EXTI_LINE_NB.

Return values

- **HAL:** Status.

HAL_EXTI_IRQHandler

Function name

void HAL_EXTI_IRQHandler (EXTI_HandleTypeDef * hexti)

Function description

Handle EXTI interrupt request.

Parameters

- **hexti:** Exti handle.

Return values

- **none.:**

HAL_EXTI_GetPending

Function name

uint32_t HAL_EXTI_GetPending (EXTI_HandleTypeDef * hexti, uint32_t Edge)

Function description

Get interrupt pending bit of a dedicated line.

Parameters

- **hexti:** Exti handle.
- **Edge:** Specify which pending edge as to be checked. This parameter can be one of the following values:
  – EXTI_TRIGGER_RISING_FALLING This parameter is kept for compatibility with other series.

Return values

- **1:** if interrupt is pending else 0.

HAL_EXTI_ClearPending

Function name

void HAL_EXTI_ClearPending (EXTI_HandleTypeDef * hexti, uint32_t Edge)

Function description

Clear interrupt pending bit of a dedicated line.

Parameters

- **hexti:** Exti handle.
- **Edge:** Specify which pending edge as to be clear. This parameter can be one of the following values:
  – EXTI_TRIGGER_RISING_FALLING This parameter is kept for compatibility with other series.

Return values

- **None.:**

HAL_EXTI_GenerateSWI

Function name

void HAL_EXTI_GenerateSWI (EXTI_HandleTypeDef * hexti)

Function description

Generate a software interrupt for a dedicated line.

Parameters

- **hexti:** Exti handle.

Return values

- **None.:**

29.3 EXTI Firmware driver defines

The following section lists the various define and macros of the module.

29.3.1 EXTI

EXTI

Event Input Config

EXTI_MODE_IT

EXTI_MODE_EVT

EXTI_RISING_EDGE

EXTI_FALLING_EDGE

IS_EXTI_EDGE_LINE

IS_EXTI_MODE_LINE

EXTI_LINE0
External interrupt LINE 0

EXTI_LINE1
External interrupt LINE 1

EXTI_LINE2
External interrupt LINE 2

EXTI_LINE3
External interrupt LINE 3

EXTI_LINE4
External interrupt LINE 4

EXTI_LINE5
External interrupt LINE 5

EXTI_LINE6
External interrupt LINE 6

EXTI_LINE7
External interrupt LINE 7

EXTI_LINE8
External interrupt LINE 8

EXTI_LINE9
External interrupt LINE 9

EXTI_LINE10
External interrupt LINE 10

EXTI_LINE11
External interrupt LINE 11

EXTI_LINE12
External interrupt LINE 12

EXTI_LINE13
External interrupt LINE 13

EXTI_LINE14
External interrupt LINE 14

EXTI_LINE15
External interrupt LINE 15

EXTI_LINE16

EXTI_LINE17

EXTI_LINE18

EXTI_LINE19

EXTI_LINE20

EXTI_LINE21

EXTI_LINE22

EXTI_LINE23

EXTI_LINE24

EXTI_LINE25

EXTI_LINE26

EXTI_LINE27

EXTI_LINE28

EXTI_LINE29

EXTI_LINE30

EXTI_LINE31

EXTI_LINE32

EXTI_LINE33

EXTI_LINE34

EXTI_LINE35

EXTI_LINE36

EXTI_LINE37

EXTI_LINE38

EXTI_LINE39

EXTI_LINE40

EXTI_LINE41

EXTI_LINE42

EXTI_LINE43

EXTI_LINE44

EXTI_LINE46

EXTI_LINE47

EXTI_LINE48

EXTI_LINE49

EXTI_LINE50

EXTI_LINE51

EXTI_LINE52

EXTI_LINE53

EXTI_LINE54

EXTI_LINE55

EXTI_LINE56

EXTI_LINE57

EXTI_LINE58

EXTI_LINE59

EXTI_LINE60

EXTI_LINE61

EXTI_LINE62

EXTI_LINE63

EXTI_LINE64

EXTI_LINE65

EXTI_LINE66

EXTI_LINE67

EXTI_LINE68

EXTI_LINE69

EXTI_LINE70

EXTI_LINE71

EXTI_LINE72

EXTI_LINE73

EXTI_LINE74

EXTI_LINE75

EXTI_LINE76

EXTI_LINE77

EXTI_LINE78

EXTI_LINE79

EXTI_LINE80

EXTI_LINE82

EXTI_LINE84

EXTI_LINE85

EXTI_LINE86

EXTI_LINE87

EXTI_LINE88

EXTI_LINE89

EXTI_LINE90

EXTI_LINE91

IS_HAL_EXTI_CONFIG_LINE

IS_EXTI_ALL_LINE

IS_EXTI_D1_LINE

IS_EXTI_D2_LINE

IS_EXTI_D3_LINE

BDMA_CH6_CLEAR

BDMA ch6 event selected as D3 domain pendclear source

BDMA_CH7_CLEAR

BDMA ch7 event selected as D3 domain pendclear source

LPTIM4_OUT_CLEAR

LPTIM4 out selected as D3 domain pendclear source

LPTIM5_OUT_CLEAR

LPTIM5 out selected as D3 domain pendclear source

IS_EXTI_D3_CLEAR

EXTI GPIOSel

EXTI_GPIOA

EXTI_GPIOB

EXTI_GPIOC

EXTI_GPIOD

EXTI_GPIOE

EXTI_GPIOF

EXTI_GPIOG

EXTI_GPIOH

EXTI_GPIOI

EXTI_GPIOJ

EXTI_GPIOK

EXTI Line

EXTI_LINE_0

EXTI_LINE_1

EXTI_LINE_2

EXTI_LINE_3

EXTI_LINE_4

EXTI_LINE_5

EXTI_LINE_6

EXTI_LINE_7

EXTI_LINE_8

EXTI_LINE_9

EXTI_LINE_10

EXTI_LINE_11

EXTI_LINE_12

EXTI_LINE_13

EXTI_LINE_14

EXTI_LINE_15

EXTI_LINE_16

EXTI_LINE_17

EXTI_LINE_18

EXTI_LINE_19

EXTI_LINE_20

EXTI_LINE_21

EXTI_LINE_22

EXTI_LINE_23

EXTI_LINE_24

EXTI_LINE_25

EXTI_LINE_26

EXTI_LINE_27

EXTI_LINE_28

EXTI_LINE_29

EXTI_LINE_30

EXTI_LINE_31

EXTI_LINE_32

EXTI_LINE_33

EXTI_LINE_34

EXTI_LINE_35

EXTI_LINE_36

EXTI_LINE_37

EXTI_LINE_38

EXTI_LINE_39

EXTI_LINE_40

EXTI_LINE_41

EXTI_LINE_42

EXTI_LINE_43

EXTI_LINE_44

EXTI_LINE_45

EXTI_LINE_46

EXTI_LINE_47

EXTI_LINE_48

EXTI_LINE_49

EXTI_LINE_50

EXTI_LINE_51

EXTI_LINE_52

EXTI_LINE_53

EXTI_LINE_54

EXTI_LINE_55

EXTI_LINE_56

EXTI_LINE_57

EXTI_LINE_58

EXTI_LINE_59

EXTI_LINE_60

EXTI_LINE_61

EXTI_LINE_62

EXTI_LINE_63

EXTI_LINE_64

EXTI_LINE_65

EXTI_LINE_66

EXTI_LINE_67

EXTI_LINE_68

EXTI_LINE_69

EXTI_LINE_70

EXTI_LINE_71

EXTI_LINE_72

EXTI_LINE_73

EXTI_LINE_74

EXTI_LINE_75

EXTI_LINE_76

EXTI_LINE_77

EXTI_LINE_78

EXTI_LINE_79

EXTI_LINE_80

EXTI_LINE_81

EXTI_LINE_82

EXTI_LINE_83

EXTI_LINE_84

EXTI_LINE_85

EXTI_LINE_86

EXTI_LINE_87

EXTI Mode

EXTI_MODE_NONE

EXTI_MODE_INTERRUPT

EXTI_MODE_EVENT

EXTI_MODE_CORE1_INTERRUPT

EXTI_MODE_CORE1_EVENT

EXTI_MODE_CORE2_INTERRUPT

EXTI_MODE_CORE2_EVENT

EXTI PendClear Source

EXTI_D3_PENDCLR_SRC_NONE

No D3 domain pendclear source , PMRx register to be set to zero

EXTI_D3_PENDCLR_SRC_DMACH6

DMA ch6 event selected as D3 domain pendclear source, PMRx register to be set to 1

EXTI_D3_PENDCLR_SRC_DMACH7

DMA ch7 event selected as D3 domain pendclear source, PMRx register to be set to 1

EXTI_D3_PENDCLR_SRC_LPTIM4

LPTIM4 out selected as D3 domain pendclear source, PMRx register to be set to 1

EXTI_D3_PENDCLR_SRC_LPTIM5

LPTIM5 out selected as D3 domain pendclear source, PMRx register to be set to 1

EXTI Trigger

EXTI_TRIGGER_NONE

EXTI_TRIGGER_RISING

EXTI_TRIGGER_FALLING

EXTI_TRIGGER_RISING_FALLING

30 HAL FDCAN Generic Driver

30.1 FDCAN Firmware driver registers structures

30.1.1 FDCAN_InitTypeDef

FDCAN_InitTypeDef is defined in the stm32h7xx_hal_fdcan.h

Data Fields

- ***uint32_t FrameFormat***
- ***uint32_t Mode***
- ***FunctionalState AutoRetransmission***
- ***FunctionalState TransmitPause***
- ***FunctionalState ProtocolException***
- ***uint32_t NominalPrescaler***
- ***uint32_t NominalSyncJumpWidth***
- ***uint32_t NominalTimeSeg1***
- ***uint32_t NominalTimeSeg2***
- ***uint32_t DataPrescaler***
- ***uint32_t DataSyncJumpWidth***
- ***uint32_t DataTimeSeg1***
- ***uint32_t DataTimeSeg2***
- ***uint32_t MessageRAMOffset***
- ***uint32_t StdFiltersNbr***
- ***uint32_t ExtFiltersNbr***
- ***uint32_t RxFifo0ElmtsNbr***
- ***uint32_t RxFifo0ElmtSize***
- ***uint32_t RxFifo1ElmtsNbr***
- ***uint32_t RxFifo1ElmtSize***
- ***uint32_t RxBuffersNbr***
- ***uint32_t RxBufferSize***
- ***uint32_t TxEventsNbr***
- ***uint32_t TxBuffersNbr***
- ***uint32_t TxFifoQueueElmtsNbr***
- ***uint32_t TxFifoQueueMode***
- ***uint32_t TxElmtSize***

Field Documentation

- ***uint32_t FDCAN_InitTypeDef::FrameFormat***
  Specifies the FDCAN frame format. This parameter can be a value of ***FDCAN_frame_format***
- ***uint32_t FDCAN_InitTypeDef::Mode***
  Specifies the FDCAN mode. This parameter can be a value of ***FDCAN_operating_mode***
- ***FunctionalState FDCAN_InitTypeDef::AutoRetransmission***
  Enable or disable the automatic retransmission mode. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState FDCAN_InitTypeDef::TransmitPause***
  Enable or disable the Transmit Pause feature. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState FDCAN_InitTypeDef::ProtocolException***
  Enable or disable the Protocol Exception Handling. This parameter can be set to ENABLE or DISABLE

- ***uint32_t FDCAN_InitTypeDef::NominalPrescaler***
  Specifies the value by which the oscillator frequency is divided for generating the nominal bit time quanta. This parameter must be a number between 1 and 512
- ***uint32_t FDCAN_InitTypeDef::NominalSyncJumpWidth***
  Specifies the maximum number of time quanta the FDCAN hardware is allowed to lengthen or shorten a bit to perform resynchronization. This parameter must be a number between 1 and 128
- ***uint32_t FDCAN_InitTypeDef::NominalTimeSeg1***
  Specifies the number of time quanta in Bit Segment 1. This parameter must be a number between 2 and 256
- ***uint32_t FDCAN_InitTypeDef::NominalTimeSeg2***
  Specifies the number of time quanta in Bit Segment 2. This parameter must be a number between 2 and 128
- ***uint32_t FDCAN_InitTypeDef::DataPrescaler***
  Specifies the value by which the oscillator frequency is divided for generating the data bit time quanta. This parameter must be a number between 1 and 32
- ***uint32_t FDCAN_InitTypeDef::DataSyncJumpWidth***
  Specifies the maximum number of time quanta the FDCAN hardware is allowed to lengthen or shorten a data bit to perform resynchronization. This parameter must be a number between 1 and 16
- ***uint32_t FDCAN_InitTypeDef::DataTimeSeg1***
  Specifies the number of time quanta in Data Bit Segment 1. This parameter must be a number between 1 and 32
- ***uint32_t FDCAN_InitTypeDef::DataTimeSeg2***
  Specifies the number of time quanta in Data Bit Segment 2. This parameter must be a number between 1 and 16
- ***uint32_t FDCAN_InitTypeDef::MessageRAMOffset***
  Specifies the message RAM start address. This parameter must be a number between 0 and 2560
- ***uint32_t FDCAN_InitTypeDef::StdFiltersNbr***
  Specifies the number of standard Message ID filters. This parameter must be a number between 0 and 128
- ***uint32_t FDCAN_InitTypeDef::ExtFiltersNbr***
  Specifies the number of extended Message ID filters. This parameter must be a number between 0 and 64
- ***uint32_t FDCAN_InitTypeDef::RxFifo0ElmtsNbr***
  Specifies the number of Rx FIFO0 Elements. This parameter must be a number between 0 and 64
- ***uint32_t FDCAN_InitTypeDef::RxFifo0ElmtSize***
  Specifies the Data Field Size in an Rx FIFO 0 element. This parameter can be a value of ***FDCAN_data_field_size***
- ***uint32_t FDCAN_InitTypeDef::RxFifo1ElmtsNbr***
  Specifies the number of Rx FIFO 1 Elements. This parameter must be a number between 0 and 64
- ***uint32_t FDCAN_InitTypeDef::RxFifo1ElmtSize***
  Specifies the Data Field Size in an Rx FIFO 1 element. This parameter can be a value of ***FDCAN_data_field_size***
- ***uint32_t FDCAN_InitTypeDef::RxBuffersNbr***
  Specifies the number of Dedicated Rx Buffer elements. This parameter must be a number between 0 and 64
- ***uint32_t FDCAN_InitTypeDef::RxBufferSize***
  Specifies the Data Field Size in an Rx Buffer element. This parameter can be a value of ***FDCAN_data_field_size***
- ***uint32_t FDCAN_InitTypeDef::TxEventsNbr***
  Specifies the number of Tx Event FIFO elements. This parameter must be a number between 0 and 32
- ***uint32_t FDCAN_InitTypeDef::TxBuffersNbr***
  Specifies the number of Dedicated Tx Buffers. This parameter must be a number between 0 and 32
- ***uint32_t FDCAN_InitTypeDef::TxFifoQueueElmtsNbr***
  Specifies the number of Tx Buffers used for Tx FIFO/Queue. This parameter must be a number between 0 and 32
- ***uint32_t FDCAN_InitTypeDef::TxFifoQueueMode***
  Tx FIFO/Queue Mode selection. This parameter can be a value of ***FDCAN_txFifoQueue_Mode***

- ***uint32_t FDCAN_InitTypeDef::TxElmtSize***
  Specifies the Data Field Size in a Tx Element. This parameter can be a value of ***FDCAN_data_field_size***

30.1.2 FDCAN_ClkCalUnitTypeDef

FDCAN_ClkCalUnitTypeDef is defined in the stm32h7xx_hal_fdcan.h

Data Fields

- ***uint32_t ClockCalibration***
- ***uint32_t ClockDivider***
- ***uint32_t MinOscClkPeriods***
- ***uint32_t CalFieldLength***
- ***uint32_t TimeQuantaPerBitTime***
- ***uint32_t WatchdogStartValue***

Field Documentation

- ***uint32_t FDCAN_ClkCalUnitTypeDef::ClockCalibration***
  Enable or disable the clock calibration. This parameter can be a value of ***FDCAN_clock_calibration***.
- ***uint32_t FDCAN_ClkCalUnitTypeDef::ClockDivider***
  Specifies the FDCAN kernel clock divider when the clock calibration is bypassed. This parameter can be a value of ***FDCAN_clock_divider***
- ***uint32_t FDCAN_ClkCalUnitTypeDef::MinOscClkPeriods***
  Configures the minimum number of periods in two CAN bit times. The actual configured number of periods is MinOscClkPeriods x 32. This parameter must be a number between 0x00 and 0xFF
- ***uint32_t FDCAN_ClkCalUnitTypeDef::CalFieldLength***
  Specifies the calibration field length. This parameter can be a value of ***FDCAN_calibration_field_length***
- ***uint32_t FDCAN_ClkCalUnitTypeDef::TimeQuantaPerBitTime***
  Configures the number of time quanta per bit time. This parameter must be a number between 4 and 25
- ***uint32_t FDCAN_ClkCalUnitTypeDef::WatchdogStartValue***
  Start value of the Calibration Watchdog Counter. If set to zero the counter is disabled. This parameter must be a number between 0x0000 and 0xFFFF

30.1.3 FDCAN_FilterTypeDef

FDCAN_FilterTypeDef is defined in the stm32h7xx_hal_fdcan.h

Data Fields

- ***uint32_t IdType***
- ***uint32_t FilterIndex***
- ***uint32_t FilterType***
- ***uint32_t FilterConfig***
- ***uint32_t FilterID1***
- ***uint32_t FilterID2***
- ***uint32_t RxBufferIndex***
- ***uint32_t IsCalibrationMsg***

Field Documentation

- ***uint32_t FDCAN_FilterTypeDef::IdType***
  Specifies the identifier type. This parameter can be a value of ***FDCAN_id_type***
- ***uint32_t FDCAN_FilterTypeDef::FilterIndex***
  Specifies the filter which will be initialized. This parameter must be a number between:
  – 0 and 127, if IdType is FDCAN_STANDARD_ID
  – 0 and 63, if IdType is FDCAN_EXTENDED_ID

- ***uint32_t FDCAN_FilterTypeDef::FilterType***

  Specifies the filter type. This parameter can be a value of ***FDCAN_filter_type***. The value FDCAN_EXT_FILTER_RANGE_NO_EIDM is permitted only when IdType is FDCAN_EXTENDED_ID. This parameter is ignored if FilterConfig is set to FDCAN_FILTER_TO_RXBUFFER
- ***uint32_t FDCAN_FilterTypeDef::FilterConfig***

  Specifies the filter configuration. This parameter can be a value of ***FDCAN_filter_config***
- ***uint32_t FDCAN_FilterTypeDef::FilterID1***

  Specifies the filter identification 1. This parameter must be a number between:
  - 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
  - 0 and 0x1FFFFFFF, if IdType is FDCAN_EXTENDED_ID
- ***uint32_t FDCAN_FilterTypeDef::FilterID2***

  Specifies the filter identification 2. This parameter is ignored if FilterConfig is set to FDCAN_FILTER_TO_RXBUFFER. This parameter must be a number between:
  - 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
  - 0 and 0x1FFFFFFF, if IdType is FDCAN_EXTENDED_ID
- ***uint32_t FDCAN_FilterTypeDef::RxBufferIndex***

  Contains the index of the Rx buffer in which the matching message will be stored. This parameter must be a number between 0 and 63. This parameter is ignored if FilterConfig is different from FDCAN_FILTER_TO_RXBUFFER
- ***uint32_t FDCAN_FilterTypeDef::IsCalibrationMsg***

  Specifies whether the filter is configured for calibration messages. This parameter is ignored if FilterConfig is different from FDCAN_FILTER_TO_RXBUFFER. This parameter can be:
  - 0 : ordinary message
  - 1 : calibration message

30.1.4 FDCAN_TxHeaderTypeDef

FDCAN_TxHeaderTypeDef is defined in the stm32h7xx_hal_fdcan.h

Data Fields

- ***uint32_t Identifier***
- ***uint32_t IdType***
- ***uint32_t TxFrameType***
- ***uint32_t DataLength***
- ***uint32_t ErrorStateIndicator***
- ***uint32_t BitRateSwitch***
- ***uint32_t FDFormat***
- ***uint32_t TxEventFifoControl***
- ***uint32_t MessageMarker***

Field Documentation

- ***uint32_t FDCAN_TxHeaderTypeDef::Identifier***

  Specifies the identifier. This parameter must be a number between:
  - 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
  - 0 and 0x1FFFFFFF, if IdType is FDCAN_EXTENDED_ID
- ***uint32_t FDCAN_TxHeaderTypeDef::IdType***

  Specifies the identifier type for the message that will be transmitted. This parameter can be a value of ***FDCAN_id_type***
- ***uint32_t FDCAN_TxHeaderTypeDef::TxFrameType***

  Specifies the frame type of the message that will be transmitted. This parameter can be a value of ***FDCAN_frame_type***
- ***uint32_t FDCAN_TxHeaderTypeDef::DataLength***

  Specifies the length of the frame that will be transmitted. This parameter can be a value of ***FDCAN_data_length_code***

- ***uint32_t FDCAN_TxHeaderTypeDef::ErrorStateIndicator***
  Specifies the error state indicator. This parameter can be a value of ***FDCAN_error_state_indicator***
- ***uint32_t FDCAN_TxHeaderTypeDef::BitRateSwitch***
  Specifies whether the Tx frame will be transmitted with or without bit rate switching. This parameter can be a value of ***FDCAN_bit_rate_switching***
- ***uint32_t FDCAN_TxHeaderTypeDef::FDFormat***
  Specifies whether the Tx frame will be transmitted in classic or FD format. This parameter can be a value of ***FDCAN_format***
- ***uint32_t FDCAN_TxHeaderTypeDef::TxEventFifoControl***
  Specifies the event FIFO control. This parameter can be a value of ***Section 30.3.1 FDCAN_EFC***
- ***uint32_t FDCAN_TxHeaderTypeDef::MessageMarker***
  Specifies the message marker to be copied into Tx Event FIFO element for identification of Tx message status. This parameter must be a number between 0 and 0xFF

30.1.5 FDCAN_RxHeaderTypeDef

FDCAN_RxHeaderTypeDef is defined in the stm32h7xx_hal_fdcan.h

Data Fields

- ***uint32_t Identifier***
- ***uint32_t IdType***
- ***uint32_t RxFrameType***
- ***uint32_t DataLength***
- ***uint32_t ErrorStateIndicator***
- ***uint32_t BitRateSwitch***
- ***uint32_t FDFormat***
- ***uint32_t RxTimestamp***
- ***uint32_t FilterIndex***
- ***uint32_t IsFilterMatchingFrame***

Field Documentation

- ***uint32_t FDCAN_RxHeaderTypeDef::Identifier***
  Specifies the identifier. This parameter must be a number between:
  – 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
  – 0 and 0x1FFFFFFF, if IdType is FDCAN_EXTENDED_ID
- ***uint32_t FDCAN_RxHeaderTypeDef::IdType***
  Specifies the identifier type of the received message. This parameter can be a value of ***FDCAN_id_type***
- ***uint32_t FDCAN_RxHeaderTypeDef::RxFrameType***
  Specifies the the received message frame type. This parameter can be a value of ***FDCAN_frame_type***
- ***uint32_t FDCAN_RxHeaderTypeDef::DataLength***
  Specifies the received frame length. This parameter can be a value of ***FDCAN_data_length_code***
- ***uint32_t FDCAN_RxHeaderTypeDef::ErrorStateIndicator***
  Specifies the error state indicator. This parameter can be a value of ***FDCAN_error_state_indicator***
- ***uint32_t FDCAN_RxHeaderTypeDef::BitRateSwitch***
  Specifies whether the Rx frame is received with or without bit rate switching. This parameter can be a value of ***FDCAN_bit_rate_switching***
- ***uint32_t FDCAN_RxHeaderTypeDef::FDFormat***
  Specifies whether the Rx frame is received in classic or FD format. This parameter can be a value of ***FDCAN_format***
- ***uint32_t FDCAN_RxHeaderTypeDef::RxTimestamp***
  Specifies the timestamp counter value captured on start of frame reception. This parameter must be a number between 0 and 0xFFFF

- ***uint32_t FDCAN_RxHeaderTypeDef::FilterIndex***

  Specifies the index of matching Rx acceptance filter element. This parameter must be a number between:
  - 0 and 127, if IdType is FDCAN_STANDARD_ID
  - 0 and 63, if IdType is FDCAN_EXTENDED_ID When the frame is a Non-Filter matching frame, this parameter is unused.
- ***uint32_t FDCAN_RxHeaderTypeDef::IsFilterMatchingFrame***

  Specifies whether the accepted frame did not match any Rx filter. Acceptance of non-matching frames may be enabled via **HAL_FDCAN_ConfigGlobalFilter()**. This parameter takes 0 if the frame matched an Rx filter or 1 if it did not match any Rx filter

30.1.6 FDCAN_TxEventFifoTypeDef

FDCAN_TxEventFifoTypeDef is defined in the stm32h7xx_hal_fdcan.h

Data Fields

- ***uint32_t Identifier***
- ***uint32_t IdType***
- ***uint32_t TxFrameType***
- ***uint32_t DataLength***
- ***uint32_t ErrorStateIndicator***
- ***uint32_t BitRateSwitch***
- ***uint32_t FDFormat***
- ***uint32_t TxTimestamp***
- ***uint32_t MessageMarker***
- ***uint32_t EventType***

Field Documentation

- ***uint32_t FDCAN_TxEventFifoTypeDef::Identifier***

  Specifies the identifier. This parameter must be a number between:
  - 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
  - 0 and 0x1FFFFFFF, if IdType is FDCAN_EXTENDED_ID
- ***uint32_t FDCAN_TxEventFifoTypeDef::IdType***

  Specifies the identifier type for the transmitted message. This parameter can be a value of ***FDCAN_id_type***
- ***uint32_t FDCAN_TxEventFifoTypeDef::TxFrameType***

  Specifies the frame type of the transmitted message. This parameter can be a value of ***FDCAN_frame_type***
- ***uint32_t FDCAN_TxEventFifoTypeDef::DataLength***

  Specifies the length of the transmitted frame. This parameter can be a value of ***FDCAN_data_length_code***
- ***uint32_t FDCAN_TxEventFifoTypeDef::ErrorStateIndicator***

  Specifies the error state indicator. This parameter can be a value of ***FDCAN_error_state_indicator***
- ***uint32_t FDCAN_TxEventFifoTypeDef::BitRateSwitch***

  Specifies whether the Tx frame is transmitted with or without bit rate switching. This parameter can be a value of ***FDCAN_bit_rate_switching***
- ***uint32_t FDCAN_TxEventFifoTypeDef::FDFormat***

  Specifies whether the Tx frame is transmitted in classic or FD format. This parameter can be a value of ***FDCAN_format***
- ***uint32_t FDCAN_TxEventFifoTypeDef::TxTimestamp***

  Specifies the timestamp counter value captured on start of frame transmission. This parameter must be a number between 0 and 0xFFFF
- ***uint32_t FDCAN_TxEventFifoTypeDef::MessageMarker***

  Specifies the message marker copied into Tx Event FIFO element for identification of Tx message status. This parameter must be a number between 0 and 0xFF
- ***uint32_t FDCAN_TxEventFifoTypeDef::EventType***

  Specifies the event type. This parameter can be a value of ***FDCAN_event_type***

30.1.7 FDCAN_HpMsgStatusTypeDef

FDCAN_HpMsgStatusTypeDef is defined in the stm32h7xx_hal_fdcan.h

Data Fields

- ***uint32_t FilterList***
- ***uint32_t FilterIndex***
- ***uint32_t MessageStorage***
- ***uint32_t MessageIndex***

Field Documentation

- ***uint32_t FDCAN_HpMsgStatusTypeDef::FilterList***
  Specifies the filter list of the matching filter element. This parameter can be:
  – 0 : Standard Filter List
  – 1 : Extended Filter List
- ***uint32_t FDCAN_HpMsgStatusTypeDef::FilterIndex***
  Specifies the index of matching filter element. This parameter can be a number between:
  – 0 and 127, if FilterList is 0 (Standard)
  – 0 and 63, if FilterList is 1 (Extended)
- ***uint32_t FDCAN_HpMsgStatusTypeDef::MessageStorage***
  Specifies the HP Message Storage. This parameter can be a value of ***FDCAN_hp_msg_storage***
- ***uint32_t FDCAN_HpMsgStatusTypeDef::MessageIndex***
  Specifies the Index of Rx FIFO element to which the message was stored. This parameter is valid only when MessageStorage is: FDCAN_HP_STORAGE_RXFIFO0 or FDCAN_HP_STORAGE_RXFIFO1

30.1.8 FDCAN_ProtocolStatusTypeDef

FDCAN_ProtocolStatusTypeDef is defined in the stm32h7xx_hal_fdcan.h

Data Fields

- ***uint32_t LastErrorCode***
- ***uint32_t DataLastErrorCode***
- ***uint32_t Activity***
- ***uint32_t ErrorPassive***
- ***uint32_t Warning***
- ***uint32_t BusOff***
- ***uint32_t RxESIflag***
- ***uint32_t RxBRSflag***
- ***uint32_t RxFDFflag***
- ***uint32_t ProtocolException***
- ***uint32_t TDCvalue***

Field Documentation

- ***uint32_t FDCAN_ProtocolStatusTypeDef::LastErrorCode***
  Specifies the type of the last error that occurred on the FDCAN bus. This parameter can be a value of ***FDCAN_protocol_error_code***
- ***uint32_t FDCAN_ProtocolStatusTypeDef::DataLastErrorCode***
  Specifies the type of the last error that occurred in the data phase of a CAN FD format frame with its BRS flag set. This parameter can be a value of ***FDCAN_protocol_error_code***
- ***uint32_t FDCAN_ProtocolStatusTypeDef::Activity***
  Specifies the FDCAN module communication state. This parameter can be a value of ***FDCAN_communication_state***
- ***uint32_t FDCAN_ProtocolStatusTypeDef::ErrorPassive***
  Specifies the FDCAN module error status. This parameter can be:
  – 0 : The FDCAN is in Error_Active state
  – 1 : The FDCAN is in Error_Passive state

- ***uint32_t FDCAN_ProtocolStatusTypeDef::Warning***
  Specifies the FDCAN module warning status. This parameter can be:
  – 0 : error counters (RxErrorCnt and TxErrorCnt) are below the Error_Warning limit of 96
  – 1 : at least one of error counters has reached the Error_Warning limit of 96
- ***uint32_t FDCAN_ProtocolStatusTypeDef::BusOff***
  Specifies the FDCAN module Bus_Off status. This parameter can be:
  – 0 : The FDCAN is not in Bus_Off state
  – 1 : The FDCAN is in Bus_Off state
- ***uint32_t FDCAN_ProtocolStatusTypeDef::RxESIflag***
  Specifies ESI flag of last received CAN FD message. This parameter can be:
  – 0 : Last received CAN FD message did not have its ESI flag set
  – 1 : Last received CAN FD message had its ESI flag set
- ***uint32_t FDCAN_ProtocolStatusTypeDef::RxBRSflag***
  Specifies BRS flag of last received CAN FD message. This parameter can be:
  – 0 : Last received CAN FD message did not have its BRS flag set
  – 1 : Last received CAN FD message had its BRS flag set
- ***uint32_t FDCAN_ProtocolStatusTypeDef::RxFDFflag***
  Specifies if CAN FD message (FDF flag set) has been received since last protocol status. This parameter can be:
  – 0 : no CAN FD message received
  – 1 : CAN FD message received
- ***uint32_t FDCAN_ProtocolStatusTypeDef::ProtocolException***
  Specifies the FDCAN module Protocol Exception status. This parameter can be:
  – 0 : No protocol exception event occurred since last read access
  – 1 : Protocol exception event occurred
- ***uint32_t FDCAN_ProtocolStatusTypeDef::TDCvalue***
  Specifies the Transmitter Delay Compensation Value. This parameter can be a number between 0 and 127

30.1.9 FDCAN_ErrorCountersTypeDef

FDCAN_ErrorCountersTypeDef is defined in the stm32h7xx_hal_fdcan.h

Data Fields

- ***uint32_t TxErrorCnt***
- ***uint32_t RxErrorCnt***
- ***uint32_t RxErrorPassive***
- ***uint32_t ErrorLogging***

Field Documentation

- ***uint32_t FDCAN_ErrorCountersTypeDef::TxErrorCnt***
  Specifies the Transmit Error Counter Value. This parameter can be a number between 0 and 255
- ***uint32_t FDCAN_ErrorCountersTypeDef::RxErrorCnt***
  Specifies the Receive Error Counter Value. This parameter can be a number between 0 and 127
- ***uint32_t FDCAN_ErrorCountersTypeDef::RxErrorPassive***
  Specifies the Receive Error Passive status. This parameter can be:
  – 0 : The Receive Error Counter (RxErrorCnt) is below the error passive level of 128
  – 1 : The Receive Error Counter (RxErrorCnt) has reached the error passive level of 128
- ***uint32_t FDCAN_ErrorCountersTypeDef::ErrorLogging***
  Specifies the Transmit/Receive error logging counter value. This parameter can be a number between 0 and 255. This counter is incremented each time when a FDCAN protocol error causes the TxErrorCnt or the RxErrorCnt to be incremented. The counter stops at 255; the next increment of TxErrorCnt or RxErrorCnt sets interrupt flag FDCAN_FLAG_ERROR_LOGGING_OVERFLOW

30.1.10 FDCAN_TT_ConfigTypeDef

FDCAN_TT_ConfigTypeDef is defined in the stm32h7xx_hal_fdcan.h

Data Fields

- ***uint32_t OperationMode***
- ***uint32_t GapEnable***
- ***uint32_t TimeMaster***
- ***uint32_t SyncDevLimit***
- ***uint32_t InitRefTrigOffset***
- ***uint32_t ExternalClkSync***
- ***uint32_t AppWdgLimit***
- ***uint32_t GlobalTimeFilter***
- ***uint32_t ClockCalibration***
- ***uint32_t EvtTrigPolarity***
- ***uint32_t BasicCyclesNbr***
- ***uint32_t CycleStartSync***
- ***uint32_t TxEnableWindow***
- ***uint32_t ExpTxTrigNbr***
- ***uint32_t TURNumerator***
- ***uint32_t TURDenominator***
- ***uint32_t TriggerMemoryNbr***
- ***uint32_t StopWatchTrigSel***
- ***uint32_t EventTrigSel***

Field Documentation

- ***uint32_t FDCAN_TT_ConfigTypeDef::OperationMode***

  Specifies the FDCAN Operation Mode. This parameter can be a value of ***FDCAN_operation_mode***
- ***uint32_t FDCAN_TT_ConfigTypeDef::GapEnable***

  Specifies the FDCAN TT Operation. This parameter can be a value of ***FDCAN_TT_operation***. This parameter is ignored if OperationMode is set to FDCAN_TT_COMMUNICATION_LEVEL0
- ***uint32_t FDCAN_TT_ConfigTypeDef::TimeMaster***

  Specifies whether the instance is a slave or a potential master. This parameter can be a value of ***FDCAN_TT_time_master***
- ***uint32_t FDCAN_TT_ConfigTypeDef::SyncDevLimit***

  Specifies the Synchronization Deviation Limit SDL of the TUR numerator : TUR = (Numerator +/- SDL) / Denominator. With : SDL = 2^(SyncDevLimit+5). This parameter must be a number between 0 and 7
- ***uint32_t FDCAN_TT_ConfigTypeDef::InitRefTrigOffset***

  Specifies the Initial Reference Trigger Offset. This parameter must be a number between 0 and 127
- ***uint32_t FDCAN_TT_ConfigTypeDef::ExternalClkSync***

  Enable or disable External Clock Synchronization. This parameter can be a value of ***FDCAN_TT_external_clk_sync***. This parameter is ignored if OperationMode is set to FDCAN_TT_COMMUNICATION_LEVEL1
- ***uint32_t FDCAN_TT_ConfigTypeDef::AppWdgLimit***

  Specifies the Application Watchdog Limit : maximum time after which the application has to serve the application watchdog. The application watchdog is incremented once each 256 NTUs. The application watchdog can be disabled by setting AppWdgLimit to 0. This parameter must be a number between 0 and 255. This parameter is ignored if OperationMode is set to FDCAN_TT_COMMUNICATION_LEVEL0
- ***uint32_t FDCAN_TT_ConfigTypeDef::GlobalTimeFilter***

  Enable or disable Global Time Filtering. This parameter can be a value of ***FDCAN_TT_global_time_filtering***. This parameter is ignored if OperationMode is set to FDCAN_TT_COMMUNICATION_LEVEL1

- ***uint32_t FDCAN_TT_ConfigTypeDef::ClockCalibration***

  Enable or disable Automatic Clock Calibration. This parameter can be a value of ***FDCAN_TT_auto_clk_calibration***. This parameter is ignored if OperationMode is set to FDCAN_TT_COMMUNICATION_LEVEL1
- ***uint32_t FDCAN_TT_ConfigTypeDef::EvtTrigPolarity***

  Specifies the Event Trigger Polarity. This parameter can be a value of ***FDCAN_TT_event_trig_polarity***. This parameter is ignored if OperationMode is set to FDCAN_TT_COMMUNICATION_LEVEL0
- ***uint32_t FDCAN_TT_ConfigTypeDef::BasicCyclesNbr***

  Specifies the number of basic cycles in the system matrix. This parameter can be a value of ***FDCAN_TT_basic_cycle_number***
- ***uint32_t FDCAN_TT_ConfigTypeDef::CycleStartSync***

  Enable or disable synchronization pulse output at pin fdcan1_soc. This parameter can be a value of ***FDCAN_TT_cycle_start_sync***
- ***uint32_t FDCAN_TT_ConfigTypeDef::TxEnableWindow***

  Specifies the length of Tx enable window in NTUs. This parameter must be a number between 1 and 16
- ***uint32_t FDCAN_TT_ConfigTypeDef::ExpTxTrigNbr***

  Specifies the number of expected Tx_Triggers in the system matrix. This is the sum of Tx_Triggers for exclusive, single arbitrating and merged arbitrating windows. This parameter must be a number between 0 and 4095
- ***uint32_t FDCAN_TT_ConfigTypeDef::TURNumerator***

  Specifies the TUR (Time Unit Ratio) numerator. It is advised to set this parameter to the largest applicable value. This parameter must be a number between 0x10000 and 0x1FFFF
- ***uint32_t FDCAN_TT_ConfigTypeDef::TURDenominator***

  Specifies the TUR (Time Unit Ratio) denominator. This parameter must be a number between 0x0001 and 0x3FFF
- ***uint32_t FDCAN_TT_ConfigTypeDef::TriggerMemoryNbr***

  Specifies the number of trigger memory elements. This parameter must be a number between 0 and 64
- ***uint32_t FDCAN_TT_ConfigTypeDef::StopWatchTrigSel***

  Specifies the input to be used as stop watch trigger. This parameter can be a value of ***FDCAN_TT_stop_watch_trig_selection***
- ***uint32_t FDCAN_TT_ConfigTypeDef::EventTrigSel***

  Specifies the input to be used as event trigger. This parameter can be a value of ***FDCAN_TT_event_trig_selection***

30.1.11 FDCAN_TriggerTypeDef

FDCAN_TriggerTypeDef is defined in the stm32h7xx_hal_fdcan.h

Data Fields

- ***uint32_t TriggerIndex***
- ***uint32_t TimeMark***
- ***uint32_t RepeatFactor***
- ***uint32_t StartCycle***
- ***uint32_t TmEventInt***
- ***uint32_t TmEventExt***
- ***uint32_t TriggerType***
- ***uint32_t FilterType***
- ***uint32_t TxBufferIndex***
- ***uint32_t FilterIndex***

Field Documentation

- ***uint32_t FDCAN_TriggerTypeDef::TriggerIndex***

  Specifies the trigger which will be configured. This parameter must be a number between 0 and 63

- ***uint32_t FDCAN_TriggerTypeDef::TimeMark***

  Specifies the cycle time for which the trigger becomes active. This parameter must be a number between 0 and 0xFFFF
- ***uint32_t FDCAN_TriggerTypeDef::RepeatFactor***

  Specifies the trigger repeat factor. This parameter can be a value of ***FDCAN_TT_Repeat_Factor***
- ***uint32_t FDCAN_TriggerTypeDef::StartCycle***

  Specifies the index of the first cycle in which the trigger becomes active. This parameter is ignored if RepeatFactor is set to FDCAN_TT_REPEAT_EVERY_CYCLE. This parameter must be a number between 0 and RepeatFactor
- ***uint32_t FDCAN_TriggerTypeDef::TmEventInt***

  Enable or disable the internal time mark event. If enabled, FDCAN_TT_FLAG_TRIG_TIME_MARK flag is set when trigger memory element becomes active. This parameter can be a value of ***FDCAN_TT_Time_Mark_Event_Internal***
- ***uint32_t FDCAN_TriggerTypeDef::TmEventExt***

  Enable or disable the external time mark event. If enabled, and if TTOCN.TTIE is set, a pulse is generated at fdcan1_tmp when trigger memory element becomes active. This parameter can be a value of ***FDCAN_TT_Time_Mark_Event_External***
- ***uint32_t FDCAN_TriggerTypeDef::TriggerType***

  Specifies the trigger type. This parameter can be a value of ***FDCAN_TT_Trigger_Type***
- ***uint32_t FDCAN_TriggerTypeDef::FilterType***

  Specifies the filter identifier type. This parameter can be a value of ***FDCAN_id_type***
- ***uint32_t FDCAN_TriggerTypeDef::TxBufferIndex***

  Specifies the index of the Tx buffer for which the trigger is valid. This parameter can be a value of ***FDCAN_Tx_location***. This parameter is taken in consideration only if the trigger is configured for transmission.
- ***uint32_t FDCAN_TriggerTypeDef::FilterIndex***

  Specifies the filter for which the trigger is valid. This parameter is taken in consideration only if the trigger is configured for reception. This parameter must be a number between:
  - 0 and 127, if FilterType is FDCAN_STANDARD_ID
  - 0 and 63, if FilterType is FDCAN_EXTENDED_ID

30.1.12 FDCAN_TTOperationStatusTypeDef

FDCAN_TTOperationStatusTypeDef is defined in the stm32h7xx_hal_fdcan.h

Data Fields

- ***uint32_t ErrorLevel***
- ***uint32_t MasterState***
- ***uint32_t SyncState***
- ***uint32_t GTimeQuality***
- ***uint32_t ClockQuality***
- ***uint32_t RefTrigOffset***
- ***uint32_t GTimeDiscPending***
- ***uint32_t GapFinished***
- ***uint32_t MasterPriority***
- ***uint32_t GapStarted***
- ***uint32_t WaitForEvt***
- ***uint32_t AppWdgEvt***
- ***uint32_t ECSPending***
- ***uint32_t PhaseLock***

Field Documentation

- ***uint32_t FDCAN_TTOperationStatusTypeDef::ErrorLevel***

  Specifies the type of the TT operation error level. This parameter can be a value of ***FDCAN_TT_error_level***

- ***uint32_t FDCAN_TTOperationStatusTypeDef::MasterState***
  Specifies the type of the TT master state. This parameter can be a value of ***FDCAN_TT_master_state***
- ***uint32_t FDCAN_TTOperationStatusTypeDef::SyncState***
  Specifies the type of the TT synchronization state. This parameter can be a value of ***FDCAN_TT_sync_state***
- ***uint32_t FDCAN_TTOperationStatusTypeDef::GTimeQuality***
  Specifies the Quality of Global Time Phase. This parameter is only relevant in Level 0 and Level 2, otherwise fixed to 0. This parameter can be:
  – 0 : Global time not valid
  – 1 : Global time in phase with Time Master
- ***uint32_t FDCAN_TTOperationStatusTypeDef::ClockQuality***
  Specifies the Quality of Clock Speed. This parameter is only relevant in Level 0 and Level 2, otherwise fixed to 1. This parameter can be:
  – 0 : Local clock speed not synchronized to Time Master clock speed
  – 1 : Synchronization Deviation = SDL
- ***uint32_t FDCAN_TTOperationStatusTypeDef::RefTrigOffset***
  Specifies the Actual Reference Trigger Offset Value. This parameter can be a number between 0 and 0xFF
- ***uint32_t FDCAN_TTOperationStatusTypeDef::GTimeDiscPending***
  Specifies the Global Time Discontinuity State. This parameter can be:
  – 0 : No global time preset pending
  – 1 : Node waits for the global time preset to take effect
- ***uint32_t FDCAN_TTOperationStatusTypeDef::GapFinished***
  Specifies whether a Gap is finished. This parameter can be:
  – 0 : Reset at the end of each reference message
  – 1 : Gap finished
- ***uint32_t FDCAN_TTOperationStatusTypeDef::MasterPriority***
  Specifies the Priority of actual Time Master. This parameter can be a number between 0 and 0x7
- ***uint32_t FDCAN_TTOperationStatusTypeDef::GapStarted***
  Specifies whether a Gap is started. This parameter can be:
  – 0 : No Gap in schedule
  – 1 : Gap time after Basic Cycle has started
- ***uint32_t FDCAN_TTOperationStatusTypeDef::WaitForEvt***
  Specifies whether a Gap is announced. This parameter can be:
  – 0 : No Gap announced, reset by a reference message with Next_is_Gap = 0
  – 1 : Reference message with Next_is_Gap = 1 received
- ***uint32_t FDCAN_TTOperationStatusTypeDef::AppWdgEvt***
  Specifies the Application Watchdog State. This parameter can be:
  – 0 : Application Watchdog served in time
  – 1 : Failed to serve Application Watchdog in time
- ***uint32_t FDCAN_TTOperationStatusTypeDef::ECSPending***
  Specifies the External Clock Synchronization State. This parameter can be:
  – 0 : No external clock synchronization pending
  – 1 : Node waits for external clock synchronization to take effect
- ***uint32_t FDCAN_TTOperationStatusTypeDef::PhaseLock***
  Specifies the Phase Lock State. This parameter can be:
  – 0 : Phase outside range
  – 1 : Phase inside range

30.1.13 FDCAN_MsgRamAddressTypeDef

FDCAN_MsgRamAddressTypeDef is defined in the stm32h7xx_hal_fdcan.h

Data Fields

- ***uint32_t StandardFilterSA***
- ***uint32_t ExtendedFilterSA***
- ***uint32_t RxFIFO0SA***
- ***uint32_t RxFIFO1SA***
- ***uint32_t RxBufferSA***
- ***uint32_t TxEventFIFOSA***
- ***uint32_t TxBufferSA***
- ***uint32_t TxFIFOQSA***
- ***uint32_t TTMemorySA***
- ***uint32_t EndAddress***

Field Documentation

- ***uint32_t FDCAN_MsgRamAddressTypeDef::StandardFilterSA***
  Specifies the Standard Filter List Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::ExtendedFilterSA***
  Specifies the Extended Filter List Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::RxFIFO0SA***
  Specifies the Rx FIFO 0 Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::RxFIFO1SA***
  Specifies the Rx FIFO 1 Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::RxBufferSA***
  Specifies the Rx Buffer Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::TxEventFIFOSA***
  Specifies the Tx Event FIFO Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::TxBufferSA***
  Specifies the Tx Buffers Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::TxFIFOQSA***
  Specifies the Tx FIFO/Queue Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::TTMemorySA***
  Specifies the Trigger Memory Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::EndAddress***
  Specifies the End Address of the allocated RAM. This parameter must be a 32-bit word address

30.1.14 __FDCAN_HandleTypeDef

__FDCAN_HandleTypeDef is defined in the stm32h7xx_hal_fdcan.h

Data Fields

- ***FDCAN_GlobalTypeDef * Instance***
- ***TTCAN_TypeDef * ttcan***
- ***FDCAN_InitTypeDef Init***
- ***FDCAN_MsgRamAddressTypeDef msgRam***
- ***uint32_t LatestTxFifoQRequest***
- ***__IO HAL_FDCAN_StateTypeDef State***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t ErrorCode***
- ***void(* ClockCalibrationCallback***
- ***void(* TxEventFifoCallback***
- ***void(* RxFifo0Callback***
- ***void(* RxFifo1Callback***
- ***void(* TxFifoEmptyCallback***

- ***void(* TxBufferCompleteCallback***
- ***void(* TxBufferAbortCallback***
- ***void(* RxBufferNewMessageCallback***
- ***void(* HighPriorityMessageCallback***
- ***void(* TimestampWraparoundCallback***
- ***void(* TimeoutOccurredCallback***
- ***void(* ErrorCallback***
- ***void(* ErrorStatusCallback***
- ***void(* TT_ScheduleSyncCallback***
- ***void(* TT_TimeMarkCallback***
- ***void(* TT_StopWatchCallback***
- ***void(* TT_GlobalTimeCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***FDCAN_GlobalTypeDef* __FDCAN_HandleTypeDef::Instance***
  Register base address
- ***TTCAN_TypeDef* __FDCAN_HandleTypeDef::ttcan***
  TT register base address
- ***FDCAN_InitTypeDef __FDCAN_HandleTypeDef::Init***
  FDCAN required parameters
- ***FDCAN_MsgRamAddressTypeDef __FDCAN_HandleTypeDef::msgRam***
  FDCAN Message RAM blocks
- ***uint32_t __FDCAN_HandleTypeDef::LatestTxFifoQRequest***
  FDCAN Tx buffer index of latest Tx FIFO/Queue request
- ***__IO HAL_FDCAN_StateTypeDef __FDCAN_HandleTypeDef::State***
  FDCAN communication state
- ***HAL_LockTypeDef __FDCAN_HandleTypeDef::Lock***
  FDCAN locking object
- ***__IO uint32_t __FDCAN_HandleTypeDef::ErrorCode***
  FDCAN Error code
- ***void(* __FDCAN_HandleTypeDef::ClockCalibrationCallback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t ClkCalibrationITs)***
  FDCAN Clock Calibration callback
- ***void(* __FDCAN_HandleTypeDef::TxEventFifoCallback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t TxEventFifoITs)***
  FDCAN Tx Event Fifo callback
- ***void(* __FDCAN_HandleTypeDef::RxFifo0Callback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t RxFifo0ITs)***
  FDCAN Rx Fifo 0 callback
- ***void(* __FDCAN_HandleTypeDef::RxFifo1Callback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t RxFifo1ITs)***
  FDCAN Rx Fifo 1 callback
- ***void(* __FDCAN_HandleTypeDef::TxFifoEmptyCallback)(struct __FDCAN_HandleTypeDef *hfdcan)***
  FDCAN Tx Fifo Empty callback
- ***void(* __FDCAN_HandleTypeDef::TxBufferCompleteCallback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t BufferIndexes)***
  FDCAN Tx Buffer complete callback

- ***void(* __FDCAN_HandleTypeDef::TxBufferAbortCallback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t BufferIndexes)***
  FDCAN Tx Buffer abort callback
- ***void(* __FDCAN_HandleTypeDef::RxBufferNewMessageCallback)(struct __FDCAN_HandleTypeDef *hfdcan)***
  FDCAN Rx Buffer New Message callback
- ***void(* __FDCAN_HandleTypeDef::HighPriorityMessageCallback)(struct __FDCAN_HandleTypeDef *hfdcan)***
  FDCAN High priority message callback
- ***void(* __FDCAN_HandleTypeDef::TimestampWraparoundCallback)(struct __FDCAN_HandleTypeDef *hfdcan)***
  FDCAN Timestamp wraparound callback
- ***void(* __FDCAN_HandleTypeDef::TimeoutOccurredCallback)(struct __FDCAN_HandleTypeDef *hfdcan)***
  FDCAN Timeout occurred callback
- ***void(* __FDCAN_HandleTypeDef::ErrorCallback)(struct __FDCAN_HandleTypeDef *hfdcan)***
  FDCAN Error callback
- ***void(* __FDCAN_HandleTypeDef::ErrorStatusCallback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t ErrorStatusITs)***
  FDCAN Error status callback
- ***void(* __FDCAN_HandleTypeDef::TT_ScheduleSyncCallback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t TTSchedSyncITs)***
  FDCAN T Schedule Synchronization callback
- ***void(* __FDCAN_HandleTypeDef::TT_TimeMarkCallback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t TTTimeMarkITs)***
  FDCAN TT Time Mark callback
- ***void(* __FDCAN_HandleTypeDef::TT_StopWatchCallback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t SWTime, uint32_t SWCycleCount)***
  FDCAN TT Stop Watch callback
- ***void(* __FDCAN_HandleTypeDef::TT_GlobalTimeCallback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t TTGlobTimeITs)***
  FDCAN TT Global Time callback
- ***void(* __FDCAN_HandleTypeDef::MspInitCallback)(struct __FDCAN_HandleTypeDef *hfdcan)***
  FDCAN Msp Init callback
- ***void(* __FDCAN_HandleTypeDef::MspDeInitCallback)(struct __FDCAN_HandleTypeDef *hfdcan)***
  FDCAN Msp DeInit callback

30.2 FDCAN Firmware driver API description

The following section lists the various functions of the FDCAN library.

30.2.1 How to use this driver

1. Initialize the FDCAN peripheral using HAL_FDCAN_Init function.

2. If needed , configure the reception filters and optional features using the following configuration functions:
   - HAL_FDCAN_ConfigClockCalibration
   - HAL_FDCAN_ConfigFilter
   - HAL_FDCAN_ConfigGlobalFilter
   - HAL_FDCAN_ConfigExtendedIdMask
   - HAL_FDCAN_ConfigRxFifoOverwrite
   - HAL_FDCAN_ConfigFifoWatermark
   - HAL_FDCAN_ConfigRamWatchdog
   - HAL_FDCAN_ConfigTimestampCounter
   - HAL_FDCAN_EnableTimestampCounter
   - HAL_FDCAN_DisableTimestampCounter
   - HAL_FDCAN_ConfigTimeoutCounter
   - HAL_FDCAN_EnableTimeoutCounter
   - HAL_FDCAN_DisableTimeoutCounter
   - HAL_FDCAN_ConfigTxDelayCompensation
   - HAL_FDCAN_EnableTxDelayCompensation
   - HAL_FDCAN_DisableTxDelayCompensation
   - HAL_FDCAN_EnableISOMode
   - HAL_FDCAN_DisableISOMode
   - HAL_FDCAN_EnableEdgeFiltering
   - HAL_FDCAN_DisableEdgeFiltering
   - HAL_FDCAN_TT_ConfigOperation
   - HAL_FDCAN_TT_ConfigReferenceMessage
   - HAL_FDCAN_TT_ConfigTrigger
3. Start the FDCAN module using HAL_FDCAN_Start function. At this level the node is active on the bus: it can send and receive messages.
4. The following Tx control functions can only be called when the FDCAN module is started:
   - HAL_FDCAN_AddMessageToTxFifoQ
   - HAL_FDCAN_EnableTxBufferRequest
   - HAL_FDCAN_AbortTxRequest
5. After having submitted a Tx request in Tx Fifo or Queue, it is possible to get Tx buffer location used to place the Tx request thanks to HAL_FDCAN_GetLatestTxFifoQRequestBuffer API. It is then possible to abort later on the corresponding Tx Request using HAL_FDCAN_AbortTxRequest API.
6. When a message is received into the FDCAN message RAM, it can be retrieved using the HAL_FDCAN_GetRxMessage function.
7. Calling the HAL_FDCAN_Stop function stops the FDCAN module by entering it to initialization mode and re-enabling access to configuration registers through the configuration functions listed here above.
8. All other control functions can be called any time after initialization phase, no matter if the FDCAN module is started or stopped.

Polling mode operation

1. Reception and transmission states can be monitored via the following functions:
   - HAL_FDCAN_IsRxBufferMessageAvailable
   - HAL_FDCAN_IsTxBufferMessagePending
   - HAL_FDCAN_GetRxFifoFillLevel
   - HAL_FDCAN_GetTxFifoFreeLevel

Interrupt mode operation

1. There are two interrupt lines: line 0 and 1. By default, all interrupts are assigned to line 0. Interrupt lines can be configured using HAL_FDCAN_ConfigInterruptLines function.

2. Notifications are activated using HAL_FDCAN_ActivateNotification function. Then, the process can be controlled through one of the available user callbacks: HAL_FDCAN_xxxCallback.

Callback registration

30.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the FDCAN.
- De-initialize the FDCAN.
- Enter FDCAN peripheral in power down mode.
- Exit power down mode.
- Register callbacks.
- Unregister callbacks.

This section contains the following APIs:

- ***HAL_FDCAN_Init()***
- ***HAL_FDCAN_DeInit()***
- ***HAL_FDCAN_MspInit()***
- ***HAL_FDCAN_MspDeInit()***
- ***HAL_FDCAN_EnterPowerDownMode()***
- ***HAL_FDCAN_ExitPowerDownMode()***
- ***HAL_FDCAN_RegisterCallback()***
- ***HAL_FDCAN_UnRegisterCallback()***
- ***HAL_FDCAN_RegisterClockCalibrationCallback()***
- ***HAL_FDCAN_UnRegisterClockCalibrationCallback()***
- ***HAL_FDCAN_RegisterTxEventFifoCallback()***
- ***HAL_FDCAN_UnRegisterTxEventFifoCallback()***
- ***HAL_FDCAN_RegisterRxFifo0Callback()***
- ***HAL_FDCAN_UnRegisterRxFifo0Callback()***
- ***HAL_FDCAN_RegisterRxFifo1Callback()***
- ***HAL_FDCAN_UnRegisterRxFifo1Callback()***
- ***HAL_FDCAN_RegisterTxBufferCompleteCallback()***
- ***HAL_FDCAN_UnRegisterTxBufferCompleteCallback()***
- ***HAL_FDCAN_RegisterTxBufferAbortCallback()***
- ***HAL_FDCAN_UnRegisterTxBufferAbortCallback()***
- ***HAL_FDCAN_RegisterErrorStatusCallback()***
- ***HAL_FDCAN_UnRegisterErrorStatusCallback()***
- ***HAL_FDCAN_RegisterTTScheduleSyncCallback()***
- ***HAL_FDCAN_UnRegisterTTScheduleSyncCallback()***
- ***HAL_FDCAN_RegisterTTTimeMarkCallback()***
- ***HAL_FDCAN_UnRegisterTTTimeMarkCallback()***
- ***HAL_FDCAN_RegisterTTStopWatchCallback()***
- ***HAL_FDCAN_UnRegisterTTStopWatchCallback()***
- ***HAL_FDCAN_RegisterTTGlobalTimeCallback()***
- ***HAL_FDCAN_UnRegisterTTGlobalTimeCallback()***
- ***HAL_FDCAN_RegisterCallback()***

30.2.3 Configuration functions

This section provides functions allowing to:

- HAL_FDCAN_ConfigClockCalibration : Configure the FDCAN clock calibration unit

- HAL_FDCAN_GetClockCalibrationState : Get the clock calibration state
- HAL_FDCAN_ResetClockCalibrationState : Reset the clock calibration state
- HAL_FDCAN_GetClockCalibrationCounter : Get the clock calibration counters values
- HAL_FDCAN_ConfigFilter : Configure the FDCAN reception filters
- HAL_FDCAN_ConfigGlobalFilter : Configure the FDCAN global filter
- HAL_FDCAN_ConfigExtendedIdMask : Configure the extended ID mask
- HAL_FDCAN_ConfigRxFifoOverwrite : Configure the Rx FIFO operation mode
- HAL_FDCAN_ConfigFifoWatermark : Configure the FIFO watermark
- HAL_FDCAN_ConfigRamWatchdog : Configure the RAM watchdog
- HAL_FDCAN_ConfigTimestampCounter : Configure the timestamp counter
- HAL_FDCAN_EnableTimestampCounter : Enable the timestamp counter
- HAL_FDCAN_DisableTimestampCounter : Disable the timestamp counter
- HAL_FDCAN_GetTimestampCounter : Get the timestamp counter value
- HAL_FDCAN_ResetTimestampCounter : Reset the timestamp counter to zero
- HAL_FDCAN_ConfigTimeoutCounter : Configure the timeout counter
- HAL_FDCAN_EnableTimeoutCounter : Enable the timeout counter
- HAL_FDCAN_DisableTimeoutCounter : Disable the timeout counter
- HAL_FDCAN_GetTimeoutCounter : Get the timeout counter value
- HAL_FDCAN_ResetTimeoutCounter : Reset the timeout counter to its start value
- HAL_FDCAN_ConfigTxDelayCompensation : Configure the transmitter delay compensation
- HAL_FDCAN_EnableTxDelayCompensation : Enable the transmitter delay compensation
- HAL_FDCAN_DisableTxDelayCompensation : Disable the transmitter delay compensation
- HAL_FDCAN_EnableISOMode : Enable ISO 11898-1 protocol mode
- HAL_FDCAN_DisableISOMode : Disable ISO 11898-1 protocol mode
- HAL_FDCAN_EnableEdgeFiltering : Enable edge filtering during bus integration
- HAL_FDCAN_DisableEdgeFiltering : Disable edge filtering during bus integration

This section contains the following APIs:

- ***HAL_FDCAN_ConfigClockCalibration()***
- ***HAL_FDCAN_GetClockCalibrationState()***
- ***HAL_FDCAN_ResetClockCalibrationState()***
- ***HAL_FDCAN_GetClockCalibrationCounter()***
- ***HAL_FDCAN_ConfigFilter()***
- ***HAL_FDCAN_ConfigGlobalFilter()***
- ***HAL_FDCAN_ConfigExtendedIdMask()***
- ***HAL_FDCAN_ConfigRxFifoOverwrite()***
- ***HAL_FDCAN_ConfigFifoWatermark()***
- ***HAL_FDCAN_ConfigRamWatchdog()***
- ***HAL_FDCAN_ConfigTimestampCounter()***
- ***HAL_FDCAN_EnableTimestampCounter()***
- ***HAL_FDCAN_DisableTimestampCounter()***
- ***HAL_FDCAN_GetTimestampCounter()***
- ***HAL_FDCAN_ResetTimestampCounter()***
- ***HAL_FDCAN_ConfigTimeoutCounter()***
- ***HAL_FDCAN_EnableTimeoutCounter()***
- ***HAL_FDCAN_DisableTimeoutCounter()***
- ***HAL_FDCAN_GetTimeoutCounter()***
- ***HAL_FDCAN_ResetTimeoutCounter()***
- ***HAL_FDCAN_ConfigTxDelayCompensation()***
- ***HAL_FDCAN_EnableTxDelayCompensation()***

- *HAL_FDCAN_DisableTxDelayCompensation()*
- *HAL_FDCAN_EnableISOMode()*
- *HAL_FDCAN_DisableISOMode()*
- *HAL_FDCAN_EnableEdgeFiltering()*
- *HAL_FDCAN_DisableEdgeFiltering()*

30.2.4 Control functions

This section provides functions allowing to:

- HAL_FDCAN_Start : Start the FDCAN module
- HAL_FDCAN_Stop : Stop the FDCAN module and enable access to configuration registers
- HAL_FDCAN_AddMessageToTxFifoQ : Add a message to the Tx FIFO/Queue and activate the corresponding transmission request
- HAL_FDCAN_AddMessageToTxBuffer : Add a message to a dedicated Tx buffer
- HAL_FDCAN_EnableTxBufferRequest : Enable transmission request
- HAL_FDCAN_GetLatestTxFifoQRequestBuffer : Get Tx buffer index of latest Tx FIFO/Queue request
- HAL_FDCAN_AbortTxRequest : Abort transmission request
- HAL_FDCAN_GetRxMessage : Get an FDCAN frame from the Rx Buffer/FIFO zone into the message RAM
- HAL_FDCAN_GetTxEvent : Get an FDCAN Tx event from the Tx Event FIFO zone into the message RAM
- HAL_FDCAN_GetHighPriorityMessageStatus : Get high priority message status
- HAL_FDCAN_GetProtocolStatus : Get protocol status
- HAL_FDCAN_GetErrorCounters : Get error counter values
- HAL_FDCAN_IsRxBufferMessageAvailable : Check if a new message is received in the selected Rx buffer
- HAL_FDCAN_IsTxBufferMessagePending : Check if a transmission request is pending on the selected Tx buffer
- HAL_FDCAN_GetRxFifoFillLevel : Return Rx FIFO fill level
- HAL_FDCAN_GetTxFifoFreeLevel : Return Tx FIFO free level
- HAL_FDCAN_IsRestrictedOperationMode : Check if the FDCAN peripheral entered Restricted Operation Mode
- HAL_FDCAN_ExitRestrictedOperationMode : Exit Restricted Operation Mode

This section contains the following APIs:

- *HAL_FDCAN_Start()*
- *HAL_FDCAN_Stop()*
- *HAL_FDCAN_AddMessageToTxFifoQ()*
- *HAL_FDCAN_AddMessageToTxBuffer()*
- *HAL_FDCAN_EnableTxBufferRequest()*
- *HAL_FDCAN_GetLatestTxFifoQRequestBuffer()*
- *HAL_FDCAN_AbortTxRequest()*
- *HAL_FDCAN_GetRxMessage()*
- *HAL_FDCAN_GetTxEvent()*
- *HAL_FDCAN_GetHighPriorityMessageStatus()*
- *HAL_FDCAN_GetProtocolStatus()*
- *HAL_FDCAN_GetErrorCounters()*
- *HAL_FDCAN_IsRxBufferMessageAvailable()*
- *HAL_FDCAN_IsTxBufferMessagePending()*
- *HAL_FDCAN_GetRxFifoFillLevel()*
- *HAL_FDCAN_GetTxFifoFreeLevel()*
- *HAL_FDCAN_IsRestrictedOperationMode()*
- *HAL_FDCAN_ExitRestrictedOperationMode()*

30.2.5 TT Configuration and control functions

This section provides functions allowing to:

- HAL_FDCAN_TT_ConfigOperation : Initialize TT operation parameters
- HAL_FDCAN_TT_ConfigReferenceMessage : Configure the reference message
- HAL_FDCAN_TT_ConfigTrigger : Configure the FDCAN trigger
- HAL_FDCAN_TT_SetGlobalTime : Schedule global time adjustment
- HAL_FDCAN_TT_SetClockSynchronization : Schedule TUR numerator update
- HAL_FDCAN_TT_ConfigStopWatch : Configure stop watch source and polarity
- HAL_FDCAN_TT_ConfigRegisterTimeMark : Configure register time mark pulse generation
- HAL_FDCAN_TT_EnableRegisterTimeMarkPulse : Enable register time mark pulse generation
- HAL_FDCAN_TT_DisableRegisterTimeMarkPulse : Disable register time mark pulse generation
- HAL_FDCAN_TT_EnableTriggerTimeMarkPulse : Enable trigger time mark pulse generation
- HAL_FDCAN_TT_DisableTriggerTimeMarkPulse : Disable trigger time mark pulse generation
- HAL_FDCAN_TT_EnableHardwareGapControl : Enable gap control by input pin fdcan1_evt
- HAL_FDCAN_TT_DisableHardwareGapControl : Disable gap control by input pin fdcan1_evt
- HAL_FDCAN_TT_EnableTimeMarkGapControl : Enable gap control (finish only) by register time mark IT
- HAL_FDCAN_TT_DisableTimeMarkGapControl : Disable gap control by register time mark interrupt
- HAL_FDCAN_TT_SetNextIsGap : Transmit next reference message with Next_is_Gap = "1"
- HAL_FDCAN_TT_SetEndOfGap : Finish a Gap by requesting start of reference message
- HAL_FDCAN_TT_ConfigExternalSyncPhase : Configure target phase used for external synchronization
- HAL_FDCAN_TT_EnableExternalSynchronization : Synchronize the phase of the FDCAN schedule to an external schedule
- HAL_FDCAN_TT_DisableExternalSynchronization : Disable external schedule synchronization
- HAL_FDCAN_TT_GetOperationStatus : Get TT operation status

This section contains the following APIs:

- ***HAL_FDCAN_TT_ConfigOperation()***
- ***HAL_FDCAN_TT_ConfigReferenceMessage()***
- ***HAL_FDCAN_TT_ConfigTrigger()***
- ***HAL_FDCAN_TT_SetGlobalTime()***
- ***HAL_FDCAN_TT_SetClockSynchronization()***
- ***HAL_FDCAN_TT_ConfigStopWatch()***
- ***HAL_FDCAN_TT_ConfigRegisterTimeMark()***
- ***HAL_FDCAN_TT_EnableRegisterTimeMarkPulse()***
- ***HAL_FDCAN_TT_DisableRegisterTimeMarkPulse()***
- ***HAL_FDCAN_TT_EnableTriggerTimeMarkPulse()***
- ***HAL_FDCAN_TT_DisableTriggerTimeMarkPulse()***
- ***HAL_FDCAN_TT_EnableHardwareGapControl()***
- ***HAL_FDCAN_TT_DisableHardwareGapControl()***
- ***HAL_FDCAN_TT_EnableTimeMarkGapControl()***
- ***HAL_FDCAN_TT_DisableTimeMarkGapControl()***
- ***HAL_FDCAN_TT_SetNextIsGap()***
- ***HAL_FDCAN_TT_SetEndOfGap()***
- ***HAL_FDCAN_TT_ConfigExternalSyncPhase()***
- ***HAL_FDCAN_TT_EnableExternalSynchronization()***
- ***HAL_FDCAN_TT_DisableExternalSynchronization()***
- ***HAL_FDCAN_TT_GetOperationStatus()***

30.2.6 Interrupts management

This section provides functions allowing to:

- HAL_FDCAN_ConfigInterruptLines : Assign interrupts to either Interrupt line 0 or 1
- HAL_FDCAN_TT_ConfigInterruptLines : Assign TT interrupts to either Interrupt line 0 or 1
- HAL_FDCAN_ActivateNotification : Enable interrupts
- HAL_FDCAN_DeactivateNotification : Disable interrupts
- HAL_FDCAN_TT_ActivateNotification : Enable TT interrupts
- HAL_FDCAN_TT_DeactivateNotification : Disable TT interrupts
- HAL_FDCAN_IRQHandler : Handles FDCAN interrupt request

This section contains the following APIs:

- ***HAL_FDCAN_ConfigInterruptLines()***
- ***HAL_FDCAN_TT_ConfigInterruptLines()***
- ***HAL_FDCAN_ActivateNotification()***
- ***HAL_FDCAN_DeactivateNotification()***
- ***HAL_FDCAN_TT_ActivateNotification()***
- ***HAL_FDCAN_TT_DeactivateNotification()***
- ***HAL_FDCAN_IRQHandler()***

30.2.7 Callback functions

This subsection provides the following callback functions:

- HAL_FDCAN_ClockCalibrationCallback
- HAL_FDCAN_TxEventFifoCallback
- HAL_FDCAN_RxFifo0Callback
- HAL_FDCAN_RxFifo1Callback
- HAL_FDCAN_TxFifoEmptyCallback
- HAL_FDCAN_TxBufferCompleteCallback
- HAL_FDCAN_TxBufferAbortCallback
- HAL_FDCAN_RxBufferNewMessageCallback
- HAL_FDCAN_HighPriorityMessageCallback
- HAL_FDCAN_TimestampWraparoundCallback
- HAL_FDCAN_TimeoutOccurredCallback
- HAL_FDCAN_ErrorCallback
- HAL_FDCAN_ErrorStatusCallback
- HAL_FDCAN_TT_ScheduleSyncCallback
- HAL_FDCAN_TT_TimeMarkCallback
- HAL_FDCAN_TT_StopWatchCallback
- HAL_FDCAN_TT_GlobalTimeCallback

This section contains the following APIs:

- ***HAL_FDCAN_ClockCalibrationCallback()***
- ***HAL_FDCAN_TxEventFifoCallback()***
- ***HAL_FDCAN_RxFifo0Callback()***
- ***HAL_FDCAN_RxFifo1Callback()***
- ***HAL_FDCAN_TxFifoEmptyCallback()***
- ***HAL_FDCAN_TxBufferCompleteCallback()***
- ***HAL_FDCAN_TxBufferAbortCallback()***
- ***HAL_FDCAN_RxBufferNewMessageCallback()***
- ***HAL_FDCAN_TimestampWraparoundCallback()***
- ***HAL_FDCAN_TimeoutOccurredCallback()***
- ***HAL_FDCAN_HighPriorityMessageCallback()***
- ***HAL_FDCAN_ErrorCallback()***
- ***HAL_FDCAN_ErrorStatusCallback()***

- *HAL_FDCAN_TT_ScheduleSyncCallback()*
- *HAL_FDCAN_TT_TimeMarkCallback()*
- *HAL_FDCAN_TT_StopWatchCallback()*
- *HAL_FDCAN_TT_GlobalTimeCallback()*

30.2.8 Peripheral State functions

This subsection provides functions allowing to :

- HAL_FDCAN_GetState() : Return the FDCAN state.
- HAL_FDCAN_GetError() : Return the FDCAN error code if any.

This section contains the following APIs:

- *HAL_FDCAN_GetState()*
- *HAL_FDCAN_GetError()*

30.2.9 Detailed description of functions

HAL_FDCAN_Init

Function name

HAL_StatusTypeDef HAL_FDCAN_Init (FDCAN_HandleTypeDef * hfdcan)

Function description

Initializes the FDCAN peripheral according to the specified parameters in the FDCAN_InitTypeDef structure.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_DeInit

Function name

HAL_StatusTypeDef HAL_FDCAN_DeInit (FDCAN_HandleTypeDef * hfdcan)

Function description

Deinitializes the FDCAN peripheral registers to their default reset values.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_MspInit

Function name

void HAL_FDCAN_MspInit (FDCAN_HandleTypeDef * hfdcan)

Function description

Initializes the FDCAN MSP.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_MspDeInit

Function name

void HAL_FDCAN_MspDeInit (FDCAN_HandleTypeDef * hfdcan)

Function description

DeInitializes the FDCAN MSP.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_EnterPowerDownMode

Function name

HAL_StatusTypeDef HAL_FDCAN_EnterPowerDownMode (FDCAN_HandleTypeDef * hfdcan)

Function description

Enter FDCAN peripheral in sleep mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_ExitPowerDownMode

Function name

HAL_StatusTypeDef HAL_FDCAN_ExitPowerDownMode (FDCAN_HandleTypeDef * hfdcan)

Function description

Exit power down mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_RegisterCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterCallback (FDCAN_HandleTypeDef * hfdcan, HAL_FDCAN_CallbackIDTypeDef CallbackID, pFDCAN_CallbackTypeDef pCallback)

Function description

HAL_FDCAN_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterCallback (FDCAN_HandleTypeDef * hfdcan, HAL_FDCAN_CallbackIDTypeDef CallbackID)

Function description

Unregister a FDCAN CallBack.

Parameters

- **hfdcan:** pointer to a FDCAN_HandleTypeDef structure that contains the configuration information for FDCAN module
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_FDCAN_TX_FIFO_EMPTY_CB_ID Tx Fifo Empty callback ID
  - HAL_FDCAN_RX_BUFFER_NEW_MSG_CB_ID Rx buffer new message callback ID
  - HAL_FDCAN_HIGH_PRIO_MESSAGE_CB_ID High priority message callback ID
  - HAL_FDCAN_TIMESTAMP_WRAPAROUND_CB_ID Timestamp wraparound callback ID
  - HAL_FDCAN_TIMEOUT_OCCURRED_CB_ID Timeout occurred callback ID
  - HAL_FDCAN_ERROR_CALLBACK_CB_ID Error callback ID
  - HAL_FDCAN_MSPINIT_CB_ID MspInit callback ID
  - HAL_FDCAN_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_FDCAN_RegisterClockCalibrationCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterClockCalibrationCallback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_ClockCalibrationCallbackTypeDef pCallback)

Function description

Register Clock Calibration FDCAN Callback To be used instead of the weak HAL_FDCAN_ClockCalibrationCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle
- **pCallback:** pointer to the Clock Calibration Callback function

Return values

- **HAL:** status

HAL_FDCAN_UnRegisterClockCalibrationCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterClockCalibrationCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

UnRegister the Clock Calibration FDCAN Callback Clock Calibration FDCAN Callback is redirected to the weak HAL_FDCAN_ClockCalibrationCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle

Return values

- **HAL:** status

HAL_FDCAN_RegisterTxEventFifoCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterTxEventFifoCallback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_TxEventFifoCallbackTypeDef pCallback)

Function description

Register Tx Event Fifo FDCAN Callback To be used instead of the weak HAL_FDCAN_TxEventFifoCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle
- **pCallback:** pointer to the Tx Event Fifo Callback function

Return values

- **HAL:** status

HAL_FDCAN_UnRegisterTxEventFifoCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterTxEventFifoCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

UnRegister the Tx Event Fifo FDCAN Callback Tx Event Fifo FDCAN Callback is redirected to the weak HAL_FDCAN_TxEventFifoCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle

Return values

- **HAL:** status

HAL_FDCAN_RegisterRxFifo0Callback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterRxFifo0Callback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_RxFifo0CallbackTypeDef pCallback)

Function description

Register Rx Fifo 0 FDCAN Callback To be used instead of the weak HAL_FDCAN_RxFifo0Callback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle
- **pCallback:** pointer to the Rx Fifo 0 Callback function

Return values

- **HAL:** status

HAL_FDCAN_UnRegisterRxFifo0Callback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterRxFifo0Callback (FDCAN_HandleTypeDef * hfdcan)

Function description

UnRegister the Rx Fifo 0 FDCAN Callback Rx Fifo 0 FDCAN Callback is redirected to the weak HAL_FDCAN_RxFifo0Callback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle

Return values

- **HAL:** status

HAL_FDCAN_RegisterRxFifo1Callback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterRxFifo1Callback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_RxFifo1CallbackTypeDef pCallback)

Function description

Register Rx Fifo 1 FDCAN Callback To be used instead of the weak HAL_FDCAN_RxFifo1Callback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle
- **pCallback:** pointer to the Rx Fifo 1 Callback function

Return values

- **HAL:** status

HAL_FDCAN_UnRegisterRxFifo1Callback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterRxFifo1Callback (FDCAN_HandleTypeDef * hfdcan)

Function description

UnRegister the Rx Fifo 1 FDCAN Callback Rx Fifo 1 FDCAN Callback is redirected to the weak HAL_FDCAN_RxFifo1Callback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle

Return values

- **HAL:** status

HAL_FDCAN_RegisterTxBufferCompleteCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterTxBufferCompleteCallback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_TxBufferCompleteCallbackTypeDef pCallback)

Function description

Register Tx Buffer Complete FDCAN Callback To be used instead of the weak HAL_FDCAN_TxBufferCompleteCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle
- **pCallback:** pointer to the Tx Buffer Complete Callback function

Return values

- **HAL:** status

HAL_FDCAN_UnRegisterTxBufferCompleteCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterTxBufferCompleteCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

UnRegister the Tx Buffer Complete FDCAN Callback Tx Buffer Complete FDCAN Callback is redirected to the weak HAL_FDCAN_TxBufferCompleteCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle

Return values

- **HAL:** status

HAL_FDCAN_RegisterTxBufferAbortCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterTxBufferAbortCallback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_TxBufferAbortCallbackTypeDef pCallback)

Function description

Register Tx Buffer Abort FDCAN Callback To be used instead of the weak HAL_FDCAN_TxBufferAbortCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle
- **pCallback:** pointer to the Tx Buffer Abort Callback function

Return values

- **HAL:** status

HAL_FDCAN_UnRegisterTxBufferAbortCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterTxBufferAbortCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

UnRegister the Tx Buffer Abort FDCAN Callback Tx Buffer Abort FDCAN Callback is redirected to the weak HAL_FDCAN_TxBufferAbortCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle

Return values

- **HAL:** status

HAL_FDCAN_RegisterErrorStatusCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterErrorStatusCallback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_ErrorStatusCallbackTypeDef pCallback)

Function description

Register Error Status FDCAN Callback To be used instead of the weak HAL_FDCAN_ErrorStatusCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle
- **pCallback:** pointer to the Error Status Callback function

Return values

- **HAL:** status

HAL_FDCAN_UnRegisterErrorStatusCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterErrorStatusCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

UnRegister the Error Status FDCAN Callback Error Status FDCAN Callback is redirected to the weak HAL_FDCAN_ErrorStatusCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle

Return values

- **HAL:** status

HAL_FDCAN_RegisterTTScheduleSyncCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterTTScheduleSyncCallback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_TT_ScheduleSyncCallbackTypeDef pCallback)

Function description

Register TT Schedule Synchronization FDCAN Callback To be used instead of the weak HAL_FDCAN_TT_ScheduleSyncCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle
- **pCallback:** pointer to the TT Schedule Synchronization Callback function

Return values

- **HAL:** status

HAL_FDCAN_UnRegisterTTScheduleSyncCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterTTScheduleSyncCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

UnRegister the TT Schedule Synchronization FDCAN Callback TT Schedule Synchronization Callback is redirected to the weak HAL_FDCAN_TT_ScheduleSyncCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle

Return values

- **HAL:** status

HAL_FDCAN_RegisterTTTimeMarkCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterTTTimeMarkCallback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_TT_TimeMarkCallbackTypeDef pCallback)

Function description

Register TT Time Mark FDCAN Callback To be used instead of the weak HAL_FDCAN_TT_TimeMarkCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle
- **pCallback:** pointer to the TT Time Mark Callback function

Return values

- **HAL:** status

HAL_FDCAN_UnRegisterTTTimeMarkCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterTTTimeMarkCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

UnRegister the TT Time Mark FDCAN Callback TT Time Mark Callback is redirected to the weak HAL_FDCAN_TT_TimeMarkCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle

Return values

- **HAL:** status

HAL_FDCAN_RegisterTTStopWatchCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterTTStopWatchCallback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_TT_StopWatchCallbackTypeDef pCallback)

Function description

Register TT Stop Watch FDCAN Callback To be used instead of the weak HAL_FDCAN_TT_StopWatchCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle
- **pCallback:** pointer to the TT Stop Watch Callback function

Return values

- **HAL:** status

HAL_FDCAN_UnRegisterTTStopWatchCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterTTStopWatchCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

UnRegister the TT Stop Watch FDCAN Callback TT Stop Watch Callback is redirected to the weak HAL_FDCAN_TT_StopWatchCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle

Return values

- **HAL:** status

HAL_FDCAN_RegisterTTGlobalTimeCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterTTGlobalTimeCallback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_TT_GlobalTimeCallbackTypeDef pCallback)

Function description

Register TT Global Time FDCAN Callback To be used instead of the weak HAL_FDCAN_TT_GlobalTimeCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle
- **pCallback:** pointer to the TT Global Time Callback function

Return values

- **HAL:** status

HAL_FDCAN_UnRegisterTTGlobalTimeCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterTTGlobalTimeCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

UnRegister the TT Global Time FDCAN Callback TT Global Time Callback is redirected to the weak HAL_FDCAN_TT_GlobalTimeCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle

Return values

- **HAL:** status

HAL_FDCAN_ConfigClockCalibration

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigClockCalibration (FDCAN_HandleTypeDef * hfdcan, FDCAN_ClkCalUnitTypeDef * sCcuConfig)

Function description

Configure the FDCAN clock calibration unit according to the specified parameters in the FDCAN_ClkCalUnitTypeDef structure.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **sCcuConfig:** pointer to an FDCAN_ClkCalUnitTypeDef structure that contains the clock calibration information

Return values

- **HAL:** status

HAL_FDCAN_GetClockCalibrationState

Function name

uint32_t HAL_FDCAN_GetClockCalibrationState (FDCAN_HandleTypeDef * hfdcan)

Function description

Get the clock calibration state.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **State:** clock calibration state (can be a value of
  - FDCAN_calibration_state)

HAL_FDCAN_ResetClockCalibrationState

Function name

HAL_StatusTypeDef HAL_FDCAN_ResetClockCalibrationState (FDCAN_HandleTypeDef * hfdcan)

Function description

Reset the clock calibration state.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_GetClockCalibrationCounter

Function name

uint32_t HAL_FDCAN_GetClockCalibrationCounter (FDCAN_HandleTypeDef * hfdcan, uint32_t Counter)

Function description

Get the clock calibration counter value.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **Counter:** clock calibration counter. This parameter can be a value of
  - FDCAN_calibration_counter.

Return values

- **Value:** clock calibration counter value

HAL_FDCAN_ConfigFilter

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigFilter (FDCAN_HandleTypeDef * hfdcan, FDCAN_FilterTypeDef * sFilterConfig)

Function description

Configure the FDCAN reception filter according to the specified parameters in the FDCAN_FilterTypeDef structure.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **sFilterConfig:** pointer to an FDCAN_FilterTypeDef structure that contains the filter configuration information

Return values

- **HAL:** status

HAL_FDCAN_ConfigGlobalFilter

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigGlobalFilter (FDCAN_HandleTypeDef * hfdcan, uint32_t NonMatchingStd, uint32_t NonMatchingExt, uint32_t RejectRemoteStd, uint32_t RejectRemoteExt)

Function description

Configure the FDCAN global filter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **NonMatchingStd:** Defines how received messages with 11-bit IDs that do not match any element of the filter list are treated. This parameter can be a value of
  - FDCAN_Non_Matching_Frames.
- **NonMatchingExt:** Defines how received messages with 29-bit IDs that do not match any element of the filter list are treated. This parameter can be a value of
  - FDCAN_Non_Matching_Frames.
- **RejectRemoteStd:** Filter or reject all the remote 11-bit IDs frames. This parameter can be a value of
  - FDCAN_Reject_Remote_Frames.
- **RejectRemoteExt:** Filter or reject all the remote 29-bit IDs frames. This parameter can be a value of
  - FDCAN_Reject_Remote_Frames.

Return values

- **HAL:** status

HAL_FDCAN_ConfigExtendedIdMask

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigExtendedIdMask (FDCAN_HandleTypeDef * hfdcan, uint32_t Mask)

Function description

Configure the extended ID mask.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **Mask:** Extended ID Mask. This parameter must be a number between 0 and 0x1FFFFFFF

Return values

- **HAL:** status

HAL_FDCAN_ConfigRxFifoOverwrite

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigRxFifoOverwrite (FDCAN_HandleTypeDef * hfdcan, uint32_t RxFifo, uint32_t OperationMode)

Function description

Configure the Rx FIFO operation mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **RxFifo:** Rx FIFO. This parameter can be one of the following values:
  - FDCAN_RX_FIFO0: Rx FIFO 0
  - FDCAN_RX_FIFO1: Rx FIFO 1
- **OperationMode:** operation mode. This parameter can be a value of
  - FDCAN_Rx_FIFO_operation_mode.

Return values

- **HAL:** status

HAL_FDCAN_ConfigFifoWatermark

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigFifoWatermark (FDCAN_HandleTypeDef * hfdcan, uint32_t FIFO, uint32_t Watermark)

Function description

Configure the FIFO watermark.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **FIFO:** select the FIFO to be configured. This parameter can be a value of
  - FDCAN_FIFO_watermark.
- **Watermark:** level for FIFO watermark interrupt. This parameter must be a number between:
  - 0 and 32, if FIFO is FDCAN_CFG_TX_EVENT_FIFO
  - 0 and 64, if FIFO is FDCAN_CFG_RX_FIFO0 or FDCAN_CFG_RX_FIFO1

Return values

- **HAL:** status

HAL_FDCAN_ConfigRamWatchdog

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigRamWatchdog (FDCAN_HandleTypeDef * hfdcan, uint32_t CounterStartValue)

Function description

Configure the RAM watchdog.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **CounterStartValue:** Start value of the Message RAM Watchdog Counter, This parameter must be a number between 0x00 and 0xFF, with the reset value of 0x00 the counter is disabled.

Return values

- **HAL:** status

HAL_FDCAN_ConfigTimestampCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigTimestampCounter (FDCAN_HandleTypeDef * hfdcan, uint32_t TimestampPrescaler)

Function description

Configure the timestamp counter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TimestampPrescaler:** Timestamp Counter Prescaler. This parameter can be a value of
  - FDCAN_Timestamp_Prescaler.

Return values

- **HAL:** status

HAL_FDCAN_EnableTimestampCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_EnableTimestampCounter (FDCAN_HandleTypeDef * hfdcan, uint32_t TimestampOperation)

Function description

Enable the timestamp counter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TimestampOperation:** Timestamp counter operation. This parameter can be a value of
  - FDCAN_Timestamp.

Return values

- **HAL:** status

HAL_FDCAN_DisableTimestampCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_DisableTimestampCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

Disable the timestamp counter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_GetTimestampCounter

Function name

uint16_t HAL_FDCAN_GetTimestampCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

Get the timestamp counter value.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **Value:** Timestamp counter value

HAL_FDCAN_ResetTimestampCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_ResetTimestampCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

Reset the timestamp counter to zero.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_ConfigTimeoutCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigTimeoutCounter (FDCAN_HandleTypeDef * hfdcan, uint32_t TimeoutOperation, uint32_t TimeoutPeriod)

Function description

Configure the timeout counter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TimeoutOperation:** Timeout counter operation. This parameter can be a value of
  - FDCAN_Timeout_Operation.
- **TimeoutPeriod:** Start value of the timeout down-counter. This parameter must be a number between 0x0000 and 0xFFFF

Return values

- **HAL:** status

HAL_FDCAN_EnableTimeoutCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_EnableTimeoutCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

Enable the timeout counter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_DisableTimeoutCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_DisableTimeoutCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

Disable the timeout counter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_GetTimeoutCounter

Function name

uint16_t HAL_FDCAN_GetTimeoutCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

Get the timeout counter value.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **Value:** Timeout counter value

HAL_FDCAN_ResetTimeoutCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_ResetTimeoutCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

Reset the timeout counter to its start value.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_ConfigTxDelayCompensation

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigTxDelayCompensation (FDCAN_HandleTypeDef * hfdcan, uint32_t TdcOffset, uint32_t TdcFilter)

Function description

Configure the transmitter delay compensation.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TdcOffset:** Transmitter Delay Compensation Offset. This parameter must be a number between 0x00 and 0x7F.
- **TdcFilter:** Transmitter Delay Compensation Filter Window Length. This parameter must be a number between 0x00 and 0x7F.

Return values

- **HAL:** status

HAL_FDCAN_EnableTxDelayCompensation

Function name

HAL_StatusTypeDef HAL_FDCAN_EnableTxDelayCompensation (FDCAN_HandleTypeDef * hfdcan)

Function description

Enable the transmitter delay compensation.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_DisableTxDelayCompensation

Function name

HAL_StatusTypeDef HAL_FDCAN_DisableTxDelayCompensation (FDCAN_HandleTypeDef * hfdcan)

Function description

Disable the transmitter delay compensation.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_EnableISOMode

Function name

HAL_StatusTypeDef HAL_FDCAN_EnableISOMode (FDCAN_HandleTypeDef * hfdcan)

Function description

Enable ISO 11898-1 protocol mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_DisableISOMode

Function name

HAL_StatusTypeDef HAL_FDCAN_DisableISOMode (FDCAN_HandleTypeDef * hfdcan)

Function description

Disable ISO 11898-1 protocol mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_EnableEdgeFiltering

Function name

HAL_StatusTypeDef HAL_FDCAN_EnableEdgeFiltering (FDCAN_HandleTypeDef * hfdcan)

Function description

Enable edge filtering during bus integration.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_DisableEdgeFiltering

Function name

HAL_StatusTypeDef HAL_FDCAN_DisableEdgeFiltering (FDCAN_HandleTypeDef * hfdcan)

Function description

Disable edge filtering during bus integration.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_Start

Function name

HAL_StatusTypeDef HAL_FDCAN_Start (FDCAN_HandleTypeDef * hfdcan)

Function description

Start the FDCAN module.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_Stop

Function name

HAL_StatusTypeDef HAL_FDCAN_Stop (FDCAN_HandleTypeDef * hfdcan)

Function description

Stop the FDCAN module and enable access to configuration registers.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_AddMessageToTxFifoQ

Function name

HAL_StatusTypeDef HAL_FDCAN_AddMessageToTxFifoQ (FDCAN_HandleTypeDef * hfdcan, FDCAN_TxHeaderTypeDef * pTxHeader, uint8_t * pTxData)

Function description

Add a message to the Tx FIFO/Queue and activate the corresponding transmission request.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **pTxHeader:** pointer to a FDCAN_TxHeaderTypeDef structure.
- **pTxData:** pointer to a buffer containing the payload of the Tx frame.

Return values

- **HAL:** status

HAL_FDCAN_AddMessageToTxBuffer

Function name

HAL_StatusTypeDef HAL_FDCAN_AddMessageToTxBuffer (FDCAN_HandleTypeDef * hfdcan, FDCAN_TxHeaderTypeDef * pTxHeader, uint8_t * pTxData, uint32_t BufferIndex)

Function description

Add a message to a dedicated Tx buffer.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **pTxHeader:** pointer to a FDCAN_TxHeaderTypeDef structure.
- **pTxData:** pointer to a buffer containing the payload of the Tx frame.
- **BufferIndex:** index of the buffer to be configured. This parameter can be a value of
  - FDCAN_Tx_location.

Return values

- **HAL:** status

HAL_FDCAN_EnableTxBufferRequest

Function name

HAL_StatusTypeDef HAL_FDCAN_EnableTxBufferRequest (FDCAN_HandleTypeDef * hfdcan, uint32_t BufferIndex)

Function description

Enable transmission request.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **BufferIndex:** buffer index. This parameter can be any combination of
  - FDCAN_Tx_location.

Return values

- **HAL:** status

HAL_FDCAN_GetLatestTxFifoQRequestBuffer

Function name

uint32_t HAL_FDCAN_GetLatestTxFifoQRequestBuffer (FDCAN_HandleTypeDef * hfdcan)

Function description

Get Tx buffer index of latest Tx FIFO/Queue request.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **Tx:** buffer index of last Tx FIFO/Queue request
  - Any value of
    - FDCAN_Tx_location if Tx request has been submitted.
  - 0 if no Tx FIFO/Queue request have been submitted.

HAL_FDCAN_AbortTxRequest

Function name

HAL_StatusTypeDef HAL_FDCAN_AbortTxRequest (FDCAN_HandleTypeDef * hfdcan, uint32_t BufferIndex)

Function description

Abort transmission request.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **BufferIndex:** buffer index. This parameter can be any combination of
  - FDCAN_Tx_location.

Return values

- **HAL:** status

HAL_FDCAN_GetRxMessage

Function name

HAL_StatusTypeDef HAL_FDCAN_GetRxMessage (FDCAN_HandleTypeDef * hfdcan, uint32_t RxLocation, FDCAN_RxHeaderTypeDef * pRxHeader, uint8_t * pRxData)

Function description

Get an FDCAN frame from the Rx Buffer/FIFO zone into the message RAM.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **RxLocation:** Location of the received message to be read. This parameter can be a value of
  - FDCAN_Rx_location.
- **pRxHeader:** pointer to a FDCAN_RxHeaderTypeDef structure.
- **pRxData:** pointer to a buffer where the payload of the Rx frame will be stored.

Return values

- **HAL:** status

HAL_FDCAN_GetTxEvent

Function name

HAL_StatusTypeDef HAL_FDCAN_GetTxEvent (FDCAN_HandleTypeDef * hfdcan, FDCAN_TxEventFifoTypeDef * pTxEvent)

Function description

Get an FDCAN Tx event from the Tx Event FIFO zone into the message RAM.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **pTxEvent:** pointer to a FDCAN_TxEventFifoTypeDef structure.

Return values

- **HAL:** status

HAL_FDCAN_GetHighPriorityMessageStatus

Function name

HAL_StatusTypeDef HAL_FDCAN_GetHighPriorityMessageStatus (FDCAN_HandleTypeDef * hfdcan, FDCAN_HpMsgStatusTypeDef * HpMsgStatus)

Function description

Get high priority message status.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **HpMsgStatus:** pointer to an FDCAN_HpMsgStatusTypeDef structure.

Return values

- **HAL:** status

HAL_FDCAN_GetProtocolStatus

Function name

HAL_StatusTypeDef HAL_FDCAN_GetProtocolStatus (FDCAN_HandleTypeDef * hfdcan, FDCAN_ProtocolStatusTypeDef * ProtocolStatus)

Function description

Get protocol status.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **ProtocolStatus:** pointer to an FDCAN_ProtocolStatusTypeDef structure.

Return values

- **HAL:** status

HAL_FDCAN_GetErrorCounters

Function name

HAL_StatusTypeDef HAL_FDCAN_GetErrorCounters (FDCAN_HandleTypeDef * hfdcan, FDCAN_ErrorCountersTypeDef * ErrorCounters)

Function description

Get error counter values.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **ErrorCounters:** pointer to an FDCAN_ErrorCountersTypeDef structure.

Return values

- **HAL:** status

HAL_FDCAN_IsRxBufferMessageAvailable

Function name

uint32_t HAL_FDCAN_IsRxBufferMessageAvailable (FDCAN_HandleTypeDef * hfdcan, uint32_t RxBufferIndex)

Function description

Check if a new message is received in the selected Rx buffer.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **RxBufferIndex:** Rx buffer index. This parameter must be a number between 0 and 63.

Return values

- **Status:**
  - 0 : No new message on RxBufferIndex.
  - 1 : New message received on RxBufferIndex.

HAL_FDCAN_IsTxBufferMessagePending

Function name

uint32_t HAL_FDCAN_IsTxBufferMessagePending (FDCAN_HandleTypeDef * hfdcan, uint32_t TxBufferIndex)

Function description

Check if a transmission request is pending on the selected Tx buffer.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TxBufferIndex:** Tx buffer index. This parameter can be any combination of
  - FDCAN_Tx_location.

Return values

- **Status:**
  - 0 : No pending transmission request on TxBufferIndex.
  - 1 : Pending transmission request on TxBufferIndex.

HAL_FDCAN_GetRxFifoFillLevel

Function name

uint32_t HAL_FDCAN_GetRxFifoFillLevel (FDCAN_HandleTypeDef * hfdcan, uint32_t RxFifo)

Function description

Return Rx FIFO fill level.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **RxFifo:** Rx FIFO. This parameter can be one of the following values:
  - FDCAN_RX_FIFO0: Rx FIFO 0
  - FDCAN_RX_FIFO1: Rx FIFO 1

Return values

- **Level:** Rx FIFO fill level.

HAL_FDCAN_GetTxFifoFreeLevel

Function name

uint32_t HAL_FDCAN_GetTxFifoFreeLevel (FDCAN_HandleTypeDef * hfdcan)

Function description

Return Tx FIFO free level: number of consecutive free Tx FIFO elements starting from Tx FIFO GetIndex.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **Level:** Tx FIFO free level.

HAL_FDCAN_IsRestrictedOperationMode

Function name

uint32_t HAL_FDCAN_IsRestrictedOperationMode (FDCAN_HandleTypeDef * hfdcan)

Function description

Check if the FDCAN peripheral entered Restricted Operation Mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **Status:**
  - 0 : Normal FDCAN operation.
  - 1 : Restricted Operation Mode active.

HAL_FDCAN_ExitRestrictedOperationMode

Function name

HAL_StatusTypeDef HAL_FDCAN_ExitRestrictedOperationMode (FDCAN_HandleTypeDef * hfdcan)

Function description

Exit Restricted Operation Mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_TT_ConfigOperation

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_ConfigOperation (FDCAN_HandleTypeDef * hfdcan, FDCAN_TT_ConfigTypeDef * pTTParams)

Function description

Initialize TT operation parameters.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **pTTParams:** pointer to a FDCAN_TT_ConfigTypeDef structure.

Return values

- **HAL:** status

HAL_FDCAN_TT_ConfigReferenceMessage

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_ConfigReferenceMessage (FDCAN_HandleTypeDef * hfdcan, uint32_t IdType, uint32_t Identifier, uint32_t Payload)

Function description

Configure the reference message.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **IdType:** Identifier Type. This parameter can be a value of
  - FDCAN_id_type.
- **Identifier:** Reference Identifier. This parameter must be a number between:
  - 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
  - 0 and 0x1FFFFFFF, if IdType is FDCAN_EXTENDED_ID
- **Payload:** Enable or disable the additional payload. This parameter can be a value of
  - FDCAN_TT_Reference_Message_Payload. This parameter is ignored in case of time slaves. If this parameter is set to FDCAN_TT_REF_MESSAGE_ADD_PAYLOAD, the following elements are taken from Tx Buffer 0:
    - MessageMarker
    - TxEventFifoControl
    - DataLength
    - Data Bytes (payload):
      - bytes 2-8, for Level 1
      - bytes 5-8, for Level 0 and Level 2

Return values

- **HAL:** status

HAL_FDCAN_TT_ConfigTrigger

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_ConfigTrigger (FDCAN_HandleTypeDef * hfdcan, FDCAN_TriggerTypeDef * sTriggerConfig)

Function description

Configure the FDCAN trigger according to the specified parameters in the FDCAN_TriggerTypeDef structure.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **sTriggerConfig:** pointer to an FDCAN_TriggerTypeDef structure that contains the trigger configuration information

Return values

- **HAL:** status

HAL_FDCAN_TT_SetGlobalTime

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_SetGlobalTime (FDCAN_HandleTypeDef * hfdcan, uint32_t TimePreset)

Function description

Schedule global time adjustment for the next reference message.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TimePreset:** time preset value. This parameter must be a number between:
  - 0x0000 and 0x7FFF, Next_Master_Ref_Mark = Current_Master_Ref_Mark + TimePreset or
  - 0x8001 and 0xFFFF, Next_Master_Ref_Mark = Current_Master_Ref_Mark - (0x10000 - TimePreset)

Return values

- **HAL:** status

HAL_FDCAN_TT_SetClockSynchronization

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_SetClockSynchronization (FDCAN_HandleTypeDef * hfdcan, uint32_t NewTURNumerator)

Function description

Schedule TUR numerator update for the next reference message.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **NewTURNumerator:** new value of the TUR numerator. This parameter must be a number between 0x10000 and 0x1FFFF.

Return values

- **HAL:** status

HAL_FDCAN_TT_ConfigStopWatch

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_ConfigStopWatch (FDCAN_HandleTypeDef * hfdcan, uint32_t Source, uint32_t Polarity)

Function description

Configure stop watch source and polarity.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **Source:** stop watch source. This parameter can be a value of
  - FDCAN_TT_stop_watch_source.
- **Polarity:** stop watch polarity. This parameter can be a value of
  - FDCAN_TT_stop_watch_polarity.

Return values

- **HAL:** status

HAL_FDCAN_TT_ConfigRegisterTimeMark

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_ConfigRegisterTimeMark (FDCAN_HandleTypeDef * hfdcan, uint32_t TimeMarkSource, uint32_t TimeMarkValue, uint32_t RepeatFactor, uint32_t StartCycle)

Function description

Configure register time mark pulse generation.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TimeMarkSource:** time mark source. This parameter can be a value of
  - FDCAN_TT_time_mark_source.
- **TimeMarkValue:** time mark value (reference). This parameter must be a number between 0 and 0xFFFF.
- **RepeatFactor:** repeat factor of the cycle for which the time mark is valid. This parameter can be a value of
  - FDCAN_TT_Repeat_Factor.
- **StartCycle:** index of the first cycle in which the time mark becomes valid. This parameter is ignored if RepeatFactor is set to FDCAN_TT_REPEAT_EVERY_CYCLE. This parameter must be a number between 0 and RepeatFactor.

Return values

- **HAL:** status

HAL_FDCAN_TT_EnableRegisterTimeMarkPulse

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_EnableRegisterTimeMarkPulse (FDCAN_HandleTypeDef * hfdcan)

Function description

Enable register time mark pulse generation.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_TT_DisableRegisterTimeMarkPulse

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_DisableRegisterTimeMarkPulse (FDCAN_HandleTypeDef * hfdcan)

Function description

Disable register time mark pulse generation.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_TT_EnableTriggerTimeMarkPulse

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_EnableTriggerTimeMarkPulse (FDCAN_HandleTypeDef * hfdcan)

Function description

Enable trigger time mark pulse generation.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_TT_DisableTriggerTimeMarkPulse

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_DisableTriggerTimeMarkPulse (FDCAN_HandleTypeDef * hfdcan)

Function description

Disable trigger time mark pulse generation.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_TT_EnableHardwareGapControl

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_EnableHardwareGapControl (FDCAN_HandleTypeDef * hfdcan)

Function description

Enable gap control by input pin fdcan1_evt.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_TT_DisableHardwareGapControl

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_DisableHardwareGapControl (FDCAN_HandleTypeDef * hfdcan)

Function description

Disable gap control by input pin fdcan1_evt.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_TT_EnableTimeMarkGapControl

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_EnableTimeMarkGapControl (FDCAN_HandleTypeDef * hfdcan)

Function description

Enable gap control (finish only) by register time mark interrupt.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_TT_DisableTimeMarkGapControl

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_DisableTimeMarkGapControl (FDCAN_HandleTypeDef * hfdcan)

Function description

Disable gap control by register time mark interrupt.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_TT_SetNextIsGap

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_SetNextIsGap (FDCAN_HandleTypeDef * hfdcan)

Function description

Transmit next reference message with Next_is_Gap = "1".

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_TT_SetEndOfGap

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_SetEndOfGap (FDCAN_HandleTypeDef * hfdcan)

Function description

Finish a Gap by requesting start of reference message.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_TT_ConfigExternalSyncPhase

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_ConfigExternalSyncPhase (FDCAN_HandleTypeDef * hfdcan, uint32_t TargetPhase)

Function description

Configure target phase used for external synchronization by event trigger input pin fdcan1_evt.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TargetPhase:** defines target value of cycle time when a rising edge of fdcan1_evt is expected. This parameter must be a number between 0 and 0xFFFF.

Return values

- **HAL:** status

HAL_FDCAN_TT_EnableExternalSynchronization

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_EnableExternalSynchronization (FDCAN_HandleTypeDef * hfdcan)

Function description

Synchronize the phase of the FDCAN schedule to an external schedule using event trigger input pin fdcan1_evt.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_TT_DisableExternalSynchronization

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_DisableExternalSynchronization (FDCAN_HandleTypeDef * hfdcan)

Function description

Disable external schedule synchronization.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_TT_GetOperationStatus

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_GetOperationStatus (FDCAN_HandleTypeDef * hfdcan, FDCAN_TTOperationStatusTypeDef * TTOpStatus)

Function description

Get TT operation status.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TTOpStatus:** pointer to an FDCAN_TTOperationStatusTypeDef structure.

Return values

- **HAL:** status

HAL_FDCAN_ConfigInterruptLines

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigInterruptLines (FDCAN_HandleTypeDef * hfdcan, uint32_t ITList, uint32_t InterruptLine)

Function description

Assign interrupts to either Interrupt line 0 or 1.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **ITList:** indicates which interrupts will be assigned to the selected interrupt line. This parameter can be any combination of
  - FDCAN_Interrupts.
- **InterruptLine:** Interrupt line. This parameter can be a value of
  - FDCAN_Interrupt_Line.

Return values

- **HAL:** status

HAL_FDCAN_TT_ConfigInterruptLines

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_ConfigInterruptLines (FDCAN_HandleTypeDef * hfdcan, uint32_t TTITList, uint32_t InterruptLine)

Function description

Assign TT interrupts to either Interrupt line 0 or 1.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TTITList:** indicates which interrupts will be assigned to the selected interrupt line. This parameter can be any combination of
  - FDCAN_TTInterrupts.
- **InterruptLine:** Interrupt line. This parameter can be a value of
  - FDCAN_Interrupt_Line.

Return values

- **HAL:** status

HAL_FDCAN_ActivateNotification

Function name

HAL_StatusTypeDef HAL_FDCAN_ActivateNotification (FDCAN_HandleTypeDef * hfdcan, uint32_t ActiveITs, uint32_t BufferIndexes)

Function description

Enable interrupts.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **ActiveITs:** indicates which interrupts will be enabled. This parameter can be any combination of
  - FDCAN_Interrupts.
- **BufferIndexes:** Tx Buffer Indexes. This parameter can be any combination of
  - FDCAN_Tx_location. This parameter is ignored if ActiveITs does not include one of the following:
    - FDCAN_IT_TX_COMPLETE
    - FDCAN_IT_TX_ABORT_COMPLETE

Return values

- **HAL:** status

HAL_FDCAN_DeactivateNotification

Function name

HAL_StatusTypeDef HAL_FDCAN_DeactivateNotification (FDCAN_HandleTypeDef * hfdcan, uint32_t InactiveITs)

Function description

Disable interrupts.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **InactiveITs:** indicates which interrupts will be disabled. This parameter can be any combination of
  - FDCAN_Interrupts.

Return values

- **HAL:** status

HAL_FDCAN_TT_ActivateNotification

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_ActivateNotification (FDCAN_HandleTypeDef * hfdcan, uint32_t ActiveTTITs)

Function description

Enable TT interrupts.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **ActiveTTITs:** indicates which TT interrupts will be enabled. This parameter can be any combination of
  - FDCAN_TTInterrupts.

Return values

- **HAL:** status

HAL_FDCAN_TT_DeactivateNotification

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_DeactivateNotification (FDCAN_HandleTypeDef * hfdcan, uint32_t InactiveTTITs)

Function description

Disable TT interrupts.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **InactiveTTITs:** indicates which TT interrupts will be disabled. This parameter can be any combination of
  – FDCAN_TTInterrupts.

Return values

- **HAL:** status

HAL_FDCAN_IRQHandler

Function name

void HAL_FDCAN_IRQHandler (FDCAN_HandleTypeDef * hfdcan)

Function description

Handles FDCAN interrupt request.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_ClockCalibrationCallback

Function name

void HAL_FDCAN_ClockCalibrationCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t ClkCalibrationITs)

Function description

Clock Calibration callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **ClkCalibrationITs:** indicates which Clock Calibration interrupts are signaled. This parameter can be any combination of
  – FDCAN_Clock_Calibration_Interrupts.

Return values

- **None:**

HAL_FDCAN_TxEventFifoCallback

Function name

void HAL_FDCAN_TxEventFifoCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t TxEventFifoITs)

Function description

Tx Event callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TxEventFifoITs:** indicates which Tx Event FIFO interrupts are signaled. This parameter can be any combination of
  – FDCAN_Tx_Event_Fifo_Interrupts.

Return values

- **None:**

HAL_FDCAN_RxFifo0Callback

Function name

void HAL_FDCAN_RxFifo0Callback (FDCAN_HandleTypeDef * hfdcan, uint32_t RxFifo0ITs)

Function description

Rx FIFO 0 callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **RxFifo0ITs:** indicates which Rx FIFO 0 interrupts are signaled. This parameter can be any combination of
  – FDCAN_Rx_Fifo0_Interrupts.

Return values

- **None:**

HAL_FDCAN_RxFifo1Callback

Function name

void HAL_FDCAN_RxFifo1Callback (FDCAN_HandleTypeDef * hfdcan, uint32_t RxFifo1ITs)

Function description

Rx FIFO 1 callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **RxFifo1ITs:** indicates which Rx FIFO 1 interrupts are signaled. This parameter can be any combination of
  – FDCAN_Rx_Fifo1_Interrupts.

Return values

- **None:**

HAL_FDCAN_TxFifoEmptyCallback

Function name

void HAL_FDCAN_TxFifoEmptyCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

Tx FIFO Empty callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_TxBufferCompleteCallback

Function name

void HAL_FDCAN_TxBufferCompleteCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t BufferIndexes)

Function description

Transmission Complete callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **BufferIndexes:** Indexes of the transmitted buffers. This parameter can be any combination of
  - FDCAN_Tx_location.

Return values

- **None:**

HAL_FDCAN_TxBufferAbortCallback

Function name

void HAL_FDCAN_TxBufferAbortCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t BufferIndexes)

Function description

Transmission Cancellation callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **BufferIndexes:** Indexes of the aborted buffers. This parameter can be any combination of
  - FDCAN_Tx_location.

Return values

- **None:**

HAL_FDCAN_RxBufferNewMessageCallback

Function name

void HAL_FDCAN_RxBufferNewMessageCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

Rx Buffer New Message callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_HighPriorityMessageCallback

Function name

void HAL_FDCAN_HighPriorityMessageCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

High Priority Message callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_TimestampWraparoundCallback

Function name

void HAL_FDCAN_TimestampWraparoundCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

Timestamp Wraparound callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_TimeoutOccurredCallback

Function name

void HAL_FDCAN_TimeoutOccurredCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

Timeout Occurred callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_ErrorCallback

Function name

void HAL_FDCAN_ErrorCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

Error callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_ErrorStatusCallback

Function name

void HAL_FDCAN_ErrorStatusCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t ErrorStatusITs)

Function description

Error status callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **ErrorStatusITs:** indicates which Error Status interrupts are signaled. This parameter can be any combination of
  - FDCAN_Error_Status_Interrupts.

Return values

- **None:**

HAL_FDCAN_TT_ScheduleSyncCallback

Function name

void HAL_FDCAN_TT_ScheduleSyncCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t TTSchedSyncITs)

Function description

TT Schedule Synchronization callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TTSchedSyncITs:** indicates which TT Schedule Synchronization interrupts are signaled. This parameter can be any combination of
  - FDCAN_TTScheduleSynchronization_Interrupts.

Return values

- **None:**

HAL_FDCAN_TT_TimeMarkCallback

Function name

void HAL_FDCAN_TT_TimeMarkCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t TTTimeMarkITs)

Function description

TT Time Mark callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TTTimeMarkITs:** indicates which TT Schedule Synchronization interrupts are signaled. This parameter can be any combination of
  - FDCAN_TTTimeMark_Interrupts.

Return values

- **None:**

HAL_FDCAN_TT_StopWatchCallback

Function name

void HAL_FDCAN_TT_StopWatchCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t SWTime, uint32_t SWCycleCount)

Function description

TT Stop Watch callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **SWTime:** Time Value captured at the Stop Watch Trigger pin (fdcan1_swt) falling/rising edge (as configured via HAL_FDCAN_TTConfigStopWatch). This parameter is a number between 0 and 0xFFFF.
- **SWCycleCount:** Cycle count value captured together with SWTime. This parameter is a number between 0 and 0x3F.

Return values

- **None:**

HAL_FDCAN_TT_GlobalTimeCallback

Function name

void HAL_FDCAN_TT_GlobalTimeCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t TTGlobTimeITs)

Function description

TT Global Time callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TTGlobTimeITs:** indicates which TT Global Time interrupts are signaled. This parameter can be any combination of
  - FDCAN_TTGlobalTime_Interrupts.

Return values

- **None:**

HAL_FDCAN_GetError

Function name

uint32_t HAL_FDCAN_GetError (FDCAN_HandleTypeDef * hfdcan)

Function description

Return the FDCAN error code.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **FDCAN:** Error Code

HAL_FDCAN_GetState

Function name

HAL_FDCAN_StateTypeDef HAL_FDCAN_GetState (FDCAN_HandleTypeDef * hfdcan)

Function description

Return the FDCAN state.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** state

30.3 FDCAN Firmware driver defines

The following section lists the various define and macros of the module.

30.3.1 FDCAN

FDCAN

FDCAN Bit Rate Switching

FDCAN_BRS_OFF

FDCAN frames transmitted/received without bit rate switching

FDCAN_BRS_ON

FDCAN frames transmitted/received with bit rate switching

FDCAN Calibration Counter

FDCAN_CALIB_TIME_QUANTA_COUNTER

Time Quanta Counter

FDCAN_CALIB_CLOCK_PERIOD_COUNTER

Oscillator Clock Period Counter

FDCAN_CALIB_WATCHDOG_COUNTER

Calibration Watchdog Counter

FDCAN Calibration Field Length

FDCAN_CALIB_FIELD_LENGTH_32

Calibration field length is 32 bits

FDCAN_CALIB_FIELD_LENGTH_64

Calibration field length is 64 bits

FDCAN Calibration State

FDCAN_CLOCK_NOT_CALIBRATED

Clock not calibrated

FDCAN_CLOCK_BASIC_CALIBRATED

Clock basic calibrated

FDCAN_CLOCK_PRECISION_CALIBRATED

Clock precision calibrated

FDCAN Clock Calibration

FDCAN_CLOCK_CALIBRATION_DISABLE

Disable Clock Calibration

FDCAN_CLOCK_CALIBRATION_ENABLE

Enable Clock Calibration

Clock Calibration Interrupts

FDCAN_IT_CALIB_STATE_CHANGED

Clock calibration state changed

FDCAN_IT_CALIB_WATCHDOG_EVENT

Clock calibration watchdog event occurred

FDCAN Clock Divider

FDCAN_CLOCK_DIV1

Divide kernel clock by 1

FDCAN_CLOCK_DIV2

Divide kernel clock by 2

FDCAN_CLOCK_DIV4

Divide kernel clock by 4

FDCAN_CLOCK_DIV6

Divide kernel clock by 6

FDCAN_CLOCK_DIV8

Divide kernel clock by 8

FDCAN_CLOCK_DIV10

Divide kernel clock by 10

FDCAN_CLOCK_DIV12

Divide kernel clock by 12

FDCAN_CLOCK_DIV14

Divide kernel clock by 14

FDCAN_CLOCK_DIV16

Divide kernel clock by 16

FDCAN_CLOCK_DIV18

Divide kernel clock by 18

FDCAN_CLOCK_DIV20

Divide kernel clock by 20

FDCAN_CLOCK_DIV22

Divide kernel clock by 22

FDCAN_CLOCK_DIV24

Divide kernel clock by 24

FDCAN_CLOCK_DIV26

Divide kernel clock by 26

FDCAN_CLOCK_DIV28

Divide kernel clock by 28

FDCAN_CLOCK_DIV30

Divide kernel clock by 30

FDCAN communication state

FDCAN_COM_STATE_SYNC

Node is synchronizing on CAN communication

FDCAN_COM_STATE_IDLE

Node is neither receiver nor transmitter

FDCAN_COM_STATE_RX

Node is operating as receiver

FDCAN_COM_STATE_TX

Node is operating as transmitter

FDCAN Counter Interrupts

FDCAN_IT_TIMESTAMP_WRAPAROUND

Timestamp counter wrapped around

FDCAN_IT_TIMEOUT_OCCURRED

Timeout reached

FDCAN Data Field Size

FDCAN_DATA_BYTES_8

8 bytes data field

FDCAN_DATA_BYTES_12

12 bytes data field

FDCAN_DATA_BYTES_16

16 bytes data field

FDCAN_DATA_BYTES_20

20 bytes data field

FDCAN_DATA_BYTES_24

24 bytes data field

FDCAN_DATA_BYTES_32

32 bytes data field

FDCAN_DATA_BYTES_48

48 bytes data field

FDCAN_DATA_BYTES_64

64 bytes data field

FDCAN Data Length Code

FDCAN_DLC_BYTES_0

0 bytes data field

FDCAN_DLC_BYTES_1

1 bytes data field

FDCAN_DLC_BYTES_2

2 bytes data field

FDCAN_DLC_BYTES_3

3 bytes data field

FDCAN_DLC_BYTES_4

4 bytes data field

FDCAN_DLC_BYTES_5

5 bytes data field

FDCAN_DLC_BYTES_6

6 bytes data field

FDCAN_DLC_BYTES_7

7 bytes data field

FDCAN_DLC_BYTES_8

8 bytes data field

FDCAN_DLC_BYTES_12

12 bytes data field

FDCAN_DLC_BYTES_16

16 bytes data field

FDCAN_DLC_BYTES_20

20 bytes data field

FDCAN_DLC_BYTES_24

24 bytes data field

FDCAN_DLC_BYTES_32

32 bytes data field

FDCAN_DLC_BYTES_48

48 bytes data field

FDCAN_DLC_BYTES_64

64 bytes data field

FDCAN Error Interrupts

FDCAN_IT_RAM_ACCESS_FAILURE

Message RAM access failure occurred

FDCAN_IT_ERROR_LOGGING_OVERFLOW

Overflow of FDCAN Error Logging Counter occurred

FDCAN_IT_RAM_WATCHDOG

Message RAM Watchdog event due to missing READY

FDCAN_IT_ARB_PROTOCOL_ERROR

Protocol error in arbitration phase detected

FDCAN_IT_DATA_PROTOCOL_ERROR

Protocol error in data phase detected

FDCAN_IT_RESERVED_ADDRESS_ACCESS

Access to reserved address occurred

FDCAN Error State Indicator

FDCAN_ESI_ACTIVE

Transmitting node is error active

FDCAN_ESI_PASSIVE

Transmitting node is error passive

FDCAN Error Status Interrupts

FDCAN_IT_ERROR_PASSIVE

Error_Passive status changed

FDCAN_IT_ERROR_WARNING

Error_Warning status changed

FDCAN_IT_BUS_OFF

Bus_Off status changed

FDCAN Event Type

FDCAN_TX_EVENT

Tx event

FDCAN_TX_IN_SPITE_OF_ABORT

Transmission in spite of cancellation

FDCAN Exported Macros

__HAL_FDCAN_RESET_HANDLE_STATE

Description:

- Reset FDCAN handle state.

Parameters:

- __HANDLE__: FDCAN handle.

Return value:

- None

__HAL_FDCAN_ENABLE_IT

Description:

- Enable the specified FDCAN interrupts.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: FDCAN interrupt. This parameter can be any combination of
  - FDCAN_Interrupts

Return value:

- None

__HAL_FDCAN_DISABLE_IT

Description:

- Disable the specified FDCAN interrupts.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: FDCAN interrupt. This parameter can be any combination of
  - FDCAN_Interrupts

Return value:

- None

__HAL_FDCAN_GET_IT

Description:

- Check whether the specified FDCAN interrupt is set or not.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: FDCAN interrupt. This parameter can be one of
  - FDCAN_Interrupts

Return value:

- ITStatus

__HAL_FDCAN_CLEAR_IT

Description:

- Clear the specified FDCAN interrupts.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: specifies the interrupts to clear. This parameter can be any combination of
  - FDCAN_Interrupts

Return value:

- None

__HAL_FDCAN_GET_FLAG

Description:

- Check whether the specified FDCAN flag is set or not.

Parameters:

- __HANDLE__: FDCAN handle.
- __FLAG__: FDCAN flag. This parameter can be one of
  - FDCAN_flags

Return value:

- FlagStatus

__HAL_FDCAN_CLEAR_FLAG

Description:

- Clear the specified FDCAN flags.

Parameters:

- __HANDLE__: FDCAN handle.
- __FLAG__: specifies the flags to clear. This parameter can be any combination of
  - FDCAN_flags

Return value:

- None

__HAL_FDCAN_GET_IT_SOURCE

Description:

- Check if the specified FDCAN interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: specifies the FDCAN interrupt source to check. This parameter can be a value of
  - FDCAN_Interrupts

Return value:

- ITStatus

__HAL_FDCAN_TT_ENABLE_IT

Description:

- Enable the specified FDCAN TT interrupts.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: FDCAN TT interrupt. This parameter can be any combination of
  - FDCAN_TTInterrupts

Return value:

- None

__HAL_FDCAN_TT_DISABLE_IT

Description:

- Disable the specified FDCAN TT interrupts.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: FDCAN TT interrupt. This parameter can be any combination of
  - FDCAN_TTInterrupts

Return value:

- None

__HAL_FDCAN_TT_GET_IT

Description:

- Check whether the specified FDCAN TT interrupt is set or not.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: FDCAN TT interrupt. This parameter can be one of
  - FDCAN_TTInterrupts

Return value:

- ITStatus

__HAL_FDCAN_TT_CLEAR_IT

Description:

- Clear the specified FDCAN TT interrupts.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: specifies the TT interrupts to clear. This parameter can be any combination of
  - FDCAN_TTInterrupts

Return value:

- None

__HAL_FDCAN_TT_GET_FLAG

Description:

- Check whether the specified FDCAN TT flag is set or not.

Parameters:

- __HANDLE__: FDCAN handle.
- __FLAG__: FDCAN TT flag. This parameter can be one of
  - FDCAN_TTflags

Return value:

- FlagStatus

__HAL_FDCAN_TT_CLEAR_FLAG

Description:

- Clear the specified FDCAN TT flags.

Parameters:

- __HANDLE__: FDCAN handle.
- __FLAG__: specifies the TT flags to clear. This parameter can be any combination of
  - FDCAN_TTflags

Return value:

- None

__HAL_FDCAN_TT_GET_IT_SOURCE

Description:

- Check if the specified FDCAN TT interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: specifies the FDCAN TT interrupt source to check. This parameter can be a value of
  - FDCAN_TTInterrupts

Return value:

- ITStatus

FDCAN FIFO watermark

FDCAN_CFG_TX_EVENT_FIFO

Tx event FIFO

FDCAN_CFG_RX_FIFO0

Rx FIFO0

FDCAN_CFG_RX_FIFO1

Rx FIFO1

FDCAN Filter Configuration

FDCAN_FILTER_DISABLE

Disable filter element

FDCAN_FILTER_TO_RXFIFO0

Store in Rx FIFO 0 if filter matches

FDCAN_FILTER_TO_RXFIFO1

Store in Rx FIFO 1 if filter matches

FDCAN_FILTER_REJECT

Reject ID if filter matches

FDCAN_FILTER_HP

Set high priority if filter matches

FDCAN_FILTER_TO_RXFIFO0_HP

Set high priority and store in FIFO 0 if filter matches

FDCAN_FILTER_TO_RXFIFO1_HP

Set high priority and store in FIFO 1 if filter matches

FDCAN_FILTER_TO_RXBUFFER

Store into Rx Buffer, configuration of FilterType ignored

FDCAN Filter Type

FDCAN_FILTER_RANGE

Range filter from FilterID1 to FilterID2

FDCAN_FILTER_DUAL

Dual ID filter for FilterID1 or FilterID2

FDCAN_FILTER_MASK

Classic filter: FilterID1 = filter, FilterID2 = mask

FDCAN_FILTER_RANGE_NO_EIDM

Range filter from FilterID1 to FilterID2, EIDM mask not applied

FDCAN Flags

FDCAN_FLAG_TX_COMPLETE

Transmission Completed

FDCAN_FLAG_TX_ABORT_COMPLETE

Transmission Cancellation Finished

FDCAN_FLAG_TX_FIFO_EMPTY

Tx FIFO Empty

FDCAN_FLAG_RX_HIGH_PRIORITY_MSG

High priority message received

FDCAN_FLAG_RX_BUFFER_NEW_MESSAGE

At least one received message stored into a Rx Buffer

FDCAN_FLAG_TX_EVT_FIFO_ELT_LOST

Tx Event FIFO element lost

FDCAN_FLAG_TX_EVT_FIFO_FULL

Tx Event FIFO full

FDCAN_FLAG_TX_EVT_FIFO_WATERMARK

Tx Event FIFO fill level reached watermark

FDCAN_FLAG_TX_EVT_FIFO_NEW_DATA

Tx Handler wrote Tx Event FIFO element

FDCAN_FLAG_RX_FIFO0_MESSAGE_LOST

Rx FIFO 0 message lost

FDCAN_FLAG_RX_FIFO0_FULL

Rx FIFO 0 full

FDCAN_FLAG_RX_FIFO0_WATERMARK

Rx FIFO 0 fill level reached watermark

FDCAN_FLAG_RX_FIFO0_NEW_MESSAGE

New message written to Rx FIFO 0

FDCAN_FLAG_RX_FIFO1_MESSAGE_LOST

Rx FIFO 1 message lost

FDCAN_FLAG_RX_FIFO1_FULL

Rx FIFO 1 full

FDCAN_FLAG_RX_FIFO1_WATERMARK

Rx FIFO 1 fill level reached watermark

FDCAN_FLAG_RX_FIFO1_NEW_MESSAGE

New message written to Rx FIFO 1

FDCAN_FLAG_RAM_ACCESS_FAILURE

Message RAM access failure occurred

FDCAN_FLAG_ERROR_LOGGING_OVERFLOW

Overflow of FDCAN Error Logging Counter occurred

FDCAN_FLAG_ERROR_PASSIVE

Error_Passive status changed

FDCAN_FLAG_ERROR_WARNING

Error_Warning status changed

FDCAN_FLAG_BUS_OFF

Bus_Off status changed

FDCAN_FLAG_RAM_WATCHDOG

Message RAM Watchdog event due to missing READY

FDCAN_FLAG_ARB_PROTOCOL_ERROR

Protocol error in arbitration phase detected

FDCAN_FLAG_DATA_PROTOCOL_ERROR

Protocol error in data phase detected

FDCAN_FLAG_RESERVED_ADDRESS_ACCESS

Access to reserved address occurred

FDCAN_FLAG_TIMESTAMP_WRAPAROUND

Timestamp counter wrapped around

FDCAN_FLAG_TIMEOUT_OCCURRED

Timeout reached

FDCAN_FLAG_CALIB_STATE_CHANGED

Clock calibration state changed

FDCAN_FLAG_CALIB_WATCHDOG_EVENT

Clock calibration watchdog event occurred

FDCAN format

FDCAN_CLASSIC_CAN

Frame transmitted/received in Classic CAN format

FDCAN_FD_CAN

Frame transmitted/received in FDCAN format

FDCAN Frame Format

FDCAN_FRAME_CLASSIC

Classic mode

FDCAN_FRAME_FD_NO_BRS

FD mode without BitRate Switching

FDCAN_FRAME_FD_BRS

FD mode with BitRate Switching

FDCAN Frame Type

FDCAN_DATA_FRAME

Data frame

FDCAN_REMOTE_FRAME

Remote frame

FDCAN High Priority Message Storage

FDCAN_HP_STORAGE_NO_FIFO

No FIFO selected

FDCAN_HP_STORAGE_MSG_LOST

FIFO message lost

FDCAN_HP_STORAGE_RXFIFO0

Message stored in FIFO 0

FDCAN_HP_STORAGE_RXFIFO1

Message stored in FIFO 1

FDCAN ID Type

FDCAN_STANDARD_ID

Standard ID element

FDCAN_EXTENDED_ID

Extended ID element

FDCAN interrupt line

FDCAN_INTERRUPT_LINE0

Interrupt Line 0

FDCAN_INTERRUPT_LINE1

Interrupt Line 1

FDCAN non-matching frames

FDCAN_ACCEPT_IN_RX_FIFO0

Accept in Rx FIFO 0

FDCAN_ACCEPT_IN_RX_FIFO1

Accept in Rx FIFO 1

FDCAN_REJECT

Reject

FDCAN Operating Mode

FDCAN_MODE_NORMAL

Normal mode

FDCAN_MODE_RESTRICTED_OPERATION

Restricted Operation mode

FDCAN_MODE_BUS_MONITORING

Bus Monitoring mode

FDCAN_MODE_INTERNAL_LOOPBACK

Internal LoopBack mode

FDCAN_MODE_EXTERNAL_LOOPBACK

External LoopBack mode

FDCAN Operation Mode

FDCAN_TT_COMMUNICATION_LEVEL1

Time triggered communication, level 1

FDCAN_TT_COMMUNICATION_LEVEL2

Time triggered communication, level 2

FDCAN_TT_COMMUNICATION_LEVEL0

Time triggered communication, level 0

FDCAN protocol error code

FDCAN_PROTOCOL_ERROR_NONE

No error occurred

FDCAN_PROTOCOL_ERROR_STUFF

Stuff error

FDCAN_PROTOCOL_ERROR_FORM

Form error

FDCAN_PROTOCOL_ERROR_ACK

Acknowledge error

FDCAN_PROTOCOL_ERROR_BIT1

Bit 1 (recessive) error

FDCAN_PROTOCOL_ERROR_BIT0

Bit 0 (dominant) error

FDCAN_PROTOCOL_ERROR_CRC

CRC check sum error

FDCAN_PROTOCOL_ERROR_NO_CHANGE

No change since last read

FDCAN reject remote frames

FDCAN_FILTER_REMOTE

Filter remote frames

FDCAN_REJECT_REMOTE

Reject all remote frames

FDCAN Rx FIFO 0 Interrupts

FDCAN_IT_RX_FIFO0_MESSAGE_LOST

Rx FIFO 0 message lost

FDCAN_IT_RX_FIFO0_FULL

Rx FIFO 0 full

FDCAN_IT_RX_FIFO0_WATERMARK

Rx FIFO 0 fill level reached watermark

FDCAN_IT_RX_FIFO0_NEW_MESSAGE

New message written to Rx FIFO 0

FDCAN Rx FIFO 1 Interrupts

FDCAN_IT_RX_FIFO1_MESSAGE_LOST

Rx FIFO 1 message lost

FDCAN_IT_RX_FIFO1_FULL

Rx FIFO 1 full

FDCAN_IT_RX_FIFO1_WATERMARK

Rx FIFO 1 fill level reached watermark

FDCAN_IT_RX_FIFO1_NEW_MESSAGE

New message written to Rx FIFO 1

FDCAN FIFO operation mode

FDCAN_RX_FIFO_BLOCKING

Rx FIFO blocking mode

FDCAN_RX_FIFO_OVERWRITE

Rx FIFO overwrite mode

FDCAN Rx Interrupts

FDCAN_IT_RX_HIGH_PRIORITY_MSG

High priority message received

FDCAN_IT_RX_BUFFER_NEW_MESSAGE

At least one received message stored into a Rx Buffer

FDCAN Rx Location

FDCAN_RX_FIFO0

Get received message from Rx FIFO 0

FDCAN_RX_FIFO1

Get received message from Rx FIFO 1

FDCAN_RX_BUFFER0

Get received message from Rx Buffer 0

FDCAN_RX_BUFFER1

Get received message from Rx Buffer 1

FDCAN_RX_BUFFER2

Get received message from Rx Buffer 2

FDCAN_RX_BUFFER3

Get received message from Rx Buffer 3

FDCAN_RX_BUFFER4

Get received message from Rx Buffer 4

FDCAN_RX_BUFFER5

Get received message from Rx Buffer 5

FDCAN_RX_BUFFER6

Get received message from Rx Buffer 6

FDCAN_RX_BUFFER7

Get received message from Rx Buffer 7

FDCAN_RX_BUFFER8

Get received message from Rx Buffer 8

FDCAN_RX_BUFFER9

Get received message from Rx Buffer 9

FDCAN_RX_BUFFER10

Get received message from Rx Buffer 10

FDCAN_RX_BUFFER11

Get received message from Rx Buffer 11

FDCAN_RX_BUFFER12

Get received message from Rx Buffer 12

FDCAN_RX_BUFFER13

Get received message from Rx Buffer 13

FDCAN_RX_BUFFER14

Get received message from Rx Buffer 14

FDCAN_RX_BUFFER15

Get received message from Rx Buffer 15

FDCAN_RX_BUFFER16

Get received message from Rx Buffer 16

FDCAN_RX_BUFFER17

Get received message from Rx Buffer 17

FDCAN_RX_BUFFER18

Get received message from Rx Buffer 18

FDCAN_RX_BUFFER19

Get received message from Rx Buffer 19

FDCAN_RX_BUFFER20

Get received message from Rx Buffer 20

FDCAN_RX_BUFFER21

Get received message from Rx Buffer 21

FDCAN_RX_BUFFER22

Get received message from Rx Buffer 22

FDCAN_RX_BUFFER23

Get received message from Rx Buffer 23

FDCAN_RX_BUFFER24

Get received message from Rx Buffer 24

FDCAN_RX_BUFFER25

Get received message from Rx Buffer 25

FDCAN_RX_BUFFER26

Get received message from Rx Buffer 26

FDCAN_RX_BUFFER27

Get received message from Rx Buffer 27

FDCAN_RX_BUFFER28

Get received message from Rx Buffer 28

FDCAN_RX_BUFFER29

Get received message from Rx Buffer 29

FDCAN_RX_BUFFER30

Get received message from Rx Buffer 30

FDCAN_RX_BUFFER31

Get received message from Rx Buffer 31

FDCAN_RX_BUFFER32

Get received message from Rx Buffer 32

FDCAN_RX_BUFFER33

Get received message from Rx Buffer 33

FDCAN_RX_BUFFER34

Get received message from Rx Buffer 34

FDCAN_RX_BUFFER35

Get received message from Rx Buffer 35

FDCAN_RX_BUFFER36

Get received message from Rx Buffer 36

FDCAN_RX_BUFFER37

Get received message from Rx Buffer 37

FDCAN_RX_BUFFER38

Get received message from Rx Buffer 38

FDCAN_RX_BUFFER39

Get received message from Rx Buffer 39

FDCAN_RX_BUFFER40

Get received message from Rx Buffer 40

FDCAN_RX_BUFFER41

Get received message from Rx Buffer 41

FDCAN_RX_BUFFER42

Get received message from Rx Buffer 42

FDCAN_RX_BUFFER43

Get received message from Rx Buffer 43

FDCAN_RX_BUFFER44

Get received message from Rx Buffer 44

FDCAN_RX_BUFFER45

Get received message from Rx Buffer 45

FDCAN_RX_BUFFER46

Get received message from Rx Buffer 46

FDCAN_RX_BUFFER47

Get received message from Rx Buffer 47

FDCAN_RX_BUFFER48

Get received message from Rx Buffer 48

FDCAN_RX_BUFFER49

Get received message from Rx Buffer 49

FDCAN_RX_BUFFER50

Get received message from Rx Buffer 50

FDCAN_RX_BUFFER51

Get received message from Rx Buffer 51

FDCAN_RX_BUFFER52

Get received message from Rx Buffer 52

FDCAN_RX_BUFFER53

Get received message from Rx Buffer 53

FDCAN_RX_BUFFER54

Get received message from Rx Buffer 54

FDCAN_RX_BUFFER55

Get received message from Rx Buffer 55

FDCAN_RX_BUFFER56

Get received message from Rx Buffer 56

FDCAN_RX_BUFFER57

Get received message from Rx Buffer 57

FDCAN_RX_BUFFER58

Get received message from Rx Buffer 58

FDCAN_RX_BUFFER59

Get received message from Rx Buffer 59

FDCAN_RX_BUFFER60

Get received message from Rx Buffer 60

FDCAN_RX_BUFFER61

Get received message from Rx Buffer 61

FDCAN_RX_BUFFER62

Get received message from Rx Buffer 62

FDCAN_RX_BUFFER63

Get received message from Rx Buffer 63

FDCAN timeout operation

FDCAN_TIMEOUT_CONTINUOUS

Timeout continuous operation

FDCAN_TIMEOUT_TX_EVENT_FIFO

Timeout controlled by Tx Event FIFO

FDCAN_TIMEOUT_RX_FIFO0

Timeout controlled by Rx FIFO 0

FDCAN_TIMEOUT_RX_FIFO1

Timeout controlled by Rx FIFO 1

FDCAN timestamp

FDCAN_TIMESTAMP_INTERNAL

Timestamp counter value incremented according to TCP

FDCAN_TIMESTAMP_EXTERNAL

External timestamp counter value used

FDCAN timestamp prescaler

FDCAN_TIMESTAMP_PRESC_1

Timestamp counter time unit in equal to CAN bit time

FDCAN_TIMESTAMP_PRESC_2

Timestamp counter time unit in equal to CAN bit time multiplied by 2

FDCAN_TIMESTAMP_PRESC_3

Timestamp counter time unit in equal to CAN bit time multiplied by 3

FDCAN_TIMESTAMP_PRESC_4

Timestamp counter time unit in equal to CAN bit time multiplied by 4

FDCAN_TIMESTAMP_PRESC_5

Timestamp counter time unit in equal to CAN bit time multiplied by 5

FDCAN_TIMESTAMP_PRESC_6

Timestamp counter time unit in equal to CAN bit time multiplied by 6

FDCAN_TIMESTAMP_PRESC_7

Timestamp counter time unit in equal to CAN bit time multiplied by 7

FDCAN_TIMESTAMP_PRESC_8

Timestamp counter time unit in equal to CAN bit time multiplied by 8

FDCAN_TIMESTAMP_PRESC_9

Timestamp counter time unit in equal to CAN bit time multiplied by 9

FDCAN_TIMESTAMP_PRESC_10

Timestamp counter time unit in equal to CAN bit time multiplied by 10

FDCAN_TIMESTAMP_PRESC_11

Timestamp counter time unit in equal to CAN bit time multiplied by 11

FDCAN_TIMESTAMP_PRESC_12

Timestamp counter time unit in equal to CAN bit time multiplied by 12

FDCAN_TIMESTAMP_PRESC_13

Timestamp counter time unit in equal to CAN bit time multiplied by 13

FDCAN_TIMESTAMP_PRESC_14

Timestamp counter time unit in equal to CAN bit time multiplied by 14

FDCAN_TIMESTAMP_PRESC_15

Timestamp counter time unit in equal to CAN bit time multiplied by 15

FDCAN_TIMESTAMP_PRESC_16

Timestamp counter time unit in equal to CAN bit time multiplied by 16

FDCAN TT Disturbing Error Interrupts

FDCAN_TT_IT_GLOBAL_TIME_ERROR

Global Time Error

FDCAN_TT_IT_TX_COUNT_UNDERFLOW

Tx Count Underflow

FDCAN_TT_IT_TX_COUNT_OVERFLOW

Tx Count Overflow

FDCAN_TT_IT_SCHEDULING_ERROR_1

Scheduling Error 1

FDCAN_TT_IT_SCHEDULING_ERROR_2

Scheduling Error 2

FDCAN_TT_IT_ERROR_LEVEL_CHANGE

Error Level Changed

FDCAN TT Fatal Error Interrupts

FDCAN_TT_IT_INIT_WATCH_TRIGGER

Initialization Watch Trigger

FDCAN_TT_IT_WATCH_TRIGGER

Watch Trigger

FDCAN_TT_IT_APPLICATION_WATCHDOG

Application Watchdog

FDCAN_TT_IT_CONFIG_ERROR

Configuration Error

FDCAN TT Flags

FDCAN_TT_FLAG_BASIC_CYCLE_START

Start of Basic Cycle

FDCAN_TT_FLAG_MATRIX_CYCLE_START

Start of Matrix Cycle

FDCAN_TT_FLAG_SYNC_MODE_CHANGE

Change of Synchronization Mode

FDCAN_TT_FLAG_START_OF_GAP

Start of Gap

FDCAN_TT_FLAG_REG_TIME_MARK

Register Time Mark Interrupt

FDCAN_TT_FLAG_TRIG_TIME_MARK

Trigger Time Mark Event Internal

FDCAN_TT_FLAG_STOP_WATCH

Stop Watch Event

FDCAN_TT_FLAG_GLOBAL_TIME_WRAP

Global Time Wrap

FDCAN_TT_FLAG_GLOBAL_TIME_DISC

Global Time Discontinuity

FDCAN_TT_FLAG_GLOBAL_TIME_ERROR

Global Time Error

FDCAN_TT_FLAG_TX_COUNT_UNDERFLOW

Tx Count Underflow

FDCAN_TT_FLAG_TX_COUNT_OVERFLOW

Tx Count Overflow

FDCAN_TT_FLAG_SCHEDULING_ERROR_1

Scheduling Error 1

FDCAN_TT_FLAG_SCHEDULING_ERROR_2

Scheduling Error 2

FDCAN_TT_FLAG_ERROR_LEVEL_CHANGE

Error Level Changed

FDCAN_TT_FLAG_INIT_WATCH_TRIGGER

Initialization Watch Trigger

FDCAN_TT_FLAG_WATCH_TRIGGER

Watch Trigger

FDCAN_TT_FLAG_APPLICATION_WATCHDOG

Application Watchdog

FDCAN_TT_FLAG_CONFIG_ERROR

Configuration Error

FDCAN TT Global Time Interrupts

FDCAN_TT_IT_GLOBAL_TIME_WRAP

Global Time Wrap

FDCAN_TT_IT_GLOBAL_TIME_DISC

Global Time Discontinuity

FDCAN TT Schedule Synchronization Interrupts

FDCAN_TT_IT_BASIC_CYCLE_START

Start of Basic Cycle

FDCAN_TT_IT_MATRIX_CYCLE_START

Start of Matrix Cycle

FDCAN_TT_IT_SYNC_MODE_CHANGE

Change of Synchronization Mode

FDCAN_TT_IT_START_OF_GAP

Start of Gap

FDCAN TT Stop Watch Interrupt

FDCAN_TT_IT_STOP_WATCH

Stop Watch Event

FDCAN TT Time Mark Interrupts

FDCAN_TT_IT_REG_TIME_MARK

Register Time Mark Interrupt

FDCAN_TT_IT_TRIG_TIME_MARK

Trigger Time Mark Event Internal

FDCAN TT Automatic Clock Calibration

FDCAN_TT_AUTO_CLK_CALIB_DISABLE

Automatic clock calibration in Level 0,2 disabled

FDCAN_TT_AUTO_CLK_CALIB_ENABLE

Automatic clock calibration in Level 0,2 enabled

FDCAN TT Basic Cycle Number

FDCAN_TT_CYCLES_PER_MATRIX_1

1 Basic Cycle per Matrix

FDCAN_TT_CYCLES_PER_MATRIX_2

2 Basic Cycles per Matrix

FDCAN_TT_CYCLES_PER_MATRIX_4

4 Basic Cycles per Matrix

FDCAN_TT_CYCLES_PER_MATRIX_8

8 Basic Cycles per Matrix

FDCAN_TT_CYCLES_PER_MATRIX_16

16 Basic Cycles per Matrix

FDCAN_TT_CYCLES_PER_MATRIX_32

32 Basic Cycles per Matrix

FDCAN_TT_CYCLES_PER_MATRIX_64

64 Basic Cycles per Matrix

FDCAN TT Cycle Start Sync

FDCAN_TT_NO_SYNC_PULSE

No sync pulse

FDCAN_TT_SYNC_BASIC_CYCLE_START

Sync pulse at start of basic cycle

FDCAN_TT_SYNC_MATRIX_START

Sync pulse at start of matrix

FDCAN TT Error Level

FDCAN_TT_NO_ERROR

Severity 0 - No Error

FDCAN_TT_WARNING

Severity 1 - Warning

FDCAN_TT_ERROR

Severity 2 - Error

FDCAN_TT_SEVERE_ERROR

Severity 3 - Severe Error

FDCAN TT Event Trigger Polarity

FDCAN_TT_EVT_TRIG_POL_RISING

Rising edge trigger

FDCAN_TT_EVT_TRIG_POL_FALLING

Falling edge trigger

FDCAN TT Event Trigger Selection

FDCAN_TT_EVENT_TRIGGER_0

TIM2 selected as event trigger

FDCAN_TT_EVENT_TRIGGER_1

TIM3 selected as event trigger

FDCAN_TT_EVENT_TRIGGER_2

ETH selected as event trigger

FDCAN_TT_EVENT_TRIGGER_3

HRTIM selected as event trigger

FDCAN TT External Clock Synchronization

FDCAN_TT_EXT_CLK_SYNC_DISABLE

External clock synchronization in Level 0,2 disabled

FDCAN_TT_EXT_CLK_SYNC_ENABLE

External clock synchronization in Level 0,2 enabled

FDCAN TT Global Time Filtering

FDCAN_TT_GLOB_TIME_FILT_DISABLE

Global time filtering in Level 0,2 disabled

FDCAN_TT_GLOB_TIME_FILT_ENABLE

Global time filtering in Level 0,2 enabled

FDCAN TT Master State

FDCAN_TT_MASTER_OFF

Master_Off, no master properties relevant

FDCAN_TT_TIME_SLAVE

Operating as Time Slave

FDCAN_TT_BACKUP_TIME_MASTER

Operating as Backup Time Master

FDCAN_TT_CURRENT_TIME_MASTER

Operating as current Time Master

FDCAN TT Operation

FDCAN_STRICTLY_TT_OPERATION

Strictly time-triggered operation

FDCAN_EXT_EVT_SYNC_TT_OPERATION

External event-synchronized time-triggered operation

FDCAN TT reference message payload

FDCAN_TT_REF_MESSAGE_NO_PAYLOAD

Reference message has no additional payload

FDCAN_TT_REF_MESSAGE_ADD_PAYLOAD

Additional payload is taken from Tx Buffer 0

FDCAN TT repeat factor

FDCAN_TT_REPEAT_EVERY_CYCLE

Trigger valid for all cycles

FDCAN_TT_REPEAT_EVERY_2ND_CYCLE

Trigger valid every 2dn cycle

FDCAN_TT_REPEAT_EVERY_4TH_CYCLE

Trigger valid every 4th cycle

FDCAN_TT_REPEAT_EVERY_8TH_CYCLE

Trigger valid every 8th cycle

FDCAN_TT_REPEAT_EVERY_16TH_CYCLE

Trigger valid every 16th cycle

FDCAN_TT_REPEAT_EVERY_32ND_CYCLE

Trigger valid every 32nd cycle

FDCAN_TT_REPEAT_EVERY_64TH_CYCLE

Trigger valid every 64th cycle

FDCAN TT Stop Watch Polarity

FDCAN_TT_STOP_WATCH_RISING

Selected stop watch source is captured at rising edge of fdcan1_swt

FDCAN_TT_STOP_WATCH_FALLING

Selected stop watch source is captured at falling edge of fdcan1_swt

FDCAN TT Stop Watch Source

FDCAN_TT_STOP_WATCH_DISABLED

Stop Watch disabled

FDCAN_TT_STOP_WATCH_CYCLE_TIME

Actual value of cycle time is copied to Capture Time register (TTCPT.SWV)

FDCAN_TT_STOP_WATCH_LOCAL_TIME

Actual value of local time is copied to Capture Time register (TTCPT.SWV)

FDCAN_TT_STOP_WATCH_GLOBAL_TIME

Actual value of global time is copied to Capture Time register (TTCPT.SWV)

FDCAN TT Stop Watch Trigger Selection

FDCAN_TT_STOP_WATCH_TRIGGER_0

TIM2 selected as stop watch trigger

FDCAN_TT_STOP_WATCH_TRIGGER_1

TIM3 selected as stop watch trigger

FDCAN_TT_STOP_WATCH_TRIGGER_2

ETH selected as stop watch trigger

FDCAN_TT_STOP_WATCH_TRIGGER_3

HRTIM selected as stop watch trigger

FDCAN TT Synchronization State

FDCAN_TT_OUT_OF_SYNC

Out of Synchronization

FDCAN_TT_SYNCHRONIZING

Synchronizing to communication

FDCAN_TT_IN_GAP

Schedule suspended by Gap

FDCAN_TT_IN_SCHEDULE

Synchronized to schedule

FDCAN TT time mark event external

FDCAN_TT_TM_NO_EXTERNAL_EVENT

No action

FDCAN_TT_TM_GEN_EXTERNAL_EVENT

External event (pulse) is generated when trigger becomes active

FDCAN TT time mark event internal

FDCAN_TT_TM_NO_INTERNAL_EVENT

No action

FDCAN_TT_TM_GEN_INTERNAL_EVENT

Internal event is generated when trigger becomes active

FDCAN TT Time Mark Source

FDCAN_TT_REG_TIMEMARK_DIABLED

No Register Time Mark Interrupt generated

FDCAN_TT_REG_TIMEMARK_CYC_TIME

Register Time Mark Interrupt if Time Mark = cycle time

FDCAN_TT_REG_TIMEMARK_LOC_TIME

Register Time Mark Interrupt if Time Mark = local time

FDCAN_TT_REG_TIMEMARK_GLO_TIME

Register Time Mark Interrupt if Time Mark = global time

FDCAN TT Time Master

FDCAN_TT_SLAVE

Time slave

FDCAN_TT_POTENTIAL_MASTER

Potential time master

FDCAN TT trigger type

FDCAN_TT_TX_REF_TRIGGER

Transmit reference message in strictly time-triggered operation

FDCAN_TT_TX_REF_TRIGGER_GAP

Transmit reference message in external event-synchronized time-triggered operation

FDCAN_TT_TX_TRIGGER_SINGLE

Start a single transmission in an exclusive time window

FDCAN_TT_TX_TRIGGER_CONTINUOUS

Start a continuous transmission in an exclusive time window

FDCAN_TT_TX_TRIGGER_ARBITRATION

Start a transmission in an arbitration time window

FDCAN_TT_TX_TRIGGER_MERGED

Start a merged arbitration window

FDCAN_TT_WATCH_TRIGGER

Check for missing reference messages in strictly time-triggered operation

FDCAN_TT_WATCH_TRIGGER_GAP

Check for missing reference messages in external event-synchronized time-triggered operation

FDCAN_TT_RX_TRIGGER

Check for the reception of periodic messages in exclusive time windows

FDCAN_TT_TIME_BASE_TRIGGER

Generate internal/external events depending on TmEventInt/TmEventExt configuration

FDCAN_TT_END_OF_LIST

Illegal trigger, to be assigned to the unused triggers after a FDCAN_TT_WATCH_TRIGGER or FDCAN_TT_WATCH_TRIGGER_GAP

FDCAN Tx FIFO/Queue Mode

FDCAN_TX_FIFO_OPERATION

FIFO mode

FDCAN_TX_QUEUE_OPERATION

Queue mode

FDCAN Tx Event FIFO Interrupts

FDCAN_IT_TX_EVT_FIFO_ELT_LOST

Tx Event FIFO element lost

FDCAN_IT_TX_EVT_FIFO_FULL

Tx Event FIFO full

FDCAN_IT_TX_EVT_FIFO_WATERMARK

Tx Event FIFO fill level reached watermark

FDCAN_IT_TX_EVT_FIFO_NEW_DATA

Tx Handler wrote Tx Event FIFO element

FDCAN Tx Interrupts

FDCAN_IT_TX_COMPLETE

Transmission Completed

FDCAN_IT_TX_ABORT_COMPLETE

Transmission Cancellation Finished

FDCAN_IT_TX_FIFO_EMPTY

Tx FIFO Empty

FDCAN Tx Location

FDCAN_TX_BUFFER0

Add message to Tx Buffer 0

FDCAN_TX_BUFFER1

Add message to Tx Buffer 1

FDCAN_TX_BUFFER2

Add message to Tx Buffer 2

FDCAN_TX_BUFFER3

Add message to Tx Buffer 3

FDCAN_TX_BUFFER4

Add message to Tx Buffer 4

FDCAN_TX_BUFFER5

Add message to Tx Buffer 5

FDCAN_TX_BUFFER6

Add message to Tx Buffer 6

FDCAN_TX_BUFFER7

Add message to Tx Buffer 7

FDCAN_TX_BUFFER8

Add message to Tx Buffer 8

FDCAN_TX_BUFFER9

Add message to Tx Buffer 9

FDCAN_TX_BUFFER10

Add message to Tx Buffer 10

FDCAN_TX_BUFFER11

Add message to Tx Buffer 11

FDCAN_TX_BUFFER12

Add message to Tx Buffer 12

FDCAN_TX_BUFFER13

Add message to Tx Buffer 13

FDCAN_TX_BUFFER14

Add message to Tx Buffer 14

FDCAN_TX_BUFFER15

Add message to Tx Buffer 15

FDCAN_TX_BUFFER16

Add message to Tx Buffer 16

FDCAN_TX_BUFFER17

Add message to Tx Buffer 17

FDCAN_TX_BUFFER18

Add message to Tx Buffer 18

FDCAN_TX_BUFFER19

Add message to Tx Buffer 19

FDCAN_TX_BUFFER20

Add message to Tx Buffer 20

FDCAN_TX_BUFFER21

Add message to Tx Buffer 21

FDCAN_TX_BUFFER22

Add message to Tx Buffer 22

FDCAN_TX_BUFFER23

Add message to Tx Buffer 23

FDCAN_TX_BUFFER24

Add message to Tx Buffer 24

FDCAN_TX_BUFFER25

Add message to Tx Buffer 25

FDCAN_TX_BUFFER26

Add message to Tx Buffer 26

FDCAN_TX_BUFFER27

Add message to Tx Buffer 27

FDCAN_TX_BUFFER28

Add message to Tx Buffer 28

FDCAN_TX_BUFFER29

Add message to Tx Buffer 29

FDCAN_TX_BUFFER30

Add message to Tx Buffer 30

FDCAN_TX_BUFFER31

Add message to Tx Buffer 31

31 HAL FLASH Generic Driver

31.1 FLASH Firmware driver registers structures

31.1.1 FLASH_ProcessTypeDef

FLASH_ProcessTypeDef is defined in the stm32h7xx_hal_flash.h

Data Fields

- ***__IO FLASH_ProcedureTypeDef ProcedureOnGoing***
- ***__IO uint32_t NbSectorsToErase***
- ***__IO uint32_t VoltageForErase***
- ***__IO uint32_t Sector***
- ***__IO uint32_t Address***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***__IO FLASH_ProcedureTypeDef FLASH_ProcessTypeDef::ProcedureOnGoing***
  Internal variable to indicate which procedure is ongoing or not in IT context
- ***__IO uint32_t FLASH_ProcessTypeDef::NbSectorsToErase***
  Internal variable to save the remaining sectors to erase in IT context
- ***__IO uint32_t FLASH_ProcessTypeDef::VoltageForErase***
  Internal variable to provide voltage range selected by user in IT context
- ***__IO uint32_t FLASH_ProcessTypeDef::Sector***
  Internal variable to define the current sector which is erasing
- ***__IO uint32_t FLASH_ProcessTypeDef::Address***
  Internal variable to save address selected for program
- ***HAL_LockTypeDef FLASH_ProcessTypeDef::Lock***
  FLASH locking object
- ***__IO uint32_t FLASH_ProcessTypeDef::ErrorCode***
  FLASH error code

31.2 FLASH Firmware driver API description

The following section lists the various functions of the FLASH library.

31.2.1 FLASH peripheral features

The Flash memory interface manages CPU AXI I-Code and D-Code accesses to the Flash memory. It implements the erase and program Flash memory operations and the read and write protection mechanisms.

The FLASH main features are:

- Flash memory read operations
- Flash memory program/erase operations
- Read / write protections
- Option bytes programming
- Error code correction (ECC) : Data in flash are 266-bits word (10 bits added per flash word)

31.2.2 How to use this driver

This driver provides functions and macros to configure and program the FLASH memory of all STM32H7xx devices.

1. FLASH Memory IO Programming functions:
   - Lock and Unlock the FLASH interface using HAL_FLASH_Unlock() and HAL_FLASH_Lock() functions
   - Program functions: 256-bit word only
   - There Two modes of programming :
     - Polling mode using HAL_FLASH_Program() function
     - Interrupt mode using HAL_FLASH_Program_IT() function
2. Interrupts and flags management functions :
   - Handle FLASH interrupts by calling HAL_FLASH_IRQHandler()
   - Callback functions are called when the flash operations are finished : HAL_FLASH_EndOfOperationCallback() when everything is ok, otherwise HAL_FLASH_OperationErrorCallback()
   - Get error flag status by calling HAL_FLASH_GetError()
3. Option bytes management functions :
   - Lock and Unlock the option bytes using HAL_FLASH_OB_Unlock() and HAL_FLASH_OB_Lock() functions
   - Launch the reload of the option bytes using HAL_FLASH_OB_Launch() function. In this case, a reset is generated

In addition to these functions, this driver includes a set of macros allowing to handle the following operations:

- Set the latency
- Enable/Disable the FLASH interrupts
- Monitor the FLASH flags status

Note: For any Flash memory program operation (erase or program), the CPU clock frequency (HCLK) must be at least 1MHz.

Note: The contents of the Flash memory are not guaranteed if a device reset occurs during a Flash memory operation.

Note: The application can simultaneously request a read and a write operation through each AXI interface. As the Flash memory is divided into two independent banks, the embedded Flash memory interface can drive different operations at the same time on each bank. For example a read, write or erase operation can be executed on bank 1 while another read, write or erase operation is executed on bank 2.

31.2.3 Programming operation functions

This subsection provides a set of functions allowing to manage the FLASH program operations.

This section contains the following APIs:

- ***HAL_FLASH_Program()***
- ***HAL_FLASH_Program_IT()***
- ***HAL_FLASH_IRQHandler()***
- ***HAL_FLASH_EndOfOperationCallback()***
- ***HAL_FLASH_OperationErrorCallback()***

31.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the FLASH memory operations.

This section contains the following APIs:

- ***HAL_FLASH_Unlock()***
- ***HAL_FLASH_Lock()***
- ***HAL_FLASH_OB_Unlock()***
- ***HAL_FLASH_OB_Lock()***
- ***HAL_FLASH_OB_Launch()***

31.2.5 Peripheral Errors functions

This subsection permits to get in run-time Errors of the FLASH peripheral.

This section contains the following APIs:

- *HAL_FLASH_GetError()*

31.2.6 Detailed description of functions

HAL_FLASH_Program

Function name

HAL_StatusTypeDef HAL_FLASH_Program (uint32_t TypeProgram, uint32_t FlashAddress, uint32_t DataAddress)

Function description

Program a flash word at a specified address.

Parameters

- **TypeProgram:** Indicate the way to program at a specified address. This parameter can be a value of FLASH Type Program
- **FlashAddress:** specifies the address to be programmed. This parameter shall be aligned to the Flash word:
  - 256 bits for STM32H74x/5X devices (8x 32bits words)
  - 128 bits for STM32H7Ax/BX devices (4x 32bits words)
  - 256 bits for STM32H72x/3X devices (8x 32bits words)
- **DataAddress:** specifies the address of data to be programmed. This parameter shall be 32-bit aligned

Return values

- **HAL_StatusTypeDef:** HAL Status

HAL_FLASH_Program_IT

Function name

HAL_StatusTypeDef HAL_FLASH_Program_IT (uint32_t TypeProgram, uint32_t FlashAddress, uint32_t DataAddress)

Function description

Program a flash word at a specified address with interrupt enabled.

Parameters

- **TypeProgram:** Indicate the way to program at a specified address. This parameter can be a value of FLASH Type Program
- **FlashAddress:** specifies the address to be programmed. This parameter shall be aligned to the Flash word:
  - 256 bits for STM32H74x/5X devices (8x 32bits words)
  - 128 bits for STM32H7Ax/BX devices (4x 32bits words)
  - 256 bits for STM32H72x/3X devices (8x 32bits words)
- **DataAddress:** specifies the address of data to be programmed. This parameter shall be 32-bit aligned

Return values

- **HAL:** Status

HAL_FLASH_IRQHandler

Function name

void HAL_FLASH_IRQHandler (void)

Function description

This function handles FLASH interrupt request.

Return values

- **None:**

HAL_FLASH_EndOfOperationCallback

Function name

void HAL_FLASH_EndOfOperationCallback (uint32_t ReturnValue)

Function description

FLASH end of operation interrupt callback.

Parameters

- **ReturnValue:** The value saved in this parameter depends on the ongoing procedure Mass Erase: Bank number which has been requested to erase Sectors Erase: Sector which has been erased (if 0xFFFFFFFF, it means that all the selected sectors have been erased) Program: Address which was selected for data program

Return values

- **None:**

HAL_FLASH_OperationErrorCallback

Function name

void HAL_FLASH_OperationErrorCallback (uint32_t ReturnValue)

Function description

FLASH operation error interrupt callback.

Parameters

- **ReturnValue:** The value saved in this parameter depends on the ongoing procedure Mass Erase: Bank number which has been requested to erase Sectors Erase: Sector number which returned an error Program: Address which was selected for data program

Return values

- **None:**

HAL_FLASH_Unlock

Function name

HAL_StatusTypeDef HAL_FLASH_Unlock (void)

Function description

Unlock the FLASH control registers access.

Return values

- **HAL:** Status

HAL_FLASH_Lock

Function name

HAL_StatusTypeDef HAL_FLASH_Lock (void)

Function description

Locks the FLASH control registers access.

Return values

- **HAL:** Status

HAL_FLASH_OB_Unlock

Function name

HAL_StatusTypeDef HAL_FLASH_OB_Unlock (void)

Function description

Unlock the FLASH Option Control Registers access.

Return values

- **HAL:** Status

HAL_FLASH_OB_Lock

Function name

HAL_StatusTypeDef HAL_FLASH_OB_Lock (void)

Function description

Lock the FLASH Option Control Registers access.

Return values

- **HAL:** Status

HAL_FLASH_OB_Launch

Function name

HAL_StatusTypeDef HAL_FLASH_OB_Launch (void)

Function description

Launch the option bytes loading.

Return values

- **HAL:** Status

HAL_FLASH_GetError

Function name

uint32_t HAL_FLASH_GetError (void)

Function description

Get the specific FLASH error flag.

Return values

- **HAL_FLASH_ERRORCode:** The returned value can be:
  - HAL_FLASH_ERROR_NONE : No error set
  - HAL_FLASH_ERROR_WRP_BANK1 : Write Protection Error on Bank 1
  - HAL_FLASH_ERROR_PGS_BANK1 : Program Sequence Error on Bank 1
  - HAL_FLASH_ERROR_STRB_BANK1 : Strobe Error on Bank 1
  - HAL_FLASH_ERROR_INC_BANK1 : Inconsistency Error on Bank 1
  - HAL_FLASH_ERROR_OPE_BANK1 : Operation Error on Bank 1
  - HAL_FLASH_ERROR_RDP_BANK1 : Read Protection Error on Bank 1
  - HAL_FLASH_ERROR_RDS_BANK1 : Read Secured Error on Bank 1
  - HAL_FLASH_ERROR_SNECC_BANK1: ECC Single Correction Error on Bank 1
  - HAL_FLASH_ERROR_DBECC_BANK1: ECC Double Detection Error on Bank 1
  - HAL_FLASH_ERROR_CRCRD_BANK1: CRC Read Error on Bank 1
  - HAL_FLASH_ERROR_WRP_BANK2 : Write Protection Error on Bank 2
  - HAL_FLASH_ERROR_PGS_BANK2 : Program Sequence Error on Bank 2
  - HAL_FLASH_ERROR_STRB_BANK2 : Strobe Error on Bank 2
  - HAL_FLASH_ERROR_INC_BANK2 : Inconsistency Error on Bank 2
  - HAL_FLASH_ERROR_OPE_BANK2 : Operation Error on Bank 2
  - HAL_FLASH_ERROR_RDP_BANK2 : Read Protection Error on Bank 2
  - HAL_FLASH_ERROR_RDS_BANK2 : Read Secured Error on Bank 2
  - HAL_FLASH_ERROR_SNECC_BANK2: SNECC Error on Bank 2
  - HAL_FLASH_ERROR_DBECC_BANK2: Double Detection ECC on Bank 2
  - HAL_FLASH_ERROR_CRCRD_BANK2: CRC Read Error on Bank 2

FLASH_WaitForLastOperation

Function name

HAL_StatusTypeDef FLASH_WaitForLastOperation (uint32_t Timeout, uint32_t Bank)

Function description

Wait for a FLASH operation to complete.

Parameters

- **Timeout:** maximum flash operation timeout
- **Bank:** flash FLASH_BANK_1 or FLASH_BANK_2

Return values

- **HAL_StatusTypeDef:** HAL Status

FLASH_OB_WaitForLastOperation

Function name

HAL_StatusTypeDef FLASH_OB_WaitForLastOperation (uint32_t Timeout)

Function description

Wait for a FLASH Option Bytes change operation to complete.

Parameters

- **Timeout:** maximum flash operation timeout

Return values

- **HAL_StatusTypeDef:** HAL Status

FLASH_CRC_WaitForLastOperation

Function name

HAL_StatusTypeDef FLASH_CRC_WaitForLastOperation (uint32_t Timeout, uint32_t Bank)

Function description

Wait for a FLASH CRC computation to complete.

Parameters

- **Timeout:** maximum flash operation timeout
- **Bank:** flash FLASH_BANK_1 or FLASH_BANK_2

Return values

- **HAL_StatusTypeDef:** HAL Status

31.3 FLASH Firmware driver defines

The following section lists the various define and macros of the module.

31.3.1 FLASH

FLASH

FLASH Error Code

HAL_FLASH_ERROR_NONE

No error

HAL_FLASH_ERROR_WRP

Write Protection Error

HAL_FLASH_ERROR_PGS

Program Sequence Error

HAL_FLASH_ERROR_STRB

Strobe Error

HAL_FLASH_ERROR_INC

Inconsistency Error

HAL_FLASH_ERROR_OPE

Operation Error

HAL_FLASH_ERROR_RDP

Read Protection Error

HAL_FLASH_ERROR_RDS

Read Secured Error

HAL_FLASH_ERROR_SNECC

ECC Single Correction Error

HAL_FLASH_ERROR_DBECC

ECC Double Detection Error

HAL_FLASH_ERROR_CRCRD

CRC Read Error

HAL_FLASH_ERROR_WRP_BANK1

Write Protection Error on Bank 1

HAL_FLASH_ERROR_PGS_BANK1

Program Sequence Error on Bank 1

HAL_FLASH_ERROR_STRB_BANK1

Strobe Error on Bank 1

HAL_FLASH_ERROR_INC_BANK1

Inconsistency Error on Bank 1

HAL_FLASH_ERROR_OPE_BANK1

Operation Error on Bank 1

HAL_FLASH_ERROR_RDP_BANK1

Read Protection Error on Bank 1

HAL_FLASH_ERROR_RDS_BANK1

Read Secured Error on Bank 1

HAL_FLASH_ERROR_SNECC_BANK1

ECC Single Correction Error on Bank 1

HAL_FLASH_ERROR_DBECC_BANK1

ECC Double Detection Error on Bank 1

HAL_FLASH_ERROR_CRCRD_BANK1

CRC Read Error on Bank1

HAL_FLASH_ERROR_WRP_BANK2

Write Protection Error on Bank 2

HAL_FLASH_ERROR_PGS_BANK2

Program Sequence Error on Bank 2

HAL_FLASH_ERROR_STRB_BANK2

Strobe Error on Bank 2

HAL_FLASH_ERROR_INC_BANK2

Inconsistency Error on Bank 2

HAL_FLASH_ERROR_OPE_BANK2

Operation Error on Bank 2

HAL_FLASH_ERROR_RDP_BANK2

Read Protection Error on Bank 2

HAL_FLASH_ERROR_RDS_BANK2

Read Secured Error on Bank 2

HAL_FLASH_ERROR_SNECC_BANK2

ECC Single Correction Error on Bank 2

HAL_FLASH_ERROR_DBECC_BANK2

ECC Double Detection Error on Bank 2

HAL_FLASH_ERROR_CRCRD_BANK2

CRC Read Error on Bank2

HAL_FLASH_ERROR_OB_CHANGE

Option Byte Change Error

FLASH Exported Macros

__HAL_FLASH_SET_LATENCY

Description:

- Set the FLASH Latency.

Parameters:

- __LATENCY__: FLASH Latency The value of this parameter depend on device used within the same series

Return value:

- none

__HAL_FLASH_GET_LATENCY

Description:

- Get the FLASH Latency.

Return value:

- FLASH: Latency The value of this parameter depend on device used within the same series

__HAL_FLASH_ENABLE_IT_BANK1

Description:

- Enable the specified FLASH interrupt.

Parameters:

- __INTERRUPT__: : FLASH interrupt In case of Bank 1 This parameter can be any combination of the following values:
  - FLASH_IT_EOP_BANK1 : End of FLASH Bank 1 Operation Interrupt source
  - FLASH_IT_WRPERR_BANK1 : Write Protection Error on Bank 1 Interrupt source
  - FLASH_IT_PGSERR_BANK1 : Program Sequence Error on Bank 1 Interrupt source
  - FLASH_IT_STRBERR_BANK1 : Strobe Error on Bank 1 Interrupt source
  - FLASH_IT_INCERR_BANK1 : Inconsistency Error on Bank 1 Interrupt source
  - FLASH_IT_OPERR_BANK1 : Operation Error on Bank 1 Interrupt source
  - FLASH_IT_RDPERR_BANK1 : Read protection Error on Bank 1 Interrupt source
  - FLASH_IT_RDSERR_BANK1 : Read secure Error on Bank 1 Interrupt source
  - FLASH_IT_SNECCERR_BANK1 : Single ECC Error Correction on Bank 1 Interrupt source
  - FLASH_IT_DBECCERR_BANK1 : Double Detection ECC Error on Bank 1 Interrupt source
  - FLASH_IT_CRCEND_BANK1 : CRC End on Bank 1 Interrupt source
  - FLASH_IT_CRCRDERR_BANK1 : CRC Read error on Bank 1 Interrupt source
  - FLASH_IT_ALL_BANK1 : All Bank 1 Interrupt sources
  - FLASH_IT_EOP_BANK2 : End of FLASH Bank 2 Operation Interrupt source
  - FLASH_IT_WRPERR_BANK2 : Write Protection Error on Bank 2 Interrupt source
  - FLASH_IT_PGSERR_BANK2 : Program Sequence Error on Bank 2 Interrupt source
  - FLASH_IT_STRBERR_BANK2 : Strobe Error on Bank 2 Interrupt source
  - FLASH_IT_INCERR_BANK2 : Inconsistency Error on Bank 2 Interrupt source
  - FLASH_IT_OPERR_BANK2 : Operation Error on Bank 2 Interrupt source
  - FLASH_IT_RDPERR_BANK2 : Read protection Error on Bank 2 Interrupt source
  - FLASH_IT_RDSERR_BANK2 : Read secure Error on Bank 2 Interrupt source
  - FLASH_IT_SNECCERR_BANK2 : Single ECC Error Correction on Bank 2 Interrupt source
  - FLASH_IT_DBECCERR_BANK2 : Double Detection ECC Error on Bank 2 Interrupt source
  - FLASH_IT_CRCEND_BANK2 : CRC End on Bank 2 Interrupt source
  - FLASH_IT_CRCRDERR_BANK2 : CRC Read error on Bank 2 Interrupt source
  - FLASH_IT_ALL_BANK2 : All Bank 2 Interrupt sources

Return value:

- none

__HAL_FLASH_ENABLE_IT_BANK2

__HAL_FLASH_ENABLE_IT

__HAL_FLASH_DISABLE_IT_BANK1

Description:

- Disable the specified FLASH interrupt.

Parameters:

- __INTERRUPT__: : FLASH interrupt In case of Bank 1 This parameter can be any combination of the following values:
  - FLASH_IT_EOP_BANK1 : End of FLASH Bank 1 Operation Interrupt source
  - FLASH_IT_WRPERR_BANK1 : Write Protection Error on Bank 1 Interrupt source
  - FLASH_IT_PGSERR_BANK1 : Program Sequence Error on Bank 1 Interrupt source
  - FLASH_IT_STRBERR_BANK1 : Strobe Error on Bank 1 Interrupt source
  - FLASH_IT_INCERR_BANK1 : Inconsistency Error on Bank 1 Interrupt source
  - FLASH_IT_OPERR_BANK1 : Operation Error on Bank 1 Interrupt source
  - FLASH_IT_RDPERR_BANK1 : Read protection Error on Bank 1 Interrupt source
  - FLASH_IT_RDSERR_BANK1 : Read secure Error on Bank 1 Interrupt source
  - FLASH_IT_SNECCERR_BANK1 : Single ECC Error Correction on Bank 1 Interrupt source
  - FLASH_IT_DBECCERR_BANK1 : Double Detection ECC Error on Bank 1 Interrupt source
  - FLASH_IT_CRCEND_BANK1 : CRC End on Bank 1 Interrupt source
  - FLASH_IT_CRCRDERR_BANK1 : CRC Read error on Bank 1 Interrupt source
  - FLASH_IT_ALL_BANK1 : All Bank 1 Interrupt sources
  - FLASH_IT_EOP_BANK2 : End of FLASH Bank 2 Operation Interrupt source
  - FLASH_IT_WRPERR_BANK2 : Write Protection Error on Bank 2 Interrupt source
  - FLASH_IT_PGSERR_BANK2 : Program Sequence Error on Bank 2 Interrupt source
  - FLASH_IT_STRBERR_BANK2 : Strobe Error on Bank 2 Interrupt source
  - FLASH_IT_INCERR_BANK2 : Inconsistency Error on Bank 2 Interrupt source
  - FLASH_IT_OPERR_BANK2 : Operation Error on Bank 2 Interrupt source
  - FLASH_IT_RDPERR_BANK2 : Read protection Error on Bank 2 Interrupt source
  - FLASH_IT_RDSERR_BANK2 : Read secure Error on Bank 2 Interrupt source
  - FLASH_IT_SNECCERR_BANK2 : Single ECC Error Correction on Bank 2 Interrupt source
  - FLASH_IT_DBECCERR_BANK2 : Double Detection ECC Error on Bank 2 Interrupt source
  - FLASH_IT_CRCEND_BANK2 : CRC End on Bank 2 Interrupt source
  - FLASH_IT_CRCRDERR_BANK2 : CRC Read error on Bank 2 Interrupt source
  - FLASH_IT_ALL_BANK2 : All Bank 2 Interrupt sources

Return value:

- none

__HAL_FLASH_DISABLE_IT_BANK2

__HAL_FLASH_DISABLE_IT

__HAL_FLASH_GET_FLAG_BANK1

Description:

- Checks whether the specified FLASH flag is set or not.

Parameters:

- __FLAG__: specifies the FLASH flag to check. In case of Bank 1 This parameter can be one of the following values :
  - FLASH_FLAG_BSY_BANK1 : FLASH Bank 1 Busy flag
  - FLASH_FLAG_WBNE_BANK1 : Write Buffer Not Empty on Bank 1 flag
  - FLASH_FLAG_QW_BANK1 : Wait Queue on Bank 1 flag
  - FLASH_FLAG_CRC_BUSY_BANK1 : CRC module is working on Bank 1 flag
  - FLASH_FLAG_EOP_BANK1 : End Of Program on Bank 1 flag
  - FLASH_FLAG_WRPERR_BANK1 : Write Protection Error on Bank 1 flag
  - FLASH_FLAG_PGSERR_BANK1 : Program Sequence Error on Bank 1 flag
  - FLASH_FLAG_STRBER_BANK1 : Program Alignment Error on Bank 1 flag
  - FLASH_FLAG_INCERR_BANK1 : Inconsistency Error on Bank 1 flag
  - FLASH_FLAG_OPERR_BANK1 : Operation Error on Bank 1 flag
  - FLASH_FLAG_RDPERR_BANK1 : Read Protection Error on Bank 1 flag
  - FLASH_FLAG_RDSERR_BANK1 : Read secure Error on Bank 1 flag
  - FLASH_FLAG_SNECCE_BANK1 : Single ECC Error Correction on Bank 1 flag
  - FLASH_FLAG_DBECCE_BANK1 : Double Detection ECC Error on Bank 1 flag
  - FLASH_FLAG_CRCEND_BANK1 : CRC End on Bank 1 flag
  - FLASH_FLAG_CRCRDERR_BANK1 : CRC Read error on Bank 1 flag
  - FLASH_FLAG_BSY_BANK2 : FLASH Bank 2 Busy flag
  - FLASH_FLAG_WBNE_BANK2 : Write Buffer Not Empty on Bank 2 flag
  - FLASH_FLAG_QW_BANK2 : Wait Queue on Bank 2 flag
  - FLASH_FLAG_CRC_BUSY_BANK2 : CRC module is working on Bank 2 flag
  - FLASH_FLAG_EOP_BANK2 : End Of Program on Bank 2 flag
  - FLASH_FLAG_WRPERR_BANK2 : Write Protection Error on Bank 2 flag
  - FLASH_FLAG_PGSERR_BANK2 : Program Sequence Error on Bank 2 flag
  - FLASH_FLAG_STRBER_BANK2 : Program Alignment Error on Bank 2 flag
  - FLASH_FLAG_INCERR_BANK2 : Inconsistency Error on Bank 2 flag
  - FLASH_FLAG_OPERR_BANK2 : Operation Error on Bank 2 flag
  - FLASH_FLAG_RDPERR_BANK2 : Read Protection Error on Bank 2 flag
  - FLASH_FLAG_RDSERR_BANK2 : Read secure Error on Bank 2 flag
  - FLASH_FLAG_SNECCE_BANK2 : Single ECC Error Correction on Bank 2 flag
  - FLASH_FLAG_DBECCE_BANK2 : Double Detection ECC Error on Bank 2 flag
  - FLASH_FLAG_CRCEND_BANK2 : CRC End on Bank 2 flag
  - FLASH_FLAG_CRCRDERR_BANK2 : CRC Read error on Bank 2 flag

Return value:

- The: new state of FLASH_FLAG (SET or RESET).

__HAL_FLASH_GET_FLAG_BANK2

__HAL_FLASH_GET_FLAG

__HAL_FLASH_CLEAR_FLAG_BANK1

Description:

- Clear the specified FLASH flag.

Parameters:

- __FLAG__: specifies the FLASH flags to clear. In case of Bank 1, this parameter can be any combination of the following values:
  - FLASH_FLAG_EOP_BANK1 : End Of Program on Bank 1 flag
  - FLASH_FLAG_WRPERR_BANK1 : Write Protection Error on Bank 1 flag
  - FLASH_FLAG_PGSERR_BANK1 : Program Sequence Error on Bank 1 flag
  - FLASH_FLAG_STRBER_BANK1 : Program Alignment Error on Bank 1 flag
  - FLASH_FLAG_INCERR_BANK1 : Inconsistency Error on Bank 1 flag
  - FLASH_FLAG_OPERR_BANK1 : Operation Error on Bank 1 flag
  - FLASH_FLAG_RDPERR_BANK1 : Read Protection Error on Bank 1 flag
  - FLASH_FLAG_RDSERR_BANK1 : Read secure Error on Bank 1 flag
  - FLASH_FLAG_SNECCE_BANK1 : Single ECC Error Correction on Bank 1 flag
  - FLASH_FLAG_DBECCE_BANK1 : Double Detection ECC Error on Bank 1 flag
  - FLASH_FLAG_CRCEND_BANK1 : CRC End on Bank 1 flag
  - FLASH_FLAG_CRCRDERR_BANK1 : CRC Read error on Bank 1 flag
  - FLASH_FLAG_ALL_ERRORS_BANK1 : All Bank 1 error flags
  - FLASH_FLAG_ALL_BANK1 : All Bank 1 flags
  - FLASH_FLAG_EOP_BANK2 : End Of Program on Bank 2 flag
  - FLASH_FLAG_WRPERR_BANK2 : Write Protection Error on Bank 2 flag
  - FLASH_FLAG_PGSERR_BANK2 : Program Sequence Error on Bank 2 flag
  - FLASH_FLAG_STRBER_BANK2 : Program Alignment Error on Bank 2 flag
  - FLASH_FLAG_INCERR_BANK2 : Inconsistency Error on Bank 2 flag
  - FLASH_FLAG_OPERR_BANK2 : Operation Error on Bank 2 flag
  - FLASH_FLAG_RDPERR_BANK2 : Read Protection Error on Bank 2 flag
  - FLASH_FLAG_RDSERR_BANK2 : Read secure Error on Bank 2 flag
  - FLASH_FLAG_SNECCE_BANK2 : Single ECC Error Correction on Bank 2 flag
  - FLASH_FLAG_DBECCE_BANK2 : Double Detection ECC Error on Bank 2 flag
  - FLASH_FLAG_CRCEND_BANK2 : CRC End on Bank 2 flag
  - FLASH_FLAG_CRCRDERR_BANK2 : CRC Read error on Bank 2 flag
  - FLASH_FLAG_ALL_ERRORS_BANK2 : All Bank 2 error flags
  - FLASH_FLAG_ALL_BANK2 : All Bank 2 flags

Return value:

- none

__HAL_FLASH_CLEAR_FLAG_BANK2

__HAL_FLASH_CLEAR_FLAG

FLASH Flag definition

FLASH_FLAG_BSY

FLASH Busy flag

FLASH_FLAG_WBNE

Write Buffer Not Empty flag

FLASH_FLAG_QW

Wait Queue on flag

FLASH_FLAG_CRC_BUSY

CRC Busy flag

FLASH_FLAG_EOP

End Of Program on flag

FLASH_FLAG_WRPERR

Write Protection Error on flag

FLASH_FLAG_PGSERR

Program Sequence Error on flag

FLASH_FLAG_STRBERR

Strobe Error flag

FLASH_FLAG_INCERR

Inconsistency Error on flag

FLASH_FLAG_OPERR

Operation Error on flag

FLASH_FLAG_RDPERR

Read Protection Error on flag

FLASH_FLAG_RDSERR

Read Secured Error on flag

FLASH_FLAG_SNECCERR

Single ECC Error Correction on flag

FLASH_FLAG_DBECCERR

Double Detection ECC Error on flag

FLASH_FLAG_CRCEND

CRC End of Calculation flag

FLASH_FLAG_CRCRDERR

CRC Read Error on bank flag

FLASH_FLAG_BSY_BANK1

FLASH Bank 1 Busy flag

FLASH_FLAG_WBNE_BANK1

Write Buffer Not Empty on Bank 1 flag

FLASH_FLAG_QW_BANK1

Wait Queue on Bank 1 flag

FLASH_FLAG_CRC_BUSY_BANK1

CRC Busy on Bank 1 flag

FLASH_FLAG_EOP_BANK1

End Of Program on Bank 1 flag

FLASH_FLAG_WRPERR_BANK1

Write Protection Error on Bank 1 flag

FLASH_FLAG_PGSERR_BANK1

Program Sequence Error on Bank 1 flag

FLASH_FLAG_STRBERR_BANK1

Strobe Error on Bank 1 flag

FLASH_FLAG_INCERR_BANK1

Inconsistency Error on Bank 1 flag

FLASH_FLAG_OPERR_BANK1

Operation Error on Bank 1 flag

FLASH_FLAG_RDPERR_BANK1

Read Protection Error on Bank 1 flag

FLASH_FLAG_RDSERR_BANK1

Read Secured Error on Bank 1 flag

FLASH_FLAG_SNECCERR_BANK1

Single ECC Error Correction on Bank 1 flag

FLASH_FLAG_DBECCERR_BANK1

Double Detection ECC Error on Bank 1 flag

FLASH_FLAG_CRCEND_BANK1

CRC End of Calculation on Bank 1 flag

FLASH_FLAG_CRCRDERR_BANK1

CRC Read error on Bank 1 flag

FLASH_FLAG_ALL_ERRORS_BANK1

All Bank 1 error flags

FLASH_FLAG_ALL_BANK1

All Bank 1 flags

FLASH_FLAG_BSY_BANK2

FLASH Bank 2 Busy flag

FLASH_FLAG_WBNE_BANK2

Write Buffer Not Empty on Bank 2 flag

FLASH_FLAG_QW_BANK2

Wait Queue on Bank 2 flag

FLASH_FLAG_CRC_BUSY_BANK2

CRC Busy on Bank 2 flag

FLASH_FLAG_EOP_BANK2

End Of Program on Bank 2 flag

FLASH_FLAG_WRPERR_BANK2

Write Protection Error on Bank 2 flag

FLASH_FLAG_PGSERR_BANK2

Program Sequence Error on Bank 2 flag

FLASH_FLAG_STRBERR_BANK2

Strobe Error on Bank 2 flag

FLASH_FLAG_INCERR_BANK2

Inconsistency Error on Bank 2 flag

FLASH_FLAG_OPERR_BANK2

Operation Error on Bank 2 flag

FLASH_FLAG_RDPERR_BANK2

Read Protection Error on Bank 2 flag

FLASH_FLAG_RDSERR_BANK2

Read Secured Error on Bank 2 flag

FLASH_FLAG_SNECCERR_BANK2

Single ECC Error Correction on Bank 2 flag

FLASH_FLAG_DBECCERR_BANK2

Double Detection ECC Error on Bank 2 flag

FLASH_FLAG_CRCEND_BANK2

CRC End of Calculation on Bank 2 flag

FLASH_FLAG_CRCRDERR_BANK2

CRC Read error on Bank 2 flag

FLASH_FLAG_ALL_ERRORS_BANK2

All Bank 2 error flags

FLASH_FLAG_ALL_BANK2

All Bank 2 flags

FLASH Interrupt definition

FLASH_IT_EOP_BANK1

End of FLASH Bank 1 Operation Interrupt source

FLASH_IT_WRPERR_BANK1

Write Protection Error on Bank 1 Interrupt source

FLASH_IT_PGSERR_BANK1

Program Sequence Error on Bank 1 Interrupt source

FLASH_IT_STRBERR_BANK1

Strobe Error on Bank 1 Interrupt source

FLASH_IT_INCERR_BANK1

Inconsistency Error on Bank 1 Interrupt source

FLASH_IT_OPERR_BANK1

Operation Error on Bank 1 Interrupt source

FLASH_IT_RDPERR_BANK1

Read protection Error on Bank 1 Interrupt source

FLASH_IT_RDSERR_BANK1

Read Secured Error on Bank 1 Interrupt source

FLASH_IT_SNECCERR_BANK1

Single ECC Error Correction on Bank 1 Interrupt source

FLASH_IT_DBECCERR_BANK1

Double Detection ECC Error on Bank 1 Interrupt source

FLASH_IT_CRCEND_BANK1

CRC End on Bank 1 Interrupt source

FLASH_IT_CRCRDERR_BANK1

CRC Read error on Bank 1 Interrupt source

FLASH_IT_ALL_BANK1

All Bank 1 Interrupt sources

FLASH_IT_EOP_BANK2

End of FLASH Bank 2 Operation Interrupt source

FLASH_IT_WRPERR_BANK2

Write Protection Error on Bank 2 Interrupt source

FLASH_IT_PGSERR_BANK2

Program Sequence Error on Bank 2 Interrupt source

FLASH_IT_STRBERR_BANK2

Strobe Error on Bank 2 Interrupt source

FLASH_IT_INCERR_BANK2

Inconsistency Error on Bank 2 Interrupt source

FLASH_IT_OPERR_BANK2

Operation Error on Bank 2 Interrupt source

FLASH_IT_RDPERR_BANK2

Read protection Error on Bank 2 Interrupt source

FLASH_IT_RDSERR_BANK2

Read Secured Error on Bank 2 Interrupt source

FLASH_IT_SNECCERR_BANK2

Single ECC Error Correction on Bank 2 Interrupt source

FLASH_IT_DBECCERR_BANK2

Double Detection ECC Error on Bank 2 Interrupt source

FLASH_IT_CRCEND_BANK2

CRC End on Bank 2 Interrupt source

FLASH_IT_CRCRDERR_BANK2

CRC Read Error on Bank 2 Interrupt source

FLASH_IT_ALL_BANK2

All Bank 2 Interrupt sources

FLASH Keys

FLASH_KEY1

FLASH_KEY2

FLASH_OPT_KEY1

FLASH_OPT_KEY2

FLASH Latency

FLASH_LATENCY_0

FLASH Zero Latency cycle

FLASH_LATENCY_1

FLASH One Latency cycle

FLASH_LATENCY_2

FLASH Two Latency cycles

FLASH_LATENCY_3

FLASH Three Latency cycles

FLASH_LATENCY_4

FLASH Four Latency cycles

FLASH_LATENCY_5

FLASH Five Latency cycles

FLASH_LATENCY_6

FLASH Six Latency cycles

FLASH_LATENCY_7

FLASH Seven Latency cycles

FLASH_LATENCY_8

FLASH Eight Latency cycle

FLASH_LATENCY_9

FLASH Nine Latency cycle

FLASH_LATENCY_10

FLASH Ten Latency cycles

FLASH_LATENCY_11

FLASH Eleven Latency cycles

FLASH_LATENCY_12

FLASH Twelve Latency cycles

FLASH_LATENCY_13

FLASH Thirteen Latency cycles

FLASH_LATENCY_14

FLASH Fourteen Latency cycles

FLASH_LATENCY_15

FLASH Fifteen Latency cycles

FLASH Program Parallelism

FLASH_PSIZE_BYTE

Flash program/erase by 8 bits

FLASH_PSIZE_HALF_WORD

Flash program/erase by 16 bits

FLASH_PSIZE_WORD

Flash program/erase by 32 bits

FLASH_PSIZE_DOUBLE_WORD

Flash program/erase by 64 bits

FLASH Sectors

FLASH_SECTOR_0

Sector Number 0

FLASH_SECTOR_1

Sector Number 1

FLASH_SECTOR_2

Sector Number 2

FLASH_SECTOR_3

Sector Number 3

FLASH_SECTOR_4

Sector Number 4

FLASH_SECTOR_5

Sector Number 5

FLASH_SECTOR_6

Sector Number 6

FLASH_SECTOR_7

Sector Number 7

FLASH Type Program

FLASH_TYPEPROGRAM_FLASHWORD

Program a flash word at a specified address

32 HAL FLASH Extension Driver

32.1 FLASHEx Firmware driver registers structures

32.1.1 FLASH_EraseInitTypeDef

FLASH_EraseInitTypeDef is defined in the stm32h7xx_hal_flash_ex.h

Data Fields

- ***uint32_t TypeErase***
- ***uint32_t Banks***
- ***uint32_t Sector***
- ***uint32_t NbSectors***
- ***uint32_t VoltageRange***

Field Documentation

- ***uint32_t FLASH_EraseInitTypeDef::TypeErase***
  Mass erase or sector Erase. This parameter can be a value of ***FLASHEx_Type_Erase***
- ***uint32_t FLASH_EraseInitTypeDef::Banks***
  Select banks to erase when Mass erase is enabled. This parameter must be a value of ***FLASHEx_Banks***
- ***uint32_t FLASH_EraseInitTypeDef::Sector***
  Initial FLASH sector to erase when Mass erase is disabled This parameter must be a value of ***FLASH_Sectors***
- ***uint32_t FLASH_EraseInitTypeDef::NbSectors***
  Number of sectors to be erased. This parameter must be a value between 1 and (max number of sectors - value of Initial sector)
- ***uint32_t FLASH_EraseInitTypeDef::VoltageRange***
  The device voltage range which defines the erase parallelism This parameter must be a value of ***FLASHEx_Voltage_Range***

32.1.2 FLASH_OBProgramInitTypeDef

FLASH_OBProgramInitTypeDef is defined in the stm32h7xx_hal_flash_ex.h

Data Fields

- ***uint32_t OptionType***
- ***uint32_t WRPState***
- ***uint32_t WRPSector***
- ***uint32_t RDPLevel***
- ***uint32_t BORLevel***
- ***uint32_t USERType***
- ***uint32_t USERConfig***
- ***uint32_t Banks***
- ***uint32_t PCROPConfig***
- ***uint32_t PCROPStartAddr***
- ***uint32_t PCROPEndAddr***
- ***uint32_t BootConfig***
- ***uint32_t BootAddr0***
- ***uint32_t BootAddr1***
- ***uint32_t CM4BootConfig***
- ***uint32_t CM4BootAddr0***
- ***uint32_t CM4BootAddr1***

- ***uint32_t SecureAreaConfig***
- ***uint32_t SecureAreaStartAddr***
- ***uint32_t SecureAreaEndAddr***

Field Documentation

- ***uint32_t FLASH_OBProgramInitTypeDef::OptionType***
  Option byte to be configured. This parameter can be a value of ***FLASHEx_Option_Type***
- ***uint32_t FLASH_OBProgramInitTypeDef::WRPState***
  Write protection activation or deactivation. This parameter can be a value of ***FLASHEx_WRP_State***
- ***uint32_t FLASH_OBProgramInitTypeDef::WRPSector***
  Specifies the sector(s) to be write protected. The value of this parameter depend on device used within the same series
- ***uint32_t FLASH_OBProgramInitTypeDef::RDPLevel***
  Set the read protection level. This parameter can be a value of ***FLASHEx_Option_Bytes_Read_Protection***
- ***uint32_t FLASH_OBProgramInitTypeDef::BORLevel***
  Set the BOR Level. This parameter can be a value of ***FLASHEx_BOR_Reset_Level***
- ***uint32_t FLASH_OBProgramInitTypeDef::USERType***
  User option byte(s) to be configured (used for OPTIONBYTE_USER). This parameter can be a combination of ***FLASHEx_OB_USER_Type***
- ***uint32_t FLASH_OBProgramInitTypeDef::USERConfig***
  Program the FLASH User Option Byte: IWDG_SW / RST_STOP / RST_STDBY / IWDG_FREEZE_STOP / IWDG_FREEZE_SANDBY / IO_HSLV / SWAP_BANK_OPT
- ***uint32_t FLASH_OBProgramInitTypeDef::Banks***
  Select banks for WRP , PCROP and secure area config . This parameter must be a value of ***FLASHEx_Banks***
- ***uint32_t FLASH_OBProgramInitTypeDef::PCROPConfig***
  specifies if the PCROP area shall be erased or not when RDP level decreased from Level 1 to Level 0 or during a mass erase. This parameter must be a value of ***FLASHEx_OB_PCROP_RDP*** enumeration
- ***uint32_t FLASH_OBProgramInitTypeDef::PCROPStartAddr***
  PCROP Start address (used for OPTIONBYTE_PCROP). This parameter must be a value between begin and end of a bank
- ***uint32_t FLASH_OBProgramInitTypeDef::PCROPEndAddr***
  PCROP End address (used for OPTIONBYTE_PCROP). This parameter must be a value between PCROP Start address and end of a bank
- ***uint32_t FLASH_OBProgramInitTypeDef::BootConfig***
  Specifies if the Boot Address to be configured BOOT_ADD0, BOOT_ADD1 or both. This parameter must be a value of ***FLASHEx_OB_BOOT_OPTION*** enumeration
- ***uint32_t FLASH_OBProgramInitTypeDef::BootAddr0***
  Boot Address 0. This parameter must be a value between begin and end of a bank
- ***uint32_t FLASH_OBProgramInitTypeDef::BootAddr1***
  Boot Address 1. This parameter must be a value between begin and end of a bank
- ***uint32_t FLASH_OBProgramInitTypeDef::CM4BootConfig***
  specifies if the CM4 boot Address to be configured BOOT_ADD0, BOOT_ADD1 or both. This parameter must be a value of ***FLASHEx_OB_BOOT_OPTION*** enumeration
- ***uint32_t FLASH_OBProgramInitTypeDef::CM4BootAddr0***
  CM4 Boot Address 0. This parameter must be a value between begin and end of a bank
- ***uint32_t FLASH_OBProgramInitTypeDef::CM4BootAddr1***
  CM4 Boot Address 1. This parameter must be a value between begin and end of a bank
- ***uint32_t FLASH_OBProgramInitTypeDef::SecureAreaConfig***
  specifies if the bank secured area shall be erased or not when RDP level decreased from Level 1 to Level 0 or during a mass erase. This parameter must be a value of ***FLASHEx_OB_SECURE_RDP*** enumeration

- ***uint32_t FLASH_OBProgramInitTypeDef::SecureAreaStartAddr***
  Bank Secure area Start address. This parameter must be a value between begin address and end address of bank1
- ***uint32_t FLASH_OBProgramInitTypeDef::SecureAreaEndAddr***
  Bank Secure area End address. This parameter must be a value between Secure Area Start address and end address of a bank1

32.1.3 FLASH_CRCInitTypeDef

FLASH_CRCInitTypeDef is defined in the stm32h7xx_hal_flash_ex.h

Data Fields

- ***uint32_t TypeCRC***
- ***uint32_t BurstSize***
- ***uint32_t Bank***
- ***uint32_t Sector***
- ***uint32_t NbSectors***
- ***uint32_t CRCStartAddr***
- ***uint32_t CRCEndAddr***

Field Documentation

- ***uint32_t FLASH_CRCInitTypeDef::TypeCRC***
  CRC Selection Type. This parameter can be a value of ***FLASHEx_CRC_Selection_Type***
- ***uint32_t FLASH_CRCInitTypeDef::BurstSize***
  CRC Burst Size. This parameter can be a value of ***FLASHEx_CRC_Burst_Size***
- ***uint32_t FLASH_CRCInitTypeDef::Bank***
  Select bank where CRC computation is enabled. This parameter must be FLASH_BANK_1 or FLASH_BANK_2
- ***uint32_t FLASH_CRCInitTypeDef::Sector***
  Initial FLASH sector from which starts the CRC computation This parameter must be a value of ***FLASH_Sectors***
- ***uint32_t FLASH_CRCInitTypeDef::NbSectors***
  Number of sectors to be computed. This parameter must be a value between 1 and (max number of sectors - value of Initial sector)
- ***uint32_t FLASH_CRCInitTypeDef::CRCStartAddr***
  CRC Start address. This parameter must be a value between begin address and end address of a bank
- ***uint32_t FLASH_CRCInitTypeDef::CRCEndAddr***
  CRC End address. This parameter must be a value between CRC Start address and end address of a bank

32.2 FLASHEx Firmware driver API description

The following section lists the various functions of the FLASHEx library.

32.2.1 Flash Extension features

Comparing to other previous devices, the FLASH interface for STM32H7xx devices contains the following additional features

- Capacity up to 2 Mbyte with dual bank architecture supporting read-while-write capability (RWW)
- Dual bank memory organization
- PCROP protection for all banks
- Global readout protection (RDP)
- Write protection
- Secure access only protection
- Bank / register swapping (when Dual-Bank)
- Cyclic Redundancy Check (CRC)

32.2.2 How to use this driver

This driver provides functions to configure and program the FLASH memory of all STM32H7xx devices. It includes

1. FLASH Memory Erase functions:
   - Lock and Unlock the FLASH interface using HAL_FLASH_Unlock() and HAL_FLASH_Lock() functions
   - Erase function: Sector erase, bank erase and dual-bank mass erase
   - There are two modes of erase :
     - Polling Mode using HAL_FLASHEx_Erase()
     - Interrupt Mode using HAL_FLASHEx_Erase_IT()
2. Option Bytes Programming functions: Use HAL_FLASHEx_OBProgram() to:
   - Set/Reset the write protection per bank
   - Set the Read protection Level
   - Set the BOR level
   - Program the user Option Bytes
   - PCROP protection configuration and control per bank
   - Secure area configuration and control per bank
   - Core Boot address configuration
   - TCM / AXI shared RAM configuration
   - CPU Frequency Boost configuration
3. FLASH Memory Lock and unlock per Bank: HAL_FLASHEx_Lock_Bank1(), HAL_FLASHEx_Unlock_Bank1(), HAL_FLASHEx_Lock_Bank2() and HAL_FLASHEx_Unlock_Bank2() functions
4. FLASH CRC computation function: Use HAL_FLASHEx_ComputeCRC() to:
   - Enable CRC feature
   - Program the desired burst size
   - Define the user Flash Area on which the CRC has be computed
   - Perform the CRC computation
   - Disable CRC feature

32.2.3 Extended programming operation functions

This subsection provides a set of functions allowing to manage the Extension FLASH programming operations Operations.

This section contains the following APIs:

- ***HAL_FLASHEx_Erase()***
- ***HAL_FLASHEx_Erase_IT()***
- ***HAL_FLASHEx_OBProgram()***
- ***HAL_FLASHEx_OBGetConfig()***
- ***HAL_FLASHEx_Unlock_Bank1()***
- ***HAL_FLASHEx_Lock_Bank1()***
- ***HAL_FLASHEx_Unlock_Bank2()***
- ***HAL_FLASHEx_Lock_Bank2()***
- ***HAL_FLASHEx_ComputeCRC()***

32.2.4 Detailed description of functions

HAL_FLASHEx_Erase

Function name

HAL_StatusTypeDef HAL_FLASHEx_Erase (FLASH_EraseInitTypeDef * pEraseInit, uint32_t * SectorError)

Function description

Perform a mass erase or erase the specified FLASH memory sectors.

Parameters

- **pEraseInit:** pointer to an FLASH_EraseInitTypeDef structure that contains the configuration information for the erasing.
- **SectorError:** pointer to variable that contains the configuration information on faulty sector in case of error (0xFFFFFFFF means that all the sectors have been correctly erased)

Return values

- **HAL:** Status

HAL_FLASHEx_Erase_IT

Function name

HAL_StatusTypeDef HAL_FLASHEx_Erase_IT (FLASH_EraseInitTypeDef * pEraseInit)

Function description

Perform a mass erase or erase the specified FLASH memory sectors with interrupt enabled.

Parameters

- **pEraseInit:** pointer to an FLASH_EraseInitTypeDef structure that contains the configuration information for the erasing.

Return values

- **HAL:** Status

HAL_FLASHEx_OBProgram

Function name

HAL_StatusTypeDef HAL_FLASHEx_OBProgram (FLASH_OBProgramInitTypeDef * pOBInit)

Function description

Program option bytes.

Parameters

- **pOBInit:** pointer to an FLASH_OBProgramInitTypeDef structure that contains the configuration information for the programming.

Return values

- **HAL:** Status

HAL_FLASHEx_OBGetConfig

Function name

void HAL_FLASHEx_OBGetConfig (FLASH_OBProgramInitTypeDef * pOBInit)

Function description

Get the Option byte configuration.

Parameters

- **pOBInit:** pointer to an FLASH_OBProgramInitTypeDef structure that contains the configuration information for the programming.

Return values

- **None:**

Notes

- The parameter Banks of the pOBInit structure must be set exclusively to FLASH_BANK_1 or FLASH_BANK_2, as this parameter is use to get the given Bank WRP, PCROP and secured area configuration.

HAL_FLASHEx_Unlock_Bank1

Function name

HAL_StatusTypeDef HAL_FLASHEx_Unlock_Bank1 (void)

Function description

Unlock the FLASH Bank1 control registers access.

Return values

- **HAL:** Status

HAL_FLASHEx_Lock_Bank1

Function name

HAL_StatusTypeDef HAL_FLASHEx_Lock_Bank1 (void)

Function description

Locks the FLASH Bank1 control registers access.

Return values

- **HAL:** Status

HAL_FLASHEx_Unlock_Bank2

Function name

HAL_StatusTypeDef HAL_FLASHEx_Unlock_Bank2 (void)

Function description

Unlock the FLASH Bank2 control registers access.

Return values

- **HAL:** Status

HAL_FLASHEx_Lock_Bank2

Function name

HAL_StatusTypeDef HAL_FLASHEx_Lock_Bank2 (void)

Function description

Locks the FLASH Bank2 control registers access.

Return values

- **HAL:** Status

HAL_FLASHEx_ComputeCRC

Function name

HAL_StatusTypeDef HAL_FLASHEx_ComputeCRC (FLASH_CRCInitTypeDef * pCRCInit, uint32_t * CRC_Result)

Function description

FLASH_Erase_Sector

Function name

void FLASH_Erase_Sector (uint32_t Sector, uint32_t Banks, uint32_t VoltageRange)

Function description

Erase the specified FLASH memory sector.

Parameters

- **Sector:** FLASH sector to erase This parameter can be a value of FLASH Sectors
- **Banks:** Banks to be erased This parameter can be one of the following values:
  - FLASH_BANK_1: Bank1 to be erased
  - FLASH_BANK_2: Bank2 to be erased
  - FLASH_BANK_BOTH: Bank1 and Bank2 to be erased
- **VoltageRange:** The device program/erase parallelism. This parameter can be one of the following values:
  - FLASH_VOLTAGE_RANGE_1 : Flash program/erase by 8 bits
  - FLASH_VOLTAGE_RANGE_2 : Flash program/erase by 16 bits
  - FLASH_VOLTAGE_RANGE_3 : Flash program/erase by 32 bits
  - FLASH_VOLTAGE_RANGE_4 : Flash program/erase by 64 bits

Return values

- **None:**

32.3 FLASHEx Firmware driver defines

The following section lists the various define and macros of the module.

32.3.1 FLASHEx

FLASHEx

FLASH Banks

FLASH_BANK_1

Bank 1

FLASH_BANK_2

Bank 2

FLASH_BANK_BOTH

Bank1 and Bank2

FLASH Boot Address

OB_BOOTADDR_ITCM_RAM

Boot from ITCM RAM (0x00000000)

OB_BOOTADDR_SYSTEM

Boot from System memory bootloader (0x00100000)

OB_BOOTADDR_ITCM_FLASH

Boot from Flash on ITCM interface (0x00200000)

OB_BOOTADDR_AXIM_FLASH

Boot from Flash on AXIM interface (0x08000000)

OB_BOOTADDR_DTCM_RAM

Boot from DTCM RAM (0x20000000)

OB_BOOTADDR_SRAM1

Boot from SRAM1 (0x20010000)

OB_BOOTADDR_SRAM2

Boot from SRAM2 (0x2004C000)

FLASH BOR Reset Level

OB_BOR_LEVEL0

Reset level threshold is set to 1.6V

OB_BOR_LEVEL1

Reset level threshold is set to 2.1V

OB_BOR_LEVEL2

Reset level threshold is set to 2.4V

OB_BOR_LEVEL3

Reset level threshold is set to 2.7V

FLASH CRC Burst Size

FLASH_CRC_BURST_SIZE_4

Every burst has a size of 4 Flash words (256-bit)

FLASH_CRC_BURST_SIZE_16

Every burst has a size of 16 Flash words (256-bit)

FLASH_CRC_BURST_SIZE_64

Every burst has a size of 64 Flash words (256-bit)

FLASH_CRC_BURST_SIZE_256

Every burst has a size of 256 Flash words (256-bit)

FLASH CRC Selection Type

FLASH_CRC_ADDR

CRC selection type by address

FLASH_CRC_SECTORS

CRC selection type by sectors

FLASH_CRC_BANK

CRC selection type by bank

FLASH Exported Macros

__HAL_FLASH_CALC_BOOT_BASE_ADR

Description:

- Calculate the FLASH Boot Base Address (BOOT_ADD0 or BOOT_ADD1)

Parameters:

- __ADDRESS__: FLASH Boot Address (in the range 0x0000 0000 to 0x2004 FFFF with a granularity of 16KB)

Return value:

- The: FLASH Boot Base Address

Notes:

- Returned value BOOT_ADDx[15:0] corresponds to boot address [29:14].

__HAL_FLASH_SET_PSIZE

Description:

- Set the FLASH Program/Erase parallelism.

Parameters:

- __PSIZE__: FLASH Program/Erase parallelism This parameter can be a value of
- __BANK__: Flash bank (FLASH_BANK_1 or FLASH_BANK_2)

Return value:

- none

__HAL_FLASH_GET_PSIZE

Description:

- Get the FLASH Program/Erase parallelism.

Parameters:

- __BANK__: Flash bank (FLASH_BANK_1 or FLASH_BANK_2)

Return value:

- FLASH: Program/Erase parallelism This return value can be a value of

__HAL_FLASH_SET_PROGRAM_DELAY

Description:

- Set the FLASH Programming Delay.

Parameters:

- __DELAY__: FLASH Programming Delay This parameter can be a value of

Return value:

- none

__HAL_FLASH_GET_PROGRAM_DELAY

Description:

- Get the FLASH Programming Delay.

Return value:

- FLASH: Programming Delay This return value can be a value of

FLASHEx Private macros to check input parameters

IS_FLASH_TYPEERASE

IS_VOLTAGERANGE

IS_WRPSTATE

IS_OPTIONBYTE

IS_OB_BOOT_ADDRESS

IS_OB_RDP_LEVEL

IS_OB_IWDG_SOURCE

IS_OB_STOP_SOURCE

IS_OB_STDBY_SOURCE

IS_OB_IWDG_STOP_FREEZE

IS_OB_IWDG_STDBY_FREEZE

IS_OB_BOR_LEVEL

IS_FLASH_LATENCY

IS_FLASH_SECTOR

IS_OB_WRP_SECTOR

IS_OB_PCROP_RDP

IS_OB_SECURE_RDP

IS_OB_USER_SWAP_BANK

IS_OB_USER_IOHSLV

IS_OB_IWDG1_SOURCE

IS_OB_IWDG2_SOURCE

IS_OB_STOP_D1_RESET

IS_OB_STDBY_D1_RESET

IS_OB_USER_IWDG_STOP

IS_OB_USER_IWDG_STDBY

IS_OB_USER_ST_RAM_SIZE

IS_OB_USER_SECURITY

IS_OB_USER_BCM4

IS_OB_USER_BCM7

IS_OB_STOP_D2_RESET

IS_OB_STDBY_D2_RESET

IS_OB_USER_TYPE

IS_OB_BOOT_ADD_OPTION

IS_FLASH_TYPECRC

FLASHEx OB BCM4

OB_BCM4_DISABLE

CM4 Boot disabled

OB_BCM4_ENABLE

CM4 Boot enabled

FLASHEx OB BCM7

OB_BCM7_DISABLE

CM7 Boot disabled

OB_BCM7_ENABLE

CM7 Boot enabled

FLASHEx OB BOOT OPTION

OB_BOOT_ADD0

Select Boot Address 0

OB_BOOT_ADD1

Select Boot Address 1

OB_BOOT_ADD_BOTH

Select Boot Address 0 and 1

FLASHEx OB IOHSLV

OB_IOHSLV_DISABLE

IOHSLV disabled

OB_IOHSLV_ENABLE

IOHSLV enabled

FLASHEx OB IWDG1 SW

OB_IWDG1_SW

Hardware independent watchdog 1

OB_IWDG1_HW

Software independent watchdog 1

FLASHEx OB IWDG2 SW

OB_IWDG2_SW

Hardware independent watchdog 2

OB_IWDG2_HW

Software independent watchdog 2

FLASHEx OB NRST STDBY D1

OB_STDBY_RST_D1

Reset generated when entering the D1 to standby mode

OB_STDBY_NO_RST_D1

No reset generated when entering the D1 to standby mode

FLASHEx OB NRST STDBY D2

OB_STDBY_RST_D2

Reset generated when entering the D2 to standby mode

OB_STDBY_NO_RST_D2

No reset generated when entering the D2 to standby mode

FLASHEx OB NRST STOP D1

OB_STOP_RST_D1

Reset generated when entering the D1 to stop mode

OB_STOP_NO_RST_D1

No reset generated when entering the D1 to stop mode

FLASHEx OB NRST STOP D2

OB_STOP_RST_D2

Reset generated when entering the D2 to stop mode

OB_STOP_NO_RST_D2

No reset generated when entering the D2 to stop mode

FLASHEx OB PCROP RDP

OB_PCROP_RDP_NOT_ERASE

PCROP area is not erased when the RDP level is decreased from Level 1 to Level 0 or during a mass erase

OB_PCROP_RDP_ERASE

PCROP area is erased when the RDP level is decreased from Level 1 to Level 0 (full mass erase)

FLASHEx OB SECURE RDP

OB_SECURE_RDP_NOT_ERASE

Secure area is not erased when the RDP level is decreased from Level 1 to Level 0 or during a mass erase

OB_SECURE_RDP_ERASE

Secure area is erased when the RDP level is decreased from Level 1 to Level 0 (full mass erase)

FLASHEx OB SECURITY

OB_SECURITY_DISABLE

security enabled

OB_SECURITY_ENABLE

security disabled

FLASHEx OB ST RAM SIZE

OB_ST_RAM_SIZE_2KB

2 Kbytes reserved to ST code

OB_ST_RAM_SIZE_4KB

4 Kbytes reserved to ST code

OB_ST_RAM_SIZE_8KB

8 Kbytes reserved to ST code

OB_ST_RAM_SIZE_16KB

16 Kbytes reserved to ST code

FLASHEx OB SWAP BANK

OB_SWAP_BANK_DISABLE

Bank swap disabled

OB_SWAP_BANK_ENABLE

Bank swap enabled

FLASHEx OB USER Type

OB_USER_IWDG1_SW

Independent watchdog selection

OB_USER_NRST_STOP_D1

Reset when entering Stop mode selection

OB_USER_NRST_STDBY_D1

Reset when entering standby mode selection

OB_USER_IWDG_STOP

Independent watchdog counter freeze in stop mode

OB_USER_IWDG_STDBY

Independent watchdog counter freeze in standby mode

OB_USER_ST_RAM_SIZE

dedicated DTCM Ram size selection

OB_USER_SECURITY

security selection

OB_USER_IOHSLV

IO HSLV selection

OB_USER_SWAP_BANK

Bank swap selection

OB_USER_IWDG2_SW

Window watchdog selection

OB_USER_BCM4

CM4 boot selection

OB_USER_BCM7

CM7 boot selection

OB_USER_NRST_STOP_D2

Reset when entering Stop mode selection

OB_USER_NRST_STDBY_D2

Reset when entering standby mode selection

OB_USER_ALL

FLASH Option Bytes IWatchdog

OB_IWDG_SW

Software IWDG selected

OB_IWDG_HW

Hardware IWDG selected

FLASH IWDG Counter Freeze in STANDBY

OB_IWDG_STDBY_FREEZE

Freeze IWDG counter in STANDBY mode

OB_IWDG_STDBY_ACTIVE

IWDG counter active in STANDBY mode

FLASH IWDG Counter Freeze in STOP

OB_IWDG_STOP_FREEZE

Freeze IWDG counter in STOP mode

OB_IWDG_STOP_ACTIVE

IWDG counter active in STOP mode

FLASH Option Bytes nRST_STDBY

OB_STDBY_NO_RST

No reset generated when entering in STANDBY

OB_STDBY_RST

Reset generated when entering in STANDBY

FLASH Option Bytes nRST_STOP

OB_STOP_NO_RST

No reset generated when entering in STOP

OB_STOP_RST

Reset generated when entering in STOP

FLASH Option Bytes Read Protection

OB_RDP_LEVEL_0

OB_RDP_LEVEL_1

OB_RDP_LEVEL_2

Warning: When enabling read protection level 2 it s no more possible to go back to level 1 or 0

FLASH Option Bytes Write Protection

OB_WRP_SECTOR_0

Write protection of Sector0

OB_WRP_SECTOR_1

Write protection of Sector1

OB_WRP_SECTOR_2

Write protection of Sector2

OB_WRP_SECTOR_3

Write protection of Sector3

OB_WRP_SECTOR_4

Write protection of Sector4

OB_WRP_SECTOR_5

Write protection of Sector5

OB_WRP_SECTOR_6

Write protection of Sector6

OB_WRP_SECTOR_7

Write protection of Sector7

OB_WRP_SECTOR_ALL

Write protection of all Sectors

FLASH Option Type

OPTIONBYTE_WRP

WRP option byte configuration

OPTIONBYTE_RDP

RDP option byte configuration

OPTIONBYTE_USER

USER option byte configuration

OPTIONBYTE_PCROP

PCROP option byte configuration

OPTIONBYTE_BOR

BOR option byte configuration

OPTIONBYTE_SECURE_AREA

secure area option byte configuration

OPTIONBYTE_CM7_BOOTADD

CM7 BOOT ADD option byte configuration

OPTIONBYTE_CM4_BOOTADD

CM4 BOOT ADD option byte configuration

OPTIONBYTE_BOOTADD

BOOT ADD option byte configuration

OPTIONBYTE_ALL

All option byte configuration

FLASH Programming Delay

FLASH_PROGRAMMING_DELAY_0

programming delay set for Flash running at 70 MHz or below

FLASH_PROGRAMMING_DELAY_1

programming delay set for Flash running between 70 MHz and 185 MHz

FLASH_PROGRAMMING_DELAY_2

programming delay set for Flash running between 185 MHz and 225 MHz

FLASH_PROGRAMMING_DELAY_3

programming delay set for Flash at startup

FLASH Type Erase

FLASH_TYPEERASE_SECTORS

Sectors erase only

FLASH_TYPEERASE_MASSERASE

Flash Mass erase activation

FLASH Voltage Range

FLASH_VOLTAGE_RANGE_1

Flash program/erase by 8 bits

FLASH_VOLTAGE_RANGE_2

Flash program/erase by 16 bits

FLASH_VOLTAGE_RANGE_3

Flash program/erase by 32 bits

FLASH_VOLTAGE_RANGE_4

Flash program/erase by 64 bits

FLASH WRP State

OB_WRPSTATE_DISABLE

Disable the write protection of the desired bank 1 sectors

OB_WRPSTATE_ENABLE

Enable the write protection of the desired bank 1 sectors

33 HAL FMAC Generic Driver

33.1 FMAC Firmware driver registers structures

33.1.1 __FMAC_HandleTypeDef

__FMAC_HandleTypeDef is defined in the stm32h7xx_hal_fmac.h

Data Fields

- ***FMAC_TypeDef * Instance***
- ***uint32_t FilterParam***
- ***uint8_t InputAccess***
- ***uint8_t OutputAccess***
- ***int16_t * pInput***
- ***uint16_t InputCurrentSize***
- ***uint16_t * pInputSize***
- ***int16_t * pOutput***
- ***uint16_t OutputCurrentSize***
- ***uint16_t * pOutputSize***
- ***DMA_HandleTypeDef * hdmaIn***
- ***DMA_HandleTypeDef * hdmaOut***
- ***DMA_HandleTypeDef * hdmaPreload***
- ***void(* ErrorCallback***
- ***void(* HalfGetDataCallback***
- ***void(* GetDataCallback***
- ***void(* HalfOutputDataReadyCallback***
- ***void(* OutputDataReadyCallback***
- ***void(* FilterConfigCallback***
- ***void(* FilterPreloadCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_FMAC_StateTypeDef State***
- ***__IO HAL_FMAC_StateTypeDef RdState***
- ***__IO HAL_FMAC_StateTypeDef WrState***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***FMAC_TypeDef* __FMAC_HandleTypeDef::Instance***
  Register base address
- ***uint32_t __FMAC_HandleTypeDef::FilterParam***
  Filter configuration (operation and parameters). Set to 0 if no valid configuration was applied.
- ***uint8_t __FMAC_HandleTypeDef::InputAccess***
  Access to the input buffer (internal memory area): DMA, IT, Polling, None. This parameter can be a value of ***FMAC_Buffer_Access***.
- ***uint8_t __FMAC_HandleTypeDef::OutputAccess***
  Access to the output buffer (internal memory area): DMA, IT, Polling, None. This parameter can be a value of ***FMAC_Buffer_Access***.
- ***int16_t* __FMAC_HandleTypeDef::pInput***
  Pointer to FMAC input data buffer

- ***uint16_t __FMAC_HandleTypeDef::InputCurrentSize***
  Number of the input elements already written into FMAC
- ***uint16_t* __FMAC_HandleTypeDef::pInputSize***
  Number of input elements to write (memory allocated to pInput). In case of early interruption of the filter operation, its value will be updated.
- ***int16_t* __FMAC_HandleTypeDef::pOutput***
  Pointer to FMAC output data buffer
- ***uint16_t __FMAC_HandleTypeDef::OutputCurrentSize***
  Number of the output elements already read from FMAC
- ***uint16_t* __FMAC_HandleTypeDef::pOutputSize***
  Number of output elements to read (memory allocated to pOutput). In case of early interruption of the filter operation, its value will be updated.
- ***DMA_HandleTypeDef* __FMAC_HandleTypeDef::hdmaIn***
  FMAC peripheral input data DMA handle parameters
- ***DMA_HandleTypeDef* __FMAC_HandleTypeDef::hdmaOut***
  FMAC peripheral output data DMA handle parameters
- ***DMA_HandleTypeDef* __FMAC_HandleTypeDef::hdmaPreload***
  FMAC peripheral preloaded data (X1, X2 and Y) DMA handle parameters
- ***void(* __FMAC_HandleTypeDef::ErrorCallback)(struct __FMAC_HandleTypeDef *hfmac)***
  FMAC error callback
- ***void(* __FMAC_HandleTypeDef::HalfGetDataCallback)(struct __FMAC_HandleTypeDef *hfmac)***
  FMAC get half data callback
- ***void(* __FMAC_HandleTypeDef::GetDataCallback)(struct __FMAC_HandleTypeDef *hfmac)***
  FMAC get data callback
- ***void(* __FMAC_HandleTypeDef::HalfOutputDataReadyCallback)(struct __FMAC_HandleTypeDef *hfmac)***
  FMAC half output data ready callback
- ***void(* __FMAC_HandleTypeDef::OutputDataReadyCallback)(struct __FMAC_HandleTypeDef *hfmac)***
  FMAC output data ready callback
- ***void(* __FMAC_HandleTypeDef::FilterConfigCallback)(struct __FMAC_HandleTypeDef *hfmac)***
  FMAC filter configuration callback
- ***void(* __FMAC_HandleTypeDef::FilterPreloadCallback)(struct __FMAC_HandleTypeDef *hfmac)***
  FMAC filter preload callback
- ***void(* __FMAC_HandleTypeDef::MspInitCallback)(struct __FMAC_HandleTypeDef *hfmac)***
  FMAC Msp Init callback
- ***void(* __FMAC_HandleTypeDef::MspDeInitCallback)(struct __FMAC_HandleTypeDef *hfmac)***
  FMAC Msp DeInit callback
- ***HAL_LockTypeDef __FMAC_HandleTypeDef::Lock***
  FMAC locking object
- ***__IO HAL_FMAC_StateTypeDef __FMAC_HandleTypeDef::State***
  FMAC state related to global handle management This parameter can be a value of **HAL_FMAC_StateTypeDef**
- ***__IO HAL_FMAC_StateTypeDef __FMAC_HandleTypeDef::RdState***
  FMAC state related to read operations (access to Y buffer) This parameter can be a value of **HAL_FMAC_StateTypeDef**
- ***__IO HAL_FMAC_StateTypeDef __FMAC_HandleTypeDef::WrState***
  FMAC state related to write operations (access to X1 buffer) This parameter can be a value of **HAL_FMAC_StateTypeDef**
- ***__IO uint32_t __FMAC_HandleTypeDef::ErrorCode***
  FMAC peripheral error code This parameter can be a value of ***FMAC_Error_Code***

33.1.2 FMAC_FilterConfigTypeDef

FMAC_FilterConfigTypeDef is defined in the stm32h7xx_hal_fmac.h

Data Fields

- ***uint8_t InputBaseAddress***
- ***uint8_t InputBufferSize***
- ***uint32_t InputThreshold***
- ***uint8_t CoeffBaseAddress***
- ***uint8_t CoeffBufferSize***
- ***uint8_t OutputBaseAddress***
- ***uint8_t OutputBufferSize***
- ***uint32_t OutputThreshold***
- ***int16_t * pCoeffA***
- ***uint8_t CoeffASize***
- ***int16_t * pCoeffB***
- ***uint8_t CoeffBSize***
- ***uint8_t InputAccess***
- ***uint8_t OutputAccess***
- ***uint32_t Clip***
- ***uint32_t Filter***
- ***uint8_t P***
- ***uint8_t Q***
- ***uint8_t R***

Field Documentation

- ***uint8_t FMAC_FilterConfigTypeDef::InputBaseAddress***

  Base address of the input buffer (X1) within the internal memory (0x00 to 0xFF). Ignored if InputBufferSize is set to 0 (previous configuration kept). Note: the buffers can overlap or even coincide exactly.
- ***uint8_t FMAC_FilterConfigTypeDef::InputBufferSize***

  Number of 16-bit words allocated to the input buffer (including the optional "headroom"). 0 if a previous configuration should be kept.
- ***uint32_t FMAC_FilterConfigTypeDef::InputThreshold***

  Input threshold: the buffer full flag will be set if the number of free spaces in the buffer is lower than this threshold. This parameter can be a value of ***FMAC_Data_Buffer_Threshold***.
- ***uint8_t FMAC_FilterConfigTypeDef::CoeffBaseAddress***

  Base address of the coefficient buffer (X2) within the internal memory (0x00 to 0xFF). Ignored if CoeffBufferSize is set to 0 (previous configuration kept). Note: the buffers can overlap or even coincide exactly.
- ***uint8_t FMAC_FilterConfigTypeDef::CoeffBufferSize***

  Number of 16-bit words allocated to the coefficient buffer. 0 if a previous configuration should be kept.
- ***uint8_t FMAC_FilterConfigTypeDef::OutputBaseAddress***

  Base address of the output buffer (Y) within the internal memory (0x00 to 0xFF). Ignored if OuputBufferSize is set to 0 (previous configuration kept). Note: the buffers can overlap or even coincide exactly.
- ***uint8_t FMAC_FilterConfigTypeDef::OutputBufferSize***

  Number of 16-bit words allocated to the output buffer (including the optional "headroom"). 0 if a previous configuration should be kept.
- ***uint32_t FMAC_FilterConfigTypeDef::OutputThreshold***

  Output threshold: the buffer empty flag will be set if the number of unread values in the buffer is lower than this threshold. This parameter can be a value of ***FMAC_Data_Buffer_Threshold***.
- ***int16_t* FMAC_FilterConfigTypeDef::pCoeffA***

  [IIR only] Initialization of the coefficient vector A. If not needed, it should be set to NULL.
- ***uint8_t FMAC_FilterConfigTypeDef::CoeffASize***

  Size of the coefficient vector A.

- ***int16_t* FMAC_FilterConfigTypeDef::pCoeffB***

  Initialization of the coefficient vector B. If not needed (re-use of a previously loaded buffer), it should be set to NULL.
- ***uint8_t FMAC_FilterConfigTypeDef::CoeffBSize***

  Size of the coefficient vector B.
- ***uint8_t FMAC_FilterConfigTypeDef::InputAccess***

  Access to the input buffer (internal memory area): DMA, IT, Polling, None. This parameter can be a value of ***FMAC_Buffer_Access***.
- ***uint8_t FMAC_FilterConfigTypeDef::OutputAccess***

  Access to the output buffer (internal memory area): DMA, IT, Polling, None. This parameter can be a value of ***FMAC_Buffer_Access***.
- ***uint32_t FMAC_FilterConfigTypeDef::Clip***

  Enable or disable the clipping feature. If the q1.15 range is exceeded, wrapping is done when the clipping feature is disabled and saturation is done when the clipping feature is enabled. This parameter can be a value of ***FMAC_Clip_State***.
- ***uint32_t FMAC_FilterConfigTypeDef::Filter***

  Filter type. This parameter can be a value of ***FMAC_Functions*** (filter related values).
- ***uint8_t FMAC_FilterConfigTypeDef::P***

  Parameter P (vector length, number of filter taps, etc.).
- ***uint8_t FMAC_FilterConfigTypeDef::Q***

  Parameter Q (vector length, etc.). Ignored if not needed.
- ***uint8_t FMAC_FilterConfigTypeDef::R***

  Parameter R (gain, etc.). Ignored if not needed.

33.2 FMAC Firmware driver API description

The following section lists the various functions of the FMAC library.

33.2.1 How to use this driver

The FMAC HAL driver can be used as follows:

1. Initialize the FMAC low level resources by implementing the HAL_FMAC_MspInit():
   - Enable the FMAC interface clock using __HAL_RCC_FMAC_CLK_ENABLE().
   - In case of using interrupts (e.g. access configured as FMAC_BUFFER_ACCESS_IT):
     - Configure the FMAC interrupt priority using HAL_NVIC_SetPriority().
     - Enable the FMAC IRQ handler using HAL_NVIC_EnableIRQ().
     - In FMAC IRQ handler, call HAL_FMAC_IRQHandler().
   - In case of using DMA to control data transfer (e.g. access configured as FMAC_BUFFER_ACCESS_DMA):
     - Enable the DMA interface clock using __HAL_RCC_DMA1_CLK_ENABLE() or __HAL_RCC_DMA2_CLK_ENABLE() depending on the used DMA instance.
     - Enable the DMAMUX1 interface clock using __HAL_RCC_DMAMUX1_CLK_ENABLE().
     - If the initialization of the internal buffers (coefficients, input, output) is done via DMA, configure and enable one DMA channel for managing data transfer from memory to memory (preload channel).
     - If the input buffer is accessed via DMA, configure and enable one DMA channel for managing data transfer from memory to peripheral (input channel).
     - If the output buffer is accessed via DMA, configure and enable one DMA channel for managing data transfer from peripheral to memory (output channel).
     - Associate the initialized DMA handle(s) to the FMAC DMA handle(s) using __HAL_LINKDMA().
     - Configure the priority and enable the NVIC for the transfer complete interrupt on the enabled DMA channel(s) using HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().

2. Initialize the FMAC HAL using HAL_FMAC_Init(). This function resorts to HAL_FMAC_MspInit() for low-level initialization.
3. Configure the FMAC processing (filter) using HAL_FMAC_FilterConfig() or HAL_FMAC_FilterConfig_DMA(). This function:
   - Defines the memory area within the FMAC internal memory (input, coefficients, output) and the associated threshold (input, output).
   - Configures the filter and its parameters:
     - Finite Impulse Response (FIR) filter (also known as convolution).
     - Infinite Impulse Response (IIR) filter (direct form 1).
   - Choose the way to access to the input and output buffers: none, polling, DMA, IT. "none" means the input and/or output data will be handled by another IP (ADC, DAC, etc.).
   - Enable the error interruptions in the input access and/or the output access is done through IT/DMA. If an error occurs, the interruption will be triggered in loop. In order to recover, the user will have to reset the IP with the sequence HAL_FMAC_DeInit / HAL_FMAC_Init. Optionally, he can also disable the interrupt using __HAL_FMAC_DISABLE_IT; the error status will be kept, but no more interrupt will be triggered.
   - Write the provided coefficients into the internal memory using polling mode (HAL_FMAC_FilterConfig()) or DMA (HAL_FMAC_FilterConfig_DMA()). In the DMA case, HAL_FMAC_FilterConfigCallback() is called when the handling is over.
4. Optionally, the user can enable the error interruption related to saturation by calling __HAL_FMAC_ENABLE_IT. This helps in debugging the filter. If a saturation occurs, the interruption will be triggered in loop. In order to recover, the user will have to:
   - Disable the interruption by calling __HAL_FMAC_DISABLE_IT if the user wishes to continue all the same.
   - Reset the IP with the sequence HAL_FMAC_DeInit / HAL_FMAC_Init.
5. Optionally, preload input (FIR, IIR) and output (IIR) data using HAL_FMAC_FilterPreload() or HAL_FMAC_FilterPreload_DMA(). In the DMA case, HAL_FMAC_FilterPreloadCallback() is called when the handling is over. This step is optional as the filter can be started without preloaded data.
6. Start the FMAC processing (filter) using HAL_FMAC_FilterStart(). This function also configures the output buffer that will be filled from the circular internal output buffer. The function returns immediately without updating the provided buffer. The IP processing will be active until HAL_FMAC_FilterStop() is called.
7. If the input internal buffer is accessed via DMA, HAL_FMAC_HalfGetDataCallback() will be called to indicate that half of the input buffer has been handled.
8. If the input internal buffer is accessed via DMA or interrupt, HAL_FMAC_GetDataCallback() will be called to require new input data. It will be provided through HAL_FMAC_AppendFilterData() if the DMA isn't in circular mode.
9. If the output internal buffer is accessed via DMA, HAL_FMAC_HalfOutputDataReadyCallback() will be called to indicate that half of the output buffer has been handled.
10. If the output internal buffer is accessed via DMA or interrupt, HAL_FMAC_OutputDataReadyCallback() will be called to require a new output buffer. It will be provided through HAL_FMAC_ConfigFilterOutputBuffer() if the DMA isn't in circular mode.
11. In all modes except none, provide new input data to be processed via HAL_FMAC_AppendFilterData(). This function should only be called once the previous input data has been handled (the preloaded input data isn't concerned).
12. In all modes except none, provide a new output buffer to be filled via HAL_FMAC_ConfigFilterOutputBuffer(). This function should only be called once the previous user's output buffer has been filled.
13. In polling mode, handle the input and output data using HAL_FMAC_PollFilterData(). This function:
    - Write the user's input data (provided via HAL_FMAC_AppendFilterData()) into the FMAC input memory area.
    - Read the FMAC output memory area and write it into the user's output buffer. It will return either when:
    - the user's output buffer is filled.
    - the user's input buffer has been handled. The unused data (unread input data or free output data) will not be saved. The user will have to use the updated input and output sizes to keep track of them.
14. Stop the FMAC processing (filter) using HAL_FMAC_FilterStop().

15. Call HAL_FMAC_DeInit() to de-initialize the FMAC peripheral. This function resorts to HAL_FMAC_MspDeInit() for low-level de-initialization.

33.2.2 Callback registration

The compilation define USE_HAL_FMAC_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_FMAC_RegisterCallback() to register a user callback. Function HAL_FMAC_RegisterCallback() allows to register following callbacks:

- ErrorCallback : Error Callback.
- HalfGetDataCallback : Get Half Data Callback.
- GetDataCallback : Get Data Callback.
- HalfOutputDataReadyCallback : Half Output Data Ready Callback.
- OutputDataReadyCallback : Output Data Ready Callback.
- FilterConfigCallback : Filter Configuration Callback.
- FilterPreloadCallback : Filter Preload Callback.
- MspInitCallback : FMAC MspInit.
- MspDeInitCallback : FMAC MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_FMAC_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. HAL_FMAC_UnRegisterCallback() takes as parameters the HAL peripheral handle and the Callback ID. This function allows to reset following callbacks:

- ErrorCallback : Error Callback.
- HalfGetDataCallback : Get Half Data Callback.
- GetDataCallback : Get Data Callback.
- HalfOutputDataReadyCallback : Half Output Data Ready Callback.
- OutputDataReadyCallback : Output Data Ready Callback.
- FilterConfigCallback : Filter Configuration Callback.
- FilterPreloadCallback : Filter Preload Callback.
- MspInitCallback : FMAC MspInit.
- MspDeInitCallback : FMAC MspDeInit.

By default, after the HAL_FMAC_Init() and when the state is HAL_FMAC_STATE_RESET all callbacks are set to the corresponding weak (surcharged) functions: examples GetDataCallback(), OutputDataReadyCallback(). Exception done for MspInit and MspDeInit functions that are respectively reset to the legacy weak (surcharged) functions in the HAL_FMAC_Init() and HAL_FMAC_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_FMAC_Init() and HAL_FMAC_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_FMAC_STATE_READY state only. Exception done MspInit/ MspDeInit that can be registered/unregistered in HAL_FMAC_STATE_READY or HAL_FMAC_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_FMAC_RegisterCallback() before calling HAL_FMAC_DeInit() or HAL_FMAC_Init() function.

When the compilation define USE_HAL_FMAC_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and weak (surcharged) callbacks are used.

33.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the FMAC peripheral and the associated handle
- DeInitialize the FMAC peripheral
- Initialize the FMAC MSP (MCU Specific Package)
- De-Initialize the FMAC MSP
- Register a User FMAC Callback
- Unregister a FMAC CallBack

This section contains the following APIs:

- *HAL_FMAC_Init()*
- *HAL_FMAC_DeInit()*
- *HAL_FMAC_MspInit()*
- *HAL_FMAC_MspDeInit()*
- *HAL_FMAC_RegisterCallback()*
- *HAL_FMAC_UnRegisterCallback()*

33.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure the FMAC peripheral: memory area, filter type and parameters, way to access to the input and output memory area (none, polling, IT, DMA).
- Start the FMAC processing (filter).
- Handle the input data that will be provided into FMAC.
- Handle the output data provided by FMAC.
- Stop the FMAC processing (filter).

This section contains the following APIs:

- *HAL_FMAC_FilterConfig()*
- *HAL_FMAC_FilterConfig_DMA()*
- *HAL_FMAC_FilterPreload()*
- *HAL_FMAC_FilterPreload_DMA()*
- *HAL_FMAC_FilterStart()*
- *HAL_FMAC_AppendFilterData()*
- *HAL_FMAC_ConfigFilterOutputBuffer()*
- *HAL_FMAC_PollFilterData()*
- *HAL_FMAC_FilterStop()*

33.2.5 Callback functions

This section provides Interruption and DMA callback functions:

- DMA or Interrupt: the user's input data is half written (DMA only) or completely written.
- DMA or Interrupt: the user's output buffer is half filled (DMA only) or completely filled.
- DMA or Interrupt: error handling.

This section contains the following APIs:

- *HAL_FMAC_ErrorCallback()*
- *HAL_FMAC_HalfGetDataCallback()*
- *HAL_FMAC_GetDataCallback()*
- *HAL_FMAC_HalfOutputDataReadyCallback()*
- *HAL_FMAC_OutputDataReadyCallback()*
- *HAL_FMAC_FilterConfigCallback()*
- *HAL_FMAC_FilterPreloadCallback()*

33.2.6 IRQ handler management

This section provides IRQ handler function.

This section contains the following APIs:

- *HAL_FMAC_IRQHandler()*

33.2.7 Peripheral State and Error functions

This subsection provides functions allowing to

- Check the FMAC state
- Get error code

This section contains the following APIs:

- *HAL_FMAC_GetState()*
- *HAL_FMAC_GetError()*

33.2.8 Detailed description of functions

HAL_FMAC_Init

Function name

HAL_StatusTypeDef HAL_FMAC_Init (FMAC_HandleTypeDef * hfmac)

Function description

Initialize the FMAC peripheral and the associated handle.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure.

Return values

- **HAL_StatusTypeDef:** HAL status

HAL_FMAC_DeInit

Function name

HAL_StatusTypeDef HAL_FMAC_DeInit (FMAC_HandleTypeDef * hfmac)

Function description

De-initialize the FMAC peripheral.

Parameters

- **hfmac:** pointer to a FMAC structure.

Return values

- **HAL_StatusTypeDef:** HAL status

HAL_FMAC_MspInit

Function name

void HAL_FMAC_MspInit (FMAC_HandleTypeDef * hfmac)

Function description

Initialize the FMAC MSP.

Parameters

- **hfmac:** FMAC handle.

Return values

- **None:**

HAL_FMAC_MspDeInit

Function name

void HAL_FMAC_MspDeInit (FMAC_HandleTypeDef * hfmac)

Function description

De-initialize the FMAC MSP.

Parameters

- **hfmac:** FMAC handle.

Return values

- **None:**

HAL_FMAC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_FMAC_RegisterCallback (FMAC_HandleTypeDef * hfmac, HAL_FMAC_CallbackIDTypeDef CallbackID, pFMAC_CallbackTypeDef pCallback)

Function description

Register a User FMAC Callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **CallbackID:** ID of the callback to be registered. This parameter can be one of the following values:
  - HAL_FMAC_ERROR_CB_ID Error Callback ID
  - HAL_FMAC_HALF_GET_DATA_CB_ID Get Half Data Callback ID
  - HAL_FMAC_GET_DATA_CB_ID Get Data Callback ID
  - HAL_FMAC_HALF_OUTPUT_DATA_READY_CB_ID Half Output Data Ready Callback ID
  - HAL_FMAC_OUTPUT_DATA_READY_CB_ID Output Data Ready Callback ID
  - HAL_FMAC_FILTER_CONFIG_CB_ID Filter Configuration Callback ID
  - HAL_FMAC_FILTER_PRELOAD_CB_ID Filter Preload Callback ID
  - HAL_FMAC_MSPINIT_CB_ID FMAC MspInit ID
  - HAL_FMAC_MSPDEINIT_CB_ID FMAC MspDeInit ID
- **pCallback:** pointer to the Callback function.

Return values

- **HAL_StatusTypeDef:** HAL status

Notes

- The User FMAC Callback is to be used instead of the weak predefined callback.

HAL_FMAC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_FMAC_UnRegisterCallback (FMAC_HandleTypeDef * hfmac, HAL_FMAC_CallbackIDTypeDef CallbackID)

Function description

Unregister a FMAC CallBack.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module
- **CallbackID:** ID of the callback to be unregistered. This parameter can be one of the following values:
  - HAL_FMAC_ERROR_CB_ID Error Callback ID
  - HAL_FMAC_HALF_GET_DATA_CB_ID Get Half Data Callback ID
  - HAL_FMAC_GET_DATA_CB_ID Get Data Callback ID
  - HAL_FMAC_HALF_OUTPUT_DATA_READY_CB_ID Half Output Data Ready Callback ID
  - HAL_FMAC_OUTPUT_DATA_READY_CB_ID Output Data Ready Callback ID
  - HAL_FMAC_FILTER_CONFIG_CB_ID Filter Configuration Callback ID
  - HAL_FMAC_FILTER_PRELOAD_CB_ID Filter Preload Callback ID
  - HAL_FMAC_MSPINIT_CB_ID FMAC MspInit ID
  - HAL_FMAC_MSPDEINIT_CB_ID FMAC MspDeInit ID

Return values

- **HAL_StatusTypeDef:** HAL status

Notes

- The FMAC callback is redirected to the weak predefined callback.

HAL_FMAC_FilterConfig

Function name

HAL_StatusTypeDef HAL_FMAC_FilterConfig (FMAC_HandleTypeDef * hfmac, FMAC_FilterConfigTypeDef * pConfig)

Function description

Configure the FMAC filter.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **pConfig:** pointer to a FMAC_FilterConfigTypeDef structure that contains the FMAC configuration information.

Return values

- **HAL_StatusTypeDef:** HAL status

Notes

- The configuration is done according to the parameters specified in the FMAC_FilterConfigTypeDef structure. The provided data will be loaded using polling mode.

HAL_FMAC_FilterConfig_DMA

Function name

HAL_StatusTypeDef HAL_FMAC_FilterConfig_DMA (FMAC_HandleTypeDef * hfmac, FMAC_FilterConfigTypeDef * pConfig)

Function description

Configure the FMAC filter.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **pConfig:** pointer to a FMAC_FilterConfigTypeDef structure that contains the FMAC configuration information.

Return values

- **HAL_StatusTypeDef:** HAL status

Notes

- The configuration is done according to the parameters specified in the FMAC_FilterConfigTypeDef structure. The provided data will be loaded using DMA.

HAL_FMAC_FilterPreload

Function name

HAL_StatusTypeDef HAL_FMAC_FilterPreload (FMAC_HandleTypeDef * hfmac, int16_t * pInput, uint8_t InputSize, int16_t * pOutput, uint8_t OutputSize)

Function description

Preload the input (FIR, IIR) and output data (IIR) of the FMAC filter.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **pInput:** Preloading of the first elements of the input buffer (X1). If not needed (no data available when starting), it should be set to NULL.
- **InputSize:** Size of the input vector. As pInput is used for preloading data, it cannot be bigger than the input memory area.
- **pOutput:** [IIR] Preloading of the first elements of the output vector (Y). If not needed, it should be set to NULL.
- **OutputSize:** Size of the output vector. As pOutput is used for preloading data, it cannot be bigger than the output memory area.

Return values

- **HAL_StatusTypeDef:** HAL status

Notes

- The set(s) of data will be used by FMAC as soon as HAL_FMAC_FilterStart is called. The provided data will be loaded using polling mode.
- The input and the output buffers can be filled by calling several times HAL_FMAC_FilterPreload (each call filling partly the buffers). In case of overflow (too much data provided through all these calls), an error will be returned.

HAL_FMAC_FilterPreload_DMA

Function name

HAL_StatusTypeDef HAL_FMAC_FilterPreload_DMA (FMAC_HandleTypeDef * hfmac, int16_t * pInput, uint8_t InputSize, int16_t * pOutput, uint8_t OutputSize)

Function description

Preload the input (FIR, IIR) and output data (IIR) of the FMAC filter.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **pInput:** Preloading of the first elements of the input buffer (X1). If not needed (no data available when starting), it should be set to NULL.
- **InputSize:** Size of the input vector. As pInput is used for preloading data, it cannot be bigger than the input memory area.
- **pOutput:** [IIR] Preloading of the first elements of the output vector (Y). If not needed, it should be set to NULL.
- **OutputSize:** Size of the output vector. As pOutput is used for preloading data, it cannot be bigger than the output memory area.

Return values

- **HAL_StatusTypeDef:** HAL status

Notes

- The set(s) of data will be used by FMAC as soon as HAL_FMAC_FilterStart is called. The provided data will be loaded using DMA.
- The input and the output buffers can be filled by calling several times HAL_FMAC_FilterPreload (each call filling partly the buffers). In case of overflow (too much data provided through all these calls), an error will be returned.

HAL_FMAC_FilterStart

Function name

HAL_StatusTypeDef HAL_FMAC_FilterStart (FMAC_HandleTypeDef * hfmac, int16_t * pOutput, uint16_t * pOutputSize)

Function description

Start the FMAC processing according to the existing FMAC configuration.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **pOutput:** pointer to buffer where output data of FMAC processing will be stored in the next steps. If it is set to NULL, the output will not be read and it will be up to an external IP to empty the output buffer.
- **pOutputSize:** pointer to the size of the output buffer. The number of read data will be written here.

Return values

- **HAL_StatusTypeDef:** HAL status

HAL_FMAC_AppendFilterData

Function name

HAL_StatusTypeDef HAL_FMAC_AppendFilterData (FMAC_HandleTypeDef * hfmac, int16_t * pInput, uint16_t * pInputSize)

Function description

Provide a new input buffer that will be loaded into the FMAC input memory area.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **pInput:** New input vector (additional input data).
- **pInputSize:** Size of the input vector (if all the data can't be written, it will be updated with the number of data read from FMAC).

Return values

- **HAL_StatusTypeDef:** HAL status

HAL_FMAC_ConfigFilterOutputBuffer

Function name

HAL_StatusTypeDef HAL_FMAC_ConfigFilterOutputBuffer (FMAC_HandleTypeDef * hfmac, int16_t * pOutput, uint16_t * pOutputSize)

Function description

Provide a new output buffer to be filled with the data computed by FMAC unit.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **pOutput:** New output vector.
- **pOutputSize:** Size of the output vector (if the vector can't be entirely filled, pOutputSize will be updated with the number of data read from FMAC).

Return values

- **HAL_StatusTypeDef:** HAL status

HAL_FMAC_PollFilterData

Function name

HAL_StatusTypeDef HAL_FMAC_PollFilterData (FMAC_HandleTypeDef * hfmac, uint32_t Timeout)

Function description

Handle the input and/or output data in polling mode.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **Timeout:** timeout value.

Return values

- **HAL_StatusTypeDef:** HAL status

Notes

- This function writes the previously provided user's input data and fills the previously provided user's output buffer, according to the existing FMAC configuration (polling mode only). The function returns when the input data has been handled or when the output data is filled. The possible unused data isn't kept. It will be up to the user to handle it. The previously provided pInputSize and pOutputSize will be used to indicate to the size of the read/written data to the user.

HAL_FMAC_FilterStop

Function name

HAL_StatusTypeDef HAL_FMAC_FilterStop (FMAC_HandleTypeDef * hfmac)

Function description

Stop the FMAC processing.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **HAL_StatusTypeDef:** HAL status

HAL_FMAC_ErrorCallback

Function name

void HAL_FMAC_ErrorCallback (FMAC_HandleTypeDef * hfmac)

Function description

FMAC error callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_HalfGetDataCallback

Function name

void HAL_FMAC_HalfGetDataCallback (FMAC_HandleTypeDef * hfmac)

Function description

FMAC get half data callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_GetDataCallback

Function name

void HAL_FMAC_GetDataCallback (FMAC_HandleTypeDef * hfmac)

Function description

FMAC get data callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_HalfOutputDataReadyCallback

Function name

void HAL_FMAC_HalfOutputDataReadyCallback (FMAC_HandleTypeDef * hfmac)

Function description

FMAC half output data ready callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_OutputDataReadyCallback

Function name

void HAL_FMAC_OutputDataReadyCallback (FMAC_HandleTypeDef * hfmac)

Function description

FMAC output data ready callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_FilterConfigCallback

Function name

void HAL_FMAC_FilterConfigCallback (FMAC_HandleTypeDef * hfmac)

Function description

FMAC filter configuration callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_FilterPreloadCallback

Function name

void HAL_FMAC_FilterPreloadCallback (FMAC_HandleTypeDef * hfmac)

Function description

FMAC filter preload callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_IRQHandler

Function name

void HAL_FMAC_IRQHandler (FMAC_HandleTypeDef * hfmac)

Function description

Handle FMAC interrupt request.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_GetState

Function name

HAL_FMAC_StateTypeDef HAL_FMAC_GetState (FMAC_HandleTypeDef * hfmac)

Function description

Return the FMAC state.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **HAL_FMAC_StateTypeDef:** FMAC state

HAL_FMAC_GetError

Function name

uint32_t HAL_FMAC_GetError (FMAC_HandleTypeDef * hfmac)

Function description

Return the FMAC peripheral error.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **uint32_t:** Error bit-map based on FMAC Error code

Notes

- The returned error is a bit-map combination of possible errors.

33.3 FMAC Firmware driver defines

The following section lists the various define and macros of the module.

33.3.1 FMAC

FMAC

FMAC Buffer Access

FMAC_BUFFER_ACCESS_NONE

Buffer handled by an external IP (ADC for instance)

FMAC_BUFFER_ACCESS_DMA

Buffer accessed through DMA

FMAC_BUFFER_ACCESS_POLLING

Buffer accessed through polling

FMAC_BUFFER_ACCESS_IT

Buffer accessed through interruptions

FMAC Clip State

FMAC_CLIP_DISABLED

Clipping disabled

FMAC_CLIP_ENABLED

Clipping enabled

FMAC Data Buffer Threshold

FMAC_THRESHOLD_1

Input: Buffer full flag set if the number of free spaces in the buffer is less than 1. Output: Buffer empty flag set if the number of unread values in the buffer is less than 1.

FMAC_THRESHOLD_2

Input: Buffer full flag set if the number of free spaces in the buffer is less than 2. Output: Buffer empty flag set if the number of unread values in the buffer is less than 2.

FMAC_THRESHOLD_4

Input: Buffer full flag set if the number of free spaces in the buffer is less than 4. Output: Buffer empty flag set if the number of unread values in the buffer is less than 4.

FMAC_THRESHOLD_8

Input: Buffer full flag set if the number of free spaces in the buffer is less than 8. Output: Buffer empty flag set if the number of unread values in the buffer is less than 8.

FMAC_THRESHOLD_NO_VALUE

The configured threshold value shouldn't be changed

FMAC Error code

HAL_FMAC_ERROR_NONE

No error

HAL_FMAC_ERROR_SAT

Saturation error

HAL_FMAC_ERROR_UNFL

Underflow error

HAL_FMAC_ERROR_OVFL

Overflow error

HAL_FMAC_ERROR_DMA

DMA error

HAL_FMAC_ERROR_RESET

Reset error

HAL_FMAC_ERROR_PARAM

Parameter error

HAL_FMAC_ERROR_INVALID_CALLBACK

Invalid Callback error

HAL_FMAC_ERROR_TIMEOUT

Timeout error

FMAC Exported Macros

__HAL_FMAC_RESET_HANDLE_STATE

Description:

- Reset FMAC handle state.

Parameters:

- __HANDLE__: FMAC handle.

Return value:

- None

__HAL_FMAC_ENABLE_IT

Description:

- Enable the specified FMAC interrupt.

Parameters:

- __HANDLE__: FMAC handle.
- __INTERRUPT__: FMAC Interrupt. This parameter can be any combination of the following values:
  - FMAC_IT_RIEN Read interrupt enable
  - FMAC_IT_WIEN Write interrupt enable
  - FMAC_IT_OVFLIEN Overflow error interrupt enable
  - FMAC_IT_UNFLIEN Underflow error interrupt enable
  - FMAC_IT_SATIEN Saturation error interrupt enable (this helps in debugging a filter)

Return value:

- None

__HAL_FMAC_DISABLE_IT

Description:

- Disable the FMAC interrupt.

Parameters:

- __HANDLE__: FMAC handle.
- __INTERRUPT__: FMAC Interrupt. This parameter can be any combination of the following values:
  - FMAC_IT_RIEN Read interrupt enable
  - FMAC_IT_WIEN Write interrupt enable
  - FMAC_IT_OVFLIEN Overflow error interrupt enable
  - FMAC_IT_UNFLIEN Underflow error interrupt enable
  - FMAC_IT_SATIEN Saturation error interrupt enable (this helps in debugging a filter)

Return value:

- None

__HAL_FMAC_GET_IT

Description:

- Check whether the specified FMAC interrupt occurred or not.

Parameters:

- __HANDLE__: FMAC handle.
- __INTERRUPT__: FMAC interrupt to check. This parameter can be any combination of the following values:
  - FMAC_FLAG_YEMPTY Y Buffer Empty Flag
  - FMAC_FLAG_X1FULL X1 Buffer Full Flag
  - FMAC_FLAG_OVFL Overflow Error Flag
  - FMAC_FLAG_UNFL Underflow Error Flag
  - FMAC_FLAG_SAT Saturation Error Flag

Return value:

- SET: (interrupt occurred) or RESET (interrupt did not occurred)

__HAL_FMAC_CLEAR_IT

Description:

- Clear specified FMAC interrupt status.

Parameters:

- __HANDLE__: FMAC handle.
- __INTERRUPT__: FMAC interrupt to clear.

Return value:

- None

__HAL_FMAC_GET_FLAG

Description:

- Check whether the specified FMAC status flag is set or not.

Parameters:

- __HANDLE__: FMAC handle.
- __FLAG__: FMAC flag to check. This parameter can be any combination of the following values:
  - FMAC_FLAG_YEMPTY Y Buffer Empty Flag
  - FMAC_FLAG_X1FULL X1 Buffer Full Flag
  - FMAC_FLAG_OVFL Overflow Error Flag
  - FMAC_FLAG_UNFL Underflow Error Flag
  - FMAC_FLAG_SAT Saturation error Flag

Return value:

- SET: (flag is set) or RESET (flag is reset)

__HAL_FMAC_CLEAR_FLAG

Description:

- Clear specified FMAC status flag.

Parameters:

- __HANDLE__: FMAC handle.
- __FLAG__: FMAC flag to clear.

Return value:

- None

__HAL_FMAC_GET_IT_SOURCE

Description:

- Check whether the specified FMAC interrupt is enabled or not.

Parameters:

- __HANDLE__: FMAC handle.
- __INTERRUPT__: FMAC interrupt to check. This parameter can be one of the following values:
  - FMAC_IT_RIEN Read interrupt enable
  - FMAC_IT_WIEN Write interrupt enable
  - FMAC_IT_OVFLIEN Overflow error interrupt enable
  - FMAC_IT_UNFLIEN Underflow error interrupt enable
  - FMAC_IT_SATIEN Saturation error interrupt enable (this helps in debugging a filter)

Return value:

- FlagStatus

FMAC status flags

FMAC_FLAG_YEMPTY

Y Buffer Empty Flag

FMAC_FLAG_X1FULL

X1 Buffer Full Flag

FMAC_FLAG_OVFL

Overflow Error Flag

FMAC_FLAG_UNFL

Underflow Error Flag

FMAC_FLAG_SAT

Saturation Error Flag (this helps in debugging a filter)

FMAC Functions

FMAC_FUNC_LOAD_X1

Load X1 buffer

FMAC_FUNC_LOAD_X2

Load X2 buffer

FMAC_FUNC_LOAD_Y

Load Y buffer

FMAC_FUNC_CONVO_FIR

Convolution (FIR filter)

FMAC_FUNC_IIR_DIRECT_FORM_1

IIR filter (direct form 1)

FMAC Interrupts Enable bit

FMAC_IT_RIEN

Read Interrupt Enable

FMAC_IT_WIEN

Write Interrupt Enable

FMAC_IT_OVFLIEN

Overflow Error Interrupt Enable

FMAC_IT_UNFLIEN

Underflow Error Interrupt Enable

FMAC_IT_SATIEN

Saturation Error Interrupt Enable (this helps in debugging a filter)

34 HAL GFXMMU Generic Driver

34.1 GFXMMU Firmware driver registers structures

34.1.1 GFXMMU_BuffersTypeDef

GFXMMU_BuffersTypeDef is defined in the stm32h7xx_hal_gfxmmu.h

Data Fields

- ***uint32_t Buf0Address***
- ***uint32_t Buf1Address***
- ***uint32_t Buf2Address***
- ***uint32_t Buf3Address***

Field Documentation

- ***uint32_t GFXMMU_BuffersTypeDef::Buf0Address***
  Physical address of buffer 0.
- ***uint32_t GFXMMU_BuffersTypeDef::Buf1Address***
  Physical address of buffer 1.
- ***uint32_t GFXMMU_BuffersTypeDef::Buf2Address***
  Physical address of buffer 2.
- ***uint32_t GFXMMU_BuffersTypeDef::Buf3Address***
  Physical address of buffer 3.

34.1.2 GFXMMU_CachePrefetchTypeDef

GFXMMU_CachePrefetchTypeDef is defined in the stm32h7xx_hal_gfxmmu.h

Data Fields

- ***FunctionalState Activation***
- ***uint32_t CacheLock***
- ***uint32_t CacheLockBuffer***
- ***uint32_t CacheForce***
- ***uint32_t OutterBufferability***
- ***uint32_t OutterCachability***
- ***uint32_t Prefetch***

Field Documentation

- ***FunctionalState GFXMMU_CachePrefetchTypeDef::Activation***
  Cache and pre-fetch enable/disable.
  Note:
  – : All following parameters are useful only if cache and pre-fetch are enabled.
- ***uint32_t GFXMMU_CachePrefetchTypeDef::CacheLock***
  Locking the cache to a buffer. This parameter can be a value of ***GFXMMU_CacheLock***.
- ***uint32_t GFXMMU_CachePrefetchTypeDef::CacheLockBuffer***
  Buffer on which the cache is locked. This parameter can be a value of ***GFXMMU_CacheLockBuffer***.
  Note:
  – : Useful only when lock of the cache is enabled.
- ***uint32_t GFXMMU_CachePrefetchTypeDef::CacheForce***
  Forcing the cache regardless MPU attributes. This parameter can be a value of ***GFXMMU_CacheForce***.
  Note:
  – : Useful only when lock of the cache is enabled.

- ***uint32_t GFXMMU_CachePrefetchTypeDef::OutterBufferability***
  Bufferability of an access generated by the GFXMMU cache. This parameter can be a value of ***GFXMMU_OutterBufferability***.
- ***uint32_t GFXMMU_CachePrefetchTypeDef::OutterCachability***
  Cachability of an access generated by the GFXMMU cache. This parameter can be a value of ***GFXMMU_OutterCachability***.
- ***uint32_t GFXMMU_CachePrefetchTypeDef::Prefetch***
  Pre-fetch enable/disable. This parameter can be a value of ***GFXMMU_Prefetch***.

34.1.3 GFXMMU_InterruptsTypeDef

GFXMMU_InterruptsTypeDef is defined in the stm32h7xx_hal_gfxmmu.h

Data Fields

- ***FunctionalState Activation***
- ***uint32_t UsedInterrupts***

Field Documentation

- ***FunctionalState GFXMMU_InterruptsTypeDef::Activation***
  Interrupts enable/disable
- ***uint32_t GFXMMU_InterruptsTypeDef::UsedInterrupts***
  Interrupts used. This parameter can be a values combination of ***GFXMMU_Interrupts***.
  Note:
  – : Useful only when interrupts are enabled.

34.1.4 GFXMMU_InitTypeDef

GFXMMU_InitTypeDef is defined in the stm32h7xx_hal_gfxmmu.h

Data Fields

- ***uint32_t BlocksPerLine***
- ***uint32_t DefaultValue***
- ***GFXMMU_BuffersTypeDef Buffers***
- ***GFXMMU_CachePrefetchTypeDef CachePrefetch***
- ***GFXMMU_InterruptsTypeDef Interrupts***

Field Documentation

- ***uint32_t GFXMMU_InitTypeDef::BlocksPerLine***
  Number of blocks of 16 bytes per line. This parameter can be a value of ***GFXMMU_BlocksPerLine***.
- ***uint32_t GFXMMU_InitTypeDef::DefaultValue***
  Value returned when virtual memory location not physically mapped.
- ***GFXMMU_BuffersTypeDef GFXMMU_InitTypeDef::Buffers***
  Physical buffers addresses.
- ***GFXMMU_CachePrefetchTypeDef GFXMMU_InitTypeDef::CachePrefetch***
  Cache and pre-fetch parameters.
- ***GFXMMU_InterruptsTypeDef GFXMMU_InitTypeDef::Interrupts***
  Interrupts parameters.

34.1.5 __GFXMMU_HandleTypeDef

__GFXMMU_HandleTypeDef is defined in the stm32h7xx_hal_gfxmmu.h

Data Fields

- ***GFXMMU_TypeDef * Instance***
- ***GFXMMU_InitTypeDef Init***
- ***HAL_GFXMMU_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***GFXMMU_TypeDef* __GFXMMU_HandleTypeDef::Instance***
  GFXMMU instance
- ***GFXMMU_InitTypeDef __GFXMMU_HandleTypeDef::Init***
  GFXMMU init parameters
- ***HAL_GFXMMU_StateTypeDef __GFXMMU_HandleTypeDef::State***
  GFXMMU state
- ***__IO uint32_t __GFXMMU_HandleTypeDef::ErrorCode***
  GFXMMU error code
- ***void(* __GFXMMU_HandleTypeDef::ErrorCallback)(struct __GFXMMU_HandleTypeDef *hgfxmmu)***
  GFXMMU error callback
- ***void(* __GFXMMU_HandleTypeDef::MspInitCallback)(struct __GFXMMU_HandleTypeDef *hgfxmmu)***
  GFXMMU MSP init callback
- ***void(* __GFXMMU_HandleTypeDef::MspDeInitCallback)(struct __GFXMMU_HandleTypeDef *hgfxmmu)***
  GFXMMU MSP de-init callback

34.1.6 GFXMMU_LutLineTypeDef

GFXMMU_LutLineTypeDef is defined in the stm32h7xx_hal_gfxmmu.h

Data Fields

- ***uint32_t LineNumber***
- ***uint32_t LineStatus***
- ***uint32_t FirstVisibleBlock***
- ***uint32_t LastVisibleBlock***
- ***int32_t LineOffset***

Field Documentation

- ***uint32_t GFXMMU_LutLineTypeDef::LineNumber***
  LUT line number. This parameter must be a number between Min_Data = 0 and Max_Data = 1023.
- ***uint32_t GFXMMU_LutLineTypeDef::LineStatus***
  LUT line enable/disable. This parameter can be a value of ***GFXMMU_LutLineStatus***.
- ***uint32_t GFXMMU_LutLineTypeDef::FirstVisibleBlock***
  First visible block on this line. This parameter must be a number between Min_Data = 0 and Max_Data = 255.
- ***uint32_t GFXMMU_LutLineTypeDef::LastVisibleBlock***
  Last visible block on this line. This parameter must be a number between Min_Data = 0 and Max_Data = 255.
- ***int32_t GFXMMU_LutLineTypeDef::LineOffset***
  Offset of block 0 of the current line in physical buffer. This parameter must be a number between Min_Data = -4080 and Max_Data = 4190208.

  Note:

  – : Line offset has to be computed with the following formula: LineOffset = [(Blocks already used) - (1st visible block)]*BlockSize.

34.2 GFXMMU Firmware driver API description

The following section lists the various functions of the GFXMMU library.

34.2.1 How to use this driver

Initialization

1. As prerequisite, fill in the HAL_GFXMMU_MspInit() :
   – Enable GFXMMU clock interface with __HAL_RCC_GFXMMU_CLK_ENABLE().
   – If interrupts are used, enable and configure GFXMMU global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().
2. Configure the number of blocks per line, default value, physical buffer addresses, cache and pre-fetch parameters and interrupts using the HAL_GFXMMU_Init() function.

LUT configuration

1. Use HAL_GFXMMU_DisableLutLines() to deactivate all LUT lines (or a range of lines).
2. Use HAL_GFXMMU_ConfigLut() to copy LUT from flash to look up RAM.
3. Use HAL_GFXMMU_ConfigLutLine() to configure one line of LUT.

Force flush and/or invalidate of cache

1. Use HAL_GFXMMU_ConfigForceCache() to flush and/or invalidate cache.

Modify physical buffer addresses

1. Use HAL_GFXMMU_ModifyBuffers() to modify physical buffer addresses.

Modify cache and pre-fetch parameters

1. Use HAL_GFXMMU_ModifyCachePrefetch() to modify cache and pre-fetch parameters.

Error management

1. If interrupts are used, HAL_GFXMMU_IRQHandler() will be called when an error occurs. This function will call HAL_GFXMMU_ErrorCallback(). Use HAL_GFXMMU_GetError() to get the error code.

De-initialization

1. As prerequisite, fill in the HAL_GFXMMU_MspDeInit() :
   – Disable GFXMMU clock interface with __HAL_RCC_GFXMMU_CLK_ENABLE().
   – If interrupts has been used, disable GFXMMU global interrupt with HAL_NVIC_DisableIRQ().
2. De-initialize GFXMMU using the HAL_GFXMMU_DeInit() function.

Callback registration

The compilation define USE_HAL_GFXMMU_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use functions HAL_GFXMMU_RegisterCallback() to register a user callback.

Function HAL_GFXMMU_RegisterCallback() allows to register following callbacks:

- ErrorCallback : GFXMMU error.
- MspInitCallback : GFXMMU MspInit.
- MspDeInitCallback : GFXMMU MspDeInit.

This function takes as parameters the HAL peripheral handle, the callback ID and a pointer to the user callback function.

Use function HAL_GFXMMU_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. HAL_GFXMMU_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the callback ID.

This function allows to reset following callbacks:

- ErrorCallback : GFXMMU error.
- MspInitCallback : GFXMMU MspInit.
- MspDeInitCallback : GFXMMU MspDeInit.

By default, after the HAL_GFXMMU_Init and if the state is HAL_GFXMMU_STATE_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions: examples HAL_GFXMMU_ErrorCallback(). Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the HAL_GFXMMU_Init and HAL_GFXMMU_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_GFXMMU_Init and HAL_GFXMMU_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_GFXMMU_RegisterCallback before calling HAL_GFXMMU_DeInit or HAL_GFXMMU_Init function.

When the compilation define USE_HAL_GFXMMU_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

34.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the GFXMMU.
- De-initialize the GFXMMU.

This section contains the following APIs:

- ***HAL_GFXMMU_Init()***
- ***HAL_GFXMMU_DeInit()***
- ***HAL_GFXMMU_MspInit()***
- ***HAL_GFXMMU_MspDeInit()***
- ***HAL_GFXMMU_RegisterCallback()***
- ***HAL_GFXMMU_UnRegisterCallback()***

34.2.3 Operation functions

This section provides functions allowing to:

- Configure LUT.
- Force flush and/or invalidate of cache.
- Modify physical buffer addresses.
- Modify cache and pre-fetch parameters.
- Manage error.

This section contains the following APIs:

- ***HAL_GFXMMU_ConfigLut()***
- ***HAL_GFXMMU_DisableLutLines()***
- ***HAL_GFXMMU_ConfigLutLine()***
- ***HAL_GFXMMU_ConfigForceCache()***
- ***HAL_GFXMMU_ModifyBuffers()***
- ***HAL_GFXMMU_ModifyCachePrefetch()***
- ***HAL_GFXMMU_IRQHandler()***
- ***HAL_GFXMMU_ErrorCallback()***

34.2.4 State functions

This section provides functions allowing to:

- Get GFXMMU handle state.
- Get GFXMMU error code.

This section contains the following APIs:

- ***HAL_GFXMMU_GetState()***
- ***HAL_GFXMMU_GetError()***

34.2.5 Detailed description of functions

HAL_GFXMMU_Init

Function name

HAL_StatusTypeDef HAL_GFXMMU_Init (GFXMMU_HandleTypeDef * hgfxmmu)

Function description

Initialize the GFXMMU according to the specified parameters in the GFXMMU_InitTypeDef structure and initialize the associated handle.

Parameters

- **hgfxmmu:** GFXMMU handle.

Return values

- **HAL:** status.

HAL_GFXMMU_DeInit

Function name

HAL_StatusTypeDef HAL_GFXMMU_DeInit (GFXMMU_HandleTypeDef * hgfxmmu)

Function description

De-initialize the GFXMMU.

Parameters

- **hgfxmmu:** GFXMMU handle.

Return values

- **HAL:** status.

HAL_GFXMMU_MspInit

Function name

void HAL_GFXMMU_MspInit (GFXMMU_HandleTypeDef * hgfxmmu)

Function description

Initialize the GFXMMU MSP.

Parameters

- **hgfxmmu:** GFXMMU handle.

Return values

- **None.:**

HAL_GFXMMU_MspDeInit

Function name

void HAL_GFXMMU_MspDeInit (GFXMMU_HandleTypeDef * hgfxmmu)

Function description

De-initialize the GFXMMU MSP.

Parameters

- **hgfxmmu:** GFXMMU handle.

Return values

- **None.:**

HAL_GFXMMU_RegisterCallback

Function name

HAL_StatusTypeDef HAL_GFXMMU_RegisterCallback (GFXMMU_HandleTypeDef * hgfxmmu, HAL_GFXMMU_CallbackIDTypeDef CallbackID, pGFXMMU_CallbackTypeDef pCallback)

Function description

Register a user GFXMMU callback to be used instead of the weak predefined callback.

Parameters

- **hgfxmmu:** GFXMMU handle.
- **CallbackID:** ID of the callback to be registered. This parameter can be one of the following values:
  - HAL_GFXMMU_ERROR_CB_ID error callback ID.
  - HAL_GFXMMU_MSPINIT_CB_ID MSP init callback ID.
  - HAL_GFXMMU_MSPDEINIT_CB_ID MSP de-init callback ID.
- **pCallback:** pointer to the callback function.

Return values

- **HAL:** status.

HAL_GFXMMU_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_GFXMMU_UnRegisterCallback (GFXMMU_HandleTypeDef * hgfxmmu, HAL_GFXMMU_CallbackIDTypeDef CallbackID)

Function description

Unregister a user GFXMMU callback.

Parameters

- **hgfxmmu:** GFXMMU handle.
- **CallbackID:** ID of the callback to be unregistered. This parameter can be one of the following values:
  - HAL_GFXMMU_ERROR_CB_ID error callback ID.
  - HAL_GFXMMU_MSPINIT_CB_ID MSP init callback ID.
  - HAL_GFXMMU_MSPDEINIT_CB_ID MSP de-init callback ID.

Return values

- **HAL:** status.

HAL_GFXMMU_ConfigLut

Function name

HAL_StatusTypeDef HAL_GFXMMU_ConfigLut (GFXMMU_HandleTypeDef * hgfxmmu, uint32_t FirstLine, uint32_t LinesNumber, uint32_t Address)

Function description

This function allows to copy LUT from flash to look up RAM.

Parameters

- **hgfxmmu:** GFXMMU handle.
- **FirstLine:** First line enabled on LUT. This parameter must be a number between Min_Data = 0 and Max_Data = 1023.
- **LinesNumber:** Number of lines enabled on LUT. This parameter must be a number between Min_Data = 1 and Max_Data = 1024.
- **Address:** Start address of LUT in flash.

Return values

- **HAL:** status.

HAL_GFXMMU_DisableLutLines

Function name

HAL_StatusTypeDef HAL_GFXMMU_DisableLutLines (GFXMMU_HandleTypeDef * hgfxmmu, uint32_t FirstLine, uint32_t LinesNumber)

Function description

This function allows to disable a range of LUT lines.

Parameters

- **hgfxmmu:** GFXMMU handle.
- **FirstLine:** First line to disable on LUT. This parameter must be a number between Min_Data = 0 and Max_Data = 1023.
- **LinesNumber:** Number of lines to disable on LUT. This parameter must be a number between Min_Data = 1 and Max_Data = 1024.

Return values

- **HAL:** status.

HAL_GFXMMU_ConfigLutLine

Function name

HAL_StatusTypeDef HAL_GFXMMU_ConfigLutLine (GFXMMU_HandleTypeDef * hgfxmmu, GFXMMU_LutLineTypeDef * lutLine)

Function description

This function allows to configure one line of LUT.

Parameters

- **hgfxmmu:** GFXMMU handle.
- **lutLine:** LUT line parameters.

Return values

- **HAL:** status.

HAL_GFXMMU_ConfigForceCache

Function name

HAL_StatusTypeDef HAL_GFXMMU_ConfigForceCache (GFXMMU_HandleTypeDef * hgfxmmu, uint32_t ForceParam)

Function description

This function allows to force flush and/or invalidate of cache.

Parameters

- **hgfxmmu:** GFXMMU handle.
- **ForceParam:** Force cache parameter. This parameter can be a values combination of GFXMMU cache force parameter.

Return values

- **HAL:** status.

HAL_GFXMMU_ModifyBuffers

Function name

HAL_StatusTypeDef HAL_GFXMMU_ModifyBuffers (GFXMMU_HandleTypeDef * hgfxmmu, GFXMMU_BuffersTypeDef * Buffers)

Function description

This function allows to modify physical buffer addresses.

Parameters

- **hgfxmmu:** GFXMMU handle.
- **Buffers:** Buffers parameters.

Return values

- **HAL:** status.

HAL_GFXMMU_ModifyCachePrefetch

Function name

HAL_StatusTypeDef HAL_GFXMMU_ModifyCachePrefetch (GFXMMU_HandleTypeDef * hgfxmmu, GFXMMU_CachePrefetchTypeDef * CachePrefetch)

Function description

This function allows to modify cache and pre-fetch parameters.

Parameters

- **hgfxmmu:** GFXMMU handle.
- **CachePrefetch:** Cache and pre-fetch parameters.

Return values

- **HAL:** status.

HAL_GFXMMU_IRQHandler

Function name

void HAL_GFXMMU_IRQHandler (GFXMMU_HandleTypeDef * hgfxmmu)

Function description

This function handles the GFXMMU interrupts.

Parameters

- **hgfxmmu:** GFXMMU handle.

Return values

- **None.:**

HAL_GFXMMU_ErrorCallback

Function name

void HAL_GFXMMU_ErrorCallback (GFXMMU_HandleTypeDef * hgfxmmu)

Function description

Error callback.

Parameters

- **hgfxmmu:** GFXMMU handle.

Return values

- **None.:**

HAL_GFXMMU_GetState

Function name

HAL_GFXMMU_StateTypeDef HAL_GFXMMU_GetState (GFXMMU_HandleTypeDef * hgfxmmu)

Function description

This function allows to get the current GFXMMU handle state.

Parameters

- **hgfxmmu:** GFXMMU handle.

Return values

- **GFXMMU:** state.

HAL_GFXMMU_GetError

Function name

uint32_t HAL_GFXMMU_GetError (GFXMMU_HandleTypeDef * hgfxmmu)

Function description

This function allows to get the current GFXMMU error code.

Parameters

- **hgfxmmu:** GFXMMU handle.

Return values

- **GFXMMU:** error code.

34.3 GFXMMU Firmware driver defines

The following section lists the various define and macros of the module.

34.3.1 GFXMMU

GFXMMU

GFXMMU blocks per line

GFXMMU_256BLOCKS

256 blocks of 16 bytes per line

GFXMMU_192BLOCKS

192 blocks of 16 bytes per line

GFXMMU cache force

GFXMMU_CACHE_FORCE_DISABLE

Caching not forced

GFXMMU_CACHE_FORCE_ENABLE

Caching forced

GFXMMU cache force parameter

GFXMMU_CACHE_FORCE_FLUSH

Force cache flush

GFXMMU_CACHE_FORCE_INVALIDATE

Force cache invalidate

GFXMMU cache lock

GFXMMU_CACHE_LOCK_DISABLE

Cache not locked to a buffer

GFXMMU_CACHE_LOCK_ENABLE

Cache locked to a buffer

GFXMMU cache lock buffer

GFXMMU_CACHE_LOCK_BUFFER0

Cache locked to buffer 0

GFXMMU_CACHE_LOCK_BUFFER1

Cache locked to buffer 1

GFXMMU_CACHE_LOCK_BUFFER2

Cache locked to buffer 2

GFXMMU_CACHE_LOCK_BUFFER3

Cache locked to buffer 3

GFXMMU Error Code

GFXMMU_ERROR_NONE

No error

GFXMMU_ERROR_BUFFER0_OVERFLOW

Buffer 0 overflow

GFXMMU_ERROR_BUFFER1_OVERFLOW

Buffer 1 overflow

GFXMMU_ERROR_BUFFER2_OVERFLOW

Buffer 2 overflow

GFXMMU_ERROR_BUFFER3_OVERFLOW

Buffer 3 overflow

GFXMMU_ERROR_AHB_MASTER

AHB master error

GFXMMU_ERROR_INVALID_CALLBACK

Invalid callback error

GFXMMU Exported Macros

__HAL_GFXMMU_RESET_HANDLE_STATE

Description:

- Reset GFXMMU handle state.

Parameters:

- __HANDLE__: GFXMMU handle.

Return value:

- None

GFXMMU interrupts

GFXMMU_AHB_MASTER_ERROR_IT

AHB master error interrupt

GFXMMU_BUFFER0_OVERFLOW_IT

Buffer 0 overflow interrupt

GFXMMU_BUFFER1_OVERFLOW_IT

Buffer 1 overflow interrupt

GFXMMU_BUFFER2_OVERFLOW_IT

Buffer 2 overflow interrupt

GFXMMU_BUFFER3_OVERFLOW_IT

Buffer 3 overflow interrupt

GFXMMU LUT line status

GFXMMU_LUT_LINE_DISABLE

LUT line disabled

GFXMMU_LUT_LINE_ENABLE

LUT line enabled

GFXMMU outer bufferability

GFXMMU_OUTTER_BUFFERABILITY_DISABLE

No bufferable

GFXMMU_OUTTER_BUFFERABILITY_ENABLE

Bufferable

GFXMMU outer cachability

GFXMMU_OUTTER_CACHABILITY_DISABLE

No cacheable

GFXMMU_OUTTER_CACHABILITY_ENABLE

Cacheable

GFXMMU pre-fetch

GFXMMU_PREFETCH_DISABLE

Pre-fetch disable

GFXMMU_PREFETCH_ENABLE

Pre-fetch enable

35 HAL GPIO Generic Driver

35.1 GPIO Firmware driver registers structures

35.1.1 GPIO_InitTypeDef

GPIO_InitTypeDef is defined in the stm32h7xx_hal_gpio.h

Data Fields

- ***uint32_t Pin***
- ***uint32_t Mode***
- ***uint32_t Pull***
- ***uint32_t Speed***
- ***uint32_t Alternate***

Field Documentation

- ***uint32_t GPIO_InitTypeDef::Pin***
  Specifies the GPIO pins to be configured. This parameter can be any value of ***GPIO_pins_define***
- ***uint32_t GPIO_InitTypeDef::Mode***
  Specifies the operating mode for the selected pins. This parameter can be a value of ***GPIO_mode_define***
- ***uint32_t GPIO_InitTypeDef::Pull***
  Specifies the Pull-up or Pull-Down activation for the selected pins. This parameter can be a value of ***GPIO_pull_define***
- ***uint32_t GPIO_InitTypeDef::Speed***
  Specifies the speed for the selected pins. This parameter can be a value of ***GPIO_speed_define***
- ***uint32_t GPIO_InitTypeDef::Alternate***
  Peripheral to be connected to the selected pins. This parameter can be a value of ***GPIO_Alternate_function_selection***

35.2 GPIO Firmware driver API description

The following section lists the various functions of the GPIO library.

35.2.1 GPIO Peripheral features

- Each port bit of the general-purpose I/O (GPIO) ports can be individually configured by software in several modes:
  - Input mode
  - Analog mode
  - Output mode
  - Alternate function mode
  - External interrupt/event lines
- During and just after reset, the alternate functions and external interrupt lines are not active and the I/O ports are configured in input floating mode.
- All GPIO pins have weak internal pull-up and pull-down resistors, which can be activated or not.
- In Output or Alternate mode, each IO can be configured on open-drain or push-pull type and the IO speed can be selected depending on the VDD value.
- The microcontroller IO pins are connected to onboard peripherals/modules through a multiplexer that allows only one peripheral alternate function (AF) connected to an IO pin at a time. In this way, there can be no conflict between peripherals sharing the same IO pin.

- All ports have external interrupt/event capability. To use external interrupt lines, the port must be configured in input mode. All available GPIO pins are connected to the 16 external interrupt/event lines from EXTI0 to EXTI15. The external interrupt/event controller consists of up to 23 edge detectors (16 lines are connected to GPIO) for generating event/interrupt requests (each input line can be independently configured to select the type (interrupt or event) and the corresponding trigger event (rising or falling or both). Each line can also be masked independently.

35.2.2 How to use this driver

1. Enable the GPIO AHB clock using the following function: __HAL_RCC_GPIOx_CLK_ENABLE().
2. Configure the GPIO pin(s) using HAL_GPIO_Init().
   - Configure the IO mode using "Mode" member from GPIO_InitTypeDef structure
   - Activate Pull-up, Pull-down resistor using "Pull" member from GPIO_InitTypeDef structure.
   - In case of Output or alternate function mode selection: the speed is configured through "Speed" member from GPIO_InitTypeDef structure.
   - In alternate mode is selection, the alternate function connected to the IO is configured through "Alternate" member from GPIO_InitTypeDef structure.
   - Analog mode is required when a pin is to be used as ADC channel or DAC output.
   - In case of external interrupt/event selection the "Mode" member from GPIO_InitTypeDef structure select the type (interrupt or event) and the corresponding trigger event (rising or falling or both).
3. In case of external interrupt/event mode selection, configure NVIC IRQ priority mapped to the EXTI line using HAL_NVIC_SetPriority() and enable it using HAL_NVIC_EnableIRQ().
4. To get the level of a pin configured in input mode use HAL_GPIO_ReadPin().
5. To set/reset the level of a pin configured in output mode use HAL_GPIO_WritePin()/HAL_GPIO_TogglePin().
6. To lock pin configuration until next reset use HAL_GPIO_LockPin().
7. During and just after reset, the alternate functions are not active and the GPIO pins are configured in input floating mode (except JTAG pins).
8. The LSE oscillator pins OSC32_IN and OSC32_OUT can be used as general purpose (PC14 and PC15, respectively) when the LSE oscillator is off. The LSE has priority over the GPIO function.
9. The HSE oscillator pins OSC_IN/OSC_OUT can be used as general purpose PH0 and PH1, respectively, when the HSE oscillator is off. The HSE has priority over the GPIO function.

35.2.3 Initialization and de-initialization functions

This section provides functions allowing to initialize and de-initialize the GPIOs to be ready for use.

This section contains the following APIs:

- *HAL_GPIO_Init()*
- *HAL_GPIO_DeInit()*

35.2.4 IO operation functions

This section contains the following APIs:

- *HAL_GPIO_ReadPin()*
- *HAL_GPIO_WritePin()*
- *HAL_GPIO_TogglePin()*
- *HAL_GPIO_LockPin()*
- *HAL_GPIO_EXTI_IRQHandler()*
- *HAL_GPIO_EXTI_Callback()*

35.2.5 Detailed description of functions

HAL_GPIO_Init

Function name

void HAL_GPIO_Init (GPIO_TypeDef * GPIOx, GPIO_InitTypeDef * GPIO_Init)

Function description

Initializes the GPIOx peripheral according to the specified parameters in the GPIO_Init.

Parameters

- **GPIOx:** where x can be (A..K) to select the GPIO peripheral.
- **GPIO_Init:** pointer to a GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.

Return values

- **None:**

HAL_GPIO_DeInit

Function name

void HAL_GPIO_DeInit (GPIO_TypeDef * GPIOx, uint32_t GPIO_Pin)

Function description

De-initializes the GPIOx peripheral registers to their default reset values.

Parameters

- **GPIOx:** where x can be (A..K) to select the GPIO peripheral.
- **GPIO_Pin:** specifies the port bit to be written. This parameter can be one of GPIO_PIN_x where x can be (0..15).

Return values

- **None:**

HAL_GPIO_ReadPin

Function name

GPIO_PinState HAL_GPIO_ReadPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)

Function description

Reads the specified input port pin.

Parameters

- **GPIOx:** where x can be (A..K) to select the GPIO peripheral.
- **GPIO_Pin:** specifies the port bit to read. This parameter can be GPIO_PIN_x where x can be (0..15).

Return values

- **The:** input port pin value.

HAL_GPIO_WritePin

Function name

void HAL_GPIO_WritePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)

Function description

Sets or clears the selected data port bit.

Parameters

- **GPIOx:** where x can be (A..K) to select the GPIO peripheral.
- **GPIO_Pin:** specifies the port bit to be written. This parameter can be one of GPIO_PIN_x where x can be (0..15).
- **PinState:** specifies the value to be written to the selected bit. This parameter can be one of the GPIO_PinState enum values:
  - GPIO_PIN_RESET: to clear the port pin
  - GPIO_PIN_SET: to set the port pin

Return values

- **None:**

Notes

- This function uses GPIOx_BSRR register to allow atomic read/modify accesses. In this way, there is no risk of an IRQ occurring between the read and the modify access.

HAL_GPIO_TogglePin

Function name

void HAL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)

Function description

Toggles the specified GPIO pins.

Parameters

- **GPIOx:** Where x can be (A..K) to select the GPIO peripheral.
- **GPIO_Pin:** Specifies the pins to be toggled.

Return values

- **None:**

HAL_GPIO_LockPin

Function name

HAL_StatusTypeDef HAL_GPIO_LockPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)

Function description

Locks GPIO Pins configuration registers.

Parameters

- **GPIOx:** where x can be (A..K) to select the GPIO peripheral for STM32H7 family
- **GPIO_Pin:** specifies the port bit to be locked. This parameter can be any combination of GPIO_PIN_x where x can be (0..15).

Return values

- **None:**

Notes

- The locked registers are GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR, GPIOx_PUPDR, GPIOx_AFRL and GPIOx_AFRH.
- The configuration of the locked GPIO pins can no longer be modified until the next reset.

HAL_GPIO_EXTI_IRQHandler

Function name

void HAL_GPIO_EXTI_IRQHandler (uint16_t GPIO_Pin)

Function description

Handle EXTI interrupt request.

Parameters

- **GPIO_Pin:** Specifies the port pin connected to corresponding EXTI line.

Return values

- **None:**

HAL_GPIO_EXTI_Callback

Function name

void HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin)

Function description

EXTI line detection callback.

Parameters

- **GPIO_Pin:** Specifies the port pin connected to corresponding EXTI line.

Return values

- **None:**

35.3 GPIO Firmware driver defines

The following section lists the various define and macros of the module.

35.3.1 GPIO

GPIO

GPIO Alternate Function Selection

GPIO_AF0_RTC_50Hz

GPIO_AF0_MCO

GPIO_AF0_SWJ

GPIO_AF0_LCDBIAS

GPIO_AF0_TRACE

GPIO_AF0_C1DSLEEP

GPIO_AF0_C1SLEEP

GPIO_AF0_D1PWREN

GPIO_AF0_D2PWREN

GPIO_AF0_C2DSLEEP

GPIO_AF0_C2SLEEP

GPIO_AF1_TIM1

GPIO_AF1_TIM2

GPIO_AF1_TIM16

GPIO_AF1_TIM17

GPIO_AF1_LPTIM1

GPIO_AF1_HRTIM1

GPIO_AF1_SAI4

GPIO_AF1_FMC

GPIO_AF2_TIM3

GPIO_AF2_TIM4

GPIO_AF2_TIM5

GPIO_AF2_TIM12

GPIO_AF2_SAI1

GPIO_AF2_HRTIM1

GPIO_AF2_TIM15

GPIO_AF3_TIM8

GPIO_AF3_LPTIM2

GPIO_AF3_DFSDM1

GPIO_AF3_LPTIM3

GPIO_AF3_LPTIM4

GPIO_AF3_LPTIM5

GPIO_AF3_LPUART

GPIO_AF3_HRTIM1

GPIO_AF3_LTDC

GPIO_AF4_I2C1

GPIO_AF4_I2C2

GPIO_AF4_I2C3

GPIO_AF4_I2C4

GPIO_AF4_TIM15

GPIO_AF4_CEC

GPIO_AF4_LPTIM2

GPIO_AF4_USART1

GPIO_AF4_DFSDM1

GPIO_AF4_DCMI

GPIO_AF5_SPI1

GPIO_AF5_SPI2

GPIO_AF5_SPI3

GPIO_AF5_SPI4

GPIO_AF5_SPI5

GPIO_AF5_SPI6

GPIO_AF5_CEC

GPIO_AF6_SPI2

GPIO_AF6_SPI3

GPIO_AF6_SAI1

GPIO_AF6_I2C4

GPIO_AF6_DFSDM1

GPIO_AF6_UART4

GPIO_AF6_SAI3

GPIO_AF7_SPI2

GPIO_AF7_SPI3

GPIO_AF7_SPI6

GPIO_AF7_USART1

GPIO_AF7_USART2

GPIO_AF7_USART3

GPIO_AF7_USART6

GPIO_AF7_UART7

GPIO_AF7_SDMMC1

GPIO_AF8_SPI6

GPIO_AF8_SAI2

GPIO_AF8_UART4

GPIO_AF8_UART5

GPIO_AF8_UART8

GPIO_AF8_SPDIF

GPIO_AF8_LPUART

GPIO_AF8_SDMMC1

GPIO_AF8_SAI4

GPIO_AF9_FDCAN1

GPIO_AF9_FDCAN2

GPIO_AF9_TIM13

GPIO_AF9_TIM14

GPIO_AF9_SDMMC2

GPIO_AF9_LTDC

GPIO_AF9_SPDIF

GPIO_AF9_FMC

GPIO_AF9_QUADSPI

GPIO_AF9_SAI4

GPIO_AF10_SAI2

GPIO_AF10_SDMMC2

GPIO_AF10_OTG2_FS

GPIO_AF10_COMP1

GPIO_AF10_COMP2

GPIO_AF10_LTDC

GPIO_AF10_CRS_SYNC

GPIO_AF10_QUADSPI

GPIO_AF10_SAI4

GPIO_AF10_OTG1_HS

GPIO_AF10_TIM8

GPIO_AF10_FMC

GPIO_AF11_SWP

GPIO_AF11_MDIOS

GPIO_AF11_UART7

GPIO_AF11_SDMMC2

GPIO_AF11_DFSDM1

GPIO_AF11_COMP1

GPIO_AF11_COMP2

GPIO_AF11_TIM1

GPIO_AF11_TIM8

GPIO_AF11_I2C4

GPIO_AF11_ETH

GPIO_AF11_LTDC

GPIO_AF12_FMC

GPIO_AF12_SDMMC1

GPIO_AF12_MDIOS

GPIO_AF12_COMP1

GPIO_AF12_COMP2

GPIO_AF12_TIM1

GPIO_AF12_TIM8

GPIO_AF12_LTDC

GPIO_AF12_OTG1_FS

GPIO_AF13_DCMI

GPIO_AF13_COMP1

GPIO_AF13_COMP2

GPIO_AF13_LTDC

GPIO_AF13_TIM1

GPIO_AF14_LTDC

GPIO_AF14_UART5

GPIO_AF15_EVENTOUT

IS_GPIO_AF

GPIO Exported Macros

__HAL_GPIO_EXTI_GET_FLAG

Description:

- Checks whether the specified EXTI line flag is set or not.

Parameters:

- __EXTI_LINE__: specifies the EXTI line flag to check. This parameter can be GPIO_PIN_x where x can be(0..15)

Return value:

- The: new state of __EXTI_LINE__ (SET or RESET).

__HAL_GPIO_EXTI_CLEAR_FLAG

Description:

- Clears the EXTI's line pending flags.

Parameters:

- __EXTI_LINE__: specifies the EXTI lines flags to clear. This parameter can be any combination of GPIO_PIN_x where x can be (0..15)

Return value:

- None

__HAL_GPIO_EXTI_GET_IT

Description:

- Checks whether the specified EXTI line is asserted or not.

Parameters:

- __EXTI_LINE__: specifies the EXTI line to check. This parameter can be GPIO_PIN_x where x can be(0..15)

Return value:

- The: new state of __EXTI_LINE__ (SET or RESET).

__HAL_GPIO_EXTI_CLEAR_IT

Description:

- Clears the EXTI's line pending bits.

Parameters:

- __EXTI_LINE__: specifies the EXTI lines to clear. This parameter can be any combination of GPIO_PIN_x where x can be (0..15)

Return value:

- None

__HAL_GPIO_EXTID2_GET_FLAG

Description:

- Checks whether the specified EXTI line flag is set or not.

Parameters:

- __EXTI_LINE__: specifies the EXTI line flag to check. This parameter can be GPIO_PIN_x where x can be(0..15)

Return value:

- The: new state of __EXTI_LINE__ (SET or RESET).

__HAL_GPIO_EXTID2_CLEAR_FLAG

Description:

- Clears the EXTI's line pending flags.

Parameters:

- __EXTI_LINE__: specifies the EXTI lines flags to clear. This parameter can be any combination of GPIO_PIN_x where x can be (0..15)

Return value:

- None

__HAL_GPIO_EXTID2_GET_IT

Description:

- Checks whether the specified EXTI line is asserted or not.

Parameters:

- __EXTI_LINE__: specifies the EXTI line to check. This parameter can be GPIO_PIN_x where x can be(0..15)

Return value:

- The: new state of __EXTI_LINE__ (SET or RESET).

__HAL_GPIO_EXTID2_CLEAR_IT

Description:

- Clears the EXTI's line pending bits.

Parameters:

- __EXTI_LINE__: specifies the EXTI lines to clear. This parameter can be any combination of GPIO_PIN_x where x can be (0..15)

Return value:

- None

__HAL_GPIO_EXTI_GENERATE_SWIT

Description:

- Generates a Software interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the EXTI line to check. This parameter can be GPIO_PIN_x where x can be(0..15)

Return value:

- None

GPIO mode define

GPIO_MODE_INPUT

Input Floating Mode

GPIO_MODE_OUTPUT_PP

Output Push Pull Mode

GPIO_MODE_OUTPUT_OD

Output Open Drain Mode

GPIO_MODE_AF_PP

Alternate Function Push Pull Mode

GPIO_MODE_AF_OD

Alternate Function Open Drain Mode

GPIO_MODE_ANALOG

Analog Mode

GPIO_MODE_IT_RISING

External Interrupt Mode with Rising edge trigger detection

GPIO_MODE_IT_FALLING

External Interrupt Mode with Falling edge trigger detection

GPIO_MODE_IT_RISING_FALLING

External Interrupt Mode with Rising/Falling edge trigger detection

GPIO_MODE_EVT_RISING

External Event Mode with Rising edge trigger detection

GPIO_MODE_EVT_FALLING

External Event Mode with Falling edge trigger detection

GPIO_MODE_EVT_RISING_FALLING

External Event Mode with Rising/Falling edge trigger detection

GPIO pins define

GPIO_PIN_0

GPIO_PIN_1

GPIO_PIN_2

GPIO_PIN_3

GPIO_PIN_4

GPIO_PIN_5

GPIO_PIN_6

GPIO_PIN_7

GPIO_PIN_8

GPIO_PIN_9

GPIO_PIN_10

GPIO_PIN_11

GPIO_PIN_12

GPIO_PIN_13

GPIO_PIN_14

GPIO_PIN_15

GPIO_PIN_All

GPIO_PIN_MASK

GPIO pull define

GPIO_NOPULL

No Pull-up or Pull-down activation

GPIO_PULLUP

Pull-up activation

GPIO_PULLDOWN

Pull-down activation

GPIO speed define

GPIO_SPEED_FREQ_LOW

Low speed

GPIO_SPEED_FREQ_MEDIUM

Medium speed

GPIO_SPEED_FREQ_HIGH

Fast speed

GPIO_SPEED_FREQ_VERY_HIGH

High speed

36 HAL GPIO Extension Driver

36.1 GPIOEx Firmware driver defines

The following section lists the various define and macros of the module.

36.1.1 GPIOEx

GPIOEx

GPIO Get Port Index

GPIO_GET_INDEX

37 HAL HASH Generic Driver

37.1 HASH Firmware driver registers structures

37.1.1 HASH_InitTypeDef

HASH_InitTypeDef is defined in the stm32h7xx_hal_hash.h

Data Fields

- ***uint32_t DataType***
- ***uint32_t KeySize***
- ***uint8_t * pKey***

Field Documentation

- ***uint32_t HASH_InitTypeDef::DataType***
  32-bit data, 16-bit data, 8-bit data or 1-bit data. This parameter can be a value of ***HASH_Data_Type***.
- ***uint32_t HASH_InitTypeDef::KeySize***
  The key size is used only in HMAC operation.
- ***uint8_t* HASH_InitTypeDef::pKey***
  The key is used only in HMAC operation.

37.1.2 __HASH_HandleTypeDef

__HASH_HandleTypeDef is defined in the stm32h7xx_hal_hash.h

Data Fields

- ***HASH_InitTypeDef Init***
- ***uint8_t * pHashInBuffPtr***
- ***uint8_t * pHashOutBuffPtr***
- ***uint8_t * pHashKeyBuffPtr***
- ***uint8_t * pHashMsgBuffPtr***
- ***uint32_t HashBuffSize***
- ***__IO uint32_t HashInCount***
- ***__IO uint32_t HashITCounter***
- ***__IO uint32_t HashKeyCount***
- ***HAL_StatusTypeDef Status***
- ***HAL_HASH_PhaseTypeDef Phase***
- ***DMA_HandleTypeDef * hdmain***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_HASH_StateTypeDef State***
- ***HAL_HASH_SuspendTypeDef SuspendRequest***
- ***FlagStatus DigestCalculationDisable***
- ***__IO uint32_t NbWordsAlreadyPushed***
- ***__IO uint32_t ErrorCode***
- ***__IO uint32_t Accumulation***
- ***void(* InCpltCallback***
- ***void(* DgstCpltCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***HASH_InitTypeDef __HASH_HandleTypeDef::Init***
  HASH required parameters
- ***uint8_t* __HASH_HandleTypeDef::pHashInBuffPtr***
  Pointer to input buffer
- ***uint8_t* __HASH_HandleTypeDef::pHashOutBuffPtr***
  Pointer to output buffer (digest)
- ***uint8_t* __HASH_HandleTypeDef::pHashKeyBuffPtr***
  Pointer to key buffer (HMAC only)
- ***uint8_t* __HASH_HandleTypeDef::pHashMsgBuffPtr***
  Pointer to message buffer (HMAC only)
- ***uint32_t __HASH_HandleTypeDef::HashBuffSize***
  Size of buffer to be processed
- ***__IO uint32_t __HASH_HandleTypeDef::HashInCount***
  Counter of inputted data
- ***__IO uint32_t __HASH_HandleTypeDef::HashITCounter***
  Counter of issued interrupts
- ***__IO uint32_t __HASH_HandleTypeDef::HashKeyCount***
  Counter for Key inputted data (HMAC only)
- ***HAL_StatusTypeDef __HASH_HandleTypeDef::Status***
  HASH peripheral status
- ***HAL_HASH_PhaseTypeDef __HASH_HandleTypeDef::Phase***
  HASH peripheral phase
- ***DMA_HandleTypeDef* __HASH_HandleTypeDef::hdmain***
  HASH In DMA Handle parameters
- ***HAL_LockTypeDef __HASH_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_HASH_StateTypeDef __HASH_HandleTypeDef::State***
  HASH peripheral state
- ***HAL_HASH_SuspendTypeDef __HASH_HandleTypeDef::SuspendRequest***
  HASH peripheral suspension request flag
- ***FlagStatus __HASH_HandleTypeDef::DigestCalculationDisable***
  Digest calculation phase skip (MDMAT bit control) for multi-buffers DMA-based HMAC computation
- ***__IO uint32_t __HASH_HandleTypeDef::NbWordsAlreadyPushed***
  Numbers of words already pushed in FIFO before inputting new block
- ***__IO uint32_t __HASH_HandleTypeDef::ErrorCode***
  HASH Error code
- ***__IO uint32_t __HASH_HandleTypeDef::Accumulation***
  HASH multi buffers accumulation flag
- ***void(* __HASH_HandleTypeDef::InCpltCallback)(struct __HASH_HandleTypeDef *hhash)***
  HASH input completion callback
- ***void(* __HASH_HandleTypeDef::DgstCpltCallback)(struct __HASH_HandleTypeDef *hhash)***
  HASH digest computation completion callback
- ***void(* __HASH_HandleTypeDef::ErrorCallback)(struct __HASH_HandleTypeDef *hhash)***
  HASH error callback
- ***void(* __HASH_HandleTypeDef::MspInitCallback)(struct __HASH_HandleTypeDef *hhash)***
  HASH Msp Init callback
- ***void(* __HASH_HandleTypeDef::MspDeInitCallback)(struct __HASH_HandleTypeDef *hhash)***
  HASH Msp DeInit callback

37.2 HASH Firmware driver API description

The following section lists the various functions of the HASH library.

37.2.1 How to use this driver

The HASH HAL driver can be used as follows:

1. Initialize the HASH low level resources by implementing the HAL_HASH_MspInit():
   a. Enable the HASH interface clock using __HASH_CLK_ENABLE()
   b. When resorting to interrupt-based APIs (e.g. HAL_HASH_xxx_Start_IT())
      - Configure the HASH interrupt priority using HAL_NVIC_SetPriority()
      - Enable the HASH IRQ handler using HAL_NVIC_EnableIRQ()
      - In HASH IRQ handler, call HAL_HASH_IRQHandler() API
   c. When resorting to DMA-based APIs (e.g. HAL_HASH_xxx_Start_DMA())
      - Enable the DMAx interface clock using __DMAx_CLK_ENABLE()
      - Configure and enable one DMA stream to manage data transfer from memory to peripheral (input stream). Managing data transfer from peripheral to memory can be performed only using CPU.
      - Associate the initialized DMA handle to the HASH DMA handle using __HAL_LINKDMA()
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA stream: use HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ()
2. Initialize the HASH HAL using HAL_HASH_Init(). This function:
   a. resorts to HAL_HASH_MspInit() for low-level initialization,
   b. configures the data type: 1-bit, 8-bit, 16-bit or 32-bit.
3. Three processing schemes are available:
   a. Polling mode: processing APIs are blocking functions i.e. they process the data and wait till the digest computation is finished, e.g. HAL_HASH_xxx_Start() for HASH or HAL_HMAC_xxx_Start() for HMAC
   b. Interrupt mode: processing APIs are not blocking functions i.e. they process the data under interrupt, e.g. HAL_HASH_xxx_Start_IT() for HASH or HAL_HMAC_xxx_Start_IT() for HMAC
   c. DMA mode: processing APIs are not blocking functions and the CPU is not used for data transfer i.e. the data transfer is ensured by DMA, e.g. HAL_HASH_xxx_Start_DMA() for HASH or HAL_HMAC_xxx_Start_DMA() for HMAC. Note that in DMA mode, a call to HAL_HASH_xxx_Finish() is then required to retrieve the digest.
4. When the processing function is called after HAL_HASH_Init(), the HASH peripheral is initialized and processes the buffer fed in input. When the input data have all been fed to the Peripheral, the digest computation can start.
5. Multi-buffer processing is possible in polling, interrupt and DMA modes.
   a. In polling mode, only multi-buffer HASH processing is possible. API HAL_HASH_xxx_Accumulate() must be called for each input buffer, except for the last one. User must resort to HAL_HASH_xxx_Accumulate_End() to enter the last one and retrieve as well the computed digest.
   b. In interrupt mode, API HAL_HASH_xxx_Accumulate_IT() must be called for each input buffer, except for the last one. User must resort to HAL_HASH_xxx_Accumulate_End_IT() to enter the last one and retrieve as well the computed digest.
   c. In DMA mode, multi-buffer HASH and HMAC processing are possible.
      - HASH processing: once initialization is done, MDMAT bit must be set through __HAL_HASH_SET_MDMAT() macro. From that point, each buffer can be fed to the Peripheral through HAL_HASH_xxx_Start_DMA() API. Before entering the last buffer, reset the MDMAT bit with __HAL_HASH_RESET_MDMAT() macro then wrap-up the HASH processing in feeding the last input buffer through the same API HAL_HASH_xxx_Start_DMA(). The digest can then be retrieved with a call to API HAL_HASH_xxx_Finish().
      - HMAC processing (requires to resort to extended functions): after initialization, the key and the first input buffer are entered in the Peripheral with the API HAL_HMACEx_xxx_Step1_2_DMA(). This carries out HMAC step 1 and starts step 2. The following buffers are next entered with the API HAL_HMACEx_xxx_Step2_DMA(). At this point, the HMAC processing is still carrying out step 2. Then, step 2 for the last input buffer and step 3 are carried out by a single call to HAL_HMACEx_xxx_Step2_3_DMA(). The digest can finally be retrieved with a call to API HAL_HASH_xxx_Finish().

6. Context swapping.
   a. Two APIs are available to suspend HASH or HMAC processing:
      - HAL_HASH_SwFeed_ProcessSuspend() when data are entered by software (polling or IT mode),
      - HAL_HASH_DMAFeed_ProcessSuspend() when data are entered by DMA.
   b. When HASH or HMAC processing is suspended, HAL_HASH_ContextSaving() allows to save in memory the Peripheral context. This context can be restored afterwards to resume the HASH processing thanks to HAL_HASH_ContextRestoring().
   c. Once the HASH Peripheral has been restored to the same configuration as that at suspension time, processing can be restarted with the same API call (same API, same handle, same parameters) as done before the suspension. Relevant parameters to restart at the proper location are internally saved in the HASH handle.
7. Call HAL_HASH_DeInit() to deinitialize the HASH peripheral.

Remarks on message length

1. HAL in interruption mode (interruptions driven)
   a. Due to HASH peripheral hardware design, the peripheral interruption is triggered every 64 bytes. This is why, for driver implementation simplicity's sake, user is requested to enter a message the length of which is a multiple of 4 bytes.
   b. When the message length (in bytes) is not a multiple of words, a specific field exists in HASH_STR to specify which bits to discard at the end of the complete message to process only the message bits and not extra bits.
   c. If user needs to perform a hash computation of a large input buffer that is spread around various places in memory and where each piece of this input buffer is not necessarily a multiple of 4 bytes in size, it becomes necessary to use a temporary buffer to format the data accordingly before feeding them to the Peripheral. It is advised to the user to
      - achieve the first formatting operation by software then enter the data
      - while the Peripheral is processing the first input set, carry out the second formatting operation by software, to be ready when DINIS occurs.
      - repeat step 2 until the whole message is processed.

1. HAL in DMA mode
   a. Again, due to hardware design, the DMA transfer to feed the data can only be done on a word-basis. The same field described above in HASH_STR is used to specify which bits to discard at the end of the DMA transfer to process only the message bits and not extra bits. Due to hardware implementation, this is possible only at the end of the complete message. When several DMA transfers are needed to enter the message, this is not applicable at the end of the intermediary transfers.
   b. Similarly to the interruption-driven mode, it is suggested to the user to format the consecutive chunks of data by software while the DMA transfer and processing is on-going for the first parts of the message. Due to the 32-bit alignment required for the DMA transfer, it is underlined that the software formatting operation is more complex than in the IT mode.

Callback registration

1. The compilation define USE_HAL_HASH_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use function HAL_HASH_RegisterCallback() to register a user callback.
2. Function HAL_HASH_RegisterCallback() allows to register following callbacks: (+) InCpltCallback : callback for input completion. (+) DgstCpltCallback : callback for digest computation completion. (+) ErrorCallback : callback for error. (+) MspInitCallback : HASH MspInit. (+) MspDeInitCallback : HASH MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.
3. Use function HAL_HASH_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. HAL_HASH_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks: (+) InCpltCallback : callback for input completion. (+) DgstCpltCallback : callback for digest computation completion. (+) ErrorCallback : callback for error. (+) MspInitCallback : HASH MspInit. (+) MspDeInitCallback : HASH MspDeInit.

4. By default, after the HAL_HASH_Init and if the state is HAL_HASH_STATE_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions: examples HAL_HASH_InCpltCallback(), HAL_HASH_DgstCpltCallback() Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the HAL_HASH_Init and HAL_HASH_DeInit only when these callbacks are null (not registered beforehand) If not, MspInit or MspDeInit are not null, the HAL_HASH_Init and HAL_HASH_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand). Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/ MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_HASH_RegisterCallback before calling HAL_HASH_DeInit or HAL_HASH_Init function. When The compilation define USE_HAL_HASH_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

(#) The compilation define USE_HAL_HASH_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use function HAL_HASH_RegisterCallback() to register a user callback. (#) Function HAL_HASH_RegisterCallback() allows to register following callbacks:

- InCpltCallback : callback for input completion.
- DgstCpltCallback : callback for digest computation completion.
- ErrorCallback : callback for error.
- MspInitCallback : HASH MspInit.
- MspDeInitCallback : HASH MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. (#) Use function HAL_HASH_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. HAL_HASH_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:
- InCpltCallback : callback for input completion.
- DgstCpltCallback : callback for digest computation completion.
- ErrorCallback : callback for error.
- MspInitCallback : HASH MspInit.
- MspDeInitCallback : HASH MspDeInit. (#) By default, after the HAL_HASH_Init and if the state is HAL_HASH_STATE_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions: examples HAL_HASH_InCpltCallback(), HAL_HASH_DgstCpltCallback() Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the HAL_HASH_Init and HAL_HASH_DeInit only when these callbacks are null (not registered beforehand) If not, MspInit or MspDeInit are not null, the HAL_HASH_Init and HAL_HASH_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand). Callbacks can be registered/ unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_HASH_RegisterCallback before calling HAL_HASH_DeInit or HAL_HASH_Init function. When The compilation define USE_HAL_HASH_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

37.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the HASH according to the specified parameters in the HASH_InitTypeDef and create the associated handle
- DeInitialize the HASH peripheral
- Initialize the HASH MCU Specific Package (MSP)
- DeInitialize the HASH MSP

This section provides as well call back functions definitions for user code to manage:

- Input data transfer to Peripheral completion
- Calculated digest retrieval completion
- Error management

This section contains the following APIs:

- ***HAL_HASH_Init()***
- ***HAL_HASH_DeInit()***

- *HAL_HASH_MspInit()*
- *HAL_HASH_MspDeInit()*
- *HAL_HASH_InCpltCallback()*
- *HAL_HASH_DgstCpltCallback()*
- *HAL_HASH_ErrorCallback()*
- *HAL_HASH_RegisterCallback()*
- *HAL_HASH_UnRegisterCallback()*

37.2.3 Polling mode HASH processing functions

This section provides functions allowing to calculate in polling mode the hash value using one of the following algorithms:

- MD5
  - HAL_HASH_MD5_Start()
  - HAL_HASH_MD5_Accmlt()
  - HAL_HASH_MD5_Accmlt_End()
- SHA1
  - HAL_HASH_SHA1_Start()
  - HAL_HASH_SHA1_Accmlt()
  - HAL_HASH_SHA1_Accmlt_End()

For a single buffer to be hashed, user can resort to HAL_HASH_xxx_Start().

In case of multi-buffer HASH processing (a single digest is computed while several buffers are fed to the Peripheral), the user can resort to successive calls to HAL_HASH_xxx_Accumulate() and wrap-up the digest computation by a call to HAL_HASH_xxx_Accumulate_End().

This section contains the following APIs:

- *HAL_HASH_MD5_Start()*
- *HAL_HASH_MD5_Accmlt()*
- *HAL_HASH_MD5_Accmlt_End()*
- *HAL_HASH_SHA1_Start()*
- *HAL_HASH_SHA1_Accmlt()*
- *HAL_HASH_SHA1_Accmlt_End()*

37.2.4 Interruption mode HASH processing functions

This section provides functions allowing to calculate in interrupt mode the hash value using one of the following algorithms:

- MD5
  - HAL_HASH_MD5_Start_IT()
  - HAL_HASH_MD5_Accmlt_IT()
  - HAL_HASH_MD5_Accmlt_End_IT()
- SHA1
  - HAL_HASH_SHA1_Start_IT()
  - HAL_HASH_SHA1_Accmlt_IT()
  - HAL_HASH_SHA1_Accmlt_End_IT()

API HAL_HASH_IRQHandler() manages each HASH interruption.

Note that HAL_HASH_IRQHandler() manages as well HASH Peripheral interruptions when in HMAC processing mode.

This section contains the following APIs:

- *HAL_HASH_MD5_Start_IT()*
- *HAL_HASH_MD5_Accmlt_IT()*
- *HAL_HASH_MD5_Accmlt_End_IT()*
- *HAL_HASH_SHA1_Start_IT()*

- *HAL_HASH_SHA1_Accmlt_IT()*
- *HAL_HASH_SHA1_Accmlt_End_IT()*
- *HAL_HASH_IRQHandler()*

37.2.5 DMA mode HASH processing functions

This section provides functions allowing to calculate in DMA mode the hash value using one of the following algorithms:

- MD5
  - HAL_HASH_MD5_Start_DMA()
  - HAL_HASH_MD5_Finish()
- SHA1
  - HAL_HASH_SHA1_Start_DMA()
  - HAL_HASH_SHA1_Finish()

When resorting to DMA mode to enter the data in the Peripheral, user must resort to HAL_HASH_xxx_Start_DMA() then read the resulting digest with HAL_HASH_xxx_Finish().

In case of multi-buffer HASH processing, MDMAT bit must first be set before the successive calls to HAL_HASH_xxx_Start_DMA(). Then, MDMAT bit needs to be reset before the last call to HAL_HASH_xxx_Start_DMA(). Digest is finally retrieved thanks to HAL_HASH_xxx_Finish().

This section contains the following APIs:

- *HAL_HASH_MD5_Start_DMA()*
- *HAL_HASH_MD5_Finish()*
- *HAL_HASH_SHA1_Start_DMA()*
- *HAL_HASH_SHA1_Finish()*

37.2.6 Polling mode HMAC processing functions

This section provides functions allowing to calculate in polling mode the HMAC value using one of the following algorithms:

- MD5
  - HAL_HMAC_MD5_Start()
- SHA1
  - HAL_HMAC_SHA1_Start()

This section contains the following APIs:

- *HAL_HMAC_MD5_Start()*
- *HAL_HMAC_SHA1_Start()*

37.2.7 Interrupt mode HMAC processing functions

This section provides functions allowing to calculate in interrupt mode the HMAC value using one of the following algorithms:

- MD5
  - HAL_HMAC_MD5_Start_IT()
- SHA1
  - HAL_HMAC_SHA1_Start_IT()

This section contains the following APIs:

- *HAL_HMAC_MD5_Start_IT()*
- *HAL_HMAC_SHA1_Start_IT()*

37.2.8 DMA mode HMAC processing functions

This section provides functions allowing to calculate in DMA mode the HMAC value using one of the following algorithms:

- MD5
  - HAL_HMAC_MD5_Start_DMA()
- SHA1
  - HAL_HMAC_SHA1_Start_DMA()

When resorting to DMA mode to enter the data in the Peripheral for HMAC processing, user must resort to HAL_HMAC_xxx_Start_DMA() then read the resulting digest with HAL_HASH_xxx_Finish().

This section contains the following APIs:

- ***HAL_HMAC_MD5_Start_DMA()***
- ***HAL_HMAC_SHA1_Start_DMA()***

37.2.9 Peripheral State methods

This section permits to get in run-time the state and the peripheral handle status of the peripheral:

- HAL_HASH_GetState()
- HAL_HASH_GetStatus()

Additionally, this subsection provides functions allowing to save and restore the HASH or HMAC processing context in case of calculation suspension:

- HAL_HASH_ContextSaving()
- HAL_HASH_ContextRestoring()

This subsection provides functions allowing to suspend the HASH processing

- when input are fed to the Peripheral by software
  - HAL_HASH_SwFeed_ProcessSuspend()
- when input are fed to the Peripheral by DMA
  - HAL_HASH_DMAFeed_ProcessSuspend()

This section contains the following APIs:

- ***HAL_HASH_GetState()***
- ***HAL_HASH_GetStatus()***
- ***HAL_HASH_ContextSaving()***
- ***HAL_HASH_ContextRestoring()***
- ***HAL_HASH_SwFeed_ProcessSuspend()***
- ***HAL_HASH_DMAFeed_ProcessSuspend()***
- ***HAL_HASH_GetError()***

37.2.10 Detailed description of functions

HAL_HASH_Init

Function name

HAL_StatusTypeDef HAL_HASH_Init (HASH_HandleTypeDef * hhash)

Function description

Initialize the HASH according to the specified parameters in the HASH_HandleTypeDef and create the associated handle.

Parameters

- **hhash:** HASH handle

Return values

- **HAL:** status

Notes

- Only MDMAT and DATATYPE bits of HASH Peripheral are set by HAL_HASH_Init(), other configuration bits are set by HASH or HMAC processing APIs.
- MDMAT bit is systematically reset by HAL_HASH_Init(). To set it for multi-buffer HASH processing, user needs to resort to __HAL_HASH_SET_MDMAT() macro. For HMAC multi-buffer processing, the relevant APIs manage themselves the MDMAT bit.

HAL_HASH_DeInit

Function name

HAL_StatusTypeDef HAL_HASH_DeInit (HASH_HandleTypeDef * hhash)

Function description

DeInitialize the HASH peripheral.

Parameters

- **hhash:** HASH handle.

Return values

- **HAL:** status

HAL_HASH_MspInit

Function name

void HAL_HASH_MspInit (HASH_HandleTypeDef * hhash)

Function description

Initialize the HASH MSP.

Parameters

- **hhash:** HASH handle.

Return values

- **None:**

HAL_HASH_MspDeInit

Function name

void HAL_HASH_MspDeInit (HASH_HandleTypeDef * hhash)

Function description

DeInitialize the HASH MSP.

Parameters

- **hhash:** HASH handle.

Return values

- **None:**

HAL_HASH_InCpltCallback

Function name

void HAL_HASH_InCpltCallback (HASH_HandleTypeDef * hhash)

Function description

Input data transfer complete call back.

Parameters

- **hhash:** HASH handle.

Return values

- **None:**

Notes

- HAL_HASH_InCpltCallback() is called when the complete input message has been fed to the Peripheral. This API is invoked only when input data are entered under interruption or through DMA.
- In case of HASH or HMAC multi-buffer DMA feeding case (MDMAT bit set), HAL_HASH_InCpltCallback() is called at the end of each buffer feeding to the Peripheral.

HAL_HASH_DgstCpltCallback

Function name

void HAL_HASH_DgstCpltCallback (HASH_HandleTypeDef * hhash)

Function description

Digest computation complete call back.

Parameters

- **hhash:** HASH handle.

Return values

- **None:**

Notes

- HAL_HASH_DgstCpltCallback() is used under interruption, is not relevant with DMA.

HAL_HASH_ErrorCallback

Function name

void HAL_HASH_ErrorCallback (HASH_HandleTypeDef * hhash)

Function description

Error callback.

Parameters

- **hhash:** HASH handle.

Return values

- **None:**

Notes

- Code user can resort to hhash->Status (HAL_ERROR, HAL_TIMEOUT,...) to retrieve the error type.

HAL_HASH_RegisterCallback

Function name

HAL_StatusTypeDef HAL_HASH_RegisterCallback (HASH_HandleTypeDef * hhash, HAL_HASH_CallbackIDTypeDef CallbackID, pHASH_CallbackTypeDef pCallback)

Function description

Register a User HASH Callback To be used instead of the weak (surcharged) predefined callback.

Parameters

- **hhash:** HASH handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_HASH_INPUTCPLT_CB_ID HASH input completion Callback ID
  - HAL_HASH_DGSTCPLT_CB_ID HASH digest computation completion Callback ID
  - HAL_HASH_ERROR_CB_ID HASH error Callback ID
  - HAL_HASH_MSPINIT_CB_ID HASH MspInit callback ID
  - HAL_HASH_MSPDEINIT_CB_ID HASH MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **status:**

HAL_HASH_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_HASH_UnRegisterCallback (HASH_HandleTypeDef * hhash, HAL_HASH_CallbackIDTypeDef CallbackID)

Function description

Unregister a HASH Callback HASH Callback is redirected to the weak (surcharged) predefined callback.

Parameters

- **hhash:** HASH handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_HASH_INPUTCPLT_CB_ID HASH input completion Callback ID
  - HAL_HASH_DGSTCPLT_CB_ID HASH digest computation completion Callback ID
  - HAL_HASH_ERROR_CB_ID HASH error Callback ID
  - HAL_HASH_MSPINIT_CB_ID HASH MspInit callback ID
  - HAL_HASH_MSPDEINIT_CB_ID HASH MspDeInit callback ID

Return values

- **status:**

HAL_HASH_SHA1_Start

Function name

HAL_StatusTypeDef HAL_HASH_SHA1_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

Initialize the HASH peripheral in SHA1 mode, next process pInBuffer then read the computed digest.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 20 bytes.
- **Timeout:** Timeout value

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASH_MD5_Start

Function name

HAL_StatusTypeDef HAL_HASH_MD5_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

Initialize the HASH peripheral in MD5 mode, next process pInBuffer then read the computed digest.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 16 bytes.
- **Timeout:** Timeout value

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASH_MD5_Accmlt

Function name

HAL_StatusTypeDef HAL_HASH_MD5_Accmlt (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

If not already done, initialize the HASH peripheral in MD5 mode then processes pInBuffer.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes, must be a multiple of 4.

Return values

- **HAL:** status

Notes

- Consecutive calls to HAL_HASH_MD5_Accmlt() can be used to feed several input buffers back-to-back to the Peripheral that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL_HASH_MD5_Accmlt_End().
- Field hhash->Phase of HASH handle is tested to check whether or not the Peripheral has already been initialized.
- Digest is not retrieved by this API, user must resort to HAL_HASH_MD5_Accmlt_End() to read it, feeding at the same time the last input buffer to the Peripheral.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only HAL_HASH_MD5_Accmlt_End() is able to manage the ending buffer with a length in bytes not a multiple of 4.

HAL_HASH_SHA1_Accmlt

Function name

HAL_StatusTypeDef HAL_HASH_SHA1_Accmlt (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

If not already done, initialize the HASH peripheral in SHA1 mode then processes pInBuffer.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes, must be a multiple of 4.

Return values

- **HAL:** status

Notes

- Consecutive calls to HAL_HASH_SHA1_Accmlt() can be used to feed several input buffers back-to-back to the Peripheral that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL_HASH_SHA1_Accmlt_End().
- Field hhash->Phase of HASH handle is tested to check whether or not the Peripheral has already been initialized.
- Digest is not retrieved by this API, user must resort to HAL_HASH_SHA1_Accmlt_End() to read it, feeding at the same time the last input buffer to the Peripheral.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only HAL_HASH_SHA1_Accmlt_End() is able to manage the ending buffer with a length in bytes not a multiple of 4.

HAL_HASH_MD5_Accmlt_End

Function name

HAL_StatusTypeDef HAL_HASH_MD5_Accmlt_End (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

End computation of a single HASH signature after several calls to HAL_HASH_MD5_Accmlt() API.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 16 bytes.
- **Timeout:** Timeout value

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASH_SHA1_Accmlt_End

Function name

HAL_StatusTypeDef HAL_HASH_SHA1_Accmlt_End (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

End computation of a single HASH signature after several calls to HAL_HASH_SHA1_Accmlt() API.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 20 bytes.
- **Timeout:** Timeout value

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASH_SHA1_Start_IT

Function name

HAL_StatusTypeDef HAL_HASH_SHA1_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)

Function description

Initialize the HASH peripheral in SHA1 mode, next process pInBuffer then read the computed digest in interruption mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 20 bytes.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASH_SHA1_Accmlt_IT

Function name

HAL_StatusTypeDef HAL_HASH_SHA1_Accmlt_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

If not already done, initialize the HASH peripheral in SHA1 mode then processes pInBuffer in interruption mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes, must be a multiple of 4.

Return values

- **HAL:** status

Notes

- Consecutive calls to HAL_HASH_SHA1_Accmlt_IT() can be used to feed several input buffers back-to-back to the Peripheral that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL_HASH_SHA1_Accmlt_End_IT().
- Field hhash->Phase of HASH handle is tested to check whether or not the Peripheral has already been initialized.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only HAL_HASH_SHA1_Accmlt_End_IT() is able to manage the ending buffer with a length in bytes not a multiple of 4.

HAL_HASH_SHA1_Accmlt_End_IT

Function name

HAL_StatusTypeDef HAL_HASH_SHA1_Accmlt_End_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)

Function description

End computation of a single HASH signature after several calls to HAL_HASH_SHA1_Accmlt_IT() API.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 20 bytes.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASH_MD5_Start_IT

Function name

HAL_StatusTypeDef HAL_HASH_MD5_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)

Function description

Initialize the HASH peripheral in MD5 mode, next process pInBuffer then read the computed digest in interruption mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 16 bytes.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASH_MD5_Accmlt_IT

Function name

HAL_StatusTypeDef HAL_HASH_MD5_Accmlt_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

If not already done, initialize the HASH peripheral in MD5 mode then processes pInBuffer in interruption mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes, must be a multiple of 4.

Return values

- **HAL:** status

Notes

- Consecutive calls to HAL_HASH_MD5_Accmlt_IT() can be used to feed several input buffers back-to-back to the Peripheral that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL_HASH_MD5_Accmlt_End_IT().
- Field hhash->Phase of HASH handle is tested to check whether or not the Peripheral has already been initialized.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only HAL_HASH_MD5_Accmlt_End_IT() is able to manage the ending buffer with a length in bytes not a multiple of 4.

HAL_HASH_MD5_Accmlt_End_IT

Function name

HAL_StatusTypeDef HAL_HASH_MD5_Accmlt_End_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)

Function description

End computation of a single HASH signature after several calls to HAL_HASH_MD5_Accmlt_IT() API.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 16 bytes.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASH_IRQHandler

Function name

void HAL_HASH_IRQHandler (HASH_HandleTypeDef * hhash)

Function description

Handle HASH interrupt request.

Parameters

- **hhash:** HASH handle.

Return values

- **None:**

Notes

- HAL_HASH_IRQHandler() handles interrupts in HMAC processing as well.
- In case of error reported during the HASH interruption processing, HAL_HASH_ErrorCallback() API is called so that user code can manage the error. The error type is available in hhash->Status field.

HAL_HASH_SHA1_Start_DMA

Function name

HAL_StatusTypeDef HAL_HASH_SHA1_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

Initialize the HASH peripheral in SHA1 mode then initiate a DMA transfer to feed the input buffer to the Peripheral.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Once the DMA transfer is finished, HAL_HASH_SHA1_Finish() API must be called to retrieve the computed digest.

HAL_HASH_SHA1_Finish

Function name

HAL_StatusTypeDef HAL_HASH_SHA1_Finish (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

Return the computed digest in SHA1 mode.

Parameters

- **hhash:** HASH handle.
- **pOutBuffer:** pointer to the computed digest. Digest size is 20 bytes.
- **Timeout:** Timeout value.

Return values

- **HAL:** status

Notes

- The API waits for DCIS to be set then reads the computed digest.
- HAL_HASH_SHA1_Finish() can be used as well to retrieve the digest in HMAC SHA1 mode.

HAL_HASH_MD5_Start_DMA

Function name

HAL_StatusTypeDef HAL_HASH_MD5_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

Initialize the HASH peripheral in MD5 mode then initiate a DMA transfer to feed the input buffer to the Peripheral.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Once the DMA transfer is finished, HAL_HASH_MD5_Finish() API must be called to retrieve the computed digest.

HAL_HASH_MD5_Finish

Function name

HAL_StatusTypeDef HAL_HASH_MD5_Finish (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

Return the computed digest in MD5 mode.

Parameters

- **hhash:** HASH handle.
- **pOutBuffer:** pointer to the computed digest. Digest size is 16 bytes.
- **Timeout:** Timeout value.

Return values

- **HAL:** status

Notes

- The API waits for DCIS to be set then reads the computed digest.
- HAL_HASH_MD5_Finish() can be used as well to retrieve the digest in HMAC MD5 mode.

HAL_HMAC_SHA1_Start

Function name

HAL_StatusTypeDef HAL_HMAC_SHA1_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

Initialize the HASH peripheral in HMAC SHA1 mode, next process pInBuffer then read the computed digest.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 20 bytes.
- **Timeout:** Timeout value.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HAL_HMAC_MD5_Start

Function name

HAL_StatusTypeDef HAL_HMAC_MD5_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

Initialize the HASH peripheral in HMAC MD5 mode, next process pInBuffer then read the computed digest.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 16 bytes.
- **Timeout:** Timeout value.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HAL_HMAC_MD5_Start_IT

Function name

HAL_StatusTypeDef HAL_HMAC_MD5_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)

Function description

Initialize the HASH peripheral in HMAC MD5 mode, next process pInBuffer then read the computed digest in interrupt mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 16 bytes.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HAL_HMAC_SHA1_Start_IT

Function name

HAL_StatusTypeDef HAL_HMAC_SHA1_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)

Function description

Initialize the HASH peripheral in HMAC SHA1 mode, next process pInBuffer then read the computed digest in interrupt mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 20 bytes.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HAL_HMAC_SHA1_Start_DMA

Function name

HAL_StatusTypeDef HAL_HMAC_SHA1_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

Initialize the HASH peripheral in HMAC SHA1 mode then initiate the required DMA transfers to feed the key and the input buffer to the Peripheral.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Once the DMA transfers are finished (indicated by hhash->State set back to HAL_HASH_STATE_READY), HAL_HASH_SHA1_Finish() API must be called to retrieve the computed digest.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- If MDMAT bit is set before calling this function (multi-buffer HASH processing case), the input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. For the processing of the last buffer of the thread, MDMAT bit must be reset and the buffer length (in bytes) doesn't have to be a multiple of 4.

HAL_HMAC_MD5_Start_DMA

Function name

HAL_StatusTypeDef HAL_HMAC_MD5_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

Initialize the HASH peripheral in HMAC MD5 mode then initiate the required DMA transfers to feed the key and the input buffer to the Peripheral.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Once the DMA transfers are finished (indicated by hhash->State set back to HAL_HASH_STATE_READY), HAL_HASH_MD5_Finish() API must be called to retrieve the computed digest.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- If MDMAT bit is set before calling this function (multi-buffer HASH processing case), the input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. For the processing of the last buffer of the thread, MDMAT bit must be reset and the buffer length (in bytes) doesn't have to be a multiple of 4.

HAL_HASH_GetState

Function name

HAL_HASH_StateTypeDef HAL_HASH_GetState (HASH_HandleTypeDef * hhash)

Function description

Return the HASH handle state.

Parameters

- **hhash:** HASH handle.

Return values

- **HAL:** HASH state

Notes

- The API yields the current state of the handle (BUSY, READY,...).

HAL_HASH_GetStatus

Function name

HAL_StatusTypeDef HAL_HASH_GetStatus (HASH_HandleTypeDef * hhash)

Function description

Return the HASH HAL status.

Parameters

- **hhash:** HASH handle.

Return values

- **HAL:** status

Notes

- The API yields the HAL status of the handle: it is the result of the latest HASH processing and allows to report any issue (e.g. HAL_TIMEOUT).

HAL_HASH_ContextSaving

Function name

void HAL_HASH_ContextSaving (HASH_HandleTypeDef * hhash, uint8_t * pMemBuffer)

Function description

Save the HASH context in case of processing suspension.

Parameters

- **hhash:** HASH handle.
- **pMemBuffer:** pointer to the memory buffer where the HASH context is saved.

Return values

- **None:**

Notes

- The IMR, STR, CR then all the CSR registers are saved in that order. Only the r/w bits are read to be restored later on.
- By default, all the context swap registers (there are HASH_NUMBER_OF_CSR_REGISTERS of those) are saved.
- pMemBuffer points to a buffer allocated by the user. The buffer size must be at least (HASH_NUMBER_OF_CSR_REGISTERS + 3) * 4 uint8 long.

HAL_HASH_ContextRestoring

Function name

void HAL_HASH_ContextRestoring (HASH_HandleTypeDef * hhash, uint8_t * pMemBuffer)

Function description

Restore the HASH context in case of processing resumption.

Parameters

- **hhash:** HASH handle.
- **pMemBuffer:** pointer to the memory buffer where the HASH context is stored.

Return values

- **None:**

Notes

- The IMR, STR, CR then all the CSR registers are restored in that order. Only the r/w bits are restored.
- By default, all the context swap registers (HASH_NUMBER_OF_CSR_REGISTERS of those) are restored (all of them have been saved by default beforehand).

HAL_HASH_SwFeed_ProcessSuspend

Function name

void HAL_HASH_SwFeed_ProcessSuspend (HASH_HandleTypeDef * hhash)

Function description

Initiate HASH processing suspension when in polling or interruption mode.

Parameters

- **hhash:** HASH handle.

Return values

- **None:**

Notes

- Set the handle field SuspendRequest to the appropriate value so that the on-going HASH processing is suspended as soon as the required conditions are met. Note that the actual suspension is carried out by the functions HASH_WriteData() in polling mode and HASH_IT() in interruption mode.

HAL_HASH_DMAFeed_ProcessSuspend

Function name

HAL_StatusTypeDef HAL_HASH_DMAFeed_ProcessSuspend (HASH_HandleTypeDef * hhash)

Function description

Suspend the HASH processing when in DMA mode.

Parameters

- **hhash:** HASH handle.

Return values

- **HAL:** status

Notes

- When suspension attempt occurs at the very end of a DMA transfer and all the data have already been entered in the Peripheral, hhash->State is set to HAL_HASH_STATE_READY and the API returns HAL_ERROR. It is recommended to wrap-up the processing in reading the digest as usual.

HAL_HASH_GetError

Function name

uint32_t HAL_HASH_GetError (HASH_HandleTypeDef * hhash)

Function description

Return the HASH handle error code.

Parameters

- **hhash:** pointer to a HASH_HandleTypeDef structure.

Return values

- **HASH:** Error Code

HASH_Start

Function name

HAL_StatusTypeDef HASH_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout, uint32_t Algorithm)

Function description

Initialize the HASH peripheral, next process pInBuffer then read the computed digest.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest.
- **Timeout:** Timeout value.
- **Algorithm:** HASH algorithm.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HASH_Accumulate

Function name

HAL_StatusTypeDef HASH_Accumulate (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint32_t Algorithm)

Function description

If not already done, initialize the HASH peripheral then processes pInBuffer.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes, must be a multiple of 4.
- **Algorithm:** HASH algorithm.

Return values

- **HAL:** status

Notes

- Field hhash->Phase of HASH handle is tested to check whether or not the Peripheral has already been initialized.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HASH_Accumulate_IT

Function name

HAL_StatusTypeDef HASH_Accumulate_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint32_t Algorithm)

Function description

If not already done, initialize the HASH peripheral then processes pInBuffer in interruption mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes, must be a multiple of 4.
- **Algorithm:** HASH algorithm.

Return values

- **HAL:** status

Notes

- Field hhash->Phase of HASH handle is tested to check whether or not the Peripheral has already been initialized.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HASH_Start_IT

Function name

HAL_StatusTypeDef HASH_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Algorithm)

Function description

Initialize the HASH peripheral, next process pInBuffer then read the computed digest in interruption mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest.
- **Algorithm:** HASH algorithm.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HASH_Start_DMA

Function name

HAL_StatusTypeDef HASH_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint32_t Algorithm)

Function description

Initialize the HASH peripheral then initiate a DMA transfer to feed the input buffer to the Peripheral.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **Algorithm:** HASH algorithm.

Return values

- **HAL:** status

Notes

- If MDMAT bit is set before calling this function (multi-buffer HASH processing case), the input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. For the processing of the last buffer of the thread, MDMAT bit must be reset and the buffer length (in bytes) doesn't have to be a multiple of 4.

HASH_Finish

Function name

HAL_StatusTypeDef HASH_Finish (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

Return the computed digest.

Parameters

- **hhash:** HASH handle.
- **pOutBuffer:** pointer to the computed digest.
- **Timeout:** Timeout value.

Return values

- **HAL:** status

Notes

- The API waits for DCIS to be set then reads the computed digest.

HMAC_Start

Function name

HAL_StatusTypeDef HMAC_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout, uint32_t Algorithm)

Function description

Initialize the HASH peripheral in HMAC mode, next process pInBuffer then read the computed digest.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest.
- **Timeout:** Timeout value.
- **Algorithm:** HASH algorithm.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HMAC_Start_IT

Function name

HAL_StatusTypeDef HMAC_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Algorithm)

Function description

Initialize the HASH peripheral in HMAC mode, next process pInBuffer then read the computed digest in interruption mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest.
- **Algorithm:** HASH algorithm.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HMAC_Start_DMA

Function name

HAL_StatusTypeDef HMAC_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint32_t Algorithm)

Function description

Initialize the HASH peripheral in HMAC mode then initiate the required DMA transfers to feed the key and the input buffer to the Peripheral.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **Algorithm:** HASH algorithm.

Return values

- **HAL:** status

Notes

- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- In case of multi-buffer HMAC processing, the input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only the length of the last buffer of the thread doesn't have to be a multiple of 4.

37.3 HASH Firmware driver defines

The following section lists the various define and macros of the module.

37.3.1 HASH

HASH

HASH algorithm mode

HASH_ALGOMODE_HASH

Algorithm is HASH

HASH_ALGOMODE_HMAC

Algorithm is HMAC

HASH algorithm selection

HASH_ALGOSELECTION_SHA1

HASH function is SHA1

HASH_ALGOSELECTION_MD5

HASH function is MD5

HASH_ALGOSELECTION_SHA224

HASH function is SHA224

HASH_ALGOSELECTION_SHA256

HASH function is SHA256

HASH input data type

HASH_DATATYPE_32B

32-bit data. No swapping

HASH_DATATYPE_16B

16-bit data. Each half word is swapped

HASH_DATATYPE_8B

8-bit data. All bytes are swapped

HASH_DATATYPE_1B

1-bit data. In the word all bits are swapped

HASH Digest Calculation Status

HASH_DIGEST_CALCULATION_NOT_STARTED

DCAL not set after input data written in DIN register

HASH_DIGEST_CALCULATION_STARTED

DCAL set after input data written in DIN register

HASH DMA suspension words limit

HASH_DMA_SUSPENSION_WORDS_LIMIT

Number of words below which DMA suspension is aborted

HASH Error Definition

HAL_HASH_ERROR_NONE

No error

HAL_HASH_ERROR_IT

IT-based process error

HAL_HASH_ERROR_DMA

DMA-based process error

HAL_HASH_ERROR_INVALID_CALLBACK

Invalid Callback error

HASH Exported Macros

__HAL_HASH_GET_FLAG

Description:

- Check whether or not the specified HASH flag is set.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - HASH_FLAG_DINIS A new block can be entered into the input buffer.
  - HASH_FLAG_DCIS Digest calculation complete.
  - HASH_FLAG_DMAS DMA interface is enabled (DMAE=1) or a transfer is ongoing.
  - HASH_FLAG_BUSY The hash core is Busy : processing a block of data.
  - HASH_FLAG_DINNE DIN not empty : the input buffer contains at least one word of data.

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_HASH_CLEAR_FLAG

Description:

- Clear the specified HASH flag.

Parameters:

- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
  - HASH_FLAG_DINIS A new block can be entered into the input buffer.
  - HASH_FLAG_DCIS Digest calculation complete

Return value:

- None

__HAL_HASH_ENABLE_IT

Description:

- Enable the specified HASH interrupt.

Parameters:

- __INTERRUPT__: specifies the HASH interrupt source to enable. This parameter can be one of the following values:
  - HASH_IT_DINI A new block can be entered into the input buffer (DIN)
  - HASH_IT_DCI Digest calculation complete

Return value:

- None

__HAL_HASH_DISABLE_IT

Description:

- Disable the specified HASH interrupt.

Parameters:

- __INTERRUPT__: specifies the HASH interrupt source to disable. This parameter can be one of the following values:
  - HASH_IT_DINI A new block can be entered into the input buffer (DIN)
  - HASH_IT_DCI Digest calculation complete

Return value:

- None

__HAL_HASH_RESET_HANDLE_STATE

Description:

- Reset HASH handle state.

Parameters:

- __HANDLE__: HASH handle.

Return value:

- None

__HAL_HASH_RESET_HANDLE_STATUS

Description:

- Reset HASH handle status.

Parameters:

- __HANDLE__: HASH handle.

Return value:

- None

__HAL_HASH_SET_MDMAT

Description:

- Enable the multi-buffer DMA transfer mode.

Return value:

- None

Notes:

- This bit is set when hashing large files when multiple DMA transfers are needed.

__HAL_HASH_RESET_MDMAT

Description:

- Disable the multi-buffer DMA transfer mode.

Return value:

- None

__HAL_HASH_START_DIGEST

Description:

- Start the digest computation.

Return value:

- None

__HAL_HASH_SET_NBVALIDBITS

Description:

- Set the number of valid bits in the last word written in data register DIN.

Parameters:

- __SIZE__: size in bytes of last data written in Data register.

Return value:

- None

__HAL_HASH_INIT

Description:

- Reset the HASH core.

Return value:

- None

HASH flags definitions

HASH_FLAG_DINIS

16 locations are free in the DIN : a new block can be entered in the Peripheral

HASH_FLAG_DCIS

Digest calculation complete

HASH_FLAG_DMAS

DMA interface is enabled (DMAE=1) or a transfer is ongoing

HASH_FLAG_BUSY

The hash core is Busy, processing a block of data

HASH_FLAG_DINNE

DIN not empty : the input buffer contains at least one word of data

HMAC key length type

HASH_HMAC_KEYTYPE_SHORTKEY

HMAC Key size is <= 64 bytes

HASH_HMAC_KEYTYPE_LONGKEY

HMAC Key size is > 64 bytes

HASH interrupts definitions

HASH_IT_DINI

A new block can be entered into the input buffer (DIN)

HASH_IT_DCI

Digest calculation complete

HASH Number of Context Swap Registers

HASH_NUMBER_OF_CSR_REGISTERS

Number of Context Swap Registers

HASH TimeOut Value

HASH_TIMEOUTVALUE

Time-out value

38 HAL HASH Extension Driver

38.1 HASHEx Firmware driver API description

The following section lists the various functions of the HASHEx library.

38.1.1 HASH peripheral extended features

The SHA-224 and SHA-256 HASH and HMAC processing can be carried out exactly the same way as for SHA-1 or MD-5 algorithms.

1. Three modes are available.
   a. Polling mode: processing APIs are blocking functions i.e. they process the data and wait till the digest computation is finished, e.g. HAL_HASHEx_xxx_Start()
   b. Interrupt mode: processing APIs are not blocking functions i.e. they process the data under interrupt, e.g. HAL_HASHEx_xxx_Start_IT()
   c. DMA mode: processing APIs are not blocking functions and the CPU is not used for data transfer i.e. the data transfer is ensured by DMA, e.g. HAL_HASHEx_xxx_Start_DMA(). Note that in DMA mode, a call to HAL_HASHEx_xxx_Finish() is then required to retrieve the digest.
2. Multi-buffer processing is possible in polling, interrupt and DMA modes.
   a. In polling mode, only multi-buffer HASH processing is possible. API HAL_HASHEx_xxx_Accumulate() must be called for each input buffer, except for the last one. User must resort to HAL_HASHEx_xxx_Accumulate_End() to enter the last one and retrieve as well the computed digest.
   b. In interrupt mode, API HAL_HASHEx_xxx_Accumulate_IT() must be called for each input buffer, except for the last one. User must resort to HAL_HASHEx_xxx_Accumulate_End_IT() to enter the last one and retrieve as well the computed digest.
   c. In DMA mode, multi-buffer HASH and HMAC processing are possible.
      - HASH processing: once initialization is done, MDMAT bit must be set through __HAL_HASH_SET_MDMAT() macro. From that point, each buffer can be fed to the Peripheral through HAL_HASHEx_xxx_Start_DMA() API. Before entering the last buffer, reset the MDMAT bit with __HAL_HASH_RESET_MDMAT() macro then wrap-up the HASH processing in feeding the last input buffer through the same API HAL_HASHEx_xxx_Start_DMA(). The digest can then be retrieved with a call to API HAL_HASHEx_xxx_Finish().
      - HMAC processing (MD-5, SHA-1, SHA-224 and SHA-256 must all resort to extended functions): after initialization, the key and the first input buffer are entered in the Peripheral with the API HAL_HMACEx_xxx_Step1_2_DMA(). This carries out HMAC step 1 and starts step 2. The following buffers are next entered with the API HAL_HMACEx_xxx_Step2_DMA(). At this point, the HMAC processing is still carrying out step 2. Then, step 2 for the last input buffer and step 3 are carried out by a single call to HAL_HMACEx_xxx_Step2_3_DMA(). The digest can finally be retrieved with a call to API HAL_HASH_xxx_Finish() for MD-5 and SHA-1, to HAL_HASHEx_xxx_Finish() for SHA-224 and SHA-256.

38.1.2 Polling mode HASH extended processing functions

This section provides functions allowing to calculate in polling mode the hash value using one of the following algorithms:

- SHA224
  - HAL_HASHEx_SHA224_Start()
  - HAL_HASHEx_SHA224_Accmlt()
  - HAL_HASHEx_SHA224_Accmlt_End()
- SHA256
  - HAL_HASHEx_SHA256_Start()
  - HAL_HASHEx_SHA256_Accmlt()
  - HAL_HASHEx_SHA256_Accmlt_End()

For a single buffer to be hashed, user can resort to HAL_HASH_xxx_Start().

In case of multi-buffer HASH processing (a single digest is computed while several buffers are fed to the Peripheral), the user can resort to successive calls to HAL_HASHEx_xxx_Accumulate() and wrap-up the digest computation by a call to HAL_HASHEx_xxx_Accumulate_End().

This section contains the following APIs:

- ***HAL_HASHEx_SHA224_Start()***
- ***HAL_HASHEx_SHA224_Accmlt()***
- ***HAL_HASHEx_SHA224_Accmlt_End()***
- ***HAL_HASHEx_SHA256_Start()***
- ***HAL_HASHEx_SHA256_Accmlt()***
- ***HAL_HASHEx_SHA256_Accmlt_End()***

38.1.3 Interruption mode HASH extended processing functions

This section provides functions allowing to calculate in interrupt mode the hash value using one of the following algorithms:

- SHA224
  - HAL_HASHEx_SHA224_Start_IT()
  - HAL_HASHEx_SHA224_Accmlt_IT()
  - HAL_HASHEx_SHA224_Accmlt_End_IT()
- SHA256
  - HAL_HASHEx_SHA256_Start_IT()
  - HAL_HASHEx_SHA256_Accmlt_IT()
  - HAL_HASHEx_SHA256_Accmlt_End_IT()

This section contains the following APIs:

- ***HAL_HASHEx_SHA224_Start_IT()***
- ***HAL_HASHEx_SHA224_Accmlt_IT()***
- ***HAL_HASHEx_SHA224_Accmlt_End_IT()***
- ***HAL_HASHEx_SHA256_Start_IT()***
- ***HAL_HASHEx_SHA256_Accmlt_IT()***
- ***HAL_HASHEx_SHA256_Accmlt_End_IT()***

38.1.4 DMA mode HASH extended processing functions

This section provides functions allowing to calculate in DMA mode the hash value using one of the following algorithms:

- SHA224
  - HAL_HASHEx_SHA224_Start_DMA()
  - HAL_HASHEx_SHA224_Finish()
- SHA256
  - HAL_HASHEx_SHA256_Start_DMA()
  - HAL_HASHEx_SHA256_Finish()

When resorting to DMA mode to enter the data in the Peripheral, user must resort to HAL_HASHEx_xxx_Start_DMA() then read the resulting digest with HAL_HASHEx_xxx_Finish().

In case of multi-buffer HASH processing, MDMAT bit must first be set before the successive calls to HAL_HASHEx_xxx_Start_DMA(). Then, MDMAT bit needs to be reset before the last call to HAL_HASHEx_xxx_Start_DMA(). Digest is finally retrieved thanks to HAL_HASHEx_xxx_Finish().

This section contains the following APIs:

- ***HAL_HASHEx_SHA224_Start_DMA()***
- ***HAL_HASHEx_SHA224_Finish()***
- ***HAL_HASHEx_SHA256_Start_DMA()***
- ***HAL_HASHEx_SHA256_Finish()***

38.1.5 Polling mode HMAC extended processing functions

This section provides functions allowing to calculate in polling mode the HMAC value using one of the following algorithms:

- SHA224
  - HAL_HMACEx_SHA224_Start()
- SHA256
  - HAL_HMACEx_SHA256_Start()

This section contains the following APIs:

- ***HAL_HMACEx_SHA224_Start()***
- ***HAL_HMACEx_SHA256_Start()***

38.1.6 Interrupt mode HMAC extended processing functions

This section provides functions allowing to calculate in interrupt mode the HMAC value using one of the following algorithms:

- SHA224
  - HAL_HMACEx_SHA224_Start_IT()
- SHA256
  - HAL_HMACEx_SHA256_Start_IT()

This section contains the following APIs:

- ***HAL_HMACEx_SHA224_Start_IT()***
- ***HAL_HMACEx_SHA256_Start_IT()***

38.1.7 DMA mode HMAC extended processing functions

This section provides functions allowing to calculate in DMA mode the HMAC value using one of the following algorithms:

- SHA224
  - HAL_HMACEx_SHA224_Start_DMA()
- SHA256
  - HAL_HMACEx_SHA256_Start_DMA()

When resorting to DMA mode to enter the data in the Peripheral for HMAC processing, user must resort to HAL_HMACEx_xxx_Start_DMA() then read the resulting digest with HAL_HASHEx_xxx_Finish().

This section contains the following APIs:

- ***HAL_HMACEx_SHA224_Start_DMA()***
- ***HAL_HMACEx_SHA256_Start_DMA()***

38.1.8 Multi-buffer DMA mode HMAC extended processing functions

This section provides functions to manage HMAC multi-buffer DMA-based processing for MD5, SHA1, SHA224 and SHA256 algorithms.

- MD5
  - HAL_HMACEx_MD5_Step1_2_DMA()
  - HAL_HMACEx_MD5_Step2_DMA()
  - HAL_HMACEx_MD5_Step2_3_DMA()
- SHA1
  - HAL_HMACEx_SHA1_Step1_2_DMA()
  - HAL_HMACEx_SHA1_Step2_DMA()
  - HAL_HMACEx_SHA1_Step2_3_DMA()

- SHA256
  - HAL_HMACEx_SHA224_Step1_2_DMA()
  - HAL_HMACEx_SHA224_Step2_DMA()
  - HAL_HMACEx_SHA224_Step2_3_DMA()
- SHA256
  - HAL_HMACEx_SHA256_Step1_2_DMA()
  - HAL_HMACEx_SHA256_Step2_DMA()
  - HAL_HMACEx_SHA256_Step2_3_DMA()

User must first start-up the multi-buffer DMA-based HMAC computation in calling HAL_HMACEx_xxx_Step1_2_DMA(). This carries out HMAC step 1 and intiates step 2 with the first input buffer.

The following buffers are next fed to the Peripheral with a call to the API HAL_HMACEx_xxx_Step2_DMA(). There may be several consecutive calls to this API.

Multi-buffer DMA-based HMAC computation is wrapped up by a call to HAL_HMACEx_xxx_Step2_3_DMA(). This finishes step 2 in feeding the last input buffer to the Peripheral then carries out step 3.

Digest is retrieved by a call to HAL_HASH_xxx_Finish() for MD-5 or SHA-1, to HAL_HASHEx_xxx_Finish() for SHA-224 or SHA-256.

If only two buffers need to be consecutively processed, a call to HAL_HMACEx_xxx_Step1_2_DMA() followed by a call to HAL_HMACEx_xxx_Step2_3_DMA() is sufficient.

This section contains the following APIs:

- ***HAL_HMACEx_MD5_Step1_2_DMA()***
- ***HAL_HMACEx_MD5_Step2_DMA()***
- ***HAL_HMACEx_MD5_Step2_3_DMA()***
- ***HAL_HMACEx_SHA1_Step1_2_DMA()***
- ***HAL_HMACEx_SHA1_Step2_DMA()***
- ***HAL_HMACEx_SHA1_Step2_3_DMA()***
- ***HAL_HMACEx_SHA224_Step1_2_DMA()***
- ***HAL_HMACEx_SHA224_Step2_DMA()***
- ***HAL_HMACEx_SHA224_Step2_3_DMA()***
- ***HAL_HMACEx_SHA256_Step1_2_DMA()***
- ***HAL_HMACEx_SHA256_Step2_DMA()***
- ***HAL_HMACEx_SHA256_Step2_3_DMA()***

38.1.9 Detailed description of functions

HAL_HASHEx_SHA224_Start

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA224_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

Initialize the HASH peripheral in SHA224 mode, next process pInBuffer then read the computed digest.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 28 bytes.
- **Timeout:** Timeout value

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASHEx_SHA224_Accmlt

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA224_Accmlt (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

If not already done, initialize the HASH peripheral in SHA224 mode then processes pInBuffer.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes, must be a multiple of 4.

Return values

- **HAL:** status

Notes

- Consecutive calls to HAL_HASHEx_SHA224_Accmlt() can be used to feed several input buffers back-to-back to the Peripheral that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL_HASHEx_SHA224_Accmlt_End().
- Field hhash->Phase of HASH handle is tested to check whether or not the Peripheral has already been initialized.
- Digest is not retrieved by this API, user must resort to HAL_HASHEx_SHA224_Accmlt_End() to read it, feeding at the same time the last input buffer to the Peripheral.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only HAL_HASHEx_SHA224_Accmlt_End() is able to manage the ending buffer with a length in bytes not a multiple of 4.

HAL_HASHEx_SHA224_Accmlt_End

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA224_Accmlt_End (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

End computation of a single HASH signature after several calls to HAL_HASHEx_SHA224_Accmlt() API.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 28 bytes.
- **Timeout:** Timeout value

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASHEx_SHA256_Start

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA256_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

Initialize the HASH peripheral in SHA256 mode, next process pInBuffer then read the computed digest.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 32 bytes.
- **Timeout:** Timeout value

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASHEx_SHA256_Accmlt

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA256_Accmlt (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

If not already done, initialize the HASH peripheral in SHA256 mode then processes pInBuffer.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes, must be a multiple of 4.

Return values

- **HAL:** status

Notes

- Consecutive calls to HAL_HASHEx_SHA256_Accmlt() can be used to feed several input buffers back-to-back to the Peripheral that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL_HASHEx_SHA256_Accmlt_End().
- Field hhash->Phase of HASH handle is tested to check whether or not the Peripheral has already been initialized.
- Digest is not retrieved by this API, user must resort to HAL_HASHEx_SHA256_Accmlt_End() to read it, feeding at the same time the last input buffer to the Peripheral.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only HAL_HASHEx_SHA256_Accmlt_End() is able to manage the ending buffer with a length in bytes not a multiple of 4.

HAL_HASHEx_SHA256_Accmlt_End

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA256_Accmlt_End (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

End computation of a single HASH signature after several calls to HAL_HASHEx_SHA256_Accmlt() API.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 32 bytes.
- **Timeout:** Timeout value

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASHEx_SHA224_Start_IT

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA224_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)

Function description

Initialize the HASH peripheral in SHA224 mode, next process pInBuffer then read the computed digest in interruption mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 28 bytes.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASHEx_SHA224_Accmlt_IT

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA224_Accmlt_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

If not already done, initialize the HASH peripheral in SHA224 mode then processes pInBuffer in interruption mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes, must be a multiple of 4.

Return values

- **HAL:** status

Notes

- Consecutive calls to HAL_HASHEx_SHA224_Accmlt_IT() can be used to feed several input buffers back-to-back to the Peripheral that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL_HASHEx_SHA224_Accmlt_End_IT().
- Field hhash->Phase of HASH handle is tested to check whether or not the Peripheral has already been initialized.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only HAL_HASHEx_SHA224_Accmlt_End_IT() is able to manage the ending buffer with a length in bytes not a multiple of 4.

HAL_HASHEx_SHA224_Accmlt_End_IT

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA224_Accmlt_End_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)

Function description

End computation of a single HASH signature after several calls to HAL_HASHEx_SHA224_Accmlt_IT() API.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 28 bytes.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASHEx_SHA256_Start_IT

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA256_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)

Function description

Initialize the HASH peripheral in SHA256 mode, next process pInBuffer then read the computed digest in interruption mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 32 bytes.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASHEx_SHA256_Accmlt_IT

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA256_Accmlt_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

If not already done, initialize the HASH peripheral in SHA256 mode then processes pInBuffer in interruption mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes, must be a multiple of 4.

Return values

- **HAL:** status

Notes

- Consecutive calls to HAL_HASHEx_SHA256_Accmlt_IT() can be used to feed several input buffers back-to-back to the Peripheral that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL_HASHEx_SHA256_Accmlt_End_IT().
- Field hhash->Phase of HASH handle is tested to check whether or not the Peripheral has already been initialized.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only HAL_HASHEx_SHA256_Accmlt_End_IT() is able to manage the ending buffer with a length in bytes not a multiple of 4.

HAL_HASHEx_SHA256_Accmlt_End_IT

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA256_Accmlt_End_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)

Function description

End computation of a single HASH signature after several calls to HAL_HASHEx_SHA256_Accmlt_IT() API.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 32 bytes.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASHEx_SHA224_Start_DMA

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA224_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

Initialize the HASH peripheral in SHA224 mode then initiate a DMA transfer to feed the input buffer to the Peripheral.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Once the DMA transfer is finished, HAL_HASHEx_SHA224_Finish() API must be called to retrieve the computed digest.

HAL_HASHEx_SHA224_Finish

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA224_Finish (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

Return the computed digest in SHA224 mode.

Parameters

- **hhash:** HASH handle.
- **pOutBuffer:** pointer to the computed digest. Digest size is 28 bytes.
- **Timeout:** Timeout value.

Return values

- **HAL:** status

Notes

- The API waits for DCIS to be set then reads the computed digest.
- HAL_HASHEx_SHA224_Finish() can be used as well to retrieve the digest in HMAC SHA224 mode.

HAL_HASHEx_SHA256_Start_DMA

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA256_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

Initialize the HASH peripheral in SHA256 mode then initiate a DMA transfer to feed the input buffer to the Peripheral.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Once the DMA transfer is finished, HAL_HASHEx_SHA256_Finish() API must be called to retrieve the computed digest.

HAL_HASHEx_SHA256_Finish

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA256_Finish (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

Return the computed digest in SHA256 mode.

Parameters

- **hhash:** HASH handle.
- **pOutBuffer:** pointer to the computed digest. Digest size is 32 bytes.
- **Timeout:** Timeout value.

Return values

- **HAL:** status

Notes

- The API waits for DCIS to be set then reads the computed digest.
- HAL_HASHEx_SHA256_Finish() can be used as well to retrieve the digest in HMAC SHA256 mode.

HAL_HMACEx_SHA224_Start

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA224_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

Initialize the HASH peripheral in HMAC SHA224 mode, next process pInBuffer then read the computed digest.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 28 bytes.
- **Timeout:** Timeout value.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HAL_HMACEx_SHA256_Start

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA256_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

Initialize the HASH peripheral in HMAC SHA256 mode, next process pInBuffer then read the computed digest.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 32 bytes.
- **Timeout:** Timeout value.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HAL_HMACEx_SHA224_Start_IT

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA224_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)

Function description

Initialize the HASH peripheral in HMAC SHA224 mode, next process pInBuffer then read the computed digest in interrupt mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 28 bytes.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HAL_HMACEx_SHA256_Start_IT

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA256_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)

Function description

Initialize the HASH peripheral in HMAC SHA256 mode, next process pInBuffer then read the computed digest in interrupt mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 32 bytes.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HAL_HMACEx_SHA224_Start_DMA

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA224_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

Initialize the HASH peripheral in HMAC SHA224 mode then initiate the required DMA transfers to feed the key and the input buffer to the Peripheral.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Once the DMA transfers are finished (indicated by hhash->State set back to HAL_HASH_STATE_READY), HAL_HASHEx_SHA224_Finish() API must be called to retrieve the computed digest.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- If MDMAT bit is set before calling this function (multi-buffer HASH processing case), the input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. For the processing of the last buffer of the thread, MDMAT bit must be reset and the buffer length (in bytes) doesn't have to be a multiple of 4.

HAL_HMACEx_SHA256_Start_DMA

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA256_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

Initialize the HASH peripheral in HMAC SHA224 mode then initiate the required DMA transfers to feed the key and the input buffer to the Peripheral.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Once the DMA transfers are finished (indicated by hhash->State set back to HAL_HASH_STATE_READY), HAL_HASHEx_SHA256_Finish() API must be called to retrieve the computed digest.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- If MDMAT bit is set before calling this function (multi-buffer HASH processing case), the input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. For the processing of the last buffer of the thread, MDMAT bit must be reset and the buffer length (in bytes) doesn't have to be a multiple of 4.

HAL_HMACEx_MD5_Step1_2_DMA

Function name

HAL_StatusTypeDef HAL_HMACEx_MD5_Step1_2_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

MD5 HMAC step 1 completion and step 2 start in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 1 consists in writing the inner hash function key in the Peripheral, step 2 consists in writing the message text.
- The API carries out the HMAC step 1 then starts step 2 with the first buffer entered to the Peripheral. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HAL_HMACEx_MD5_Step2_DMA

Function name

HAL_StatusTypeDef HAL_HMACEx_MD5_Step2_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

MD5 HMAC step 2 in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 2 consists in writing the message text in the Peripheral.
- The API carries on the HMAC step 2, applied to the buffer entered as input parameter. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HAL_HMACEx_MD5_Step2_3_DMA

Function name

HAL_StatusTypeDef HAL_HMACEx_MD5_Step2_3_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

MD5 HMAC step 2 wrap-up and step 3 completion in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 2 consists in writing the message text in the Peripheral, step 3 consists in writing the outer hash function key.
- The API wraps up the HMAC step 2 in processing the buffer entered as input parameter (the input buffer must be the last one of the multi-buffer thread) then carries out HMAC step 3.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- Once the DMA transfers are finished (indicated by hhash->State set back to HAL_HASH_STATE_READY), HAL_HASHEx_SHA256_Finish() API must be called to retrieve the computed digest.

HAL_HMACEx_SHA1_Step1_2_DMA

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA1_Step1_2_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

SHA1 HMAC step 1 completion and step 2 start in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 1 consists in writing the inner hash function key in the Peripheral, step 2 consists in writing the message text.
- The API carries out the HMAC step 1 then starts step 2 with the first buffer entered to the Peripheral. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HAL_HMACEx_SHA1_Step2_DMA

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA1_Step2_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

SHA1 HMAC step 2 in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 2 consists in writing the message text in the Peripheral.
- The API carries on the HMAC step 2, applied to the buffer entered as input parameter. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HAL_HMACEx_SHA1_Step2_3_DMA

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA1_Step2_3_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

SHA1 HMAC step 2 wrap-up and step 3 completion in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 2 consists in writing the message text in the Peripheral, step 3 consists in writing the outer hash function key.
- The API wraps up the HMAC step 2 in processing the buffer entered as input parameter (the input buffer must be the last one of the multi-buffer thread) then carries out HMAC step 3.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- Once the DMA transfers are finished (indicated by hhash->State set back to HAL_HASH_STATE_READY), HAL_HASHEx_SHA256_Finish() API must be called to retrieve the computed digest.

HAL_HMACEx_SHA224_Step1_2_DMA

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA224_Step1_2_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

SHA224 HMAC step 1 completion and step 2 start in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 1 consists in writing the inner hash function key in the Peripheral, step 2 consists in writing the message text.
- The API carries out the HMAC step 1 then starts step 2 with the first buffer entered to the Peripheral. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HAL_HMACEx_SHA224_Step2_DMA

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA224_Step2_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

SHA224 HMAC step 2 in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 2 consists in writing the message text in the Peripheral.
- The API carries on the HMAC step 2, applied to the buffer entered as input parameter. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HAL_HMACEx_SHA224_Step2_3_DMA

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA224_Step2_3_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

SHA224 HMAC step 2 wrap-up and step 3 completion in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 2 consists in writing the message text in the Peripheral, step 3 consists in writing the outer hash function key.
- The API wraps up the HMAC step 2 in processing the buffer entered as input parameter (the input buffer must be the last one of the multi-buffer thread) then carries out HMAC step 3.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- Once the DMA transfers are finished (indicated by hhash->State set back to HAL_HASH_STATE_READY), HAL_HASHEx_SHA256_Finish() API must be called to retrieve the computed digest.

HAL_HMACEx_SHA256_Step1_2_DMA

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA256_Step1_2_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

SHA256 HMAC step 1 completion and step 2 start in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 1 consists in writing the inner hash function key in the Peripheral, step 2 consists in writing the message text.
- The API carries out the HMAC step 1 then starts step 2 with the first buffer entered to the Peripheral. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HAL_HMACEx_SHA256_Step2_DMA

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA256_Step2_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

SHA256 HMAC step 2 in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 2 consists in writing the message text in the Peripheral.
- The API carries on the HMAC step 2, applied to the buffer entered as input parameter. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HAL_HMACEx_SHA256_Step2_3_DMA

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA256_Step2_3_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

SHA256 HMAC step 2 wrap-up and step 3 completion in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 2 consists in writing the message text in the Peripheral, step 3 consists in writing the outer hash function key.
- The API wraps up the HMAC step 2 in processing the buffer entered as input parameter (the input buffer must be the last one of the multi-buffer thread) then carries out HMAC step 3.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- Once the DMA transfers are finished (indicated by hhash->State set back to HAL_HASH_STATE_READY), HAL_HASHEx_SHA256_Finish() API must be called to retrieve the computed digest.

39 HAL HCD Generic Driver

39.1 HCD Firmware driver registers structures

39.1.1 __HCD_HandleTypeDef

__HCD_HandleTypeDef is defined in the stm32h7xx_hal_hcd.h

Data Fields

- ***HCD_TypeDef * Instance***
- ***HCD_InitTypeDef Init***
- ***HCD_HCTypeDef hc***
- ***HAL_LockTypeDef Lock***
- ***__IO HCD_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void * pData***
- ***void(* SOFCallback***
- ***void(* ConnectCallback***
- ***void(* DisconnectCallback***
- ***void(* PortEnabledCallback***
- ***void(* PortDisabledCallback***
- ***void(* HC_NotifyURBChangeCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***HCD_TypeDef* __HCD_HandleTypeDef::Instance***
  Register base address
- ***HCD_InitTypeDef __HCD_HandleTypeDef::Init***
  HCD required parameters
- ***HCD_HCTypeDef __HCD_HandleTypeDef::hc[16]***
  Host channels parameters
- ***HAL_LockTypeDef __HCD_HandleTypeDef::Lock***
  HCD peripheral status
- ***__IO HCD_StateTypeDef __HCD_HandleTypeDef::State***
  HCD communication state
- ***__IO uint32_t __HCD_HandleTypeDef::ErrorCode***
  HCD Error code
- ***void* __HCD_HandleTypeDef::pData***
  Pointer Stack Handler
- ***void(* __HCD_HandleTypeDef::SOFCallback)(struct __HCD_HandleTypeDef *hhcd)***
  USB OTG HCD SOF callback
- ***void(* __HCD_HandleTypeDef::ConnectCallback)(struct __HCD_HandleTypeDef *hhcd)***
  USB OTG HCD Connect callback
- ***void(* __HCD_HandleTypeDef::DisconnectCallback)(struct __HCD_HandleTypeDef *hhcd)***
  USB OTG HCD Disconnect callback
- ***void(* __HCD_HandleTypeDef::PortEnabledCallback)(struct __HCD_HandleTypeDef *hhcd)***
  USB OTG HCD Port Enable callback

- ***void(* __HCD_HandleTypeDef::PortDisabledCallback)(struct __HCD_HandleTypeDef *hhcd)***
  USB OTG HCD Port Disable callback
- ***void(* __HCD_HandleTypeDef::HC_NotifyURBChangeCallback)(struct __HCD_HandleTypeDef *hhcd, uint8_t chnum, HCD_URBStateTypeDef urb_state)***
  USB OTG HCD Host Channel Notify URB Change callback
- ***void(* __HCD_HandleTypeDef::MspInitCallback)(struct __HCD_HandleTypeDef *hhcd)***
  USB OTG HCD Msp Init callback
- ***void(* __HCD_HandleTypeDef::MspDeInitCallback)(struct __HCD_HandleTypeDef *hhcd)***
  USB OTG HCD Msp DeInit callback

39.2 HCD Firmware driver API description

The following section lists the various functions of the HCD library.

39.2.1 How to use this driver

1. Declare a HCD_HandleTypeDef handle structure, for example: HCD_HandleTypeDef hhcd;
2. Fill parameters of Init structure in HCD handle
3. Call HAL_HCD_Init() API to initialize the HCD peripheral (Core, Host core, ...)
4. Initialize the HCD low level resources through the HAL_HCD_MspInit() API:
   a. Enable the HCD/USB Low Level interface clock using the following macros
      - __HAL_RCC_USB_OTG_FS_CLK_ENABLE();
      - __HAL_RCC_USB_OTG_HS_CLK_ENABLE(); (For High Speed Mode)
      - __HAL_RCC_USB_OTG_HS_ULPI_CLK_ENABLE(); (For High Speed Mode)
   b. Initialize the related GPIO clocks
   c. Configure HCD pin-out
   d. Configure HCD NVIC interrupt
5. Associate the Upper USB Host stack to the HAL HCD Driver:
   a. hhcd.pData = phost;
6. Enable HCD transmission and reception:
   a. HAL_HCD_Start();

39.2.2 Initialization and de-initialization functions

This section provides functions allowing to:
This section contains the following APIs:

- ***HAL_HCD_Init()***
- ***HAL_HCD_HC_Init()***
- ***HAL_HCD_HC_Halt()***
- ***HAL_HCD_DeInit()***
- ***HAL_HCD_MspInit()***
- ***HAL_HCD_MspDeInit()***
- ***HAL_HCD_RegisterCallback()***
- ***HAL_HCD_UnRegisterCallback()***
- ***HAL_HCD_RegisterHC_NotifyURBChangeCallback()***
- ***HAL_HCD_UnRegisterHC_NotifyURBChangeCallback()***

39.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the USB Host Data Transfer
This section contains the following APIs:

- ***HAL_HCD_HC_SubmitRequest()***

- *HAL_HCD_IRQHandler()*
- *HAL_HCD_SOF_Callback()*
- *HAL_HCD_Connect_Callback()*
- *HAL_HCD_Disconnect_Callback()*
- *HAL_HCD_PortEnabled_Callback()*
- *HAL_HCD_PortDisabled_Callback()*
- *HAL_HCD_HC_NotifyURBChange_Callback()*
- *HAL_HCD_RegisterCallback()*
- *HAL_HCD_UnRegisterCallback()*
- *HAL_HCD_RegisterHC_NotifyURBChangeCallback()*
- *HAL_HCD_UnRegisterHC_NotifyURBChangeCallback()*

39.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the HCD data transfers.
This section contains the following APIs:

- *HAL_HCD_Start()*
- *HAL_HCD_Stop()*
- *HAL_HCD_ResetPort()*

39.2.5 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.
This section contains the following APIs:

- *HAL_HCD_GetState()*
- *HAL_HCD_HC_GetURBState()*
- *HAL_HCD_HC_GetXferCount()*
- *HAL_HCD_HC_GetState()*
- *HAL_HCD_GetCurrentFrame()*
- *HAL_HCD_GetCurrentSpeed()*

39.2.6 Detailed description of functions

HAL_HCD_Init

Function name

HAL_StatusTypeDef HAL_HCD_Init (HCD_HandleTypeDef * hhcd)

Function description

Initialize the host driver.

Parameters

- **hhcd:** HCD handle

Return values

- **HAL:** status

HAL_HCD_DeInit

Function name

HAL_StatusTypeDef HAL_HCD_DeInit (HCD_HandleTypeDef * hhcd)

Function description

DeInitialize the host driver.

Parameters

- **hhcd:** HCD handle

Return values

- **HAL:** status

HAL_HCD_HC_Init

Function name

HAL_StatusTypeDef HAL_HCD_HC_Init (HCD_HandleTypeDef * hhcd, uint8_t ch_num, uint8_t epnum, uint8_t dev_address, uint8_t speed, uint8_t ep_type, uint16_t mps)

Function description

Initialize a host channel.

Parameters

- **hhcd:** HCD handle
- **ch_num:** Channel number. This parameter can be a value from 1 to 15
- **epnum:** Endpoint number. This parameter can be a value from 1 to 15
- **dev_address:** Current device address This parameter can be a value from 0 to 255
- **speed:** Current device speed. This parameter can be one of these values: HCD_DEVICE_SPEED_HIGH: High speed mode, HCD_DEVICE_SPEED_FULL: Full speed mode, HCD_DEVICE_SPEED_LOW: Low speed mode
- **ep_type:** Endpoint Type. This parameter can be one of these values: EP_TYPE_CTRL: Control type, EP_TYPE_ISOC: Isochronous type, EP_TYPE_BULK: Bulk type, EP_TYPE_INTR: Interrupt type
- **mps:** Max Packet Size. This parameter can be a value from 0 to32K

Return values

- **HAL:** status

HAL_HCD_HC_Halt

Function name

HAL_StatusTypeDef HAL_HCD_HC_Halt (HCD_HandleTypeDef * hhcd, uint8_t ch_num)

Function description

Halt a host channel.

Parameters

- **hhcd:** HCD handle
- **ch_num:** Channel number. This parameter can be a value from 1 to 15

Return values

- **HAL:** status

HAL_HCD_MspInit

Function name

void HAL_HCD_MspInit (HCD_HandleTypeDef * hhcd)

Function description

Initialize the HCD MSP.

Parameters

- **hhcd:** HCD handle

Return values

- **None:**

HAL_HCD_MspDeInit

Function name

void HAL_HCD_MspDeInit (HCD_HandleTypeDef * hhcd)

Function description

DeInitialize the HCD MSP.

Parameters

- **hhcd:** HCD handle

Return values

- **None:**

HAL_HCD_RegisterCallback

Function name

HAL_StatusTypeDef HAL_HCD_RegisterCallback (HCD_HandleTypeDef * hhcd, HAL_HCD_CallbackIDTypeDef CallbackID, pHCD_CallbackTypeDef pCallback)

Function description

Register a User USB HCD Callback To be used instead of the weak predefined callback.

Parameters

- **hhcd:** USB HCD handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_HCD_SOF_CB_ID USB HCD SOF callback ID
  - HAL_HCD_CONNECT_CB_ID USB HCD Connect callback ID
  - HAL_HCD_DISCONNECT_CB_ID OTG HCD Disconnect callback ID
  - HAL_HCD_PORT_ENABLED_CB_ID USB HCD Port Enable callback ID
  - HAL_HCD_PORT_DISABLED_CB_ID USB HCD Port Disable callback ID
  - HAL_HCD_MSPINIT_CB_ID MspDeInit callback ID
  - HAL_HCD_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_HCD_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_HCD_UnRegisterCallback (HCD_HandleTypeDef * hhcd, HAL_HCD_CallbackIDTypeDef CallbackID)

Function description

Unregister an USB HCD Callback USB HCD callback is redirected to the weak predefined callback.

Parameters

- **hhcd:** USB HCD handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_HCD_SOF_CB_ID USB HCD SOF callback ID
  - HAL_HCD_CONNECT_CB_ID USB HCD Connect callback ID
  - HAL_HCD_DISCONNECT_CB_ID OTG HCD Disconnect callback ID
  - HAL_HCD_PORT_ENABLED_CB_ID USB HCD Port Enabled callback ID
  - HAL_HCD_PORT_DISABLED_CB_ID USB HCD Port Disabled callback ID
  - HAL_HCD_MSPINIT_CB_ID MspDeInit callback ID
  - HAL_HCD_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_HCD_RegisterHC_NotifyURBChangeCallback

Function name

HAL_StatusTypeDef HAL_HCD_RegisterHC_NotifyURBChangeCallback (HCD_HandleTypeDef * hhcd, pHCD_HC_NotifyURBChangeCallbackTypeDef pCallback)

Function description

Register USB HCD Host Channel Notify URB Change Callback To be used instead of the weak HAL_HCD_HC_NotifyURBChange_Callback() predefined callback.

Parameters

- **hhcd:** HCD handle
- **pCallback:** pointer to the USB HCD Host Channel Notify URB Change Callback function

Return values

- **HAL:** status

HAL_HCD_UnRegisterHC_NotifyURBChangeCallback

Function name

HAL_StatusTypeDef HAL_HCD_UnRegisterHC_NotifyURBChangeCallback (HCD_HandleTypeDef * hhcd)

Function description

Unregister the USB HCD Host Channel Notify URB Change Callback USB HCD Host Channel Notify URB Change Callback is redirected to the weak HAL_HCD_HC_NotifyURBChange_Callback() predefined callback.

Parameters

- **hhcd:** HCD handle

Return values

- **HAL:** status

HAL_HCD_HC_SubmitRequest

Function name

HAL_StatusTypeDef HAL_HCD_HC_SubmitRequest (HCD_HandleTypeDef * hhcd, uint8_t ch_num, uint8_t direction, uint8_t ep_type, uint8_t token, uint8_t * pbuff, uint16_t length, uint8_t do_ping)

Function description

Submit a new URB for processing.

Parameters

- **hhcd:** HCD handle
- **ch_num:** Channel number. This parameter can be a value from 1 to 15
- **direction:** Channel number. This parameter can be one of these values: 0 : Output / 1 : Input
- **ep_type:** Endpoint Type. This parameter can be one of these values: EP_TYPE_CTRL: Control type/ EP_TYPE_ISOC: Isochronous type/ EP_TYPE_BULK: Bulk type/ EP_TYPE_INTR: Interrupt type/
- **token:** Endpoint Type. This parameter can be one of these values: 0: HC_PID_SETUP / 1: HC_PID_DATA1
- **pbuff:** pointer to URB data
- **length:** Length of URB data
- **do_ping:** activate do ping protocol (for high speed only). This parameter can be one of these values: 0 : do ping inactive / 1 : do ping active

Return values

- **HAL:** status

HAL_HCD_IRQHandler

Function name

void HAL_HCD_IRQHandler (HCD_HandleTypeDef * hhcd)

Function description

Handle HCD interrupt request.

Parameters

- **hhcd:** HCD handle

Return values

- **None:**

HAL_HCD_SOF_Callback

Function name

void HAL_HCD_SOF_Callback (HCD_HandleTypeDef * hhcd)

Function description

SOF callback.

Parameters

- **hhcd:** HCD handle

Return values

- **None:**

HAL_HCD_Connect_Callback

Function name

void HAL_HCD_Connect_Callback (HCD_HandleTypeDef * hhcd)

Function description

Connection Event callback.

Parameters

- **hhcd:** HCD handle

Return values

- **None:**

HAL_HCD_Disconnect_Callback

Function name

void HAL_HCD_Disconnect_Callback (HCD_HandleTypeDef * hhcd)

Function description

Disconnection Event callback.

Parameters

- **hhcd:** HCD handle

Return values

- **None:**

HAL_HCD_PortEnabled_Callback

Function name

void HAL_HCD_PortEnabled_Callback (HCD_HandleTypeDef * hhcd)

Function description

Port Enabled Event callback.

Parameters

- **hhcd:** HCD handle

Return values

- **None:**

HAL_HCD_PortDisabled_Callback

Function name

void HAL_HCD_PortDisabled_Callback (HCD_HandleTypeDef * hhcd)

Function description

Port Disabled Event callback.

Parameters

- **hhcd:** HCD handle

Return values

- **None:**

HAL_HCD_HC_NotifyURBChange_Callback

Function name

void HAL_HCD_HC_NotifyURBChange_Callback (HCD_HandleTypeDef * hhcd, uint8_t chnum, HCD_URBStateTypeDef urb_state)

Function description

Notify URB state change callback.

Parameters

- **hhcd:** HCD handle
- **chnum:** Channel number. This parameter can be a value from 1 to 15
- **urb_state:** This parameter can be one of these values: URB_IDLE/ URB_DONE/ URB_NOTREADY/ URB_NYET/ URB_ERROR/ URB_STALL/

Return values

- **None:**

HAL_HCD_ResetPort

Function name

HAL_StatusTypeDef HAL_HCD_ResetPort (HCD_HandleTypeDef * hhcd)

Function description

Reset the host port.

Parameters

- **hhcd:** HCD handle

Return values

- **HAL:** status

HAL_HCD_Start

Function name

HAL_StatusTypeDef HAL_HCD_Start (HCD_HandleTypeDef * hhcd)

Function description

Start the host driver.

Parameters

- **hhcd:** HCD handle

Return values

- **HAL:** status

HAL_HCD_Stop

Function name

HAL_StatusTypeDef HAL_HCD_Stop (HCD_HandleTypeDef * hhcd)

Function description

Stop the host driver.

Parameters

- **hhcd:** HCD handle

Return values

- **HAL:** status

HAL_HCD_GetState

Function name

HCD_StateTypeDef HAL_HCD_GetState (HCD_HandleTypeDef * hhcd)

Function description

Return the HCD handle state.

Parameters

- **hhcd:** HCD handle

Return values

- **HAL:** state

HAL_HCD_HC_GetURBState

Function name

HCD_URBStateTypeDef HAL_HCD_HC_GetURBState (HCD_HandleTypeDef * hhcd, uint8_t chnum)

Function description

Return URB state for a channel.

Parameters

- **hhcd:** HCD handle
- **chnum:** Channel number. This parameter can be a value from 1 to 15

Return values

- **URB:** state. This parameter can be one of these values: URB_IDLE/ URB_DONE/ URB_NOTREADY/ URB_NYET/ URB_ERROR/ URB_STALL

HAL_HCD_HC_GetState

Function name

HCD_HCStateTypeDef HAL_HCD_HC_GetState (HCD_HandleTypeDef * hhcd, uint8_t chnum)

Function description

Return the Host Channel state.

Parameters

- **hhcd:** HCD handle
- **chnum:** Channel number. This parameter can be a value from 1 to 15

Return values

- **Host:** channel state This parameter can be one of these values: HC_IDLE/ HC_XFRC/ HC_HALTED/ HC_NYET/ HC_NAK/ HC_STALL/ HC_XACTERR/ HC_BBLERR/ HC_DATATGLERR

HAL_HCD_HC_GetXferCount

Function name

uint32_t HAL_HCD_HC_GetXferCount (HCD_HandleTypeDef * hhcd, uint8_t chnum)

Function description

Return the last host transfer size.

Parameters

- **hhcd:** HCD handle
- **chnum:** Channel number. This parameter can be a value from 1 to 15

Return values

- **last:** transfer size in byte

HAL_HCD_GetCurrentFrame

Function name

uint32_t HAL_HCD_GetCurrentFrame (HCD_HandleTypeDef * hhcd)

Function description

Return the current Host frame number.

Parameters

- **hhcd:** HCD handle

Return values

- **Current:** Host frame number

HAL_HCD_GetCurrentSpeed

Function name

uint32_t HAL_HCD_GetCurrentSpeed (HCD_HandleTypeDef * hhcd)

Function description

Return the Host enumeration speed.

Parameters

- **hhcd:** HCD handle

Return values

- **Enumeration:** speed

39.3 HCD Firmware driver defines

The following section lists the various define and macros of the module.

39.3.1 HCD

HCD

HCD Device Speed

HCD_DEVICE_SPEED_HIGH

HCD_DEVICE_SPEED_FULL

HCD_DEVICE_SPEED_LOW

HCD Error Code definition

HAL_HCD_ERROR_INVALID_CALLBACK

Invalid Callback error

HCD Exported Macros

__HAL_HCD_ENABLE

__HAL_HCD_DISABLE

__HAL_HCD_GET_FLAG

__HAL_HCD_GET_CH_FLAG

__HAL_HCD_CLEAR_FLAG

__HAL_HCD_IS_INVALID_INTERRUPT

__HAL_HCD_CLEAR_HC_INT

__HAL_HCD_MASK_HALT_HC_INT

__HAL_HCD_UNMASK_HALT_HC_INT

__HAL_HCD_MASK_ACK_HC_INT

__HAL_HCD_UNMASK_ACK_HC_INT

HCD PHY Module

HCD_PHY_ULPI

HCD_PHY_EMBEDDED

HCD Speed

HCD_SPEED_HIGH

HCD_SPEED_FULL

HCD_SPEED_LOW

40 HAL HRTIM Generic Driver

40.1 HRTIM Firmware driver registers structures

40.1.1 HRTIM_InitTypeDef

HRTIM_InitTypeDef is defined in the stm32h7xx_hal_hrtim.h

Data Fields

- ***uint32_t HRTIMInterruptResquests***
- ***uint32_t SyncOptions***
- ***uint32_t SyncInputSource***
- ***uint32_t SyncOutputSource***
- ***uint32_t SyncOutputPolarity***

Field Documentation

- ***uint32_t HRTIM_InitTypeDef::HRTIMInterruptResquests***

  Specifies which interrupts requests must enabled for the HRTIM instance. This parameter can be any combination of ***HRTIM_Common_Interrupt_Enable***
- ***uint32_t HRTIM_InitTypeDef::SyncOptions***

  Specifies how the HRTIM instance handles the external synchronization signals. The HRTIM instance can be configured to act as a slave (waiting for a trigger to be synchronized) or a master (generating a synchronization signal) or both. This parameter can be a combination of ***HRTIM_Synchronization_Options***.
- ***uint32_t HRTIM_InitTypeDef::SyncInputSource***

  Specifies the external synchronization input source (significant only when the HRTIM instance is configured as a slave). This parameter can be a value of ***HRTIM_Synchronization_Input_Source***.
- ***uint32_t HRTIM_InitTypeDef::SyncOutputSource***

  Specifies the source and event to be sent on the external synchronization outputs (significant only when the HRTIM instance is configured as a master). This parameter can be a value of ***HRTIM_Synchronization_Output_Source***
- ***uint32_t HRTIM_InitTypeDef::SyncOutputPolarity***

  Specifies the conditioning of the event to be sent on the external synchronization outputs (significant only when the HRTIM instance is configured as a master). This parameter can be a value of ***HRTIM_Synchronization_Output_Polarity***

40.1.2 HRTIM_TimerParamTypeDef

HRTIM_TimerParamTypeDef is defined in the stm32h7xx_hal_hrtim.h

Data Fields

- ***uint32_t CaptureTrigger1***
- ***uint32_t CaptureTrigger2***
- ***uint32_t InterruptRequests***
- ***uint32_t DMARequests***
- ***uint32_t DMASrcAddress***
- ***uint32_t DMADstAddress***
- ***uint32_t DMASize***

Field Documentation

- ***uint32_t HRTIM_TimerParamTypeDef::CaptureTrigger1***

  Event(s) triggering capture unit 1. When the timer operates in Simple mode, this parameter can be a value of ***HRTIM_External_Event_Channels***. When the timer operates in Waveform mode, this parameter can be a combination of ***HRTIM_Capture_Unit_Trigger***.

- ***uint32_t HRTIM_TimerParamTypeDef::CaptureTrigger2***
  Event(s) triggering capture unit 2. When the timer operates in Simple mode, this parameter can be a value of ***HRTIM_External_Event_Channels***. When the timer operates in Waveform mode, this parameter can be a combination of ***HRTIM_Capture_Unit_Trigger***.
- ***uint32_t HRTIM_TimerParamTypeDef::InterruptRequests***
  Interrupts requests enabled for the timer.
- ***uint32_t HRTIM_TimerParamTypeDef::DMARequests***
  DMA requests enabled for the timer.
- ***uint32_t HRTIM_TimerParamTypeDef::DMASrcAddress***
  Address of the source address of the DMA transfer.
- ***uint32_t HRTIM_TimerParamTypeDef::DMADstAddress***
  Address of the destination address of the DMA transfer.
- ***uint32_t HRTIM_TimerParamTypeDef::DMASize***
  Size of the DMA transfer

40.1.3 __HRTIM_HandleTypeDef

__HRTIM_HandleTypeDef is defined in the stm32h7xx_hal_hrtim.h

Data Fields

- ***HRTIM_TypeDef * Instance***
- ***HRTIM_InitTypeDef Init***
- ***HRTIM_TimerParamTypeDef TimerParam***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_HRTIM_StateTypeDef State***
- ***DMA_HandleTypeDef * hdmaMaster***
- ***DMA_HandleTypeDef * hdmaTimerA***
- ***DMA_HandleTypeDef * hdmaTimerB***
- ***DMA_HandleTypeDef * hdmaTimerC***
- ***DMA_HandleTypeDef * hdmaTimerD***
- ***DMA_HandleTypeDef * hdmaTimerE***
- ***void(* Fault1Callback***
- ***void(* Fault2Callback***
- ***void(* Fault3Callback***
- ***void(* Fault4Callback***
- ***void(* Fault5Callback***
- ***void(* SystemFaultCallback***
- ***void(* BurstModePeriodCallback***
- ***void(* SynchronizationEventCallback***
- ***void(* ErrorCallback***
- ***void(* RegistersUpdateCallback***
- ***void(* RepetitionEventCallback***
- ***void(* Compare1EventCallback***
- ***void(* Compare2EventCallback***
- ***void(* Compare3EventCallback***
- ***void(* Compare4EventCallback***
- ***void(* Capture1EventCallback***
- ***void(* Capture2EventCallback***
- ***void(* DelayedProtectionCallback***
- ***void(* CounterResetCallback***
- ***void(* Output1SetCallback***
- ***void(* Output1ResetCallback***

- ***void(* Output2SetCallback***
- ***void(* Output2ResetCallback***
- ***void(* BurstDMATransferCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***HRTIM_TypeDef* __HRTIM_HandleTypeDef::Instance***
  Register base address
- ***HRTIM_InitTypeDef __HRTIM_HandleTypeDef::Init***
  HRTIM required parameters
- ***HRTIM_TimerParamTypeDef __HRTIM_HandleTypeDef::TimerParam[MAX_HRTIM_TIMER]***
  HRTIM timers - including the master - parameters
- ***HAL_LockTypeDef __HRTIM_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_HRTIM_StateTypeDef __HRTIM_HandleTypeDef::State***
  HRTIM communication state
- ***DMA_HandleTypeDef* __HRTIM_HandleTypeDef::hdmaMaster***
  Master timer DMA handle parameters
- ***DMA_HandleTypeDef* __HRTIM_HandleTypeDef::hdmaTimerA***
  Timer A DMA handle parameters
- ***DMA_HandleTypeDef* __HRTIM_HandleTypeDef::hdmaTimerB***
  Timer B DMA handle parameters
- ***DMA_HandleTypeDef* __HRTIM_HandleTypeDef::hdmaTimerC***
  Timer C DMA handle parameters
- ***DMA_HandleTypeDef* __HRTIM_HandleTypeDef::hdmaTimerD***
  Timer D DMA handle parameters
- ***DMA_HandleTypeDef* __HRTIM_HandleTypeDef::hdmaTimerE***
  Timer E DMA handle parameters
- ***void(* __HRTIM_HandleTypeDef::Fault1Callback)(struct __HRTIM_HandleTypeDef *hhrtim)***
  Fault 1 interrupt callback function pointer
- ***void(* __HRTIM_HandleTypeDef::Fault2Callback)(struct __HRTIM_HandleTypeDef *hhrtim)***
  Fault 2 interrupt callback function pointer
- ***void(* __HRTIM_HandleTypeDef::Fault3Callback)(struct __HRTIM_HandleTypeDef *hhrtim)***
  Fault 3 interrupt callback function pointer
- ***void(* __HRTIM_HandleTypeDef::Fault4Callback)(struct __HRTIM_HandleTypeDef *hhrtim)***
  Fault 4 interrupt callback function pointer
- ***void(* __HRTIM_HandleTypeDef::Fault5Callback)(struct __HRTIM_HandleTypeDef *hhrtim)***
  Fault 5 interrupt callback function pointer
- ***void(* __HRTIM_HandleTypeDef::SystemFaultCallback)(struct __HRTIM_HandleTypeDef *hhrtim)***
  System fault interrupt callback function pointer
- ***void(* __HRTIM_HandleTypeDef::BurstModePeriodCallback)(struct __HRTIM_HandleTypeDef *hhrtim)***
  Burst mode period interrupt callback function pointer
- ***void(* __HRTIM_HandleTypeDef::SynchronizationEventCallback)(struct __HRTIM_HandleTypeDef *hhrtim)***
  Sync Input interrupt callback function pointer
- ***void(* __HRTIM_HandleTypeDef::ErrorCallback)(struct __HRTIM_HandleTypeDef *hhrtim)***
  DMA error callback function pointer

- ***void(* __HRTIM_HandleTypeDef::RegistersUpdateCallback)(struct __HRTIM_HandleTypeDef *hhrtim, uint32_t TimerIdx)***
  Timer x Update interrupt callback function pointer
- ***void(* __HRTIM_HandleTypeDef::RepetitionEventCallback)(struct __HRTIM_HandleTypeDef *hhrtim, uint32_t TimerIdx)***
  Timer x Repetition interrupt callback function pointer
- ***void(* __HRTIM_HandleTypeDef::Compare1EventCallback)(struct __HRTIM_HandleTypeDef *hhrtim, uint32_t TimerIdx)***
  Timer x Compare 1 match interrupt callback function pointer
- ***void(* __HRTIM_HandleTypeDef::Compare2EventCallback)(struct __HRTIM_HandleTypeDef *hhrtim, uint32_t TimerIdx)***
  Timer x Compare 2 match interrupt callback function pointer
- ***void(* __HRTIM_HandleTypeDef::Compare3EventCallback)(struct __HRTIM_HandleTypeDef *hhrtim, uint32_t TimerIdx)***
  Timer x Compare 3 match interrupt callback function pointer
- ***void(* __HRTIM_HandleTypeDef::Compare4EventCallback)(struct __HRTIM_HandleTypeDef *hhrtim, uint32_t TimerIdx)***
  Timer x Compare 4 match interrupt callback function pointer
- ***void(* __HRTIM_HandleTypeDef::Capture1EventCallback)(struct __HRTIM_HandleTypeDef *hhrtim, uint32_t TimerIdx)***
  Timer x Capture 1 interrupts callback function pointer
- ***void(* __HRTIM_HandleTypeDef::Capture2EventCallback)(struct __HRTIM_HandleTypeDef *hhrtim, uint32_t TimerIdx)***
  Timer x Capture 2 interrupts callback function pointer
- ***void(* __HRTIM_HandleTypeDef::DelayedProtectionCallback)(struct __HRTIM_HandleTypeDef *hhrtim, uint32_t TimerIdx)***
  Timer x Delayed protection interrupt callback function pointer
- ***void(* __HRTIM_HandleTypeDef::CounterResetCallback)(struct __HRTIM_HandleTypeDef *hhrtim, uint32_t TimerIdx)***
  Timer x counter reset/roll-over interrupt callback function pointer
- ***void(* __HRTIM_HandleTypeDef::Output1SetCallback)(struct __HRTIM_HandleTypeDef *hhrtim, uint32_t TimerIdx)***
  Timer x output 1 set interrupt callback function pointer
- ***void(* __HRTIM_HandleTypeDef::Output1ResetCallback)(struct __HRTIM_HandleTypeDef *hhrtim, uint32_t TimerIdx)***
  Timer x output 1 reset interrupt callback function pointer
- ***void(* __HRTIM_HandleTypeDef::Output2SetCallback)(struct __HRTIM_HandleTypeDef *hhrtim, uint32_t TimerIdx)***
  Timer x output 2 set interrupt callback function pointer
- ***void(* __HRTIM_HandleTypeDef::Output2ResetCallback)(struct __HRTIM_HandleTypeDef *hhrtim, uint32_t TimerIdx)***
  Timer x output 2 reset interrupt callback function pointer
- ***void(* __HRTIM_HandleTypeDef::BurstDMATransferCallback)(struct __HRTIM_HandleTypeDef *hhrtim, uint32_t TimerIdx)***
  Timer x Burst DMA completed interrupt callback function pointer
- ***void(* __HRTIM_HandleTypeDef::MspInitCallback)(struct __HRTIM_HandleTypeDef *hhrtim)***
  HRTIM MspInit callback function pointer
- ***void(* __HRTIM_HandleTypeDef::MspDeInitCallback)(struct __HRTIM_HandleTypeDef *hhrtim)***
  HRTIM MspInit callback function pointer

40.1.4 HRTIM_TimeBaseCfgTypeDef

HRTIM_TimeBaseCfgTypeDef is defined in the stm32h7xx_hal_hrtim.h

Data Fields

- ***uint32_t Period***
- ***uint32_t RepetitionCounter***
- ***uint32_t PrescalerRatio***
- ***uint32_t Mode***

Field Documentation

- ***uint32_t HRTIM_TimeBaseCfgTypeDef::Period***

  Specifies the timer period. The period value must be above 3 periods of the fHRTIM clock. Maximum value is = 0xFFDFU
- ***uint32_t HRTIM_TimeBaseCfgTypeDef::RepetitionCounter***

  Specifies the timer repetition period. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- ***uint32_t HRTIM_TimeBaseCfgTypeDef::PrescalerRatio***

  Specifies the timer clock prescaler ratio. This parameter can be any value of ***HRTIM_Prescaler_Ratio***
- ***uint32_t HRTIM_TimeBaseCfgTypeDef::Mode***

  Specifies the counter operating mode. This parameter can be any value of ***HRTIM_Counter_Operating_Mode***

40.1.5 HRTIM_SimpleOCChannelCfgTypeDef

HRTIM_SimpleOCChannelCfgTypeDef is defined in the stm32h7xx_hal_hrtim.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t Pulse***
- ***uint32_t Polarity***
- ***uint32_t IdleLevel***

Field Documentation

- ***uint32_t HRTIM_SimpleOCChannelCfgTypeDef::Mode***

  Specifies the output compare mode (toggle, active, inactive). This parameter can be any value of of ***HRTIM_Simple_OC_Mode***
- ***uint32_t HRTIM_SimpleOCChannelCfgTypeDef::Pulse***

  Specifies the compare value to be loaded into the Compare Register. The compare value must be above or equal to 3 periods of the fHRTIM clock
- ***uint32_t HRTIM_SimpleOCChannelCfgTypeDef::Polarity***

  Specifies the output polarity. This parameter can be any value of ***HRTIM_Output_Polarity***
- ***uint32_t HRTIM_SimpleOCChannelCfgTypeDef::IdleLevel***

  Specifies whether the output level is active or inactive when in IDLE state. This parameter can be any value of ***HRTIM_Output_IDLE_Level***

40.1.6 HRTIM_SimplePWMChannelCfgTypeDef

HRTIM_SimplePWMChannelCfgTypeDef is defined in the stm32h7xx_hal_hrtim.h

Data Fields

- ***uint32_t Pulse***
- ***uint32_t Polarity***
- ***uint32_t IdleLevel***

Field Documentation

- ***uint32_t HRTIM_SimplePWMChannelCfgTypeDef::Pulse***

  Specifies the compare value to be loaded into the Compare Register. The compare value must be above or equal to 3 periods of the fHRTIM clock
- ***uint32_t HRTIM_SimplePWMChannelCfgTypeDef::Polarity***

  Specifies the output polarity. This parameter can be any value of ***HRTIM_Output_Polarity***

- ***uint32_t HRTIM_SimplePWMChannelCfgTypeDef::IdleLevel***

  Specifies whether the output level is active or inactive when in IDLE state. This parameter can be any value of ***HRTIM_Output_IDLE_Level***

40.1.7 HRTIM_SimpleCaptureChannelCfgTypeDef

HRTIM_SimpleCaptureChannelCfgTypeDef is defined in the stm32h7xx_hal_hrtim.h

Data Fields

- ***uint32_t Event***
- ***uint32_t EventPolarity***
- ***uint32_t EventSensitivity***
- ***uint32_t EventFilter***

Field Documentation

- ***uint32_t HRTIM_SimpleCaptureChannelCfgTypeDef::Event***

  Specifies the external event triggering the capture. This parameter can be any 'EEVx' value of ***HRTIM_External_Event_Channels***
- ***uint32_t HRTIM_SimpleCaptureChannelCfgTypeDef::EventPolarity***

  Specifies the polarity of the external event (in case of level sensitivity). This parameter can be a value of ***HRTIM_External_Event_Polarity***
- ***uint32_t HRTIM_SimpleCaptureChannelCfgTypeDef::EventSensitivity***

  Specifies the sensitivity of the external event. This parameter can be a value of ***HRTIM_External_Event_Sensitivity***
- ***uint32_t HRTIM_SimpleCaptureChannelCfgTypeDef::EventFilter***

  Defines the frequency used to sample the External Event and the length of the digital filter. This parameter can be a value of ***HRTIM_External_Event_Filter***

40.1.8 HRTIM_SimpleOnePulseChannelCfgTypeDef

HRTIM_SimpleOnePulseChannelCfgTypeDef is defined in the stm32h7xx_hal_hrtim.h

Data Fields

- ***uint32_t Pulse***
- ***uint32_t OutputPolarity***
- ***uint32_t OutputIdleLevel***
- ***uint32_t Event***
- ***uint32_t EventPolarity***
- ***uint32_t EventSensitivity***
- ***uint32_t EventFilter***

Field Documentation

- ***uint32_t HRTIM_SimpleOnePulseChannelCfgTypeDef::Pulse***

  Specifies the compare value to be loaded into the Compare Register. The compare value must be above or equal to 3 periods of the fHRTIM clock
- ***uint32_t HRTIM_SimpleOnePulseChannelCfgTypeDef::OutputPolarity***

  Specifies the output polarity. This parameter can be any value of ***HRTIM_Output_Polarity***
- ***uint32_t HRTIM_SimpleOnePulseChannelCfgTypeDef::OutputIdleLevel***

  Specifies whether the output level is active or inactive when in IDLE state. This parameter can be any value of ***HRTIM_Output_IDLE_Level***
- ***uint32_t HRTIM_SimpleOnePulseChannelCfgTypeDef::Event***

  Specifies the external event triggering the pulse generation. This parameter can be any 'EEVx' value of ***HRTIM_External_Event_Channels***
- ***uint32_t HRTIM_SimpleOnePulseChannelCfgTypeDef::EventPolarity***

  Specifies the polarity of the external event (in case of level sensitivity). This parameter can be a value of ***HRTIM_External_Event_Polarity***

- ***uint32_t HRTIM_SimpleOnePulseChannelCfgTypeDef::EventSensitivity***

  Specifies the sensitivity of the external event. This parameter can be a value of ***HRTIM_External_Event_Sensitivity***.
- ***uint32_t HRTIM_SimpleOnePulseChannelCfgTypeDef::EventFilter***

  Defines the frequency used to sample the External Event and the length of the digital filter. This parameter can be a value of ***HRTIM_External_Event_Filter***

40.1.9 HRTIM_TimerCfgTypeDef

HRTIM_TimerCfgTypeDef is defined in the stm32h7xx_hal_hrtim.h

Data Fields

- ***uint32_t InterruptRequests***
- ***uint32_t DMARequests***
- ***uint32_t DMASrcAddress***
- ***uint32_t DMADstAddress***
- ***uint32_t DMASize***
- ***uint32_t HalfModeEnable***
- ***uint32_t StartOnSync***
- ***uint32_t ResetOnSync***
- ***uint32_t DACSynchro***
- ***uint32_t PreloadEnable***
- ***uint32_t UpdateGating***
- ***uint32_t BurstMode***
- ***uint32_t RepetitionUpdate***
- ***uint32_t PushPull***
- ***uint32_t FaultEnable***
- ***uint32_t FaultLock***
- ***uint32_t DeadTimeInsertion***
- ***uint32_t DelayedProtectionMode***
- ***uint32_t UpdateTrigger***
- ***uint32_t ResetTrigger***
- ***uint32_t ResetUpdate***

Field Documentation

- ***uint32_t HRTIM_TimerCfgTypeDef::InterruptRequests***

  Relevant for all HRTIM timers, including the master. Specifies which interrupts requests must enabled for the timer. This parameter can be any combination of ***HRTIM_Master_Interrupt_Enable*** or ***HRTIM_Timing_Unit_Interrupt_Enable***
- ***uint32_t HRTIM_TimerCfgTypeDef::DMARequests***

  Relevant for all HRTIM timers, including the master. Specifies which DMA requests must be enabled for the timer. This parameter can be any combination of ***HRTIM_Master_DMA_Request_Enable*** or ***HRTIM_Timing_Unit_DMA_Request_Enable***
- ***uint32_t HRTIM_TimerCfgTypeDef::DMASrcAddress***

  Relevant for all HRTIM timers, including the master. Specifies the address of the source address of the DMA transfer
- ***uint32_t HRTIM_TimerCfgTypeDef::DMADstAddress***

  Relevant for all HRTIM timers, including the master. Specifies the address of the destination address of the DMA transfer
- ***uint32_t HRTIM_TimerCfgTypeDef::DMASize***

  Relevant for all HRTIM timers, including the master. Specifies the size of the DMA transfer
- ***uint32_t HRTIM_TimerCfgTypeDef::HalfModeEnable***

  Relevant for all HRTIM timers, including the master. Specifies whether or not half mode is enabled This parameter can be any value of ***HRTIM_Half_Mode_Enable***

- ***uint32_t HRTIM_TimerCfgTypeDef::StartOnSync***

  Relevant for all HRTIM timers, including the master. Specifies whether or not timer is reset by a rising edge on the synchronization input (when enabled). This parameter can be any value of ***HRTIM_Start_On_Sync_Input_Event***
- ***uint32_t HRTIM_TimerCfgTypeDef::ResetOnSync***

  Relevant for all HRTIM timers, including the master. Specifies whether or not timer is reset by a rising edge on the synchronization input (when enabled). This parameter can be any value of ***HRTIM_Reset_On_Sync_Input_Event***
- ***uint32_t HRTIM_TimerCfgTypeDef::DACSynchro***

  Relevant for all HRTIM timers, including the master. Indicates whether or not the a DAC synchronization event is generated. This parameter can be any value of ***HRTIM_DAC_Synchronization***
- ***uint32_t HRTIM_TimerCfgTypeDef::PreloadEnable***

  Relevant for all HRTIM timers, including the master. Specifies whether or not register preload is enabled. This parameter can be any value of ***HRTIM_Register_Preload_Enable***
- ***uint32_t HRTIM_TimerCfgTypeDef::UpdateGating***

  Relevant for all HRTIM timers, including the master. Specifies how the update occurs with respect to a burst DMA transaction or update enable inputs (Slave timers only). This parameter can be any value of ***HRTIM_Update_Gating***
- ***uint32_t HRTIM_TimerCfgTypeDef::BurstMode***

  Relevant for all HRTIM timers, including the master. Specifies how the timer behaves during a burst mode operation. This parameter can be any value of ***HRTIM_Timer_Burst_Mode***
- ***uint32_t HRTIM_TimerCfgTypeDef::RepetitionUpdate***

  Relevant for all HRTIM timers, including the master. Specifies whether or not registers update is triggered by the repetition event. This parameter can be any value of ***HRTIM_Timer_Repetition_Update***
- ***uint32_t HRTIM_TimerCfgTypeDef::PushPull***

  Relevant for Timer A to Timer E. Specifies whether or not the push-pull mode is enabled. This parameter can be any value of ***HRTIM_Timer_Push_Pull_Mode***
- ***uint32_t HRTIM_TimerCfgTypeDef::FaultEnable***

  Relevant for Timer A to Timer E. Specifies which fault channels are enabled for the timer. This parameter can be a combination of ***HRTIM_Timer_Fault_Enabling***
- ***uint32_t HRTIM_TimerCfgTypeDef::FaultLock***

  Relevant for Timer A to Timer E. Specifies whether or not fault enabling status is write protected. This parameter can be a value of ***HRTIM_Timer_Fault_Lock***
- ***uint32_t HRTIM_TimerCfgTypeDef::DeadTimeInsertion***

  Relevant for Timer A to Timer E. Specifies whether or not dead-time insertion is enabled for the timer. This parameter can be a value of ***HRTIM_Timer_Deadtime_Insertion***
- ***uint32_t HRTIM_TimerCfgTypeDef::DelayedProtectionMode***

  Relevant for Timer A to Timer E. Specifies the delayed protection mode. This parameter can be a value of ***HRTIM_Timer_Delayed_Protection_Mode***
- ***uint32_t HRTIM_TimerCfgTypeDef::UpdateTrigger***

  Relevant for Timer A to Timer E. Specifies source(s) triggering the timer registers update. This parameter can be a combination of ***HRTIM_Timer_Update_Trigger***
- ***uint32_t HRTIM_TimerCfgTypeDef::ResetTrigger***

  Relevant for Timer A to Timer E. Specifies source(s) triggering the timer counter reset. This parameter can be a combination of ***HRTIM_Timer_Reset_Trigger***
- ***uint32_t HRTIM_TimerCfgTypeDef::ResetUpdate***

  Relevant for Timer A to Timer E. Specifies whether or not registers update is triggered when the timer counter is reset. This parameter can be a value of ***HRTIM_Timer_Reset_Update***

40.1.10 HRTIM_CompareCfgTypeDef

HRTIM_CompareCfgTypeDef is defined in the stm32h7xx_hal_hrtim.h

Data Fields

- ***uint32_t CompareValue***

- ***uint32_t AutoDelayedMode***
- ***uint32_t AutoDelayedTimeout***

Field Documentation

- ***uint32_t HRTIM_CompareCfgTypeDef::CompareValue***

  Specifies the compare value of the timer compare unit. The minimum value must be greater than or equal to 3 periods of the fHRTIM clock. The maximum value must be less than or equal to 0xFFFFU - 1 periods of the fHRTIM clock
- ***uint32_t HRTIM_CompareCfgTypeDef::AutoDelayedMode***

  Specifies the auto delayed mode for compare unit 2 or 4. This parameter can be a value of ***HRTIM_Compare_Unit_Auto_Delayed_Mode***
- ***uint32_t HRTIM_CompareCfgTypeDef::AutoDelayedTimeout***

  Specifies compare value for timing unit 1 or 3 when auto delayed mode with time out is selected. CompareValue + AutoDelayedTimeout must be less than 0xFFFFU

40.1.11 HRTIM_CaptureCfgTypeDef

HRTIM_CaptureCfgTypeDef is defined in the stm32h7xx_hal_hrtim.h

Data Fields

- ***uint32_t Trigger***

Field Documentation

- ***uint32_t HRTIM_CaptureCfgTypeDef::Trigger***

  Specifies source(s) triggering the capture. This parameter can be a combination of ***HRTIM_Capture_Unit_Trigger***

40.1.12 HRTIM_OutputCfgTypeDef

HRTIM_OutputCfgTypeDef is defined in the stm32h7xx_hal_hrtim.h

Data Fields

- ***uint32_t Polarity***
- ***uint32_t SetSource***
- ***uint32_t ResetSource***
- ***uint32_t IdleMode***
- ***uint32_t IdleLevel***
- ***uint32_t FaultLevel***
- ***uint32_t ChopperModeEnable***
- ***uint32_t BurstModeEntryDelayed***

Field Documentation

- ***uint32_t HRTIM_OutputCfgTypeDef::Polarity***

  Specifies the output polarity. This parameter can be any value of ***HRTIM_Output_Polarity***
- ***uint32_t HRTIM_OutputCfgTypeDef::SetSource***

  Specifies the event(s) transitioning the output from its inactive level to its active level. This parameter can be a combination of ***HRTIM_Output_Set_Source***
- ***uint32_t HRTIM_OutputCfgTypeDef::ResetSource***

  Specifies the event(s) transitioning the output from its active level to its inactive level. This parameter can be a combination of ***HRTIM_Output_Reset_Source***
- ***uint32_t HRTIM_OutputCfgTypeDef::IdleMode***

  Specifies whether or not the output is affected by a burst mode operation. This parameter can be any value of ***HRTIM_Output_Idle_Mode***
- ***uint32_t HRTIM_OutputCfgTypeDef::IdleLevel***

  Specifies whether the output level is active or inactive when in IDLE state. This parameter can be any value of ***HRTIM_Output_IDLE_Level***

- ***uint32_t HRTIM_OutputCfgTypeDef::FaultLevel***
  Specifies whether the output level is active or inactive when in FAULT state. This parameter can be any value of ***HRTIM_Output_FAULT_Level***
- ***uint32_t HRTIM_OutputCfgTypeDef::ChopperModeEnable***
  Indicates whether or not the chopper mode is enabled This parameter can be any value of ***HRTIM_Output_Chopper_Mode_Enable***
- ***uint32_t HRTIM_OutputCfgTypeDef::BurstModeEntryDelayed***
  Indicates whether or not dead-time is inserted when entering the IDLE state during a burst mode operation. This parameters can be any value of ***HRTIM_Output_Burst_Mode_Entry_Delayed***

40.1.13 HRTIM_TimerEventFilteringCfgTypeDef

HRTIM_TimerEventFilteringCfgTypeDef is defined in the stm32h7xx_hal_hrtim.h

Data Fields

- ***uint32_t Filter***
- ***uint32_t Latch***

Field Documentation

- ***uint32_t HRTIM_TimerEventFilteringCfgTypeDef::Filter***
  Specifies the type of event filtering within the timing unit. This parameter can be a value of ***HRTIM_Timer_External_Event_Filter***
- ***uint32_t HRTIM_TimerEventFilteringCfgTypeDef::Latch***
  Specifies whether or not the signal is latched. This parameter can be a value of ***HRTIM_Timer_External_Event_Latch***

40.1.14 HRTIM_DeadTimeCfgTypeDef

HRTIM_DeadTimeCfgTypeDef is defined in the stm32h7xx_hal_hrtim.h

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t RisingValue***
- ***uint32_t RisingSign***
- ***uint32_t RisingLock***
- ***uint32_t RisingSignLock***
- ***uint32_t FallingValue***
- ***uint32_t FallingSign***
- ***uint32_t FallingLock***
- ***uint32_t FallingSignLock***

Field Documentation

- ***uint32_t HRTIM_DeadTimeCfgTypeDef::Prescaler***
  Specifies the dead-time prescaler. This parameter can be a value of ***HRTIM_Deadtime_Prescaler_Ratio***
- ***uint32_t HRTIM_DeadTimeCfgTypeDef::RisingValue***
  Specifies the dead-time following a rising edge. This parameter can be a number between 0x0 and 0x1FFU
- ***uint32_t HRTIM_DeadTimeCfgTypeDef::RisingSign***
  Specifies whether the dead-time is positive or negative on rising edge. This parameter can be a value of ***HRTIM_Deadtime_Rising_Sign***
- ***uint32_t HRTIM_DeadTimeCfgTypeDef::RisingLock***
  Specifies whether or not dead-time rising settings (value and sign) are write protected. This parameter can be a value of ***HRTIM_Deadtime_Rising_Lock***
- ***uint32_t HRTIM_DeadTimeCfgTypeDef::RisingSignLock***
  Specifies whether or not dead-time rising sign is write protected. This parameter can be a value of ***HRTIM_Deadtime_Rising_Sign_Lock***
- ***uint32_t HRTIM_DeadTimeCfgTypeDef::FallingValue***
  Specifies the dead-time following a falling edge. This parameter can be a number between 0x0 and 0x1FFU

- ***uint32_t HRTIM_DeadTimeCfgTypeDef::FallingSign***

  Specifies whether the dead-time is positive or negative on falling edge. This parameter can be a value of ***HRTIM_Deadtime_Falling_Sign***
- ***uint32_t HRTIM_DeadTimeCfgTypeDef::FallingLock***

  Specifies whether or not dead-time falling settings (value and sign) are write protected. This parameter can be a value of ***HRTIM_Deadtime_Falling_Lock***
- ***uint32_t HRTIM_DeadTimeCfgTypeDef::FallingSignLock***

  Specifies whether or not dead-time falling sign is write protected. This parameter can be a value of ***HRTIM_Deadtime_Falling_Sign_Lock***

40.1.15 HRTIM_ChopperModeCfgTypeDef

HRTIM_ChopperModeCfgTypeDef is defined in the stm32h7xx_hal_hrtim.h

Data Fields

- ***uint32_t CarrierFreq***
- ***uint32_t DutyCycle***
- ***uint32_t StartPulse***

Field Documentation

- ***uint32_t HRTIM_ChopperModeCfgTypeDef::CarrierFreq***

  Specifies the Timer carrier frequency value. This parameter can be a value of ***HRTIM_Chopper_Frequency***
- ***uint32_t HRTIM_ChopperModeCfgTypeDef::DutyCycle***

  Specifies the Timer chopper duty cycle value. This parameter can be a value of ***HRTIM_Chopper_Duty_Cycle***
- ***uint32_t HRTIM_ChopperModeCfgTypeDef::StartPulse***

  Specifies the Timer pulse width value. This parameter can be a value of ***HRTIM_Chopper_Start_Pulse_Width***

40.1.16 HRTIM_EventCfgTypeDef

HRTIM_EventCfgTypeDef is defined in the stm32h7xx_hal_hrtim.h

Data Fields

- ***uint32_t Source***
- ***uint32_t Polarity***
- ***uint32_t Sensitivity***
- ***uint32_t Filter***
- ***uint32_t FastMode***

Field Documentation

- ***uint32_t HRTIM_EventCfgTypeDef::Source***

  Identifies the source of the external event. This parameter can be a value of ***HRTIM_External_Event_Sources***
- ***uint32_t HRTIM_EventCfgTypeDef::Polarity***

  Specifies the polarity of the external event (in case of level sensitivity). This parameter can be a value of ***HRTIM_External_Event_Polarity***
- ***uint32_t HRTIM_EventCfgTypeDef::Sensitivity***

  Specifies the sensitivity of the external event. This parameter can be a value of ***HRTIM_External_Event_Sensitivity***
- ***uint32_t HRTIM_EventCfgTypeDef::Filter***

  Defines the frequency used to sample the External Event and the length of the digital filter. This parameter can be a value of ***HRTIM_External_Event_Filter***
- ***uint32_t HRTIM_EventCfgTypeDef::FastMode***

  Indicates whether or not low latency mode is enabled for the external event. This parameter can be a value of ***HRTIM_External_Event_Fast_Mode***

40.1.17 HRTIM_FaultCfgTypeDef

HRTIM_FaultCfgTypeDef is defined in the stm32h7xx_hal_hrtim.h

Data Fields

- ***uint32_t Source***
- ***uint32_t Polarity***
- ***uint32_t Filter***
- ***uint32_t Lock***

Field Documentation

- ***uint32_t HRTIM_FaultCfgTypeDef::Source***
  Identifies the source of the fault. This parameter can be a value of ***HRTIM_Fault_Sources***
- ***uint32_t HRTIM_FaultCfgTypeDef::Polarity***
  Specifies the polarity of the fault event. This parameter can be a value of ***HRTIM_Fault_Polarity***
- ***uint32_t HRTIM_FaultCfgTypeDef::Filter***
  Defines the frequency used to sample the Fault input and the length of the digital filter. This parameter can be a value of ***HRTIM_Fault_Filter***
- ***uint32_t HRTIM_FaultCfgTypeDef::Lock***
  Indicates whether or not fault programming bits are write protected. This parameter can be a value of ***HRTIM_Fault_Lock***

40.1.18 HRTIM_BurstModeCfgTypeDef

HRTIM_BurstModeCfgTypeDef is defined in the stm32h7xx_hal_hrtim.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t ClockSource***
- ***uint32_t Prescaler***
- ***uint32_t PreloadEnable***
- ***uint32_t Trigger***
- ***uint32_t IdleDuration***
- ***uint32_t Period***

Field Documentation

- ***uint32_t HRTIM_BurstModeCfgTypeDef::Mode***
  Specifies the burst mode operating mode. This parameter can be a value of ***HRTIM_Burst_Mode_Operating_Mode***
- ***uint32_t HRTIM_BurstModeCfgTypeDef::ClockSource***
  Specifies the burst mode clock source. This parameter can be a value of ***HRTIM_Burst_Mode_Clock_Source***
- ***uint32_t HRTIM_BurstModeCfgTypeDef::Prescaler***
  Specifies the burst mode prescaler. This parameter can be a value of ***HRTIM_Burst_Mode_Prescaler***
- ***uint32_t HRTIM_BurstModeCfgTypeDef::PreloadEnable***
  Specifies whether or not preload is enabled for burst mode related registers (HRTIM_BMCMPR and HRTIM_BMPER). This parameter can be a combination of ***HRTIM_Burst_Mode_Register_Preload_Enable***
- ***uint32_t HRTIM_BurstModeCfgTypeDef::Trigger***
  Specifies the event(s) triggering the burst operation. This parameter can be a combination of ***HRTIM_Burst_Mode_Trigger***
- ***uint32_t HRTIM_BurstModeCfgTypeDef::IdleDuration***
  Specifies number of periods during which the selected timers are in idle state. This parameter can be a number between 0x0 and 0xFFFF
- ***uint32_t HRTIM_BurstModeCfgTypeDef::Period***
  Specifies burst mode repetition period. This parameter can be a number between 0x1 and 0xFFFF

40.1.19 HRTIM_ADCTriggerCfgTypeDef

HRTIM_ADCTriggerCfgTypeDef is defined in the stm32h7xx_hal_hrtim.h

Data Fields

- ***uint32_t UpdateSource***
- ***uint32_t Trigger***

Field Documentation

- ***uint32_t HRTIM_ADCTriggerCfgTypeDef::UpdateSource***
  Specifies the ADC trigger update source. This parameter can be a value of ***HRTIM_ADC_Trigger_Update_Source***
- ***uint32_t HRTIM_ADCTriggerCfgTypeDef::Trigger***
  Specifies the event(s) triggering the ADC conversion. This parameter can be a combination of ***HRTIM_ADC_Trigger_Event***

40.2 HRTIM Firmware driver API description

The following section lists the various functions of the HRTIM library.

40.2.1 Simple mode versus waveform mode

The HRTIM HAL API is split into 2 categories:

1. Simple functions: these functions allow for using a HRTIM timer as a general purpose timer with high resolution capabilities. HRTIM simple modes are managed through the set of functions named HAL_HRTIM_Simple<Function>. These functions are similar in name and usage to the one defined for the TIM peripheral. When a HRTIM timer operates in simple mode, only a very limited set of HRTIM features are used. Following simple modes are proposed:
   – Output compare mode,
   – PWM output mode,
   – Input capture mode,
   – One pulse mode.
2. Waveform functions: These functions allow taking advantage of the HRTIM flexibility to produce numerous types of control signal. When a HRTIM timer operates in waveform mode, all the HRTIM features are accessible without any restriction. HRTIM waveform modes are managed through the set of functions named HAL_HRTIM_Waveform<Function>

40.2.2 How to use this driver

1. Initialize the HRTIM low level resources by implementing the HAL_HRTIM_MspInit() function:
   a. Enable the HRTIM clock source using __HRTIMx_CLK_ENABLE()
   b. Connect HRTIM pins to MCU I/Os
      ◦ Enable the clock for the HRTIM GPIOs using the following function: __HAL_RCC_GPIOx_CLK_ENABLE()
      ◦ Configure these GPIO pins in Alternate Function mode using HAL_GPIO_Init()
   c. When using DMA to control data transfer (e.g HAL_HRTIM_SimpleBaseStart_DMA())
      ◦ Enable the DMAx interface clock using __DMAx_CLK_ENABLE()
      ◦ Initialize the DMA handle
      ◦ Associate the initialized DMA handle to the appropriate DMA handle of the HRTIM handle using __HAL_LINKDMA()
      ◦ Initialize the DMA channel using HAL_DMA_Init()
      ◦ Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA channel using HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ()
   d. In case of using interrupt mode (e.g HAL_HRTIM_SimpleBaseStart_IT())
      ◦ Configure the priority and enable the NVIC for the concerned HRTIM interrupt using HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ()

2. Initialize the HRTIM HAL using HAL_HRTIM_Init(). The HRTIM configuration structure (field of the HRTIM handle) specifies which global interrupt of whole HRTIM must be enabled (Burst mode period, System fault, Faults). It also contains the HRTIM external synchronization configuration. HRTIM can act as a master (generating a synchronization signal) or as a slave (waiting for a trigger to be synchronized).
3. Configure HRTIM resources shared by all HRTIM timers
   a. Burst Mode Controller:
      - HAL_HRTIM_BurstModeConfig(): configures the HRTIM burst mode controller: operating mode (continuous or one-shot mode), clock (source, prescaler) , trigger(s), period, idle duration.
   b. External Events Conditioning:
      - HAL_HRTIM_EventConfig(): configures the conditioning of an external event channel: source, polarity, edge-sensitivity. External event can be used as triggers (timer reset, input capture, burst mode, ADC triggers, delayed protection) They can also be used to set or reset timer outputs. Up to 10 event channels are available.
      - HAL_HRTIM_EventPrescalerConfig(): configures the external event sampling clock (used for digital filtering).
   c. Fault Conditioning:
      - HAL_HRTIM_FaultConfig(): configures the conditioning of a fault channel: source, polarity, edge-sensitivity. Fault channels are used to disable the outputs in case of an abnormal operation. Up to 5 fault channels are available.
      - HAL_HRTIM_FaultPrescalerConfig(): configures the fault sampling clock (used for digital filtering).
      - HAL_HRTIM_FaultModeCtl(): Enables or disables fault input(s) circuitry. By default all fault inputs are disabled.
   d. ADC trigger:
      - HAL_HRTIM_ADCTriggerConfig(): configures the source triggering the update of the ADC trigger register and the ADC trigger. 4 independent triggers are available to start both the regular and the injected sequencers of the 2 ADCs
4. Configure HRTIM timer time base using HAL_HRTIM_TimeBaseConfig(). This function must be called whatever the HRTIM timer operating mode is (simple v.s. waveform). It configures mainly:
   a. The HRTIM timer counter operating mode (continuous v.s. one shot)
   b. The HRTIM timer clock prescaler
   c. The HRTIM timer period
   d. The HRTIM timer repetition counter

If the HRTIM timer operates in simple mode

1. Start or Stop simple timers
   – Simple time base: HAL_HRTIM_SimpleBaseStart(),HAL_HRTIM_SimpleBaseStop(), HAL_HRTIM_SimpleBaseStart_IT(),HAL_HRTIM_SimpleBaseStop_IT(), HAL_HRTIM_SimpleBaseStart_DMA(),HAL_HRTIM_SimpleBaseStop_DMA().
   – Simple output compare: HAL_HRTIM_SimpleOCChannelConfig(), HAL_HRTIM_SimpleOCStart(),HAL_HRTIM_SimpleOCStop(), HAL_HRTIM_SimpleOCStart_IT(),HAL_HRTIM_SimpleOCStop_IT(), HAL_HRTIM_SimpleOCStart_DMA(),HAL_HRTIM_SimpleOCStop_DMA(),
   – Simple PWM output: HAL_HRTIM_SimplePWMChannelConfig(), HAL_HRTIM_SimplePWMStart(),HAL_HRTIM_SimplePWMStop(), HAL_HRTIM_SimplePWMStart_IT(),HAL_HRTIM_SimplePWMStop_IT(), HAL_HRTIM_SimplePWMStart_DMA(),HAL_HRTIM_SimplePWMStop_DMA(),
   – Simple input capture: HAL_HRTIM_SimpleCaptureChannelConfig(), HAL_HRTIM_SimpleCaptureStart(),HAL_HRTIM_SimpleCaptureStop(), HAL_HRTIM_SimpleCaptureStart_IT(),HAL_HRTIM_SimpleCaptureStop_IT(), HAL_HRTIM_SimpleCaptureStart_DMA(),HAL_HRTIM_SimpleCaptureStop_DMA().
   – Simple one pulse: HAL_HRTIM_SimpleOnePulseChannelConfig(), HAL_HRTIM_SimpleOnePulseStart(),HAL_HRTIM_SimpleOnePulseStop(), HAL_HRTIM_SimpleOnePulseStart_IT(),HAL_HRTIM_SimpleOnePulseStop_It().

If the HRTIM timer operates in waveform mode

1. Completes waveform timer configuration
   - HAL_HRTIM_WaveformTimerConfig(): configuration of a HRTIM timer operating in wave form mode mainly consists in:
     - Enabling the HRTIM timer interrupts and DMA requests.
     - Enabling the half mode for the HRTIM timer.
     - Defining how the HRTIM timer reacts to external synchronization input.
     - Enabling the push-pull mode for the HRTIM timer.
     - Enabling the fault channels for the HRTIM timer.
     - Enabling the dead-time insertion for the HRTIM timer.
     - Setting the delayed protection mode for the HRTIM timer (source and outputs on which the delayed protection are applied).
     - Specifying the HRTIM timer update and reset triggers.
     - Specifying the HRTIM timer registers update policy (e.g. pre-load enabling).
   - HAL_HRTIM_TimerEventFilteringConfig(): configures external event blanking and windowing circuitry of a HRTIM timer:
     - Blanking: to mask external events during a defined time period a defined time period
     - Windowing, to enable external events only during a defined time period
   - HAL_HRTIM_DeadTimeConfig(): configures the dead-time insertion unit for a HRTIM timer. Allows to generate a couple of complementary signals from a single reference waveform, with programmable delays between active state.
   - HAL_HRTIM_ChopperModeConfig(): configures the parameters of the high-frequency carrier signal added on top of the timing unit output. Chopper mode can be enabled or disabled for each timer output separately (see HAL_HRTIM_WaveformOutputConfig()).
   - HAL_HRTIM_BurstDMAConfig(): configures the burst DMA burst controller. Allows having multiple HRTIM registers updated with a single DMA request. The burst DMA operation is started by calling HAL_HRTIM_BurstDMATransfer().
   - HAL_HRTIM_WaveformCompareConfig():configures the compare unit of a HRTIM timer. This operation consists in setting the compare value and possibly specifying the auto delayed mode for compare units 2 and 4 (allows to have compare events generated relatively to capture events). Note that when auto delayed mode is needed, the capture unit associated to the compare unit must be configured separately.
   - HAL_HRTIM_WaveformCaptureConfig(): configures the capture unit of a HRTIM timer. This operation consists in specifying the source(s) triggering the capture (timer register update event, external event, timer output set/reset event, other HRTIM timer related events).
   - HAL_HRTIM_WaveformOutputConfig(): configuration of a HRTIM timer output mainly consists in:
     - Setting the output polarity (active high or active low),
     - Defining the set/reset crossbar for the output,
     - Specifying the fault level (active or inactive) in IDLE and FAULT states.,
2. Set waveform timer output(s) level
   - HAL_HRTIM_WaveformSetOutputLevel(): forces the output to its active or inactive level. For example, when deadtime insertion is enabled it is necessary to force the output level by software to have the outputs in a complementary state as soon as the RUN mode is entered.
3. Enable or Disable waveform timer output(s)
   - HAL_HRTIM_WaveformOutputStart(),HAL_HRTIM_WaveformOutputStop().
4. Start or Stop waveform HRTIM timer(s).
   - HAL_HRTIM_WaveformCountStart(),HAL_HRTIM_WaveformCountStop(),
   - HAL_HRTIM_WaveformCountStart_IT(),HAL_HRTIM_WaveformCountStop_IT(),
   - HAL_HRTIM_WaveformCountStart_DMA(),HAL_HRTIM_WaveformCountStop_DMA(),
5. Burst mode controller enabling:
   - HAL_HRTIM_BurstModeCtl(): activates or de-activates the burst mode controller.

6. Some HRTIM operations can be triggered by software:
   - HAL_HRTIM_BurstModeSoftwareTrigger(): calling this function trigs the burst operation.
   - HAL_HRTIM_SoftwareCapture(): calling this function trigs the capture of the HRTIM timer counter.
   - HAL_HRTIM_SoftwareUpdate(): calling this function trigs the update of the pre-loadable registers of the HRTIM timer
   - HAL_HRTIM_SoftwareReset():calling this function resets the HRTIM timer counter.
7. Some functions can be used any time to retrieve HRTIM timer related information
   - HAL_HRTIM_GetCapturedValue(): returns actual value of the capture register of the designated capture unit.
   - HAL_HRTIM_WaveformGetOutputLevel(): returns actual level (ACTIVE/INACTIVE) of the designated timer output.
   - HAL_HRTIM_WaveformGetOutputState():returns actual state (IDLE/RUN/FAULT) of the designated timer output.
   - HAL_HRTIM_GetDelayedProtectionStatus():returns actual level (ACTIVE/INACTIVE) of the designated output when the delayed protection was triggered.
   - HAL_HRTIM_GetBurstStatus(): returns the actual status (ACTIVE/INACTIVE) of the burst mode controller.
   - HAL_HRTIM_GetCurrentPushPullStatus(): when the push-pull mode is enabled for the HRTIM timer (see HAL_HRTIM_WaveformTimerConfig()), the push-pull status indicates on which output the signal is currently active (e.g signal applied on output 1 and output 2 forced inactive or vice versa).
   - HAL_HRTIM_GetIdlePushPullStatus(): when the push-pull mode is enabled for the HRTIM timer (see HAL_HRTIM_WaveformTimerConfig()), the idle push-pull status indicates during which period the delayed protection request occurred (e.g. protection occurred when the output 1 was active and output 2 forced inactive or vice versa).
8. Some functions can be used any time to retrieve actual HRTIM status
   - HAL_HRTIM_GetState(): returns actual HRTIM instance HAL state.

Callback registration

The compilation flag USE_HAL_HRTIM_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_HRTIM_RegisterCallback() or HAL_HRTIM_TIMxRegisterCallback() to register an interrupt callback.

Function HAL_HRTIM_RegisterCallback() allows to register following callbacks:

- Fault1Callback : Fault 1 interrupt callback function
- Fault2Callback : Fault 2 interrupt callback function
- Fault3Callback : Fault 3 interrupt callback function
- Fault4Callback : Fault 4 interrupt callback function
- Fault5Callback : Fault 5 interrupt callback function
- SystemFaultCallback : System fault interrupt callback function
- BurstModePeriodCallback : Burst mode period interrupt callback function
- SynchronizationEventCallback : Sync Input interrupt callback function
- ErrorCallback : DMA error callback function
- MspInitCallback : HRTIM MspInit callback function
- MspDeInitCallback : HRTIM MspInit callback function

Function HAL_HRTIM_TIMxRegisterCallback() allows to register following callbacks:

- RegistersUpdateCallback : Timer x Update interrupt callback function
- RepetitionEventCallback : Timer x Repetition interrupt callback function
- Compare1EventCallback : Timer x Compare 1 match interrupt callback function
- Compare2EventCallback : Timer x Compare 2 match interrupt callback function
- Compare3EventCallback : Timer x Compare 3 match interrupt callback function
- Compare4EventCallback : Timer x Compare 4 match interrupt callback function
- Capture1EventCallback : Timer x Capture 1 interrupts callback function
- Capture2EventCallback : Timer x Capture 2 interrupts callback function

- DelayedProtectionCallback : Timer x Delayed protection interrupt callback function
- CounterResetCallback : Timer x counter reset/roll-over interrupt callback function
- Output1SetCallback : Timer x output 1 set interrupt callback function
- Output1ResetCallback : Timer x output 1 reset interrupt callback function
- Output2SetCallback : Timer x output 2 set interrupt callback function
- Output2ResetCallback : Timer x output 2 reset interrupt callback function
- BurstDMATransferCallback : Timer x Burst DMA completed interrupt callback function

Both functions take as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_HRTIM_UnRegisterCallback or HAL_HRTIM_TIMxUnRegisterCallback to reset a callback to the default weak function. Both functions take as parameters the HAL peripheral handle and the Callback ID.

By default, after the HAL_HRTIM_Init() and when the state is HAL_HRTIM_STATE_RESET all callbacks are set to the corresponding weak functions (e.g HAL_HRTIM_Fault1Callback) Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_HRTIM_Init()/ HAL_HRTIM_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the HAL_HRTIM_Init()/ HAL_HRTIM_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in HAL_HRTIM_STATE_READY state only. Exception done MspInit/MspDeInit functions that can be registered/unregistered in HAL_HRTIM_STATE_READY or HAL_HRTIM_STATE_RESET states, thus registered (user) MspInit/DeInit callbacks can be used during the Init/ DeInit. Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_HRTIM_RegisterCallback() before calling HAL_HRTIM_DeInit() or HAL_HRTIM_Init() function.

When the compilation flag USE_HAL_HRTIM_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

40.2.3 Initialization and Time Base Configuration functions

This section provides functions allowing to:

- Initialize a HRTIM instance
- De-initialize a HRTIM instance
- Initialize the HRTIM MSP
- De-initialize the HRTIM MSP
- Configure the time base unit of a HRTIM timer

This section contains the following APIs:

- ***HAL_HRTIM_Init()***
- ***HAL_HRTIM_DeInit()***
- ***HAL_HRTIM_MspInit()***
- ***HAL_HRTIM_MspDeInit()***
- ***HAL_HRTIM_TimeBaseConfig()***

40.2.4 Simple time base mode functions

This section provides functions allowing to:

- Start simple time base
- Stop simple time base
- Start simple time base and enable interrupt
- Stop simple time base and disable interrupt
- Start simple time base and enable DMA transfer
- Stop simple time base and disable DMA transfer

Note: When a HRTIM timer operates in simple time base mode, the timer counter counts from 0 to the period value.

This section contains the following APIs:

- ***HAL_HRTIM_SimpleBaseStart()***

- *HAL_HRTIM_SimpleBaseStop()*
- *HAL_HRTIM_SimpleBaseStart_IT()*
- *HAL_HRTIM_SimpleBaseStop_IT()*
- *HAL_HRTIM_SimpleBaseStart_DMA()*
- *HAL_HRTIM_SimpleBaseStop_DMA()*

40.2.5 Simple output compare functions

This section provides functions allowing to:

- Configure simple output channel
- Start simple output compare
- Stop simple output compare
- Start simple output compare and enable interrupt
- Stop simple output compare and disable interrupt
- Start simple output compare and enable DMA transfer
- Stop simple output compare and disable DMA transfer

Note: When a HRTIM timer operates in simple output compare mode the output level is set to a programmable value when a match is found between the compare register and the counter. Compare unit 1 is automatically associated to output 1 Compare unit 2 is automatically associated to output 2

This section contains the following APIs:

- *HAL_HRTIM_SimpleOCChannelConfig()*
- *HAL_HRTIM_SimpleOCStart()*
- *HAL_HRTIM_SimpleOCStop()*
- *HAL_HRTIM_SimpleOCStart_IT()*
- *HAL_HRTIM_SimpleOCStop_IT()*
- *HAL_HRTIM_SimpleOCStart_DMA()*
- *HAL_HRTIM_SimpleOCStop_DMA()*

40.2.6 Simple PWM output functions

This section provides functions allowing to:

- Configure simple PWM output channel
- Start simple PWM output
- Stop simple PWM output
- Start simple PWM output and enable interrupt
- Stop simple PWM output and disable interrupt
- Start simple PWM output and enable DMA transfer
- Stop simple PWM output and disable DMA transfer

Note: When a HRTIM timer operates in simple PWM output mode the output level is set to a programmable value when a match is found between the compare register and the counter and reset when the timer period is reached. Duty cycle is determined by the comparison value. Compare unit 1 is automatically associated to output 1 Compare unit 2 is automatically associated to output 2

This section contains the following APIs:

- *HAL_HRTIM_SimplePWMChannelConfig()*
- *HAL_HRTIM_SimplePWMStart()*
- *HAL_HRTIM_SimplePWMStop()*
- *HAL_HRTIM_SimplePWMStart_IT()*
- *HAL_HRTIM_SimplePWMStop_IT()*
- *HAL_HRTIM_SimplePWMStart_DMA()*
- *HAL_HRTIM_SimplePWMStop_DMA()*

40.2.7 Simple input capture functions

This section provides functions allowing to:

- Configure simple input capture channel
- Start simple input capture
- Stop simple input capture
- Start simple input capture and enable interrupt
- Stop simple input capture and disable interrupt
- Start simple input capture and enable DMA transfer
- Stop simple input capture and disable DMA transfer

Note: When a HRTIM timer operates in simple input capture mode the Capture Register (HRTIM_CPT1/2xR) is used to latch the value of the timer counter counter after a transition detected on a given external event input.

This section contains the following APIs:

- ***HAL_HRTIM_SimpleCaptureChannelConfig()***
- ***HAL_HRTIM_SimpleCaptureStart()***
- ***HAL_HRTIM_SimpleCaptureStop()***
- ***HAL_HRTIM_SimpleCaptureStart_IT()***
- ***HAL_HRTIM_SimpleCaptureStop_IT()***
- ***HAL_HRTIM_SimpleCaptureStart_DMA()***
- ***HAL_HRTIM_SimpleCaptureStop_DMA()***

40.2.8 Simple one pulse functions

This section provides functions allowing to:

- Configure one pulse channel
- Start one pulse generation
- Stop one pulse generation
- Start one pulse generation and enable interrupt
- Stop one pulse generation and disable interrupt

Note: When a HRTIM timer operates in simple one pulse mode the timer counter is started in response to transition detected on a given external event input to generate a pulse with a programmable length after a programmable delay.

This section contains the following APIs:

- ***HAL_HRTIM_SimpleOnePulseChannelConfig()***
- ***HAL_HRTIM_SimpleOnePulseStart()***
- ***HAL_HRTIM_SimpleOnePulseStop()***
- ***HAL_HRTIM_SimpleOnePulseStart_IT()***
- ***HAL_HRTIM_SimpleOnePulseStop_IT()***

40.2.9 HRTIM configuration functions

This section provides functions allowing to configure the HRTIM resources shared by all the HRTIM timers operating in waveform mode:

- Configure the burst mode controller
- Configure an external event conditioning
- Configure the external events sampling clock
- Configure a fault conditioning
- Enable or disable fault inputs
- Configure the faults sampling clock
- Configure an ADC trigger

This section contains the following APIs:

- *HAL_HRTIM_BurstModeConfig()*
- *HAL_HRTIM_EventConfig()*
- *HAL_HRTIM_EventPrescalerConfig()*
- *HAL_HRTIM_FaultConfig()*
- *HAL_HRTIM_FaultPrescalerConfig()*
- *HAL_HRTIM_FaultModeCtl()*
- *HAL_HRTIM_ADCTriggerConfig()*

40.2.10 HRTIM timer configuration and control functions

This section provides functions used to configure and control a HRTIM timer operating in waveform mode:

- Configure HRTIM timer general behavior
- Configure HRTIM timer event filtering
- Configure HRTIM timer deadtime insertion
- Configure HRTIM timer chopper mode
- Configure HRTIM timer burst DMA
- Configure HRTIM timer compare unit
- Configure HRTIM timer capture unit
- Configure HRTIM timer output
- Set HRTIM timer output level
- Enable HRTIM timer output
- Disable HRTIM timer output
- Start HRTIM timer
- Stop HRTIM timer
- Start HRTIM timer and enable interrupt
- Stop HRTIM timer and disable interrupt
- Start HRTIM timer and enable DMA transfer
- Stop HRTIM timer and disable DMA transfer
- Enable or disable the burst mode controller
- Start the burst mode controller (by software)
- Trigger a Capture (by software)
- Update the HRTIM timer preloadable registers (by software)
- Reset the HRTIM timer counter (by software)
- Start a burst DMA transfer
- Enable timer register update
- Disable timer register update

This section contains the following APIs:

- *HAL_HRTIM_WaveformTimerConfig()*
- *HAL_HRTIM_TimerEventFilteringConfig()*
- *HAL_HRTIM_DeadTimeConfig()*
- *HAL_HRTIM_ChopperModeConfig()*
- *HAL_HRTIM_BurstDMAConfig()*
- *HAL_HRTIM_WaveformCompareConfig()*
- *HAL_HRTIM_WaveformCaptureConfig()*
- *HAL_HRTIM_WaveformOutputConfig()*
- *HAL_HRTIM_WaveformSetOutputLevel()*
- *HAL_HRTIM_WaveformOutputStart()*
- *HAL_HRTIM_WaveformOutputStop()*
- *HAL_HRTIM_WaveformCountStart()*
- *HAL_HRTIM_WaveformCountStop()*

- *HAL_HRTIM_WaveformCountStart_IT()*
- *HAL_HRTIM_WaveformCountStop_IT()*
- *HAL_HRTIM_WaveformCountStart_DMA()*
- *HAL_HRTIM_WaveformCountStop_DMA()*
- *HAL_HRTIM_BurstModeCtl()*
- *HAL_HRTIM_BurstModeSoftwareTrigger()*
- *HAL_HRTIM_SoftwareCapture()*
- *HAL_HRTIM_SoftwareUpdate()*
- *HAL_HRTIM_SoftwareReset()*
- *HAL_HRTIM_BurstDMATransfer()*
- *HAL_HRTIM_UpdateEnable()*
- *HAL_HRTIM_UpdateDisable()*

40.2.11 Peripheral State functions

This section provides functions used to get HRTIM or HRTIM timer specific information:

- Get HRTIM HAL state
- Get captured value
- Get HRTIM timer output level
- Get HRTIM timer output state
- Get delayed protection status
- Get burst status
- Get current push-pull status
- Get idle push-pull status

This section contains the following APIs:

- *HAL_HRTIM_GetState()*
- *HAL_HRTIM_GetCapturedValue()*
- *HAL_HRTIM_WaveformGetOutputLevel()*
- *HAL_HRTIM_WaveformGetOutputState()*
- *HAL_HRTIM_GetDelayedProtectionStatus()*
- *HAL_HRTIM_GetBurstStatus()*
- *HAL_HRTIM_GetCurrentPushPullStatus()*
- *HAL_HRTIM_GetIdlePushPullStatus()*

40.2.12 Detailed description of functions

HAL_HRTIM_Init

Function name

HAL_StatusTypeDef HAL_HRTIM_Init (HRTIM_HandleTypeDef * hhrtim)

Function description

Initialize a HRTIM instance.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **HAL:** status

HAL_HRTIM_DeInit

Function name

HAL_StatusTypeDef HAL_HRTIM_DeInit (HRTIM_HandleTypeDef * hhrtim)

Function description

De-initialize a HRTIM instance.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **HAL:** status

HAL_HRTIM_MspInit

Function name

void HAL_HRTIM_MspInit (HRTIM_HandleTypeDef * hhrtim)

Function description

MSP initialization for a HRTIM instance.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

HAL_HRTIM_MspDeInit

Function name

void HAL_HRTIM_MspDeInit (HRTIM_HandleTypeDef * hhrtim)

Function description

MSP de-initialization of a HRTIM instance.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

HAL_HRTIM_TimeBaseConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_TimeBaseConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, const HRTIM_TimeBaseCfgTypeDef * pTimeBaseCfg)

Function description

Configure the time base unit of a timer.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **pTimeBaseCfg:** pointer to the time base configuration structure

Return values

- **HAL:** status

Notes

- This function must be called prior starting the timer
- The time-base unit initialization parameters specify: The timer counter operating mode (continuous, one shot), The timer clock prescaler, The timer period, The timer repetition counter.

HAL_HRTIM_SimpleBaseStart

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleBaseStart (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Start the counter of a timer operating in simple time base mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index. This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **HAL:** status

HAL_HRTIM_SimpleBaseStop

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleBaseStop (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Stop the counter of a timer operating in simple time base mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index. This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **HAL:** status

HAL_HRTIM_SimpleBaseStart_IT

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleBaseStart_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Start the counter of a timer operating in simple time base mode (Timer repetition interrupt is enabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index. This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **HAL:** status

HAL_HRTIM_SimpleBaseStop_IT

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleBaseStop_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Stop the counter of a timer operating in simple time base mode (Timer repetition interrupt is disabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index. This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **HAL:** status

HAL_HRTIM_SimpleBaseStart_DMA

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleBaseStart_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t SrcAddr, uint32_t DestAddr, uint32_t Length)

Function description

Start the counter of a timer operating in simple time base mode (Timer repetition DMA request is enabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index. This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **SrcAddr:** DMA transfer source address
- **DestAddr:** DMA transfer destination address
- **Length:** The length of data items (data size) to be transferred from source to destination

HAL_HRTIM_SimpleBaseStop_DMA

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleBaseStop_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Stop the counter of a timer operating in simple time base mode (Timer repetition DMA request is disabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index. This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **HAL:** status

HAL_HRTIM_SimpleOCChannelConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleOCChannelConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OCChannel, const HRTIM_SimpleOCChannelCfgTypeDef * pSimpleOCChannelCfg)

Function description

Configure an output in simple output compare mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **OCChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
- **pSimpleOCChannelCfg:** pointer to the simple output compare output configuration structure

Return values

- **HAL:** status

Notes

- When the timer operates in simple output compare mode: Output 1 is implicitly controlled by the compare unit 1 Output 2 is implicitly controlled by the compare unit 2 Output Set/Reset crossbar is set according to the selected output compare mode: Toggle: SETxyR = RSTxyR = CMPy Active: SETxyR = CMPy, RSTxyR = 0 Inactive: SETxy =0, RSTxy = CMPy

HAL_HRTIM_SimpleOCStart

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleOCStart (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OCChannel)

Function description

Start the output compare signal generation on the designed timer output.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **OCChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleOCStop

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleOCStop (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OCChannel)

Function description

Stop the output compare signal generation on the designed timer output.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **OCChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleOCStart_IT

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleOCStart_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OCChannel)

Function description

Start the output compare signal generation on the designed timer output (Interrupt is enabled (see note note below)).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **OCChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

Notes

- Interrupt enabling depends on the chosen output compare mode Output toggle: compare match interrupt is enabled Output set active: output set interrupt is enabled Output set inactive: output reset interrupt is enabled

HAL_HRTIM_SimpleOCStop_IT

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleOCStop_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OCChannel)

Function description

Stop the output compare signal generation on the designed timer output (Interrupt is disabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **OCChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleOCStart_DMA

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleOCStart_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OCChannel, uint32_t SrcAddr, uint32_t DestAddr, uint32_t Length)

Function description

Start the output compare signal generation on the designed timer output (DMA request is enabled (see note below)).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **OCChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
- **SrcAddr:** DMA transfer source address
- **DestAddr:** DMA transfer destination address
- **Length:** The length of data items (data size) to be transferred from source to destination

Return values

- **HAL:** status

Notes

- DMA request enabling depends on the chosen output compare mode Output toggle: compare match DMA request is enabled Output set active: output set DMA request is enabled Output set inactive: output reset DMA request is enabled

HAL_HRTIM_SimpleOCStop_DMA

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleOCStop_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OCChannel)

Function description

Stop the output compare signal generation on the designed timer output (DMA request is disabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **OCChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimplePWMChannelConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_SimplePWMChannelConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t PWMChannel, const HRTIM_SimplePWMChannelCfgTypeDef * pSimplePWMChannelCfg)

Function description

Configure an output in simple PWM mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **PWMChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
- **pSimplePWMChannelCfg:** pointer to the simple PWM output configuration structure

Return values

- **HAL:** status

Notes

- When the timer operates in simple PWM output mode: Output 1 is implicitly controlled by the compare unit 1 Output 2 is implicitly controlled by the compare unit 2 Output Set/Reset crossbar is set as follows: Output 1: SETx1R = CMP1, RSTx1R = PER Output 2: SETx2R = CMP2, RST2R = PER
- When Simple PWM mode is used the registers preload mechanism is enabled (otherwise the behavior is not guaranteed).

HAL_HRTIM_SimplePWMStart

Function name

HAL_StatusTypeDef HAL_HRTIM_SimplePWMStart (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t PWMChannel)

Function description

Start the PWM output signal generation on the designed timer output.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **PWMChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimplePWMStop

Function name

HAL_StatusTypeDef HAL_HRTIM_SimplePWMStop (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t PWMChannel)

Function description

Stop the PWM output signal generation on the designed timer output.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **PWMChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimplePWMStart_IT

Function name

HAL_StatusTypeDef HAL_HRTIM_SimplePWMStart_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t PWMChannel)

Function description

Start the PWM output signal generation on the designed timer output (The compare interrupt is enabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **PWMChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimplePWMStop_IT

Function name

HAL_StatusTypeDef HAL_HRTIM_SimplePWMStop_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t PWMChannel)

Function description

Stop the PWM output signal generation on the designed timer output (The compare interrupt is disabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **PWMChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimplePWMStart_DMA

Function name

HAL_StatusTypeDef HAL_HRTIM_SimplePWMStart_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t PWMChannel, uint32_t SrcAddr, uint32_t DestAddr, uint32_t Length)

Function description

Start the PWM output signal generation on the designed timer output (The compare DMA request is enabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **PWMChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
- **SrcAddr:** DMA transfer source address
- **DestAddr:** DMA transfer destination address
- **Length:** The length of data items (data size) to be transferred from source to destination

Return values

- **HAL:** status

HAL_HRTIM_SimplePWMStop_DMA

Function name

HAL_StatusTypeDef HAL_HRTIM_SimplePWMStop_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t PWMChannel)

Function description

Stop the PWM output signal generation on the designed timer output (The compare DMA request is disabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **PWMChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleCaptureChannelConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleCaptureChannelConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureChannel, const HRTIM_SimpleCaptureChannelCfgTypeDef * pSimpleCaptureChannelCfg)

Function description

Configure a simple capture.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **CaptureChannel:** Capture unit This parameter can be one of the following values:
  - HRTIM_CAPTUREUNIT_1: Capture unit 1
  - HRTIM_CAPTUREUNIT_2: Capture unit 2
- **pSimpleCaptureChannelCfg:** pointer to the simple capture configuration structure

Return values

- **HAL:** status

Notes

- When the timer operates in simple capture mode the capture is triggered by the designated external event and GPIO input is implicitly used as event source. The cature can be triggered by a rising edge, a falling edge or both edges on event channel.

HAL_HRTIM_SimpleCaptureStart

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleCaptureStart (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureChannel)

Function description

Enable a simple capture on the designed capture unit.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **CaptureChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_CAPTUREUNIT_1: Capture unit 1
  - HRTIM_CAPTUREUNIT_2: Capture unit 2

Return values

- **HAL:** status

Notes

- The external event triggering the capture is available for all timing units. It can be used directly and is active as soon as the timing unit counter is enabled.

HAL_HRTIM_SimpleCaptureStop

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleCaptureStop (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureChannel)

Function description

Disable a simple capture on the designed capture unit.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **CaptureChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_CAPTUREUNIT_1: Capture unit 1
  - HRTIM_CAPTUREUNIT_2: Capture unit 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleCaptureStart_IT

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleCaptureStart_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureChannel)

Function description

Enable a simple capture on the designed capture unit (Capture interrupt is enabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **CaptureChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_CAPTUREUNIT_1: Capture unit 1
  - HRTIM_CAPTUREUNIT_2: Capture unit 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleCaptureStop_IT

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleCaptureStop_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureChannel)

Function description

Disable a simple capture on the designed capture unit (Capture interrupt is disabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **CaptureChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_CAPTUREUNIT_1: Capture unit 1
  - HRTIM_CAPTUREUNIT_2: Capture unit 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleCaptureStart_DMA

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleCaptureStart_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureChannel, uint32_t SrcAddr, uint32_t DestAddr, uint32_t Length)

Function description

Enable a simple capture on the designed capture unit (Capture DMA request is enabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **CaptureChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_CAPTUREUNIT_1: Capture unit 1
  - HRTIM_CAPTUREUNIT_2: Capture unit 2
- **SrcAddr:** DMA transfer source address
- **DestAddr:** DMA transfer destination address
- **Length:** The length of data items (data size) to be transferred from source to destination

Return values

- **HAL:** status

HAL_HRTIM_SimpleCaptureStop_DMA

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleCaptureStop_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureChannel)

Function description

Disable a simple capture on the designed capture unit (Capture DMA request is disabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **CaptureChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_CAPTUREUNIT_1: Capture unit 1
  - HRTIM_CAPTUREUNIT_2: Capture unit 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleOnePulseChannelConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleOnePulseChannelConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OnePulseChannel, const HRTIM_SimpleOnePulseChannelCfgTypeDef * pSimpleOnePulseChannelCfg)

Function description

Configure an output simple one pulse mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **OnePulseChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
- **pSimpleOnePulseChannelCfg:** pointer to the simple one pulse output configuration structure

Return values

- **HAL:** status

Notes

- When the timer operates in simple one pulse mode: the timer counter is implicitly started by the reset event, the reset of the timer counter is triggered by the designated external event GPIO input is implicitly used as event source, Output 1 is implicitly controlled by the compare unit 1, Output 2 is implicitly controlled by the compare unit 2. Output Set/Reset crossbar is set as follows: Output 1: SETx1R = CMP1, RSTx1R = PER Output 2: SETx2R = CMP2, RST2R = PER
- If HAL_HRTIM_SimpleOnePulseChannelConfig is called for both timer outputs, the reset event related configuration data provided in the second call will override the reset event related configuration data provided in the first call.

HAL_HRTIM_SimpleOnePulseStart

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleOnePulseStart (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OnePulseChannel)

Function description

Enable the simple one pulse signal generation on the designed output.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **OnePulseChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleOnePulseStop

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleOnePulseStop (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OnePulseChannel)

Function description

Disable the simple one pulse signal generation on the designed output.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **OnePulseChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleOnePulseStart_IT

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleOnePulseStart_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OnePulseChannel)

Function description

Enable the simple one pulse signal generation on the designed output (The compare interrupt is enabled (pulse start)).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **OnePulseChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleOnePulseStop_IT

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleOnePulseStop_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OnePulseChannel)

Function description

Disable the simple one pulse signal generation on the designed output (The compare interrupt is disabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **OnePulseChannel:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

HAL_HRTIM_BurstModeConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_BurstModeConfig (HRTIM_HandleTypeDef * hhrtim, const HRTIM_BurstModeCfgTypeDef * pBurstModeCfg)

Function description

Configure the burst mode feature of the HRTIM.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **pBurstModeCfg:** pointer to the burst mode configuration structure

Return values

- **HAL:** status

Notes

- This function must be called before starting the burst mode controller

HAL_HRTIM_EventConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_EventConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t Event, const HRTIM_EventCfgTypeDef * pEventCfg)

Function description

Configure the conditioning of an external event.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Event:** external event to configure This parameter can be one of the following values:
  - HRTIM_EVENT_NONE: no external Event
  - HRTIM_EVENT_1: External event 1
  - HRTIM_EVENT_2: External event 2
  - HRTIM_EVENT_3: External event 3
  - HRTIM_EVENT_4: External event 4
  - HRTIM_EVENT_5: External event 5
  - HRTIM_EVENT_6: External event 6
  - HRTIM_EVENT_7: External event 7
  - HRTIM_EVENT_8: External event 8
  - HRTIM_EVENT_9: External event 9
  - HRTIM_EVENT_10: External event 10
- **pEventCfg:** pointer to the event conditioning configuration structure

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer

HAL_HRTIM_EventPrescalerConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_EventPrescalerConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t Prescaler)

Function description

Configure the external event conditioning block prescaler.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Prescaler:** Prescaler value This parameter can be one of the following values:
  - HRTIM_EVENTPRESCALER_DIV1: fEEVS=fHRTIM
  - HRTIM_EVENTPRESCALER_DIV2: fEEVS=fHRTIM / 2
  - HRTIM_EVENTPRESCALER_DIV4: fEEVS=fHRTIM / 4
  - HRTIM_EVENTPRESCALER_DIV8: fEEVS=fHRTIM / 8

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer

HAL_HRTIM_FaultConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_FaultConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t Fault, const HRTIM_FaultCfgTypeDef * pFaultCfg)

Function description

Configure the conditioning of fault input.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Fault:** fault input to configure This parameter can be one of the following values:
  - HRTIM_FAULT_1: Fault input 1
  - HRTIM_FAULT_2: Fault input 2
  - HRTIM_FAULT_3: Fault input 3
  - HRTIM_FAULT_4: Fault input 4
  - HRTIM_FAULT_5: Fault input 5
- **pFaultCfg:** pointer to the fault conditioning configuration structure

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer and before enabling faults inputs

HAL_HRTIM_FaultPrescalerConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_FaultPrescalerConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t Prescaler)

Function description

Configure the fault conditioning block prescaler.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Prescaler:** Prescaler value This parameter can be one of the following values:
  - HRTIM_FAULTPRESCALER_DIV1: fFLTS=fHRTIM
  - HRTIM_FAULTPRESCALER_DIV2: fFLTS=fHRTIM / 2
  - HRTIM_FAULTPRESCALER_DIV4: fFLTS=fHRTIM / 4
  - HRTIM_FAULTPRESCALER_DIV8: fFLTS=fHRTIM / 8

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer and before enabling faults inputs

HAL_HRTIM_FaultModeCtl

Function name

void HAL_HRTIM_FaultModeCtl (HRTIM_HandleTypeDef * hhrtim, uint32_t Faults, uint32_t Enable)

Function description

Enable or disables the HRTIMx Fault mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Faults:** fault input(s) to enable or disable This parameter can be any combination of the following values:
  - HRTIM_FAULT_1: Fault input 1
  - HRTIM_FAULT_2: Fault input 2
  - HRTIM_FAULT_3: Fault input 3
  - HRTIM_FAULT_4: Fault input 4
  - HRTIM_FAULT_5: Fault input 5
- **Enable:** Fault(s) enabling This parameter can be one of the following values:
  - HRTIM_FAULTMODECTL_ENABLED: Fault(s) enabled
  - HRTIM_FAULTMODECTL_DISABLED: Fault(s) disabled

Return values

- **None:**

HAL_HRTIM_ADCTriggerConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_ADCTriggerConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t ADCTrigger, const HRTIM_ADCTriggerCfgTypeDef * pADCTriggerCfg)

Function description

Configure both the ADC trigger register update source and the ADC trigger source.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **ADCTrigger:** ADC trigger to configure This parameter can be one of the following values:
  - HRTIM_ADCTRIGGER_1: ADC trigger 1
  - HRTIM_ADCTRIGGER_2: ADC trigger 2
  - HRTIM_ADCTRIGGER_3: ADC trigger 3
  - HRTIM_ADCTRIGGER_4: ADC trigger 4
- **pADCTriggerCfg:** pointer to the ADC trigger configuration structure

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer

HAL_HRTIM_WaveformTimerConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformTimerConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, const HRTIM_TimerCfgTypeDef * pTimerCfg)

Function description

Configure the general behavior of a timer operating in waveform mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **pTimerCfg:** pointer to the timer configuration structure

Return values

- **HAL:** status

Notes

- When the timer operates in waveform mode, all the features supported by the HRTIM are available without any limitation.
- This function must be called before starting the timer

HAL_HRTIM_WaveformCompareConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformCompareConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CompareUnit, const HRTIM_CompareCfgTypeDef * pCompareCfg)

Function description

Configure the compare unit of a timer operating in waveform mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **CompareUnit:** Compare unit to configure This parameter can be one of the following values:
  - HRTIM_COMPAREUNIT_1: Compare unit 1
  - HRTIM_COMPAREUNIT_2: Compare unit 2
  - HRTIM_COMPAREUNIT_3: Compare unit 3
  - HRTIM_COMPAREUNIT_4: Compare unit 4
- **pCompareCfg:** pointer to the compare unit configuration structure

Return values

- **HAL:** status

Notes

- When auto delayed mode is required for compare unit 2 or compare unit 4, application has to configure separately the capture unit. Capture unit to configure in that case depends on the compare unit auto delayed mode is applied to (see below): Auto delayed on output compare 2: capture unit 1 must be configured Auto delayed on output compare 4: capture unit 2 must be configured
- This function must be called before starting the timer

HAL_HRTIM_WaveformCaptureConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformCaptureConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureUnit, const HRTIM_CaptureCfgTypeDef * pCaptureCfg)

Function description

Configure the capture unit of a timer operating in waveform mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **CaptureUnit:** Capture unit to configure This parameter can be one of the following values:
  - HRTIM_CAPTUREUNIT_1: Capture unit 1
  - HRTIM_CAPTUREUNIT_2: Capture unit 2
- **pCaptureCfg:** pointer to the compare unit configuration structure

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer

HAL_HRTIM_WaveformOutputConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformOutputConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t Output, const HRTIM_OutputCfgTypeDef * pOutputCfg)

Function description

Configure the output of a timer operating in waveform mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **Output:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
- **pOutputCfg:** pointer to the timer output configuration structure

Return values

- **HAL:** status

Notes

- This function must be called before configuring the timer and after configuring the deadtime insertion feature (if required).

HAL_HRTIM_WaveformSetOutputLevel

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformSetOutputLevel (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t Output, uint32_t OutputLevel)

Function description

Force the timer output to its active or inactive state.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **Output:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2
- **OutputLevel:** indicates whether the output is forced to its active or inactive level This parameter can be one of the following values:
  - HRTIM_OUTPUTLEVEL_ACTIVE: output is forced to its active level
  - HRTIM_OUTPUTLEVEL_INACTIVE: output is forced to its inactive level

Return values

- **HAL:** status

Notes

- The 'software set/reset trigger' bit in the output set/reset registers is automatically reset by hardware

HAL_HRTIM_TimerEventFilteringConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_TimerEventFilteringConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t Event, const HRTIM_TimerEventFilteringCfgTypeDef * pTimerEventFilteringCfg)

Function description

Configure the event filtering capabilities of a timer (blanking, windowing)

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **Event:** external event for which timer event filtering must be configured This parameter can be one of the following values:
  - HRTIM_EVENT_1: External event 1
  - HRTIM_EVENT_2: External event 2
  - HRTIM_EVENT_3: External event 3
  - HRTIM_EVENT_4: External event 4
  - HRTIM_EVENT_5: External event 5
  - HRTIM_EVENT_6: External event 6
  - HRTIM_EVENT_7: External event 7
  - HRTIM_EVENT_8: External event 8
  - HRTIM_EVENT_9: External event 9
  - HRTIM_EVENT_10: External event 10
- **pTimerEventFilteringCfg:** pointer to the timer event filtering configuration structure

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer

HAL_HRTIM_DeadTimeConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_DeadTimeConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, const HRTIM_DeadTimeCfgTypeDef * pDeadTimeCfg)

Function description

Configure the dead-time insertion feature for a timer.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **pDeadTimeCfg:** pointer to the deadtime insertion configuration structure

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer

HAL_HRTIM_ChopperModeConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_ChopperModeConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, const HRTIM_ChopperModeCfgTypeDef * pChopperModeCfg)

Function description

Configure the chopper mode feature for a timer.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **pChopperModeCfg:** pointer to the chopper mode configuration structure

Return values

- **HAL:** status

Notes

- This function must be called before configuring the timer output(s)

HAL_HRTIM_BurstDMAConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_BurstDMAConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t RegistersToUpdate)

Function description

Configure the burst DMA controller for a timer.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **RegistersToUpdate:** registers to be written by DMA This parameter can be any combination of the following values:
  - HRTIM_BURSTDMA_CR: HRTIM_MCR or HRTIM_TIMxCR
  - HRTIM_BURSTDMA_ICR: HRTIM_MICR or HRTIM_TIMxICR
  - HRTIM_BURSTDMA_DIER: HRTIM_MDIER or HRTIM_TIMxDIER
  - HRTIM_BURSTDMA_CNT: HRTIM_MCNT or HRTIM_TIMxCNT
  - HRTIM_BURSTDMA_PER: HRTIM_MPER or HRTIM_TIMxPER
  - HRTIM_BURSTDMA_REP: HRTIM_MREP or HRTIM_TIMxREP
  - HRTIM_BURSTDMA_CMP1: HRTIM_MCMP1 or HRTIM_TIMxCMP1
  - HRTIM_BURSTDMA_CMP2: HRTIM_MCMP2 or HRTIM_TIMxCMP2
  - HRTIM_BURSTDMA_CMP3: HRTIM_MCMP3 or HRTIM_TIMxCMP3
  - HRTIM_BURSTDMA_CMP4: HRTIM_MCMP4 or HRTIM_TIMxCMP4
  - HRTIM_BURSTDMA_DTR: HRTIM_TIMxDTR
  - HRTIM_BURSTDMA_SET1R: HRTIM_TIMxSET1R
  - HRTIM_BURSTDMA_RST1R: HRTIM_TIMxRST1R
  - HRTIM_BURSTDMA_SET2R: HRTIM_TIMxSET2R
  - HRTIM_BURSTDMA_RST2R: HRTIM_TIMxRST2R
  - HRTIM_BURSTDMA_EEFR1: HRTIM_TIMxEEFR1
  - HRTIM_BURSTDMA_EEFR2: HRTIM_TIMxEEFR2
  - HRTIM_BURSTDMA_RSTR: HRTIM_TIMxRSTR
  - HRTIM_BURSTDMA_CHPR: HRTIM_TIMxCHPR
  - HRTIM_BURSTDMA_OUTR: HRTIM_TIMxOUTR
  - HRTIM_BURSTDMA_FLTR: HRTIM_TIMxFLTR

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer

HAL_HRTIM_WaveformCountStart

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformCountStart (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)

Function description

Start the counter of the designated timer(s) operating in waveform mode Timers can be combined (ORed) to allow for simultaneous counter start.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** Timer counter(s) to start This parameter can be any combination of the following values:
  - HRTIM_TIMERID_MASTER
  - HRTIM_TIMERID_TIMER_A
  - HRTIM_TIMERID_TIMER_B
  - HRTIM_TIMERID_TIMER_C
  - HRTIM_TIMERID_TIMER_D
  - HRTIM_TIMERID_TIMER_E

Return values

- **HAL:** status

HAL_HRTIM_WaveformCountStop

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformCountStop (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)

Function description

Stop the counter of the designated timer(s) operating in waveform mode Timers can be combined (ORed) to allow for simultaneous counter stop.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** Timer counter(s) to stop This parameter can be any combination of the following values:
  - HRTIM_TIMERID_MASTER
  - HRTIM_TIMERID_TIMER_A
  - HRTIM_TIMERID_TIMER_B
  - HRTIM_TIMERID_TIMER_C
  - HRTIM_TIMERID_TIMER_D
  - HRTIM_TIMERID_TIMER_E

Return values

- **HAL:** status

Notes

- The counter of a timer is stopped only if all timer outputs are disabled

HAL_HRTIM_WaveformCountStart_IT

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformCountStart_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)

Function description

Start the counter of the designated timer(s) operating in waveform mode Timers can be combined (ORed) to allow for simultaneous counter start.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** Timer counter(s) to start This parameter can be any combination of the following values:
  - HRTIM_TIMERID_MASTER
  - HRTIM_TIMERID_TIMER_A
  - HRTIM_TIMERID_TIMER_B
  - HRTIM_TIMERID_TIMER_C
  - HRTIM_TIMERID_TIMER_D
  - HRTIM_TIMERID_TIMER_E

Return values

- **HAL:** status

Notes

- HRTIM interrupts (e.g. faults interrupts) and interrupts related to the timers to start are enabled within this function. Interrupts to enable are selected through HAL_HRTIM_WaveformTimerConfig function.

HAL_HRTIM_WaveformCountStop_IT

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformCountStop_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)

Function description

Stop the counter of the designated timer(s) operating in waveform mode Timers can be combined (ORed) to allow for simultaneous counter stop.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** Timer counter(s) to stop This parameter can be any combination of the following values:
  - HRTIM_TIMERID_MASTER
  - HRTIM_TIMERID_TIMER_A
  - HRTIM_TIMERID_TIMER_B
  - HRTIM_TIMERID_TIMER_C
  - HRTIM_TIMERID_TIMER_D
  - HRTIM_TIMERID_TIMER_E

Return values

- **HAL:** status

Notes

- The counter of a timer is stopped only if all timer outputs are disabled
- All enabled timer related interrupts are disabled.

HAL_HRTIM_WaveformCountStart_DMA

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformCountStart_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)

Function description

Start the counter of the designated timer(s) operating in waveform mode Timers can be combined (ORed) to allow for simultaneous counter start.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** Timer counter(s) to start This parameter can be any combination of the following values:
  - HRTIM_TIMERID_MASTER
  - HRTIM_TIMERID_TIMER_A
  - HRTIM_TIMERID_TIMER_B
  - HRTIM_TIMERID_TIMER_C
  - HRTIM_TIMERID_TIMER_D
  - HRTIM_TIMERID_TIMER_E

Return values

- **HAL:** status

Notes

- This function enables the dma request(s) mentioned in the timer configuration data structure for every timers to start.
- The source memory address, the destination memory address and the size of each DMA transfer are specified at timer configuration time (see HAL_HRTIM_WaveformTimerConfig)

HAL_HRTIM_WaveformCountStop_DMA

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformCountStop_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)

Function description

Stop the counter of the designated timer(s) operating in waveform mode Timers can be combined (ORed) to allow for simultaneous counter stop.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** Timer counter(s) to stop This parameter can be any combination of the following values:
  - HRTIM_TIMERID_MASTER
  - HRTIM_TIMERID_TIMER_A
  - HRTIM_TIMERID_TIMER_B
  - HRTIM_TIMERID_TIMER_C
  - HRTIM_TIMERID_TIMER_D
  - HRTIM_TIMERID_TIMER_E

Return values

- **HAL:** status

Notes

- The counter of a timer is stopped only if all timer outputs are disabled
- All enabled timer related DMA requests are disabled.

HAL_HRTIM_WaveformOutputStart

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformOutputStart (HRTIM_HandleTypeDef * hhrtim, uint32_t OutputsToStart)

Function description

Enable the generation of the waveform signal on the designated output(s) Outputs can be combined (ORed) to allow for simultaneous output enabling.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **OutputsToStart:** Timer output(s) to enable This parameter can be any combination of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

HAL_HRTIM_WaveformOutputStop

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformOutputStop (HRTIM_HandleTypeDef * hhrtim, uint32_t OutputsToStop)

Function description

Disable the generation of the waveform signal on the designated output(s) Outputs can be combined (ORed) to allow for simultaneous output disabling.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **OutputsToStop:** Timer output(s) to disable This parameter can be any combination of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

HAL_HRTIM_BurstModeCtl

Function name

HAL_StatusTypeDef HAL_HRTIM_BurstModeCtl (HRTIM_HandleTypeDef * hhrtim, uint32_t Enable)

Function description

Enable or disables the HRTIM burst mode controller.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Enable:** Burst mode controller enabling This parameter can be one of the following values:
  - HRTIM_BURSTMODECTL_ENABLED: Burst mode enabled
  - HRTIM_BURSTMODECTL_DISABLED: Burst mode disabled

Return values

- **HAL:** status

Notes

- This function must be called after starting the timer(s)

HAL_HRTIM_BurstModeSoftwareTrigger

Function name

HAL_StatusTypeDef HAL_HRTIM_BurstModeSoftwareTrigger (HRTIM_HandleTypeDef * hhrtim)

Function description

Trig the burst mode operation.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **HAL:** status

HAL_HRTIM_SoftwareCapture

Function name

HAL_StatusTypeDef HAL_HRTIM_SoftwareCapture (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureUnit)

Function description

Trig a software capture on the designed capture unit.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **CaptureUnit:** Capture unit to trig This parameter can be one of the following values:
  - HRTIM_CAPTUREUNIT_1: Capture unit 1
  - HRTIM_CAPTUREUNIT_2: Capture unit 2

Return values

- **HAL:** status

Notes

- The 'software capture' bit in the capure configuration register is automatically reset by hardware

HAL_HRTIM_SoftwareUpdate

Function name

HAL_StatusTypeDef HAL_HRTIM_SoftwareUpdate (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)

Function description

Trig the update of the registers of one or several timers.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** timers concerned with the software register update This parameter can be any combination of the following values:
  - HRTIM_TIMERUPDATE_MASTER
  - HRTIM_TIMERUPDATE_A
  - HRTIM_TIMERUPDATE_B
  - HRTIM_TIMERUPDATE_C
  - HRTIM_TIMERUPDATE_D
  - HRTIM_TIMERUPDATE_E

Return values

- **HAL:** status

Notes

- The 'software update' bits in the HRTIM control register 2 register are automatically reset by hardware

HAL_HRTIM_SoftwareReset

Function name

HAL_StatusTypeDef HAL_HRTIM_SoftwareReset (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)

Function description

Trig the reset of one or several timers.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** timers concerned with the software counter reset This parameter can be any combination of the following values:
  - HRTIM_TIMERRESET_MASTER
  - HRTIM_TIMERRESET_TIMER_A
  - HRTIM_TIMERRESET_TIMER_B
  - HRTIM_TIMERRESET_TIMER_C
  - HRTIM_TIMERRESET_TIMER_D
  - HRTIM_TIMERRESET_TIMER_E

Return values

- **HAL:** status

Notes

- The 'software reset' bits in the HRTIM control register 2 are automatically reset by hardware

HAL_HRTIM_BurstDMATransfer

Function name

HAL_StatusTypeDef HAL_HRTIM_BurstDMATransfer (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t BurstBufferAddress, uint32_t BurstBufferLength)

Function description

Start a burst DMA operation to update HRTIM control registers content.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **BurstBufferAddress:** address of the buffer the HRTIM control registers content will be updated from.
- **BurstBufferLength:** size (in WORDS) of the burst buffer.

Return values

- **HAL:** status

Notes

- The TimerIdx parameter determines the dma channel to be used by the DMA burst controller (see below) HRTIM_TIMERINDEX_MASTER: DMA channel 2 is used by the DMA burst controller HRTIM_TIMERINDEX_TIMER_A: DMA channel 3 is used by the DMA burst controller HRTIM_TIMERINDEX_TIMER_B: DMA channel 4 is used by the DMA burst controller HRTIM_TIMERINDEX_TIMER_C: DMA channel 5 is used by the DMA burst controller HRTIM_TIMERINDEX_TIMER_D: DMA channel 6 is used by the DMA burst controller HRTIM_TIMERINDEX_TIMER_E: DMA channel 7 is used by the DMA burst controller

HAL_HRTIM_UpdateEnable

Function name

HAL_StatusTypeDef HAL_HRTIM_UpdateEnable (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)

Function description

Enable the transfer from preload to active registers for one or several timing units (including master timer).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** Timer(s) concerned by the register preload enabling command This parameter can be any combination of the following values:
  - HRTIM_TIMERUPDATE_MASTER
  - HRTIM_TIMERUPDATE_A
  - HRTIM_TIMERUPDATE_B
  - HRTIM_TIMERUPDATE_C
  - HRTIM_TIMERUPDATE_D
  - HRTIM_TIMERUPDATE_E

Return values

- **HAL:** status

HAL_HRTIM_UpdateDisable

Function name

HAL_StatusTypeDef HAL_HRTIM_UpdateDisable (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)

Function description

Disable the transfer from preload to active registers for one or several timing units (including master timer).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** Timer(s) concerned by the register preload disabling command This parameter can be any combination of the following values:
  - HRTIM_TIMERUPDATE_MASTER
  - HRTIM_TIMERUPDATE_A
  - HRTIM_TIMERUPDATE_B
  - HRTIM_TIMERUPDATE_C
  - HRTIM_TIMERUPDATE_D
  - HRTIM_TIMERUPDATE_E

Return values

- **HAL:** status

HAL_HRTIM_GetState

Function name

HAL_HRTIM_StateTypeDef HAL_HRTIM_GetState (const HRTIM_HandleTypeDef * hhrtim)

Function description

Return the HRTIM HAL state.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **HAL:** state

HAL_HRTIM_GetCapturedValue

Function name

uint32_t HAL_HRTIM_GetCapturedValue (const HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureUnit)

Function description

Return actual value of the capture register of the designated capture unit.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **CaptureUnit:** Capture unit to trig This parameter can be one of the following values:
  - HRTIM_CAPTUREUNIT_1: Capture unit 1
  - HRTIM_CAPTUREUNIT_2: Capture unit 2

Return values

- **Captured:** value

HAL_HRTIM_WaveformGetOutputLevel

Function name

uint32_t HAL_HRTIM_WaveformGetOutputLevel (const HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t Output)

Function description

Return actual level (active or inactive) of the designated output.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **Output:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **Output:** level

Notes

- Returned output level is taken before the output stage (chopper, polarity).

HAL_HRTIM_WaveformGetOutputState

Function name

uint32_t HAL_HRTIM_WaveformGetOutputState (const HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t Output)

Function description

Return actual state (RUN, IDLE, FAULT) of the designated output.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **Output:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **Output:** state

HAL_HRTIM_GetDelayedProtectionStatus

Function name

uint32_t HAL_HRTIM_GetDelayedProtectionStatus (const HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t Output)

Function description

Return the level (active or inactive) of the designated output when the delayed protection was triggered.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E
- **Output:** Timer output This parameter can be one of the following values:
  - HRTIM_OUTPUT_TA1: Timer A - Output 1
  - HRTIM_OUTPUT_TA2: Timer A - Output 2
  - HRTIM_OUTPUT_TB1: Timer B - Output 1
  - HRTIM_OUTPUT_TB2: Timer B - Output 2
  - HRTIM_OUTPUT_TC1: Timer C - Output 1
  - HRTIM_OUTPUT_TC2: Timer C - Output 2
  - HRTIM_OUTPUT_TD1: Timer D - Output 1
  - HRTIM_OUTPUT_TD2: Timer D - Output 2
  - HRTIM_OUTPUT_TE1: Timer E - Output 1
  - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **Delayed:** protection status

HAL_HRTIM_GetBurstStatus

Function name

uint32_t HAL_HRTIM_GetBurstStatus (const HRTIM_HandleTypeDef * hhrtim)

Function description

Return the actual status (active or inactive) of the burst mode controller.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **Burst:** mode controller status

HAL_HRTIM_GetCurrentPushPullStatus

Function name

uint32_t HAL_HRTIM_GetCurrentPushPullStatus (const HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Indicate on which output the signal is currently active (when the push pull mode is enabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **Burst:** mode controller status

HAL_HRTIM_GetIdlePushPullStatus

Function name

uint32_t HAL_HRTIM_GetIdlePushPullStatus (const HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Indicate on which output the signal was applied, in push-pull mode, balanced fault mode or delayed idle mode, when the protection was triggered.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **Idle:** Push Pull Status

HAL_HRTIM_IRQHandler

Function name

void HAL_HRTIM_IRQHandler (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

This function handles HRTIM interrupt request.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be any value of HRTIM_Timer_Index

Return values

- **None:**

HAL_HRTIM_Fault1Callback

Function name

void HAL_HRTIM_Fault1Callback (HRTIM_HandleTypeDef * hhrtim)

Function description

Callback function invoked when a fault 1 interrupt occurred.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle *

Return values

- **None:**
- **None:**

HAL_HRTIM_Fault2Callback

Function name

void HAL_HRTIM_Fault2Callback (HRTIM_HandleTypeDef * hhrtim)

Function description

Callback function invoked when a fault 2 interrupt occurred.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

HAL_HRTIM_Fault3Callback

Function name

void HAL_HRTIM_Fault3Callback (HRTIM_HandleTypeDef * hhrtim)

Function description

Callback function invoked when a fault 3 interrupt occurred.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

HAL_HRTIM_Fault4Callback

Function name

void HAL_HRTIM_Fault4Callback (HRTIM_HandleTypeDef * hhrtim)

Function description

Callback function invoked when a fault 4 interrupt occurred.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

HAL_HRTIM_Fault5Callback

Function name

void HAL_HRTIM_Fault5Callback (HRTIM_HandleTypeDef * hhrtim)

Function description

Callback function invoked when a fault 5 interrupt occurred.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

HAL_HRTIM_SystemFaultCallback

Function name

void HAL_HRTIM_SystemFaultCallback (HRTIM_HandleTypeDef * hhrtim)

Function description

Callback function invoked when a system fault interrupt occurred.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

HAL_HRTIM_BurstModePeriodCallback

Function name

void HAL_HRTIM_BurstModePeriodCallback (HRTIM_HandleTypeDef * hhrtim)

Function description

Callback function invoked when the end of the burst mode period is reached.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

HAL_HRTIM_SynchronizationEventCallback

Function name

void HAL_HRTIM_SynchronizationEventCallback (HRTIM_HandleTypeDef * hhrtim)

Function description

Callback function invoked when a synchronization input event is received.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

HAL_HRTIM_RegistersUpdateCallback

Function name

void HAL_HRTIM_RegistersUpdateCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when timer registers are updated.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **None:**

HAL_HRTIM_RepetitionEventCallback

Function name

void HAL_HRTIM_RepetitionEventCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when timer repetition period has elapsed.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **None:**

HAL_HRTIM_Compare1EventCallback

Function name

void HAL_HRTIM_Compare1EventCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer counter matches the value programmed in the compare 1 register.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **None:**

HAL_HRTIM_Compare2EventCallback

Function name

void HAL_HRTIM_Compare2EventCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer counter matches the value programmed in the compare 2 register.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **None:**

HAL_HRTIM_Compare3EventCallback

Function name

void HAL_HRTIM_Compare3EventCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer counter matches the value programmed in the compare 3 register.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **None:**

HAL_HRTIM_Compare4EventCallback

Function name

void HAL_HRTIM_Compare4EventCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer counter matches the value programmed in the compare 4 register.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **None:**

HAL_HRTIM_Capture1EventCallback

Function name

void HAL_HRTIM_Capture1EventCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer x capture 1 event occurs.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **None:**

HAL_HRTIM_Capture2EventCallback

Function name

void HAL_HRTIM_Capture2EventCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer x capture 2 event occurs.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **None:**

HAL_HRTIM_DelayedProtectionCallback

Function name

void HAL_HRTIM_DelayedProtectionCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the delayed idle or balanced idle mode is entered.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **None:**

HAL_HRTIM_CounterResetCallback

Function name

void HAL_HRTIM_CounterResetCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer x counter reset/roll-over event occurs.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **None:**

HAL_HRTIM_Output1SetCallback

Function name

void HAL_HRTIM_Output1SetCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer x output 1 is set.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **None:**

HAL_HRTIM_Output1ResetCallback

Function name

void HAL_HRTIM_Output1ResetCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer x output 1 is reset.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **None:**

HAL_HRTIM_Output2SetCallback

Function name

void HAL_HRTIM_Output2SetCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer x output 2 is set.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **None:**

HAL_HRTIM_Output2ResetCallback

Function name

void HAL_HRTIM_Output2ResetCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer x output 2 is reset.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **None:**

HAL_HRTIM_BurstDMATransferCallback

Function name

void HAL_HRTIM_BurstDMATransferCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when a DMA burst transfer is completed.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
  - HRTIM_TIMERINDEX_MASTER for master timer
  - HRTIM_TIMERINDEX_TIMER_A for timer A
  - HRTIM_TIMERINDEX_TIMER_B for timer B
  - HRTIM_TIMERINDEX_TIMER_C for timer C
  - HRTIM_TIMERINDEX_TIMER_D for timer D
  - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **None:**

HAL_HRTIM_ErrorCallback

Function name

void HAL_HRTIM_ErrorCallback (HRTIM_HandleTypeDef * hhrtim)

Function description

Callback function invoked when a DMA error occurs.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

HAL_HRTIM_RegisterCallback

Function name

HAL_StatusTypeDef HAL_HRTIM_RegisterCallback (HRTIM_HandleTypeDef * hhrtim, HAL_HRTIM_CallbackIDTypeDef CallbackID, pHRTIM_CallbackTypeDef pCallback)

Function description

HRTIM callback function registration.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **CallbackID:** ID of the HRTIM callback function to register This parameter can be one of the following values:
  - HAL_HRTIM_FAULT1CALLBACK_CB_ID
  - HAL_HRTIM_FAULT2CALLBACK_CB_ID
  - HAL_HRTIM_FAULT3CALLBACK_CB_ID
  - HAL_HRTIM_FAULT4CALLBACK_CB_ID
  - HAL_HRTIM_FAULT5CALLBACK_CB_ID
  - HAL_HRTIM_SYSTEMFAULTCALLBACK_CB_ID
  - HAL_HRTIM_BURSTMODEPERIODCALLBACK_CB_ID
  - HAL_HRTIM_SYNCHRONIZATIONEVENTCALLBACK_CB_ID
  - HAL_HRTIM_ERRORCALLBACK_CB_ID
  - HAL_HRTIM_MSPINIT_CB_ID
  - HAL_HRTIM_MSPDEINIT_CB_ID
- **pCallback:** Callback function pointer

Return values

- **HAL:** status

HAL_HRTIM_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_HRTIM_UnRegisterCallback (HRTIM_HandleTypeDef * hhrtim, HAL_HRTIM_CallbackIDTypeDef CallbackID)

Function description

HRTIM callback function un-registration.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **CallbackID:** ID of the HRTIM callback function to unregister This parameter can be one of the following values:
  - HAL_HRTIM_FAULT1CALLBACK_CB_ID
  - HAL_HRTIM_FAULT2CALLBACK_CB_ID
  - HAL_HRTIM_FAULT3CALLBACK_CB_ID
  - HAL_HRTIM_FAULT4CALLBACK_CB_ID
  - HAL_HRTIM_FAULT5CALLBACK_CB_ID
  - HAL_HRTIM_SYSTEMFAULTCALLBACK_CB_ID
  - HAL_HRTIM_BURSTMODEPERIODCALLBACK_CB_ID
  - HAL_HRTIM_SYNCHRONIZATIONEVENTCALLBACK_CB_ID
  - HAL_HRTIM_ERRORCALLBACK_CB_ID
  - HAL_HRTIM_MSPINIT_CB_ID
  - HAL_HRTIM_MSPDEINIT_CB_ID

Return values

- **HAL:** status

HAL_HRTIM_TIMxRegisterCallback

Function name

HAL_StatusTypeDef HAL_HRTIM_TIMxRegisterCallback (HRTIM_HandleTypeDef * hhrtim, HAL_HRTIM_CallbackIDTypeDef CallbackID, pHRTIM_TIMxCallbackTypeDef pCallback)

Function description

HRTIM Timer x callback function registration.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **CallbackID:** ID of the HRTIM Timer x callback function to register This parameter can be one of the following values:
  - HAL_HRTIM_REGISTERSUPDATECALLBACK_CB_ID
  - HAL_HRTIM_REPETITIONEVENTCALLBACK_CB_ID
  - HAL_HRTIM_COMPARE1EVENTCALLBACK_CB_ID
  - HAL_HRTIM_COMPARE2EVENTCALLBACK_CB_ID
  - HAL_HRTIM_COMPARE3EVENTCALLBACK_CB_ID
  - HAL_HRTIM_COMPARE4EVENTCALLBACK_CB_ID
  - HAL_HRTIM_CAPTURE1EVENTCALLBACK_CB_ID
  - HAL_HRTIM_CAPTURE2EVENTCALLBACK_CB_ID
  - HAL_HRTIM_DELAYEDPROTECTIONCALLBACK_CB_ID
  - HAL_HRTIM_COUNTERRESETCALLBACK_CB_ID
  - HAL_HRTIM_OUTPUT1SETCALLBACK_CB_ID
  - HAL_HRTIM_OUTPUT1RESETCALLBACK_CB_ID
  - HAL_HRTIM_OUTPUT2SETCALLBACK_CB_ID
  - HAL_HRTIM_OUTPUT2RESETCALLBACK_CB_ID
  - HAL_HRTIM_BURSTDMATRANSFERCALLBACK_CB_ID
- **pCallback:** Callback function pointer

Return values

- **HAL:** status

HAL_HRTIM_TIMxUnRegisterCallback

Function name

HAL_StatusTypeDef HAL_HRTIM_TIMxUnRegisterCallback (HRTIM_HandleTypeDef * hhrtim, HAL_HRTIM_CallbackIDTypeDef CallbackID)

Function description

HRTIM Timer x callback function un-registration.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **CallbackID:** ID of the HRTIM callback Timer x function to unregister This parameter can be one of the following values:
  - HAL_HRTIM_REGISTERSUPDATECALLBACK_CB_ID
  - HAL_HRTIM_REPETITIONEVENTCALLBACK_CB_ID
  - HAL_HRTIM_COMPARE1EVENTCALLBACK_CB_ID
  - HAL_HRTIM_COMPARE2EVENTCALLBACK_CB_ID
  - HAL_HRTIM_COMPARE3EVENTCALLBACK_CB_ID
  - HAL_HRTIM_COMPARE4EVENTCALLBACK_CB_ID
  - HAL_HRTIM_CAPTURE1EVENTCALLBACK_CB_ID
  - HAL_HRTIM_CAPTURE2EVENTCALLBACK_CB_ID
  - HAL_HRTIM_DELAYEDPROTECTIONCALLBACK_CB_ID
  - HAL_HRTIM_COUNTERRESETCALLBACK_CB_ID
  - HAL_HRTIM_OUTPUT1SETCALLBACK_CB_ID
  - HAL_HRTIM_OUTPUT1RESETCALLBACK_CB_ID
  - HAL_HRTIM_OUTPUT2SETCALLBACK_CB_ID
  - HAL_HRTIM_OUTPUT2RESETCALLBACK_CB_ID
  - HAL_HRTIM_BURSTDMATRANSFERCALLBACK_CB_ID

Return values

- **HAL:** status

40.3 HRTIM Firmware driver defines

The following section lists the various define and macros of the module.

40.3.1 HRTIM

HRTIM

HRTIM ADC Trigger

HRTIM_ADCTRIGGER_1

ADC trigger 1 identifier

HRTIM_ADCTRIGGER_2

ADC trigger 2 identifier

HRTIM_ADCTRIGGER_3

ADC trigger 3 identifier

HRTIM_ADCTRIGGER_4

ADC trigger 4 identifier

IS_HRTIM_ADCTRIGGER

HRTIM ADC Trigger Event

HRTIM_ADCTRIGGEREVENT13_NONE

No ADC trigger event

HRTIM_ADCTRIGGEREVENT13_MASTER_CMP1

ADC Trigger on master compare 1U

HRTIM_ADCTRIGGEREVENT13_MASTER_CMP2

ADC Trigger on master compare 2U

HRTIM_ADCTRIGGEREVENT13_MASTER_CMP3

ADC Trigger on master compare 3U

HRTIM_ADCTRIGGEREVENT13_MASTER_CMP4

ADC Trigger on master compare 4U

HRTIM_ADCTRIGGEREVENT13_MASTER_PERIOD

ADC Trigger on master period

HRTIM_ADCTRIGGEREVENT13_EVENT_1

ADC Trigger on external event 1U

HRTIM_ADCTRIGGEREVENT13_EVENT_2

ADC Trigger on external event 2U

HRTIM_ADCTRIGGEREVENT13_EVENT_3

ADC Trigger on external event 3U

HRTIM_ADCTRIGGEREVENT13_EVENT_4

ADC Trigger on external event 4U

HRTIM_ADCTRIGGEREVENT13_EVENT_5

ADC Trigger on external event 5U

HRTIM_ADCTRIGGEREVENT13_TIMERA_CMP2

ADC Trigger on Timer A compare 2U

HRTIM_ADCTRIGGEREVENT13_TIMERA_CMP3

ADC Trigger on Timer A compare 3U

HRTIM_ADCTRIGGEREVENT13_TIMERA_CMP4

ADC Trigger on Timer A compare 4U

HRTIM_ADCTRIGGEREVENT13_TIMERA_PERIOD

ADC Trigger on Timer A period

HRTIM_ADCTRIGGEREVENT13_TIMERA_RESET

ADC Trigger on Timer A reset

HRTIM_ADCTRIGGEREVENT13_TIMERB_CMP2

ADC Trigger on Timer B compare 2U

HRTIM_ADCTRIGGEREVENT13_TIMERB_CMP3

ADC Trigger on Timer B compare 3U

HRTIM_ADCTRIGGEREVENT13_TIMERB_CMP4

ADC Trigger on Timer B compare 4U

HRTIM_ADCTRIGGEREVENT13_TIMERB_PERIOD

ADC Trigger on Timer B period

HRTIM_ADCTRIGGEREVENT13_TIMERB_RESET

ADC Trigger on Timer B reset

HRTIM_ADCTRIGGEREVENT13_TIMERC_CMP2

ADC Trigger on Timer C compare 2U

HRTIM_ADCTRIGGEREVENT13_TIMERC_CMP3

ADC Trigger on Timer C compare 3U

HRTIM_ADCTRIGGEREVENT13_TIMERC_CMP4

ADC Trigger on Timer C compare 4U

HRTIM_ADCTRIGGEREVENT13_TIMERC_PERIOD

ADC Trigger on Timer C period

HRTIM_ADCTRIGGEREVENT13_TIMERD_CMP2

ADC Trigger on Timer D compare 2U

HRTIM_ADCTRIGGEREVENT13_TIMERD_CMP3

ADC Trigger on Timer D compare 3U

HRTIM_ADCTRIGGEREVENT13_TIMERD_CMP4

ADC Trigger on Timer D compare 4U

HRTIM_ADCTRIGGEREVENT13_TIMERD_PERIOD

ADC Trigger on Timer D period

HRTIM_ADCTRIGGEREVENT13_TIMERE_CMP2

ADC Trigger on Timer E compare 2U

HRTIM_ADCTRIGGEREVENT13_TIMERE_CMP3

ADC Trigger on Timer E compare 3U

HRTIM_ADCTRIGGEREVENT13_TIMERE_CMP4

ADC Trigger on Timer E compare 4U

HRTIM_ADCTRIGGEREVENT13_TIMERE_PERIOD

ADC Trigger on Timer E period

HRTIM_ADCTRIGGEREVENT24_NONE

No ADC trigger event

HRTIM_ADCTRIGGEREVENT24_MASTER_CMP1

ADC Trigger on master compare 1U

HRTIM_ADCTRIGGEREVENT24_MASTER_CMP2

ADC Trigger on master compare 2U

HRTIM_ADCTRIGGEREVENT24_MASTER_CMP3

ADC Trigger on master compare 3U

HRTIM_ADCTRIGGEREVENT24_MASTER_CMP4

ADC Trigger on master compare 4U

HRTIM_ADCTRIGGEREVENT24_MASTER_PERIOD

ADC Trigger on master period

HRTIM_ADCTRIGGEREVENT24_EVENT_6

ADC Trigger on external event 6U

HRTIM_ADCTRIGGEREVENT24_EVENT_7

ADC Trigger on external event 7U

HRTIM_ADCTRIGGEREVENT24_EVENT_8

ADC Trigger on external event 8U

HRTIM_ADCTRIGGEREVENT24_EVENT_9

ADC Trigger on external event 9U

HRTIM_ADCTRIGGEREVENT24_EVENT_10

ADC Trigger on external event 10U

HRTIM_ADCTRIGGEREVENT24_TIMERA_CMP2

ADC Trigger on Timer A compare 2U

HRTIM_ADCTRIGGEREVENT24_TIMERA_CMP3

ADC Trigger on Timer A compare 3U

HRTIM_ADCTRIGGEREVENT24_TIMERA_CMP4

ADC Trigger on Timer A compare 4U

HRTIM_ADCTRIGGEREVENT24_TIMERA_PERIOD

ADC Trigger on Timer A period

HRTIM_ADCTRIGGEREVENT24_TIMERB_CMP2

ADC Trigger on Timer B compare 2U

HRTIM_ADCTRIGGEREVENT24_TIMERB_CMP3

ADC Trigger on Timer B compare 3U

HRTIM_ADCTRIGGEREVENT24_TIMERB_CMP4

ADC Trigger on Timer B compare 4U

HRTIM_ADCTRIGGEREVENT24_TIMERB_PERIOD

ADC Trigger on Timer B period

HRTIM_ADCTRIGGEREVENT24_TIMERC_CMP2

ADC Trigger on Timer C compare 2U

HRTIM_ADCTRIGGEREVENT24_TIMERC_CMP3

ADC Trigger on Timer C compare 3U

HRTIM_ADCTRIGGEREVENT24_TIMERC_CMP4

ADC Trigger on Timer C compare 4U

HRTIM_ADCTRIGGEREVENT24_TIMERC_PERIOD

ADC Trigger on Timer C period

HRTIM_ADCTRIGGEREVENT24_TIMERC_RESET

ADC Trigger on Timer C reset

HRTIM_ADCTRIGGEREVENT24_TIMERD_CMP2

ADC Trigger on Timer D compare 2U

HRTIM_ADCTRIGGEREVENT24_TIMERD_CMP3

ADC Trigger on Timer D compare 3U

HRTIM_ADCTRIGGEREVENT24_TIMERD_CMP4

ADC Trigger on Timer D compare 4U

HRTIM_ADCTRIGGEREVENT24_TIMERD_PERIOD

ADC Trigger on Timer D period

HRTIM_ADCTRIGGEREVENT24_TIMERD_RESET

ADC Trigger on Timer D reset

HRTIM_ADCTRIGGEREVENT24_TIMERE_CMP2

ADC Trigger on Timer E compare 2U

HRTIM_ADCTRIGGEREVENT24_TIMERE_CMP3

ADC Trigger on Timer E compare 3U

HRTIM_ADCTRIGGEREVENT24_TIMERE_CMP4

ADC Trigger on Timer E compare 4U

HRTIM_ADCTRIGGEREVENT24_TIMERE_RESET

ADC Trigger on Timer E reset

HRTIM ADC Trigger Update Source

HRTIM_ADCTRIGGERUPDATE_MASTER

Master timer

HRTIM_ADCTRIGGERUPDATE_TIMER_A

Timer A

HRTIM_ADCTRIGGERUPDATE_TIMER_B

Timer B

HRTIM_ADCTRIGGERUPDATE_TIMER_C

Timer C

HRTIM_ADCTRIGGERUPDATE_TIMER_D

Timer D

HRTIM_ADCTRIGGERUPDATE_TIMER_E

Timer E

HRTIM Burst DMA Registers Update

HRTIM_BURSTDMA_NONE

No register is updated by Burst DMA accesses

HRTIM_BURSTDMA_CR

MCR or TIMxCR register is updated by Burst DMA accesses

HRTIM_BURSTDMA_ICR

MICR or TIMxICR register is updated by Burst DMA accesses

HRTIM_BURSTDMA_DIER

MDIER or TIMxDIER register is updated by Burst DMA accesses

HRTIM_BURSTDMA_CNT

MCNTR or CNTxCR register is updated by Burst DMA accesses

HRTIM_BURSTDMA_PER

MPER or PERxR register is updated by Burst DMA accesses

HRTIM_BURSTDMA_REP

MREPR or REPxR register is updated by Burst DMA accesses

HRTIM_BURSTDMA_CMP1

MCMP1R or CMP1xR register is updated by Burst DMA accesses

HRTIM_BURSTDMA_CMP2

MCMP2R or CMP2xR register is updated by Burst DMA accesses

HRTIM_BURSTDMA_CMP3

MCMP3R or CMP3xR register is updated by Burst DMA accesses

HRTIM_BURSTDMA_CMP4

MCMP4R or CMP4xR register is updated by Burst DMA accesses

HRTIM_BURSTDMA_DTR

TDxR register is updated by Burst DMA accesses

HRTIM_BURSTDMA_SET1R

SET1R register is updated by Burst DMA accesses

HRTIM_BURSTDMA_RST1R

RST1R register is updated by Burst DMA accesses

HRTIM_BURSTDMA_SET2R

SET2R register is updated by Burst DMA accesses

HRTIM_BURSTDMA_RST2R

RST1R register is updated by Burst DMA accesses

HRTIM_BURSTDMA_EEFR1

EEFxR1 register is updated by Burst DMA accesses

HRTIM_BURSTDMA_EEFR2

EEFxR2 register is updated by Burst DMA accesses

HRTIM_BURSTDMA_RSTR

RSTxR register is updated by Burst DMA accesses

HRTIM_BURSTDMA_CHPR

CHPxR register is updated by Burst DMA accesses

HRTIM_BURSTDMA_OUTR

OUTxR register is updated by Burst DMA accesses

HRTIM_BURSTDMA_FLTR

FLTxR register is updated by Burst DMA accesses

HRTIM Burst Mode Clock Source

HRTIM_BURSTMODECLOCKSOURCE_MASTER

Master timer counter reset/roll-over is used as clock source for the burst mode counter

HRTIM_BURSTMODECLOCKSOURCE_TIMER_A

Timer A counter reset/roll-over is used as clock source for the burst mode counter

HRTIM_BURSTMODECLOCKSOURCE_TIMER_B

Timer B counter reset/roll-over is used as clock source for the burst mode counter

HRTIM_BURSTMODECLOCKSOURCE_TIMER_C

Timer C counter reset/roll-over is used as clock source for the burst mode counter

HRTIM_BURSTMODECLOCKSOURCE_TIMER_D

Timer D counter reset/roll-over is used as clock source for the burst mode counter

HRTIM_BURSTMODECLOCKSOURCE_TIMER_E

Timer E counter reset/roll-over is used as clock source for the burst mode counter

HRTIM_BURSTMODECLOCKSOURCE_TIM16_OC

On-chip Event 1 (BMClk[1]), acting as a burst mode counter clock

HRTIM_BURSTMODECLOCKSOURCE_TIM17_OC

On-chip Event 2 (BMClk[2]), acting as a burst mode counter clock

HRTIM_BURSTMODECLOCKSOURCE_TIM7_TRGO

On-chip Event 3 (BMClk[3]), acting as a burst mode counter clock

HRTIM_BURSTMODECLOCKSOURCE_FHRTIM

Prescaled fHRTIM clock is used as clock source for the burst mode counter

HRTIM Burst Mode Control

HRTIM_BURSTMODECTL_DISABLED

Burst mode disabled

HRTIM_BURSTMODECTL_ENABLED

Burst mode enabled

HRTIM Burst Mode Operating Mode

HRTIM_BURSTMODE_SINGLESHOT

Burst mode operates in single shot mode

HRTIM_BURSTMODE_CONTINOUS

Burst mode operates in continuous mode

HRTIM Burst Mode Prescaler

HRTIM_BURSTMODEPRESCALER_DIV1

fBRST = fHRTIM

HRTIM_BURSTMODEPRESCALER_DIV2

fBRST = fHRTIM/2U

HRTIM_BURSTMODEPRESCALER_DIV4

fBRST = fHRTIM/4U

HRTIM_BURSTMODEPRESCALER_DIV8

fBRST = fHRTIM/8U

HRTIM_BURSTMODEPRESCALER_DIV16

fBRST = fHRTIM/16U

HRTIM_BURSTMODEPRESCALER_DIV32

fBRST = fHRTIM/32U

HRTIM_BURSTMODEPRESCALER_DIV64

fBRST = fHRTIM/64U

HRTIM_BURSTMODEPRESCALER_DIV128

fBRST = fHRTIM/128U

HRTIM_BURSTMODEPRESCALER_DIV256

fBRST = fHRTIM/256U

HRTIM_BURSTMODEPRESCALER_DIV512

fBRST = fHRTIM/512U

HRTIM_BURSTMODEPRESCALER_DIV1024

fBRST = fHRTIM/1024U

HRTIM_BURSTMODEPRESCALER_DIV2048

fBRST = fHRTIM/2048U

HRTIM_BURSTMODEPRESCALER_DIV4096

fBRST = fHRTIM/4096U

HRTIM_BURSTMODEPRESCALER_DIV8192

fBRST = fHRTIM/8192U

HRTIM_BURSTMODEPRESCALER_DIV16384

fBRST = fHRTIM/16384U

HRTIM_BURSTMODEPRESCALER_DIV32768

fBRST = fHRTIM/32768U

HRTIM Burst Mode Register Preload Enable

HRIM_BURSTMODEPRELOAD_DISABLED

Preload disabled: the write access is directly done into active registers

HRIM_BURSTMODEPRELOAD_ENABLED

Preload enabled: the write access is done into preload registers

HRTIM Burst Mode Status

HRTIM_BURSTMODESTATUS_NORMAL

Normal operation

HRTIM_BURSTMODESTATUS_ONGOING

Burst operation on-going

HRTIM Burst Mode Trigger

HRTIM_BURSTMODETRIGGER_NONE

No trigger

HRTIM_BURSTMODETRIGGER_MASTER_RESET

Master reset

HRTIM_BURSTMODETRIGGER_MASTER_REPETITION

Master repetition

HRTIM_BURSTMODETRIGGER_MASTER_CMP1

Master compare 1U

HRTIM_BURSTMODETRIGGER_MASTER_CMP2

Master compare 2U

HRTIM_BURSTMODETRIGGER_MASTER_CMP3

Master compare 3U

HRTIM_BURSTMODETRIGGER_MASTER_CMP4

Master compare 4U

HRTIM_BURSTMODETRIGGER_TIMERA_RESET

Timer A reset

HRTIM_BURSTMODETRIGGER_TIMERA_REPETITION

Timer A repetition

HRTIM_BURSTMODETRIGGER_TIMERA_CMP1

Timer A compare 1

HRTIM_BURSTMODETRIGGER_TIMERA_CMP2

Timer A compare 2

HRTIM_BURSTMODETRIGGER_TIMERB_RESET

Timer B reset

HRTIM_BURSTMODETRIGGER_TIMERB_REPETITION

Timer B repetition

HRTIM_BURSTMODETRIGGER_TIMERB_CMP1

Timer B compare 1

HRTIM_BURSTMODETRIGGER_TIMERB_CMP2

Timer B compare 2

HRTIM_BURSTMODETRIGGER_TIMERC_RESET

Timer C reset

HRTIM_BURSTMODETRIGGER_TIMERC_REPETITION

Timer C repetition

HRTIM_BURSTMODETRIGGER_TIMERC_CMP1

Timer C compare 1

HRTIM_BURSTMODETRIGGER_TIMERC_CMP2

Timer C compare 2

HRTIM_BURSTMODETRIGGER_TIMERD_RESET

Timer D reset

HRTIM_BURSTMODETRIGGER_TIMERD_REPETITION

Timer D repetition

HRTIM_BURSTMODETRIGGER_TIMERD_CMP1

Timer D compare 1

HRTIM_BURSTMODETRIGGER_TIMERD_CMP2

Timer D compare 2

HRTIM_BURSTMODETRIGGER_TIMERE_RESET

Timer E reset

HRTIM_BURSTMODETRIGGER_TIMERE_REPETITION

Timer E repetition

HRTIM_BURSTMODETRIGGER_TIMERE_CMP1

Timer E compare 1

HRTIM_BURSTMODETRIGGER_TIMERE_CMP2

Timer E compare 2

HRTIM_BURSTMODETRIGGER_TIMERA_EVENT7

Timer A period following External Event 7

HRTIM_BURSTMODETRIGGER_TIMERD_EVENT8

Timer D period following External Event 8

HRTIM_BURSTMODETRIGGER_EVENT_7

External Event 7 (timer A filters applied)

HRTIM_BURSTMODETRIGGER_EVENT_8

External Event 8 (timer D filters applied)

HRTIM_BURSTMODETRIGGER_EVENT_ONCHIP

On-chip Event

HRTIM Capture Unit

HRTIM_CAPTUREUNIT_1

Capture unit 1 identifier

HRTIM_CAPTUREUNIT_2

Capture unit 2 identifier

HRTIM Capture Unit Trigger

HRTIM_CAPTURETRIGGER_NONE

Capture trigger is disabled

HRTIM_CAPTURETRIGGER_UPDATE

The update event triggers the Capture

HRTIM_CAPTURETRIGGER_EEV_1

The External event 1 triggers the Capture

HRTIM_CAPTURETRIGGER_EEV_2

The External event 2 triggers the Capture

HRTIM_CAPTURETRIGGER_EEV_3

The External event 3 triggers the Capture

HRTIM_CAPTURETRIGGER_EEV_4

The External event 4 triggers the Capture

HRTIM_CAPTURETRIGGER_EEV_5

The External event 5 triggers the Capture

HRTIM_CAPTURETRIGGER_EEV_6

The External event 6 triggers the Capture

HRTIM_CAPTURETRIGGER_EEV_7

The External event 7 triggers the Capture

HRTIM_CAPTURETRIGGER_EEV_8

The External event 8 triggers the Capture

HRTIM_CAPTURETRIGGER_EEV_9

The External event 9 triggers the Capture

HRTIM_CAPTURETRIGGER_EEV_10

The External event 10 triggers the Capture

HRTIM_CAPTURETRIGGER_TA1_SET

Capture is triggered by TA1 output inactive to active transition

HRTIM_CAPTURETRIGGER_TA1_RESET

Capture is triggered by TA1 output active to inactive transition

HRTIM_CAPTURETRIGGER_TIMERA_CMP1

Timer A Compare 1 triggers Capture

HRTIM_CAPTURETRIGGER_TIMERA_CMP2

Timer A Compare 2 triggers Capture

HRTIM_CAPTURETRIGGER_TB1_SET

Capture is triggered by TB1 output inactive to active transition

HRTIM_CAPTURETRIGGER_TB1_RESET

Capture is triggered by TB1 output active to inactive transition

HRTIM_CAPTURETRIGGER_TIMERB_CMP1

Timer B Compare 1 triggers Capture

HRTIM_CAPTURETRIGGER_TIMERB_CMP2

Timer B Compare 2 triggers Capture

HRTIM_CAPTURETRIGGER_TC1_SET

Capture is triggered by TC1 output inactive to active transition

HRTIM_CAPTURETRIGGER_TC1_RESET

Capture is triggered by TC1 output active to inactive transition

HRTIM_CAPTURETRIGGER_TIMERC_CMP1

Timer C Compare 1 triggers Capture

HRTIM_CAPTURETRIGGER_TIMERC_CMP2

Timer C Compare 2 triggers Capture

HRTIM_CAPTURETRIGGER_TD1_SET

Capture is triggered by TD1 output inactive to active transition

HRTIM_CAPTURETRIGGER_TD1_RESET

Capture is triggered by TD1 output active to inactive transition

HRTIM_CAPTURETRIGGER_TIMERD_CMP1

Timer D Compare 1 triggers Capture

HRTIM_CAPTURETRIGGER_TIMERD_CMP2

Timer D Compare 2 triggers Capture

HRTIM_CAPTURETRIGGER_TE1_SET

Capture is triggered by TE1 output inactive to active transition

HRTIM_CAPTURETRIGGER_TE1_RESET

Capture is triggered by TE1 output active to inactive transition

HRTIM_CAPTURETRIGGER_TIMERE_CMP1

Timer E Compare 1 triggers Capture

HRTIM_CAPTURETRIGGER_TIMERE_CMP2

Timer E Compare 2 triggers Capture

HRTIM Chopper Duty Cycle

HRTIM_CHOPPER_DUTYCYCLE_0

Only 1st pulse is present

HRTIM_CHOPPER_DUTYCYCLE_125

Duty cycle of the carrier signal is 12.5U %

HRTIM_CHOPPER_DUTYCYCLE_250

Duty cycle of the carrier signal is 25U %

HRTIM_CHOPPER_DUTYCYCLE_375

Duty cycle of the carrier signal is 37.5U %

HRTIM_CHOPPER_DUTYCYCLE_500

Duty cycle of the carrier signal is 50U %

HRTIM_CHOPPER_DUTYCYCLE_625

Duty cycle of the carrier signal is 62.5U %

HRTIM_CHOPPER_DUTYCYCLE_750

Duty cycle of the carrier signal is 75U %

HRTIM_CHOPPER_DUTYCYCLE_875

Duty cycle of the carrier signal is 87.5U %

HRTIM Chopper Frequency

HRTIM_CHOPPER_PRESCALERRATIO_DIV16

fCHPFRQ = fHRTIM / 16

HRTIM_CHOPPER_PRESCALERRATIO_DIV32

fCHPFRQ = fHRTIM / 32

HRTIM_CHOPPER_PRESCALERRATIO_DIV48

fCHPFRQ = fHRTIM / 48

HRTIM_CHOPPER_PRESCALERRATIO_DIV64

fCHPFRQ = fHRTIM / 64

HRTIM_CHOPPER_PRESCALERRATIO_DIV80

fCHPFRQ = fHRTIM / 80

HRTIM_CHOPPER_PRESCALERRATIO_DIV96

fCHPFRQ = fHRTIM / 96

HRTIM_CHOPPER_PRESCALERRATIO_DIV112

fCHPFRQ = fHRTIM / 112

HRTIM_CHOPPER_PRESCALERRATIO_DIV128

fCHPFRQ = fHRTIM / 128

HRTIM_CHOPPER_PRESCALERRATIO_DIV144

fCHPFRQ = fHRTIM / 144

HRTIM_CHOPPER_PRESCALERRATIO_DIV160

fCHPFRQ = fHRTIM / 160

HRTIM_CHOPPER_PRESCALERRATIO_DIV176

fCHPFRQ = fHRTIM / 176

HRTIM_CHOPPER_PRESCALERRATIO_DIV192

fCHPFRQ = fHRTIM / 192

HRTIM_CHOPPER_PRESCALERRATIO_DIV208

fCHPFRQ = fHRTIM / 208

HRTIM_CHOPPER_PRESCALERRATIO_DIV224

fCHPFRQ = fHRTIM / 224

HRTIM_CHOPPER_PRESCALERRATIO_DIV240

fCHPFRQ = fHRTIM / 240

HRTIM_CHOPPER_PRESCALERRATIO_DIV256

fCHPFRQ = fHRTIM / 256

HRTIM Chopper Start Pulse Width

HRTIM_CHOPPER_PULSEWIDTH_16

tSTPW = tHRTIM x 16

HRTIM_CHOPPER_PULSEWIDTH_32

tSTPW = tHRTIM x 32

HRTIM_CHOPPER_PULSEWIDTH_48

tSTPW = tHRTIM x 48

HRTIM_CHOPPER_PULSEWIDTH_64

tSTPW = tHRTIM x 64

HRTIM_CHOPPER_PULSEWIDTH_80

tSTPW = tHRTIM x 80

HRTIM_CHOPPER_PULSEWIDTH_96

tSTPW = tHRTIM x 96

HRTIM_CHOPPER_PULSEWIDTH_112

tSTPW = tHRTIM x 112

HRTIM_CHOPPER_PULSEWIDTH_128

tSTPW = tHRTIM x 128

HRTIM_CHOPPER_PULSEWIDTH_144

tSTPW = tHRTIM x 144

HRTIM_CHOPPER_PULSEWIDTH_160

tSTPW = tHRTIM x 160

HRTIM_CHOPPER_PULSEWIDTH_176

tSTPW = tHRTIM x 176

HRTIM_CHOPPER_PULSEWIDTH_192

tSTPW = tHRTIM x 192

HRTIM_CHOPPER_PULSEWIDTH_208

tSTPW = tHRTIM x 208

HRTIM_CHOPPER_PULSEWIDTH_224

tSTPW = tHRTIM x 224

HRTIM_CHOPPER_PULSEWIDTH_240

tSTPW = tHRTIM x 240

HRTIM_CHOPPER_PULSEWIDTH_256

tSTPW = tHRTIM x 256

HRTIM Common Interrupt Enable

HRTIM_IT_NONE

No interrupt enabled

HRTIM_IT_FLT1

Fault 1 interrupt enable

HRTIM_IT_FLT2

Fault 2 interrupt enable

HRTIM_IT_FLT3

Fault 3 interrupt enable

HRTIM_IT_FLT4

Fault 4 interrupt enable

HRTIM_IT_FLT5

Fault 5 interrupt enable

HRTIM_IT_SYSFLT

System Fault interrupt enable

HRTIM_IT_BMPER

Burst mode period interrupt enable

HRTIM Common Interrupt Flag

HRTIM_FLAG_FLT1

Fault 1 interrupt flag

HRTIM_FLAG_FLT2

Fault 2 interrupt flag

HRTIM_FLAG_FLT3

Fault 3 interrupt flag

HRTIM_FLAG_FLT4

Fault 4 interrupt flag

HRTIM_FLAG_FLT5

Fault 5 interrupt flag

HRTIM_FLAG_SYSFLT

System Fault interrupt flag

HRTIM_FLAG_BMPER

Burst mode period interrupt flag

HRTIM Compare Unit

HRTIM_COMPAREUNIT_1

Compare unit 1 identifier

HRTIM_COMPAREUNIT_2

Compare unit 2 identifier

HRTIM_COMPAREUNIT_3

Compare unit 3 identifier

HRTIM_COMPAREUNIT_4

Compare unit 4 identifier

HRTIM Compare Unit Auto Delayed Mode

HRTIM_AUTODELAYEDMODE_REGULAR

standard compare mode

HRTIM_AUTODELAYEDMODE_AUTODELAYED_NOTIMEOUT

Compare event generated only if a capture has occurred

HRTIM_AUTODELAYEDMODE_AUTODELAYED_TIMEOUTCMP1

Compare event generated if a capture has occurred or after a Compare 1 match (timeout if capture event is missing)

HRTIM_AUTODELAYEDMODE_AUTODELAYED_TIMEOUTCMP3

Compare event generated if a capture has occurred or after a Compare 3 match (timeout if capture event is missing)

HRTIM Counter Operating Mode

HRTIM_MODE_CONTINUOUS

The timer operates in continuous (free-running) mode

HRTIM_MODE_SINGLESHOT

The timer operates in non retriggerable single-shot mode

HRTIM_MODE_SINGLESHOT_RETRIGGERABLE

The timer operates in retriggerable single-shot mode

HRTIM Current Push Pull Status

HRTIM_PUSHPULL_CURRENTSTATUS_OUTPUT1

Signal applied on output 1 and output 2 forced inactive

HRTIM_PUSHPULL_CURRENTSTATUS_OUTPUT2

Signal applied on output 2 and output 1 forced inactive

HRTIM DAC Synchronization

HRTIM_DACSYNC_NONE

No DAC synchronization event generated

HRTIM_DACSYNC_DACTRIGOUT_1

DAC synchronization event generated on DACTrigOut1 output upon timer update

HRTIM_DACSYNC_DACTRIGOUT_2

DAC synchronization event generated on DACTrigOut2 output upon timer update

HRTIM_DACSYNC_DACTRIGOUT_3

DAC update generated on DACTrigOut3 output upon timer update

HRTIM Dead-time Falling Lock

HRTIM_TIMDEADTIME_FALLINGLOCK_WRITE

Dead-time falling value and sign is writeable

HRTIM_TIMDEADTIME_FALLINGLOCK_READONLY

Dead-time falling value and sign is read-only

HRTIM Dead-time Falling Sign

HRTIM_TIMDEADTIME_FALLINGSIGN_POSITIVE

Positive dead-time on falling edge

HRTIM_TIMDEADTIME_FALLINGSIGN_NEGATIVE

Negative dead-time on falling edge

HRTIM Dead-time Falling Sign Lock

HRTIM_TIMDEADTIME_FALLINGSIGNLOCK_WRITE

Dead-time falling sign is writeable

HRTIM_TIMDEADTIME_FALLINGSIGNLOCK_READONLY

Dead-time falling sign is read-only

HRTIM Dead-time Prescaler Ratio

HRTIM_TIMDEADTIME_PRESCALERRATIO_DIV1

fDTG = fHRTIM

HRTIM_TIMDEADTIME_PRESCALERRATIO_DIV2

fDTG = fHRTIM / 2U

HRTIM_TIMDEADTIME_PRESCALERRATIO_DIV4

fDTG = fHRTIM / 4U

HRTIM_TIMDEADTIME_PRESCALERRATIO_DIV8

fDTG = fHRTIM / 8U

HRTIM_TIMDEADTIME_PRESCALERRATIO_DIV16

fDTG = fHRTIM / 16U

HRTIM Dead-time Rising Lock

HRTIM_TIMDEADTIME_RISINGLOCK_WRITE

Dead-time rising value and sign is writeable

HRTIM_TIMDEADTIME_RISINGLOCK_READONLY

Dead-time rising value and sign is read-only

HRTIM Dead-time Rising Sign

HRTIM_TIMDEADTIME_RISINGSIGN_POSITIVE

Positive dead-time on rising edge

HRTIM_TIMDEADTIME_RISINGSIGN_NEGATIVE

Negative dead-time on rising edge

HRTIM Dead-time Rising Sign Lock

HRTIM_TIMDEADTIME_RISINGSIGNLOCK_WRITE

Dead-time rising sign is writeable

HRTIM_TIMDEADTIME_RISINGSIGNLOCK_READONLY

Dead-time rising sign is read-only

HRTIM Exported Macros

__HAL_HRTIM_RESET_HANDLE_STATE

Description:

- Reset HRTIM handle state.

Parameters:

- __HANDLE__: HRTIM handle.

Return value:

- None

__HAL_HRTIM_ENABLE

Description:

- Enables or disables the timer counter(s)

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __TIMERS__: timers to enable/disable This parameter can be any combinations of the following values:
  - HRTIM_TIMERID_MASTER: Master timer identifier
  - HRTIM_TIMERID_TIMER_A: Timer A identifier
  - HRTIM_TIMERID_TIMER_B: Timer B identifier
  - HRTIM_TIMERID_TIMER_C: Timer C identifier
  - HRTIM_TIMERID_TIMER_D: Timer D identifier
  - HRTIM_TIMERID_TIMER_E: Timer E identifier

Return value:

- None

HRTIM_TAOEN_MASK

HRTIM_TBOEN_MASK

HRTIM_TCOEN_MASK

HRTIM_TDOEN_MASK

HRTIM_TEOEN_MASK

__HAL_HRTIM_DISABLE

__HAL_HRTIM_ENABLE_IT

Description:

- Enables or disables the specified HRTIM common interrupts.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - HRTIM_IT_FLT1: Fault 1 interrupt enable
  - HRTIM_IT_FLT2: Fault 2 interrupt enable
  - HRTIM_IT_FLT3: Fault 3 interrupt enable
  - HRTIM_IT_FLT4: Fault 4 interrupt enable
  - HRTIM_IT_FLT5: Fault 5 interrupt enable
  - HRTIM_IT_SYSFLT: System Fault interrupt enable
  - HRTIM_IT_BMPER: Burst mode period interrupt enable

Return value:

- None

__HAL_HRTIM_DISABLE_IT

__HAL_HRTIM_MASTER_ENABLE_IT

Description:

- Enables or disables the specified HRTIM Master timer interrupts.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - HRTIM_MASTER_IT_MCMP1: Master compare 1 interrupt enable
  - HRTIM_MASTER_IT_MCMP2: Master compare 2 interrupt enable
  - HRTIM_MASTER_IT_MCMP3: Master compare 3 interrupt enable
  - HRTIM_MASTER_IT_MCMP4: Master compare 4 interrupt enable
  - HRTIM_MASTER_IT_MREP: Master Repetition interrupt enable
  - HRTIM_MASTER_IT_SYNC: Synchronization input interrupt enable
  - HRTIM_MASTER_IT_MUPD: Master update interrupt enable

Return value:

- None

__HAL_HRTIM_MASTER_DISABLE_IT

__HAL_HRTIM_TIMER_ENABLE_IT

Description:

- Enables or disables the specified HRTIM Timerx interrupts.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __TIMER__: specified the timing unit (Timer A to E)
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - HRTIM_TIM_IT_CMP1: Timer compare 1 interrupt enable
  - HRTIM_TIM_IT_CMP2: Timer compare 2 interrupt enable
  - HRTIM_TIM_IT_CMP3: Timer compare 3 interrupt enable
  - HRTIM_TIM_IT_CMP4: Timer compare 4 interrupt enable
  - HRTIM_TIM_IT_REP: Timer repetition interrupt enable
  - HRTIM_TIM_IT_UPD: Timer update interrupt enable
  - HRTIM_TIM_IT_CPT1: Timer capture 1 interrupt enable
  - HRTIM_TIM_IT_CPT2: Timer capture 2 interrupt enable
  - HRTIM_TIM_IT_SET1: Timer output 1 set interrupt enable
  - HRTIM_TIM_IT_RST1: Timer output 1 reset interrupt enable
  - HRTIM_TIM_IT_SET2: Timer output 2 set interrupt enable
  - HRTIM_TIM_IT_RST2: Timer output 2 reset interrupt enable
  - HRTIM_TIM_IT_RST: Timer reset interrupt enable
  - HRTIM_TIM_IT_DLYPRT: Timer delay protection interrupt enable

Return value:

- None

__HAL_HRTIM_TIMER_DISABLE_IT

__HAL_HRTIM_GET_ITSTATUS

Description:

- Checks if the specified HRTIM common interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __INTERRUPT__: specifies the interrupt source to check. This parameter can be one of the following values:
  - HRTIM_IT_FLT1: Fault 1 interrupt enable
  - HRTIM_IT_FLT2: Fault 2 interrupt enable
  - HRTIM_IT_FLT3: Fault 3 enable
  - HRTIM_IT_FLT4: Fault 4 enable
  - HRTIM_IT_FLT5: Fault 5 enable
  - HRTIM_IT_SYSFLT: System Fault interrupt enable
  - HRTIM_IT_BMPER: Burst mode period interrupt enable

Return value:

- The: new state of __INTERRUPT__ (TRUE or FALSE).

__HAL_HRTIM_MASTER_GET_ITSTATUS

Description:

- Checks if the specified HRTIM Master interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __INTERRUPT__: specifies the interrupt source to check. This parameter can be one of the following values:
  - HRTIM_MASTER_IT_MCMP1: Master compare 1 interrupt enable
  - HRTIM_MASTER_IT_MCMP2: Master compare 2 interrupt enable
  - HRTIM_MASTER_IT_MCMP3: Master compare 3 interrupt enable
  - HRTIM_MASTER_IT_MCMP4: Master compare 4 interrupt enable
  - HRTIM_MASTER_IT_MREP: Master Repetition interrupt enable
  - HRTIM_MASTER_IT_SYNC: Synchronization input interrupt enable
  - HRTIM_MASTER_IT_MUPD: Master update interrupt enable

Return value:

- The: new state of __INTERRUPT__ (TRUE or FALSE).

__HAL_HRTIM_TIMER_GET_ITSTATUS

Description:

- Checks if the specified HRTIM Timerx interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __TIMER__: specified the timing unit (Timer A to E)
- __INTERRUPT__: specifies the interrupt source to check. This parameter can be one of the following values:
  - HRTIM_MASTER_IT_MCMP1: Master compare 1 interrupt enable
  - HRTIM_MASTER_IT_MCMP2: Master compare 2 interrupt enable
  - HRTIM_MASTER_IT_MCMP3: Master compare 3 interrupt enable
  - HRTIM_MASTER_IT_MCMP4: Master compare 4 interrupt enable
  - HRTIM_MASTER_IT_MREP: Master Repetition interrupt enable
  - HRTIM_MASTER_IT_SYNC: Synchronization input interrupt enable
  - HRTIM_MASTER_IT_MUPD: Master update interrupt enable
  - HRTIM_TIM_IT_CMP1: Timer compare 1 interrupt enable
  - HRTIM_TIM_IT_CMP2: Timer compare 2 interrupt enable
  - HRTIM_TIM_IT_CMP3: Timer compare 3 interrupt enable
  - HRTIM_TIM_IT_CMP4: Timer compare 4 interrupt enable
  - HRTIM_TIM_IT_REP: Timer repetition interrupt enable
  - HRTIM_TIM_IT_UPD: Timer update interrupt enable
  - HRTIM_TIM_IT_CPT1: Timer capture 1 interrupt enable
  - HRTIM_TIM_IT_CPT2: Timer capture 2 interrupt enable
  - HRTIM_TIM_IT_SET1: Timer output 1 set interrupt enable
  - HRTIM_TIM_IT_RST1: Timer output 1 reset interrupt enable
  - HRTIM_TIM_IT_SET2: Timer output 2 set interrupt enable
  - HRTIM_TIM_IT_RST2: Timer output 2 reset interrupt enable
  - HRTIM_TIM_IT_RST: Timer reset interrupt enable
  - HRTIM_TIM_IT_DLYPRT: Timer delay protection interrupt enable

Return value:

- The: new state of __INTERRUPT__ (TRUE or FALSE).

__HAL_HRTIM_CLEAR_IT

Description:

- Clears the specified HRTIM common pending flag.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
  - HRTIM_IT_FLT1: Fault 1 interrupt clear flag
  - HRTIM_IT_FLT2: Fault 2 interrupt clear flag
  - HRTIM_IT_FLT3: Fault 3 clear flag
  - HRTIM_IT_FLT4: Fault 4 clear flag
  - HRTIM_IT_FLT5: Fault 5 clear flag
  - HRTIM_IT_SYSFLT: System Fault interrupt clear flag
  - HRTIM_IT_BMPER: Burst mode period interrupt clear flag

Return value:

- None

__HAL_HRTIM_MASTER_CLEAR_IT

Description:

- Clears the specified HRTIM Master pending flag.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
  - HRTIM_MASTER_IT_MCMP1: Master compare 1 interrupt clear flag
  - HRTIM_MASTER_IT_MCMP2: Master compare 2 interrupt clear flag
  - HRTIM_MASTER_IT_MCMP3: Master compare 3 interrupt clear flag
  - HRTIM_MASTER_IT_MCMP4: Master compare 4 interrupt clear flag
  - HRTIM_MASTER_IT_MREP: Master Repetition interrupt clear flag
  - HRTIM_MASTER_IT_SYNC: Synchronization input interrupt clear flag
  - HRTIM_MASTER_IT_MUPD: Master update interrupt clear flag

Return value:

- None

__HAL_HRTIM_TIMER_CLEAR_IT

Description:

- Clears the specified HRTIM Timerx pending flag.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __TIMER__: specified the timing unit (Timer A to E)
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
  - HRTIM_TIM_IT_CMP1: Timer compare 1 interrupt clear flag
  - HRTIM_TIM_IT_CMP2: Timer compare 2 interrupt clear flag
  - HRTIM_TIM_IT_CMP3: Timer compare 3 interrupt clear flag
  - HRTIM_TIM_IT_CMP4: Timer compare 4 interrupt clear flag
  - HRTIM_TIM_IT_REP: Timer repetition interrupt clear flag
  - HRTIM_TIM_IT_UPD: Timer update interrupt clear flag
  - HRTIM_TIM_IT_CPT1: Timer capture 1 interrupt clear flag
  - HRTIM_TIM_IT_CPT2: Timer capture 2 interrupt clear flag
  - HRTIM_TIM_IT_SET1: Timer output 1 set interrupt clear flag
  - HRTIM_TIM_IT_RST1: Timer output 1 reset interrupt clear flag
  - HRTIM_TIM_IT_SET2: Timer output 2 set interrupt clear flag
  - HRTIM_TIM_IT_RST2: Timer output 2 reset interrupt clear flag
  - HRTIM_TIM_IT_RST: Timer reset interrupt clear flag
  - HRTIM_TIM_IT_DLYPRT: Timer output 1 delay protection interrupt clear flag

Return value:

- None

__HAL_HRTIM_MASTER_ENABLE_DMA

Description:

- Enables or disables the specified HRTIM Master timer DMA requests.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __DMA__: specifies the DMA request to enable or disable. This parameter can be one of the following values:
  - HRTIM_MASTER_DMA_MCMP1: Master compare 1 DMA request enable
  - HRTIM_MASTER_DMA_MCMP2: Master compare 2 DMA request enable
  - HRTIM_MASTER_DMA_MCMP3: Master compare 3 DMA request enable
  - HRTIM_MASTER_DMA_MCMP4: Master compare 4 DMA request enable
  - HRTIM_MASTER_DMA_MREP: Master Repetition DMA request enable
  - HRTIM_MASTER_DMA_SYNC: Synchronization input DMA request enable
  - HRTIM_MASTER_DMA_MUPD: Master update DMA request enable

Return value:

- None

__HAL_HRTIM_MASTER_DISABLE_DMA

__HAL_HRTIM_TIMER_ENABLE_DMA

Description:

- Enables or disables the specified HRTIM Timerx DMA requests.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __TIMER__: specified the timing unit (Timer A to E)
- __DMA__: specifies the DMA request to enable or disable. This parameter can be one of the following values:
  - HRTIM_TIM_DMA_CMP1: Timer compare 1 DMA request enable
  - HRTIM_TIM_DMA_CMP2: Timer compare 2 DMA request enable
  - HRTIM_TIM_DMA_CMP3: Timer compare 3 DMA request enable
  - HRTIM_TIM_DMA_CMP4: Timer compare 4 DMA request enable
  - HRTIM_TIM_DMA_REP: Timer repetition DMA request enable
  - HRTIM_TIM_DMA_UPD: Timer update DMA request enable
  - HRTIM_TIM_DMA_CPT1: Timer capture 1 DMA request enable
  - HRTIM_TIM_DMA_CPT2: Timer capture 2 DMA request enable
  - HRTIM_TIM_DMA_SET1: Timer output 1 set DMA request enable
  - HRTIM_TIM_DMA_RST1: Timer output 1 reset DMA request enable
  - HRTIM_TIM_DMA_SET2: Timer output 2 set DMA request enable
  - HRTIM_TIM_DMA_RST2: Timer output 2 reset DMA request enable
  - HRTIM_TIM_DMA_RST: Timer reset DMA request enable
  - HRTIM_TIM_DMA_DLYPRT: Timer delay protection DMA request enable

Return value:

- None

__HAL_HRTIM_TIMER_DISABLE_DMA

__HAL_HRTIM_GET_FLAG

__HAL_HRTIM_CLEAR_FLAG

__HAL_HRTIM_MASTER_GET_FLAG

__HAL_HRTIM_MASTER_CLEAR_FLAG

__HAL_HRTIM_TIMER_GET_FLAG

__HAL_HRTIM_TIMER_CLEAR_FLAG

__HAL_HRTIM_SETCOUNTER

Description:

- Sets the HRTIM timer Counter Register value on runtime.

Parameters:

- __HANDLE__: HRTIM Handle.
- __TIMER__: HRTIM timer This parameter can be one of the following values:
  - 0x5 for master timer
  - 0x0 to 0x4 for timers A to E
- __COUNTER__: specifies the Counter Register new value.

Return value:

- None

__HAL_HRTIM_GETCOUNTER

Description:

- Gets the HRTIM timer Counter Register value on runtime.

Parameters:

- __HANDLE__: HRTIM Handle.
- __TIMER__: HRTIM timer This parameter can be one of the following values:
  - 0x5 for master timer
  - 0x0 to 0x4 for timers A to E

Return value:

- HRTIM: timer Counter Register value

__HAL_HRTIM_SETPERIOD

Description:

- Sets the HRTIM timer Period value on runtime.

Parameters:

- __HANDLE__: HRTIM Handle.
- __TIMER__: HRTIM timer This parameter can be one of the following values:
  - 0x5 for master timer
  - 0x0 to 0x4 for timers A to E
- __PERIOD__: specifies the Period Register new value.

Return value:

- None

__HAL_HRTIM_GETPERIOD

Description:

- Gets the HRTIM timer Period Register value on runtime.

Parameters:

- __HANDLE__: HRTIM Handle.
- __TIMER__: HRTIM timer This parameter can be one of the following values:
  - 0x5 for master timer
  - 0x0 to 0x4 for timers A to E

Return value:

- timer: Period Register

__HAL_HRTIM_SETCLOCKPRESCALER

Description:

- Sets the HRTIM timer clock prescaler value on runtime.

Parameters:

- __HANDLE__: HRTIM Handle.
- __TIMER__: HRTIM timer This parameter can be one of the following values:
  - 0x5 for master timer
  - 0x0 to 0x4 for timers A to E
- __PRESCALER__: specifies the clock prescaler new value. This parameter can be one of the following values:
  - HRTIM_PRESCALERRATIO_DIV1: fHRCK: 144 MHz - Resolution: 6.95 ns - Min PWM frequency: 2.2 kHz (fHRTIM=144MHz)
  - HRTIM_PRESCALERRATIO_DIV2: fHRCK: 72 MHz - Resolution: 13.88 ns- Min PWM frequency: 1.1 kHz (fHRTIM=144MHz)
  - HRTIM_PRESCALERRATIO_DIV4: fHRCK: 36 MHz - Resolution: 27.7 ns- Min PWM frequency: 550Hz (fHRTIM=144MHz)

Return value:

- None

__HAL_HRTIM_GETCLOCKPRESCALER

Description:

- Gets the HRTIM timer clock prescaler value on runtime.

Parameters:

- __HANDLE__: HRTIM Handle.
- __TIMER__: HRTIM timer This parameter can be one of the following values:
  - 0x5 for master timer
  - 0x0 to 0x4 for timers A to E

Return value:

- timer: clock prescaler value

__HAL_HRTIM_SETCOMPARE

Description:

- Sets the HRTIM timer Compare Register value on runtime.

Parameters:

- __HANDLE__: HRTIM Handle.
- __TIMER__: HRTIM timer This parameter can be one of the following values:
  - 0x0 to 0x4 for timers A to E
- __COMPAREUNIT__: timer compare unit This parameter can be one of the following values:
  - HRTIM_COMPAREUNIT_1: Compare unit 1
  - HRTIM_COMPAREUNIT_2: Compare unit 2
  - HRTIM_COMPAREUNIT_3: Compare unit 3
  - HRTIM_COMPAREUNIT_4: Compare unit 4
- __COMPARE__: specifies the Compare new value.

Return value:

- None

__HAL_HRTIM_GETCOMPARE

Description:

- Gets the HRTIM timer Compare Register value on runtime.

Parameters:

- __HANDLE__: HRTIM Handle.
- __TIMER__: HRTIM timer This parameter can be one of the following values:
  - 0x0 to 0x4 for timers A to E
- __COMPAREUNIT__: timer compare unit This parameter can be one of the following values:
  - HRTIM_COMPAREUNIT_1: Compare unit 1
  - HRTIM_COMPAREUNIT_2: Compare unit 2
  - HRTIM_COMPAREUNIT_3: Compare unit 3
  - HRTIM_COMPAREUNIT_4: Compare unit 4

Return value:

- Compare: value

HRTIM External Event Channels

HRTIM_EVENT_NONE

Undefined event channel

HRTIM_EVENT_1

External event channel 1 identifier

HRTIM_EVENT_2

External event channel 2 identifier

HRTIM_EVENT_3

External event channel 3 identifier

HRTIM_EVENT_4

External event channel 4 identifier

HRTIM_EVENT_5

External event channel 5 identifier

HRTIM_EVENT_6

External event channel 6 identifier

HRTIM_EVENT_7

External event channel 7 identifier

HRTIM_EVENT_8

External event channel 8 identifier

HRTIM_EVENT_9

External event channel 9 identifier

HRTIM_EVENT_10

External event channel 10 identifier

HRTIM External Event Fast Mode

HRTIM_EVENTFASTMODE_DISABLE

External Event is re-synchronized by the HRTIM logic before acting on outputs

HRTIM_EVENTFASTMODE_ENABLE

External Event is acting asynchronously on outputs (low latency mode)

HRTIM External Event Filter

HRTIM_EVENTFILTER_NONE

Filter disabled

HRTIM_EVENTFILTER_1

fSAMPLING= fHRTIM, N=2U

HRTIM_EVENTFILTER_2

fSAMPLING= fHRTIM, N=4U

HRTIM_EVENTFILTER_3

fSAMPLING= fHRTIM, N=8U

HRTIM_EVENTFILTER_4

fSAMPLING= fEEVS/2U, N=6U

HRTIM_EVENTFILTER_5

fSAMPLING= fEEVS/2U, N=8U

HRTIM_EVENTFILTER_6

fSAMPLING= fEEVS/4U, N=6U

HRTIM_EVENTFILTER_7

fSAMPLING= fEEVS/4U, N=8U

HRTIM_EVENTFILTER_8

fSAMPLING= fEEVS/8U, N=6U

HRTIM_EVENTFILTER_9

fSAMPLING= fEEVS/8U, N=8U

HRTIM_EVENTFILTER_10

fSAMPLING= fEEVS/16U, N=5U

HRTIM_EVENTFILTER_11

fSAMPLING= fEEVS/16U, N=6U

HRTIM_EVENTFILTER_12

fSAMPLING= fEEVS/16U, N=8U

HRTIM_EVENTFILTER_13

fSAMPLING= fEEVS/32U, N=5U

HRTIM_EVENTFILTER_14

fSAMPLING= fEEVS/32U, N=6U

HRTIM_EVENTFILTER_15

fSAMPLING= fEEVS/32U, N=8U

HRTIM External Event Polarity

HRTIM_EVENTPOLARITY_HIGH

External event is active high

HRTIM_EVENTPOLARITY_LOW

External event is active low

HRTIM External Event Prescaler

HRTIM_EVENTPRESCALER_DIV1

fEEVS=fHRTIM

HRTIM_EVENTPRESCALER_DIV2

fEEVS=fHRTIM / 2U

HRTIM_EVENTPRESCALER_DIV4

fEEVS=fHRTIM / 4U

HRTIM_EVENTPRESCALER_DIV8

fEEVS=fHRTIM / 8U

HRTIM External Event Sensitivity

HRTIM_EVENTSENSITIVITY_LEVEL

External event is active on level

HRTIM_EVENTSENSITIVITY_RISINGEDGE

External event is active on Rising edge

HRTIM_EVENTSENSITIVITY_FALLINGEDGE

External event is active on Falling edge

HRTIM_EVENTSENSITIVITY_BOTHEDGES

External event is active on Rising and Falling edges

HRTIM External Event Sources

HRTIM_EVENTSRC_1

External event source 1U

HRTIM_EVENTSRC_2

External event source 2U

HRTIM_EVENTSRC_3

External event source 3U

HRTIM_EVENTSRC_4

External event source 4U

HRTIM External Fault Prescaler

HRTIM_FAULTPRESCALER_DIV1

fFLTS=fHRTIM

HRTIM_FAULTPRESCALER_DIV2

fFLTS=fHRTIM / 2U

HRTIM_FAULTPRESCALER_DIV4

fFLTS=fHRTIM / 4U

HRTIM_FAULTPRESCALER_DIV8

fFLTS=fHRTIM / 8U

HRTIM Fault Channel

HRTIM_FAULT_1

Fault channel 1 identifier

HRTIM_FAULT_2

Fault channel 2 identifier

HRTIM_FAULT_3

Fault channel 3 identifier

HRTIM_FAULT_4

Fault channel 4 identifier

HRTIM_FAULT_5

Fault channel 5 identifier

HRTIM Fault Filter

HRTIM_FAULTFILTER_NONE

Filter disabled

HRTIM_FAULTFILTER_1

fSAMPLING= fHRTIM, N=2U

HRTIM_FAULTFILTER_2

fSAMPLING= fHRTIM, N=4U

HRTIM_FAULTFILTER_3

fSAMPLING= fHRTIM, N=8U

HRTIM_FAULTFILTER_4

fSAMPLING= fFLTS/2U, N=6U

HRTIM_FAULTFILTER_5

fSAMPLING= fFLTS/2U, N=8U

HRTIM_FAULTFILTER_6

fSAMPLING= fFLTS/4U, N=6U

HRTIM_FAULTFILTER_7

fSAMPLING= fFLTS/4U, N=8U

HRTIM_FAULTFILTER_8

fSAMPLING= fFLTS/8U, N=6U

HRTIM_FAULTFILTER_9

fSAMPLING= fFLTS/8U, N=8U

HRTIM_FAULTFILTER_10

fSAMPLING= fFLTS/16U, N=5U

HRTIM_FAULTFILTER_11

fSAMPLING= fFLTS/16U, N=6U

HRTIM_FAULTFILTER_12

fSAMPLING= fFLTS/16U, N=8U

HRTIM_FAULTFILTER_13

fSAMPLING= fFLTS/32U, N=5U

HRTIM_FAULTFILTER_14

fSAMPLING= fFLTS/32U, N=6U

HRTIM_FAULTFILTER_15

fSAMPLING= fFLTS/32U, N=8U

HRTIM Fault Lock

HRTIM_FAULTLOCK_READWRITE

Fault settings bits are read/write

HRTIM_FAULTLOCK_READONLY

Fault settings bits are read only

HRTIM Fault Mode Control

HRTIM_FAULTMODECTL_DISABLED

Fault channel is disabled

HRTIM_FAULTMODECTL_ENABLED

Fault channel is enabled

HRTIM Fault Polarity

HRTIM_FAULTPOLARITY_LOW

Fault input is active low

HRTIM_FAULTPOLARITY_HIGH

Fault input is active high

HRTIM Fault Sources

HRTIM_FAULTSOURCE_DIGITALINPUT

Fault input is FLT input pin

HRTIM_FAULTSOURCE_INTERNAL

Fault input is FLT_Int signal (e.g. internal comparator)

HRTIM Half Mode Enable

HRTIM_HALFMODE_DISABLED

Half mode is disabled

HRTIM_HALFMODE_ENABLED

Half mode is enabled

HRTIM Idle Push Pull Status

HRTIM_PUSHPULL_IDLESTATUS_OUTPUT1

Protection occurred when the output 1 was active and output 2 forced inactive

HRTIM_PUSHPULL_IDLESTATUS_OUTPUT2

Protection occurred when the output 2 was active and output 1 forced inactive

HRTIM Master DMA Request Enable

HRTIM_MASTER_DMA_NONE

No DMA request enable

HRTIM_MASTER_DMA_MCMP1

Master compare 1 DMA request enable

HRTIM_MASTER_DMA_MCMP2

Master compare 2 DMA request enable

HRTIM_MASTER_DMA_MCMP3

Master compare 3 DMA request enable

HRTIM_MASTER_DMA_MCMP4

Master compare 4 DMA request enable

HRTIM_MASTER_DMA_MREP

Master Repetition DMA request enable

HRTIM_MASTER_DMA_SYNC

Synchronization input DMA request enable

HRTIM_MASTER_DMA_MUPD

Master update DMA request enable

HRTIM Master Interrupt Enable

HRTIM_MASTER_IT_NONE

No interrupt enabled

HRTIM_MASTER_IT_MCMP1

Master compare 1 interrupt enable

HRTIM_MASTER_IT_MCMP2

Master compare 2 interrupt enable

HRTIM_MASTER_IT_MCMP3

Master compare 3 interrupt enable

HRTIM_MASTER_IT_MCMP4

Master compare 4 interrupt enable

HRTIM_MASTER_IT_MREP

Master Repetition interrupt enable

HRTIM_MASTER_IT_SYNC

Synchronization input interrupt enable

HRTIM_MASTER_IT_MUPD

Master update interrupt enable

HRTIM Master Interrupt Flag

HRTIM_MASTER_FLAG_MCMP1

Master compare 1 interrupt flag

HRTIM_MASTER_FLAG_MCMP2

Master compare 2 interrupt flag

HRTIM_MASTER_FLAG_MCMP3

Master compare 3 interrupt flag

HRTIM_MASTER_FLAG_MCMP4

Master compare 4 interrupt flag

HRTIM_MASTER_FLAG_MREP

Master Repetition interrupt flag

HRTIM_MASTER_FLAG_SYNC

Synchronization input interrupt flag

HRTIM_MASTER_FLAG_MUPD

Master update interrupt flag

HRTIM Max Timer

MAX_HRTIM_TIMER

HRTIM Output Burst Mode Entry Delayed

HRTIM_OUTPUTBURSTMODEENTRY_REGULAR

The programmed Idle state is applied immediately to the Output

HRTIM_OUTPUTBURSTMODEENTRY_DELAYED

Dead-time is inserted on output before entering the idle mode

HRTIM Output Chopper Mode Enable

HRTIM_OUTPUTCHOPPERMODE_DISABLED

Output signal is not altered

HRTIM_OUTPUTCHOPPERMODE_ENABLED

Output signal is chopped by a carrier signal

HRTIM Output FAULT Level

HRTIM_OUTPUTFAULTLEVEL_NONE

The output is not affected by the fault input

HRTIM_OUTPUTFAULTLEVEL_ACTIVE

Output at active level when in FAULT state

HRTIM_OUTPUTFAULTLEVEL_INACTIVE

Output at inactive level when in FAULT state

HRTIM_OUTPUTFAULTLEVEL_HIGHZ

Output is tri-stated when in FAULT state

HRTIM Output IDLE Level

HRTIM_OUTPUTIDLELEVEL_INACTIVE

Output at inactive level when in IDLE state

HRTIM_OUTPUTIDLELEVEL_ACTIVE

Output at active level when in IDLE state

HRTIM Output Idle Mode

HRTIM_OUTPUTIDLEMODE_NONE

The output is not affected by the burst mode operation

HRTIM_OUTPUTIDLEMODE_IDLE

The output is in idle state when requested by the burst mode controller

HRTIM Output Level

HRTIM_OUTPUTLEVEL_ACTIVE

Force the output to its active state

HRTIM_OUTPUTLEVEL_INACTIVE

Force the output to its inactive state

IS_HRTIM_OUTPUTLEVEL

HRTIM Output Polarity

HRTIM_OUTPUTPOLARITY_HIGH

Output is active HIGH

HRTIM_OUTPUTPOLARITY_LOW

Output is active LOW

HRTIM Output Reset Source

HRTIM_OUTPUTRESET_NONE

Reset the output reset crossbar

HRTIM_OUTPUTRESET_RESYNC

Timer reset event coming solely from software or SYNC input forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMPER

Timer period event forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMCMP1

Timer compare 1 event forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMCMP2

Timer compare 2 event forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMCMP3

Timer compare 3 event forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMCMP4

Timer compare 4 event forces the output to its inactive state

HRTIM_OUTPUTRESET_MASTERPER

The master timer period event forces the output to its inactive state

HRTIM_OUTPUTRESET_MASTERCMP1

Master Timer compare 1 event forces the output to its inactive state

HRTIM_OUTPUTRESET_MASTERCMP2

Master Timer compare 2 event forces the output to its inactive state

HRTIM_OUTPUTRESET_MASTERCMP3

Master Timer compare 3 event forces the output to its inactive state

HRTIM_OUTPUTRESET_MASTERCMP4

Master Timer compare 4 event forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMEV_1

Timer event 1 forces the output to its active state

HRTIM_OUTPUTRESET_TIMEV_2

Timer event 2 forces the output to its active state

HRTIM_OUTPUTRESET_TIMEV_3

Timer event 3 forces the output to its active state

HRTIM_OUTPUTRESET_TIMEV_4

Timer event 4 forces the output to its active state

HRTIM_OUTPUTRESET_TIMEV_5

Timer event 5 forces the output to its active state

HRTIM_OUTPUTRESET_TIMEV_6

Timer event 6 forces the output to its active state

HRTIM_OUTPUTRESET_TIMEV_7

Timer event 7 forces the output to its active state

HRTIM_OUTPUTRESET_TIMEV_8

Timer event 8 forces the output to its active state

HRTIM_OUTPUTRESET_TIMEV_9

Timer event 9 forces the output to its active state

HRTIM_OUTPUTRESET_EEV_1

External event 1 forces the output to its inactive state

HRTIM_OUTPUTRESET_EEV_2

External event 2 forces the output to its inactive state

HRTIM_OUTPUTRESET_EEV_3

External event 3 forces the output to its inactive state

HRTIM_OUTPUTRESET_EEV_4

External event 4 forces the output to its inactive state

HRTIM_OUTPUTRESET_EEV_5

External event 5 forces the output to its inactive state

HRTIM_OUTPUTRESET_EEV_6

External event 6 forces the output to its inactive state

HRTIM_OUTPUTRESET_EEV_7

External event 7 forces the output to its inactive state

HRTIM_OUTPUTRESET_EEV_8

External event 8 forces the output to its inactive state

HRTIM_OUTPUTRESET_EEV_9

External event 9 forces the output to its inactive state

HRTIM_OUTPUTRESET_EEV_10

External event 10 forces the output to its inactive state

HRTIM_OUTPUTRESET_UPDATE

Timer register update event forces the output to its inactive state

HRTIM Output Set Source

HRTIM_OUTPUTSET_NONE

Reset the output set crossbar

HRTIM_OUTPUTSET_RESYNC

Timer reset event coming solely from software or SYNC input forces the output to its active state

HRTIM_OUTPUTSET_TIMPER

Timer period event forces the output to its active state

HRTIM_OUTPUTSET_TIMCMP1

Timer compare 1 event forces the output to its active state

HRTIM_OUTPUTSET_TIMCMP2

Timer compare 2 event forces the output to its active state

HRTIM_OUTPUTSET_TIMCMP3

Timer compare 3 event forces the output to its active state

HRTIM_OUTPUTSET_TIMCMP4

Timer compare 4 event forces the output to its active state

HRTIM_OUTPUTSET_MASTERPER

The master timer period event forces the output to its active state

HRTIM_OUTPUTSET_MASTERCMP1

Master Timer compare 1 event forces the output to its active state

HRTIM_OUTPUTSET_MASTERCMP2

Master Timer compare 2 event forces the output to its active state

HRTIM_OUTPUTSET_MASTERCMP3

Master Timer compare 3 event forces the output to its active state

HRTIM_OUTPUTSET_MASTERCMP4

Master Timer compare 4 event forces the output to its active state

HRTIM_OUTPUTSET_TIMEV_1

Timer event 1 forces the output to its active state

HRTIM_OUTPUTSET_TIMEV_2

Timer event 2 forces the output to its active state

HRTIM_OUTPUTSET_TIMEV_3

Timer event 3 forces the output to its active state

HRTIM_OUTPUTSET_TIMEV_4

Timer event 4 forces the output to its active state

HRTIM_OUTPUTSET_TIMEV_5

Timer event 5 forces the output to its active state

HRTIM_OUTPUTSET_TIMEV_6

Timer event 6 forces the output to its active state

HRTIM_OUTPUTSET_TIMEV_7

Timer event 7 forces the output to its active state

HRTIM_OUTPUTSET_TIMEV_8

Timer event 8 forces the output to its active state

HRTIM_OUTPUTSET_TIMEV_9

Timer event 9 forces the output to its active state

HRTIM_OUTPUTSET_EEV_1

External event 1 forces the output to its active state

HRTIM_OUTPUTSET_EEV_2

External event 2 forces the output to its active state

HRTIM_OUTPUTSET_EEV_3

External event 3 forces the output to its active state

HRTIM_OUTPUTSET_EEV_4

External event 4 forces the output to its active state

HRTIM_OUTPUTSET_EEV_5

External event 5 forces the output to its active state

HRTIM_OUTPUTSET_EEV_6

External event 6 forces the output to its active state

HRTIM_OUTPUTSET_EEV_7

External event 7 forces the output to its active state

HRTIM_OUTPUTSET_EEV_8

External event 8 forces the output to its active state

HRTIM_OUTPUTSET_EEV_9

External event 9 forces the output to its active state

HRTIM_OUTPUTSET_EEV_10

External event 10 forces the output to its active state

HRTIM_OUTPUTSET_UPDATE

Timer register update event forces the output to its active state

HRTIM Output State

HRTIM_OUTPUTSTATE_IDLE

Main operating mode, where the output can take the active or inactive level as programmed in the crossbar unit

HRTIM_OUTPUTSTATE_RUN

Default operating state (e.g. after an HRTIM reset, when the outputs are disabled by software or during a burst mode operation

HRTIM_OUTPUTSTATE_FAULT

Safety state, entered in case of a shut-down request on FAULTx inputs

HRTIM Prescaler Ratio

HRTIM_PRESCALERRATIO_DIV1

fHRCK: fHRTIM = 144 MHz - Resolution: 6.95 ns - Min PWM frequency: 2.2 kHz (fHRTIM=144MHz)

HRTIM_PRESCALERRATIO_DIV2

fHRCK: fHRTIM / 2U = 72 MHz - Resolution: 13.88 ns- Min PWM frequency: 1.1 kHz (fHRTIM=144MHz)

HRTIM_PRESCALERRATIO_DIV4

fHRCK: fHRTIM / 4U = 36 MHz - Resolution: 27.7 ns- Min PWM frequency: 550Hz (fHRTIM=144MHz)

HRTIM Register Preload Enable

HRTIM_PRELOAD_DISABLED

Preload disabled: the write access is directly done into the active register

HRTIM_PRELOAD_ENABLED

Preload enabled: the write access is done into the preload register

HRTIM Reset On Sync Input Event

HRTIM_SYNCRESET_DISABLED

Synchronization input event has effect on the timer

HRTIM_SYNCRESET_ENABLED

Synchronization input event resets the timer

HRTIM Simple OC Mode

HRTIM_BASICOCMODE_TOGGLE

Output toggles when the timer counter reaches the compare value

HRTIM_BASICOCMODE_INACTIVE

Output forced to active level when the timer counter reaches the compare value

HRTIM_BASICOCMODE_ACTIVE

Output forced to inactive level when the timer counter reaches the compare value

IS_HRTIM_BASICOCMODE

HRTIM Software Timer Reset

HRTIM_TIMERRESET_MASTER

Reset the master timer counter

HRTIM_TIMERRESET_TIMER_A

Reset the timer A counter

HRTIM_TIMERRESET_TIMER_B

Reset the timer B counter

HRTIM_TIMERRESET_TIMER_C

Reset the timer C counter

HRTIM_TIMERRESET_TIMER_D

Reset the timer D counter

HRTIM_TIMERRESET_TIMER_E

Reset the timer E counter

HRTIM Software Timer Update

HRTIM_TIMERUPDATE_MASTER

Force an immediate transfer from the preload to the active register in the master timer

HRTIM_TIMERUPDATE_A

Force an immediate transfer from the preload to the active register in the timer A

HRTIM_TIMERUPDATE_B

Force an immediate transfer from the preload to the active register in the timer B

HRTIM_TIMERUPDATE_C

Force an immediate transfer from the preload to the active register in the timer C

HRTIM_TIMERUPDATE_D

Force an immediate transfer from the preload to the active register in the timer D

HRTIM_TIMERUPDATE_E

Force an immediate transfer from the preload to the active register in the timer E

HRTIM Start On Sync Input Event

HRTIM_SYNCSTART_DISABLED

Synchronization input event has effect on the timer

HRTIM_SYNCSTART_ENABLED

Synchronization input event starts the timer

HRTIM Synchronization Input Source

HRTIM_SYNCINPUTSOURCE_NONE

disabled. HRTIM is not synchronized and runs in standalone mode

HRTIM_SYNCINPUTSOURCE_INTERNALEVENT

The HRTIM is synchronized with the on-chip timer

HRTIM_SYNCINPUTSOURCE_EXTERNALEVENT

A positive pulse on SYNCIN input triggers the HRTIM

HRTIM Synchronization Options

HRTIM_SYNCOPTION_NONE

HRTIM instance doesn't handle external synchronization signals (SYNCIN, SYNCOUT)

HRTIM_SYNCOPTION_MASTER

HRTIM instance acts as a MASTER, i.e. generates external synchronization output (SYNCOUT)

HRTIM_SYNCOPTION_SLAVE

HRTIM instance acts as a SLAVE, i.e. it is synchronized by external sources (SYNCIN)

HRTIM Synchronization Output Polarity

HRTIM_SYNCOUTPUTPOLARITY_NONE

Synchronization output event is disabled

HRTIM_SYNCOUTPUTPOLARITY_POSITIVE

SCOUT pin has a low idle level and issues a positive pulse of 16 fHRTIM clock cycles length for the synchronization

HRTIM_SYNCOUTPUTPOLARITY_NEGATIVE

SCOUT pin has a high idle level and issues a negative pulse of 16 fHRTIM clock cycles length for the synchronization

HRTIM Synchronization Output Source

HRTIM_SYNCOUTPUTSOURCE_MASTER_START

A pulse is sent on HRTIM_SCOUT output and hrtim_out_sync2 upon master timer start event

HRTIM_SYNCOUTPUTSOURCE_MASTER_CMP1

A pulse is sent on HRTIM_SCOUT output and hrtim_out_sync2 upon master timer compare 1 event

HRTIM_SYNCOUTPUTSOURCE_TIMA_START

A pulse is sent on HRTIM_SCOUT output and hrtim_out_sync2 upon timer A start or reset events

HRTIM_SYNCOUTPUTSOURCE_TIMA_CMP1

A pulse is sent on HRTIM_SCOUT output and hrtim_out_sync2 upon timer A compare 1 event

HRTIM Timer Burst Mode

HRTIM_TIMERBURSTMODE_MAINTAINCLOCK

Timer counter clock is maintained and the timer operates normally

HRTIM_TIMERBURSTMODE_RESETCOUNTER

Timer counter clock is stopped and the counter is reset

HRTIM Timer Dead-time Insertion

HRTIM_TIMDEADTIMEINSERTION_DISABLED

Output 1 and output 2 signals are independent

HRTIM_TIMDEADTIMEINSERTION_ENABLED

Dead-time is inserted between output 1 and output 2U

HRTIM Timer Delayed Protection Mode

HRTIM_TIMER_A_B_C_DELAYEDPROTECTION_DISABLED

No action

HRTIM_TIMER_A_B_C_DELAYEDPROTECTION_DELAYEDOUT1_EEV6

Timers A, B, C: Output 1 delayed Idle on external Event 6U

HRTIM_TIMER_A_B_C_DELAYEDPROTECTION_DELAYEDOUT2_EEV6

Timers A, B, C: Output 2 delayed Idle on external Event 6U

HRTIM_TIMER_A_B_C_DELAYEDPROTECTION_DELAYEDBOTH_EEV6

Timers A, B, C: Output 1 and output 2 delayed Idle on external Event 6U

HRTIM_TIMER_A_B_C_DELAYEDPROTECTION_BALANCED_EEV6

Timers A, B, C: Balanced Idle on external Event 6U

HRTIM_TIMER_A_B_C_DELAYEDPROTECTION_DELAYEDOUT1_DEEV7

Timers A, B, C: Output 1 delayed Idle on external Event 7U

HRTIM_TIMER_A_B_C_DELAYEDPROTECTION_DELAYEDOUT2_DEEV7

Timers A, B, C: Output 2 delayed Idle on external Event 7U

HRTIM_TIMER_A_B_C_DELAYEDPROTECTION_DELAYEDBOTH_EEV7

Timers A, B, C: Output 1 and output2 delayed Idle on external Event 7U

HRTIM_TIMER_A_B_C_DELAYEDPROTECTION_BALANCED_EEV7

Timers A, B, C: Balanced Idle on external Event 7U

HRTIM_TIMER_D_E_DELAYEDPROTECTION_DISABLED

No action

HRTIM_TIMER_D_E_DELAYEDPROTECTION_DELAYEDOUT1_EEV8

Timers D, E: Output 1 delayed Idle on external Event 6U

HRTIM_TIMER_D_E_DELAYEDPROTECTION_DELAYEDOUT2_EEV8

Timers D, E: Output 2 delayed Idle on external Event 6U

HRTIM_TIMER_D_E_DELAYEDPROTECTION_DELAYEDBOTH_EEV8

Timers D, E: Output 1 and output 2 delayed Idle on external Event 6U

HRTIM_TIMER_D_E_DELAYEDPROTECTION_BALANCED_EEV8

Timers D, E: Balanced Idle on external Event 6U

HRTIM_TIMER_D_E_DELAYEDPROTECTION_DELAYEDOUT1_DEEV9

Timers D, E: Output 1 delayed Idle on external Event 7U

HRTIM_TIMER_D_E_DELAYEDPROTECTION_DELAYEDOUT2_DEEV9

Timers D, E: Output 2 delayed Idle on external Event 7U

HRTIM_TIMER_D_E_DELAYEDPROTECTION_DELAYEDBOTH_EEV9

Timers D, E: Output 1 and output2 delayed Idle on external Event 7U

HRTIM_TIMER_D_E_DELAYEDPROTECTION_BALANCED_EEV9

Timers D, E: Balanced Idle on external Event 7U

HRTIM Timer External Event Filter

HRTIM_TIMEVENTFILTER_NONE

HRTIM_TIMEVENTFILTER_BLANKINGCMP1

Blanking from counter reset/roll-over to Compare 1U

HRTIM_TIMEVENTFILTER_BLANKINGCMP2

Blanking from counter reset/roll-over to Compare 2U

HRTIM_TIMEVENTFILTER_BLANKINGCMP3

Blanking from counter reset/roll-over to Compare 3U

HRTIM_TIMEVENTFILTER_BLANKINGCMP4

Blanking from counter reset/roll-over to Compare 4U

HRTIM_TIMEVENTFILTER_BLANKINGFLTR1

Blanking from another timing unit: TIMFLTR1 source

HRTIM_TIMEVENTFILTER_BLANKINGFLTR2

Blanking from another timing unit: TIMFLTR2 source

HRTIM_TIMEVENTFILTER_BLANKINGFLTR3

Blanking from another timing unit: TIMFLTR3 source

HRTIM_TIMEVENTFILTER_BLANKINGFLTR4

Blanking from another timing unit: TIMFLTR4 source

HRTIM_TIMEVENTFILTER_BLANKINGFLTR5

Blanking from another timing unit: TIMFLTR5 source

HRTIM_TIMEVENTFILTER_BLANKINGFLTR6

Blanking from another timing unit: TIMFLTR6 source

HRTIM_TIMEVENTFILTER_BLANKINGFLTR7

Blanking from another timing unit: TIMFLTR7 source

HRTIM_TIMEVENTFILTER_BLANKINGFLTR8

Blanking from another timing unit: TIMFLTR8 source

HRTIM_TIMEVENTFILTER_WINDOWINGCMP2

Windowing from counter reset/roll-over to Compare 2U

HRTIM_TIMEVENTFILTER_WINDOWINGCMP3

Windowing from counter reset/roll-over to Compare 3U

HRTIM_TIMEVENTFILTER_WINDOWINGTIM

Windowing from another timing unit: TIMWIN source

HRTIM Timer External Event Latch

HRTIM_TIMEVENTLATCH_DISABLED

Event is ignored if it happens during a blank, or passed through during a window

HRTIM_TIMEVENTLATCH_ENABLED

Event is latched and delayed till the end of the blanking or windowing period

HRTIM Timer Fault Enabling

HRTIM_TIMFAULTENABLE_NONE

No fault enabled

HRTIM_TIMFAULTENABLE_FAULT1

Fault 1 enabled

HRTIM_TIMFAULTENABLE_FAULT2

Fault 2 enabled

HRTIM_TIMFAULTENABLE_FAULT3

Fault 3 enabled

HRTIM_TIMFAULTENABLE_FAULT4

Fault 4 enabled

HRTIM_TIMFAULTENABLE_FAULT5

Fault 5 enabled

HRTIM Timer Fault Lock

HRTIM_TIMFAULTLOCK_READWRITE

Timer fault enabling bits are read/write

HRTIM_TIMFAULTLOCK_READONLY

Timer fault enabling bits are read only

HRTIM Timer identifier

HRTIM_TIMERID_MASTER

Master identifier

HRTIM_TIMERID_TIMER_A

Timer A identifier

HRTIM_TIMERID_TIMER_B

Timer B identifier

HRTIM_TIMERID_TIMER_C

Timer C identifier

HRTIM_TIMERID_TIMER_D

Timer D identifier

HRTIM_TIMERID_TIMER_E

Timer E identifier

HRTIM Timer Index

HRTIM_TIMERINDEX_TIMER_A

Index used to access timer A registers

HRTIM_TIMERINDEX_TIMER_B

Index used to access timer B registers

HRTIM_TIMERINDEX_TIMER_C

Index used to access timer C registers

HRTIM_TIMERINDEX_TIMER_D

Index used to access timer D registers

HRTIM_TIMERINDEX_TIMER_E

Index used to access timer E registers

HRTIM_TIMERINDEX_MASTER

Index used to access master registers

HRTIM_TIMERINDEX_COMMON

Index used to access HRTIM common registers

HRTIM Timer Output

HRTIM_OUTPUT_TA1

Timer A - Output 1 identifier

HRTIM_OUTPUT_TA2

Timer A - Output 2 identifier

HRTIM_OUTPUT_TB1

Timer B - Output 1 identifier

HRTIM_OUTPUT_TB2

Timer B - Output 2 identifier

HRTIM_OUTPUT_TC1

Timer C - Output 1 identifier

HRTIM_OUTPUT_TC2

Timer C - Output 2 identifier

HRTIM_OUTPUT_TD1

Timer D - Output 1 identifier

HRTIM_OUTPUT_TD2

Timer D - Output 2 identifier

HRTIM_OUTPUT_TE1

Timer E - Output 1 identifier

HRTIM_OUTPUT_TE2

Timer E - Output 2 identifier

HRTIM Timer Push Pull Mode

HRTIM_TIMPUSHPULLMODE_DISABLED

Push-Pull mode disabled

HRTIM_TIMPUSHPULLMODE_ENABLED

Push-Pull mode enabled

HRTIM Timer Repetition Update

HRTIM_UPDATEONREPETITION_DISABLED

Update on repetition disabled

HRTIM_UPDATEONREPETITION_ENABLED

Update on repetition enabled

HRTIM Timer Reset Trigger

HRTIM_TIMRESETTRIGGER_NONE

No counter reset trigger

HRTIM_TIMRESETTRIGGER_UPDATE

The timer counter is reset upon update event

HRTIM_TIMRESETTRIGGER_CMP2

The timer counter is reset upon Timer Compare 2 event

HRTIM_TIMRESETTRIGGER_CMP4

The timer counter is reset upon Timer Compare 4 event

HRTIM_TIMRESETTRIGGER_MASTER_PER

The timer counter is reset upon master timer period event

HRTIM_TIMRESETTRIGGER_MASTER_CMP1

The timer counter is reset upon master timer Compare 1 event

HRTIM_TIMRESETTRIGGER_MASTER_CMP2

The timer counter is reset upon master timer Compare 2 event

HRTIM_TIMRESETTRIGGER_MASTER_CMP3

The timer counter is reset upon master timer Compare 3 event

HRTIM_TIMRESETTRIGGER_MASTER_CMP4

The timer counter is reset upon master timer Compare 4 event

HRTIM_TIMRESETTRIGGER_EEV_1

The timer counter is reset upon external event 1U

HRTIM_TIMRESETTRIGGER_EEV_2

The timer counter is reset upon external event 2U

HRTIM_TIMRESETTRIGGER_EEV_3

The timer counter is reset upon external event 3U

HRTIM_TIMRESETTRIGGER_EEV_4

The timer counter is reset upon external event 4U

HRTIM_TIMRESETTRIGGER_EEV_5

The timer counter is reset upon external event 5U

HRTIM_TIMRESETTRIGGER_EEV_6

The timer counter is reset upon external event 6U

HRTIM_TIMRESETTRIGGER_EEV_7

The timer counter is reset upon external event 7U

HRTIM_TIMRESETTRIGGER_EEV_8

The timer counter is reset upon external event 8U

HRTIM_TIMRESETTRIGGER_EEV_9

The timer counter is reset upon external event 9U

HRTIM_TIMRESETTRIGGER_EEV_10

The timer counter is reset upon external event 10U

HRTIM_TIMRESETTRIGGER_OTHER1_CMP1

The timer counter is reset upon other timer Compare 1 event

HRTIM_TIMRESETTRIGGER_OTHER1_CMP2

The timer counter is reset upon other timer Compare 2 event

HRTIM_TIMRESETTRIGGER_OTHER1_CMP4

The timer counter is reset upon other timer Compare 4 event

HRTIM_TIMRESETTRIGGER_OTHER2_CMP1

The timer counter is reset upon other timer Compare 1 event

HRTIM_TIMRESETTRIGGER_OTHER2_CMP2

The timer counter is reset upon other timer Compare 2 event

HRTIM_TIMRESETTRIGGER_OTHER2_CMP4

The timer counter is reset upon other timer Compare 4 event

HRTIM_TIMRESETTRIGGER_OTHER3_CMP1

The timer counter is reset upon other timer Compare 1 event

HRTIM_TIMRESETTRIGGER_OTHER3_CMP2

The timer counter is reset upon other timer Compare 2 event

HRTIM_TIMRESETTRIGGER_OTHER3_CMP4

The timer counter is reset upon other timer Compare 4 event

HRTIM_TIMRESETTRIGGER_OTHER4_CMP1

The timer counter is reset upon other timer Compare 1 event

HRTIM_TIMRESETTRIGGER_OTHER4_CMP2

The timer counter is reset upon other timer Compare 2 event

HRTIM_TIMRESETTRIGGER_OTHER4_CMP4

The timer counter is reset upon other timer Compare 4 event

HRTIM Timer Reset Update

HRTIM_TIMUPDATEONRESET_DISABLED

Update by timer x reset / roll-over disabled

HRTIM_TIMUPDATEONRESET_ENABLED

Update by timer x reset / roll-over enabled

HRTIM Timer Update Trigger

HRTIM_TIMUPDATETRIGGER_NONE

Register update is disabled

HRTIM_TIMUPDATETRIGGER_MASTER

Register update is triggered by the master timer update

HRTIM_TIMUPDATETRIGGER_TIMER_A

Register update is triggered by the timer A update

HRTIM_TIMUPDATETRIGGER_TIMER_B

Register update is triggered by the timer B update

HRTIM_TIMUPDATETRIGGER_TIMER_C

Register update is triggered by the timer C update

HRTIM_TIMUPDATETRIGGER_TIMER_D

Register update is triggered by the timer D update

HRTIM_TIMUPDATETRIGGER_TIMER_E

Register update is triggered by the timer E update

HRTIM Timing Unit DMA Request Enable

HRTIM_TIM_DMA_NONE

No DMA request enable

HRTIM_TIM_DMA_CMP1

Timer compare 1 DMA request enable

HRTIM_TIM_DMA_CMP2

Timer compare 2 DMA request enable

HRTIM_TIM_DMA_CMP3

Timer compare 3 DMA request enable

HRTIM_TIM_DMA_CMP4

Timer compare 4 DMA request enable

HRTIM_TIM_DMA_REP

Timer repetition DMA request enable

HRTIM_TIM_DMA_UPD

Timer update DMA request enable

HRTIM_TIM_DMA_CPT1

Timer capture 1 DMA request enable

HRTIM_TIM_DMA_CPT2

Timer capture 2 DMA request enable

HRTIM_TIM_DMA_SET1

Timer output 1 set DMA request enable

HRTIM_TIM_DMA_RST1

Timer output 1 reset DMA request enable

HRTIM_TIM_DMA_SET2

Timer output 2 set DMA request enable

HRTIM_TIM_DMA_RST2

Timer output 2 reset DMA request enable

HRTIM_TIM_DMA_RST

Timer reset DMA request enable

HRTIM_TIM_DMA_DLYPRT

Timer delay protection DMA request enable

HRTIM Timing Unit Interrupt Enable

HRTIM_TIM_IT_NONE

No interrupt enabled

HRTIM_TIM_IT_CMP1

Timer compare 1 interrupt enable

HRTIM_TIM_IT_CMP2

Timer compare 2 interrupt enable

HRTIM_TIM_IT_CMP3

Timer compare 3 interrupt enable

HRTIM_TIM_IT_CMP4

Timer compare 4 interrupt enable

HRTIM_TIM_IT_REP

Timer repetition interrupt enable

HRTIM_TIM_IT_UPD

Timer update interrupt enable

HRTIM_TIM_IT_CPT1

Timer capture 1 interrupt enable

HRTIM_TIM_IT_CPT2

Timer capture 2 interrupt enable

HRTIM_TIM_IT_SET1

Timer output 1 set interrupt enable

HRTIM_TIM_IT_RST1

Timer output 1 reset interrupt enable

HRTIM_TIM_IT_SET2

Timer output 2 set interrupt enable

HRTIM_TIM_IT_RST2

Timer output 2 reset interrupt enable

HRTIM_TIM_IT_RST

Timer reset interrupt enable

HRTIM_TIM_IT_DLYPRT

Timer delay protection interrupt enable

HRTIM Timing Unit Interrupt Flag

HRTIM_TIM_FLAG_CMP1

Timer compare 1 interrupt flag

HRTIM_TIM_FLAG_CMP2

Timer compare 2 interrupt flag

HRTIM_TIM_FLAG_CMP3

Timer compare 3 interrupt flag

HRTIM_TIM_FLAG_CMP4

Timer compare 4 interrupt flag

HRTIM_TIM_FLAG_REP

Timer repetition interrupt flag

HRTIM_TIM_FLAG_UPD

Timer update interrupt flag

HRTIM_TIM_FLAG_CPT1

Timer capture 1 interrupt flag

HRTIM_TIM_FLAG_CPT2

Timer capture 2 interrupt flag

HRTIM_TIM_FLAG_SET1

Timer output 1 set interrupt flag

HRTIM_TIM_FLAG_RST1

Timer output 1 reset interrupt flag

HRTIM_TIM_FLAG_SET2

Timer output 2 set interrupt flag

HRTIM_TIM_FLAG_RST2

Timer output 2 reset interrupt flag

HRTIM_TIM_FLAG_RST

Timer reset interrupt flag

HRTIM_TIM_FLAG_DLYPRT

Timer delay protection interrupt flag

HRTIM Update Gating

HRTIM_UPDATEGATING_INDEPENDENT

Update done independently from the DMA burst transfer completion

HRTIM_UPDATEGATING_DMABURST

Update done when the DMA burst transfer is completed

HRTIM_UPDATEGATING_DMABURST_UPDATE

Update done on timer roll-over following a DMA burst transfer completion

HRTIM_UPDATEGATING_UPDEN1

Slave timer only - Update done on a rising edge of HRTIM update enable input 1U

HRTIM_UPDATEGATING_UPDEN2

Slave timer only - Update done on a rising edge of HRTIM update enable input 2U

HRTIM_UPDATEGATING_UPDEN3

Slave timer only - Update done on a rising edge of HRTIM update enable input 3U

HRTIM_UPDATEGATING_UPDEN1_UPDATE

Slave timer only - Update done on the update event following a rising edge of HRTIM update enable input 1U

HRTIM_UPDATEGATING_UPDEN2_UPDATE

Slave timer only - Update done on the update event following a rising edge of HRTIM update enable input 2U

HRTIM_UPDATEGATING_UPDEN3_UPDATE

Slave timer only - Update done on the update event following a rising edge of HRTIM update enable input 3U

41 HAL HSEM Generic Driver

41.1 HSEM Firmware driver API description

The following section lists the various functions of the HSEM library.

41.1.1 How to use this driver

1. Take a semaphore In 2-Step mode Using function HAL_HSEM_Take. This function takes as parameters :
   – the semaphore ID from 0 to 31
   – the process ID from 0 to 255
2. Fast Take semaphore In 1-Step mode Using function HAL_HSEM_FastTake. This function takes as parameter :
   – the semaphore ID from 0_ID to 31. Note that the process ID value is implicitly assumed as zero
3. Check if a semaphore is Taken using function HAL_HSEM_IsSemTaken. This function takes as parameter :
   – the semaphore ID from 0_ID to 31
   – It returns 1 if the given semaphore is taken otherwise (Free) zero
4. Release a semaphore using function with HAL_HSEM_Release. This function takes as parameters :
   – the semaphore ID from 0 to 31
   – the process ID from 0 to 255:
   – Note: If ProcessID and MasterID match, semaphore is freed, and an interrupt may be generated when enabled (notification activated). If ProcessID or MasterID does not match, semaphore remains taken (locked)
5. Release all semaphores at once taken by a given Master using function HAL_HSEM_Release_All This function takes as parameters :
   – the Release Key (value from 0 to 0xFFFF) can be Set or Get respectively by HAL_HSEM_SetClearKey() or HAL_HSEM_GetClearKey functions
   – the Master ID:
   – Note: If the Key and MasterID match, all semaphores taken by the given CPU that corresponds to MasterID will be freed, and an interrupt may be generated when enabled (notification activated). If the Key or the MasterID doesn't match, semaphores remains taken (locked)
6. Semaphores Release all key functions:
   – HAL_HSEM_SetClearKey() to set semaphore release all Key
   – HAL_HSEM_GetClearKey() to get release all Key
7. Semaphores notification functions :
   – HAL_HSEM_ActivateNotification to activate a notification callback on a given semaphores Mask (bitfield). When one or more semaphores defined by the mask are released the callback HAL_HSEM_FreeCallback will be asserted giving as parameters a mask of the released semaphores (bitfield).
   – HAL_HSEM_DeactivateNotification to deactivate the notification of a given semaphores Mask (bitfield).
   – See the description of the macro __HAL_HSEM_SEMID_TO_MASK to check how to calculate a semaphore mask Used by the notification functions

HSEM HAL driver macros list

Below the list of most used macros in HSEM HAL driver.

- __HAL_HSEM_SEMID_TO_MASK: Helper macro to convert a Semaphore ID to a Mask.

Example of use :

mask = __HAL_HSEM_SEMID_TO_MASK(8) | __HAL_HSEM_SEMID_TO_MASK(21) | __HAL_HSEM_SEMID_TO_MASK(25).

All next macros take as parameter a semaphore Mask (bitfiled) that can be constructed using __HAL_HSEM_SEMID_TO_MASK as the above example.

- __HAL_HSEM_ENABLE_IT: Enable the specified semaphores Mask interrupts.
- __HAL_HSEM_DISABLE_IT: Disable the specified semaphores Mask interrupts.
- __HAL_HSEM_GET_IT: Checks whether the specified semaphore interrupt has occurred or not.
- __HAL_HSEM_GET_FLAG: Get the semaphores status release flags.
- __HAL_HSEM_CLEAR_FLAG: Clear the semaphores status release flags.

41.1.2 HSEM Take and Release functions

This section provides functions allowing to:

- Take a semaphore with 2 Step method
- Fast Take a semaphore with 1 Step method
- Check semaphore state Taken or not
- Release a semaphore
- Release all semaphore at once

This section contains the following APIs:

- ***HAL_HSEM_Take()***
- ***HAL_HSEM_FastTake()***
- ***HAL_HSEM_IsSemTaken()***
- ***HAL_HSEM_Release()***
- ***HAL_HSEM_ReleaseAll()***

41.1.3 HSEM Set and Get Key functions

This section provides functions allowing to:

- Set semaphore Key
- Get semaphore Key

This section contains the following APIs:

- ***HAL_HSEM_SetClearKey()***
- ***HAL_HSEM_GetClearKey()***

41.1.4 HSEM IRQ handler management and Notification functions

This section provides HSEM IRQ handler and Notification function.

This section contains the following APIs:

- ***HAL_HSEM_ActivateNotification()***
- ***HAL_HSEM_DeactivateNotification()***
- ***HAL_HSEM_IRQHandler()***
- ***HAL_HSEM_FreeCallback()***

41.1.5 Detailed description of functions

HAL_HSEM_Take

Function name

HAL_StatusTypeDef HAL_HSEM_Take (uint32_t SemID, uint32_t ProcessID)

Function description

Take a semaphore in 2 Step mode.

Parameters

- **SemID:** semaphore ID from 0 to 31
- **ProcessID:** Process ID from 0 to 255

Return values

- **HAL:** status

HAL_HSEM_FastTake

Function name

HAL_StatusTypeDef HAL_HSEM_FastTake (uint32_t SemID)

Function description

Fast Take a semaphore with 1 Step mode.

Parameters

- **SemID:** semaphore ID from 0 to 31

Return values

- **HAL:** status

HAL_HSEM_Release

Function name

void HAL_HSEM_Release (uint32_t SemID, uint32_t ProcessID)

Function description

Release a semaphore.

Parameters

- **SemID:** semaphore ID from 0 to 31
- **ProcessID:** Process ID from 0 to 255

Return values

- **None:**

HAL_HSEM_ReleaseAll

Function name

void HAL_HSEM_ReleaseAll (uint32_t Key, uint32_t CoreID)

Function description

Release All semaphore used by a given Master .

Parameters

- **Key:** Semaphore Key , value from 0 to 0xFFFF
- **CoreID:** CoreID of the CPU that is using semaphores to be released

Return values

- **None:**

HAL_HSEM_IsSemTaken

Function name

uint32_t HAL_HSEM_IsSemTaken (uint32_t SemID)

Function description

Check semaphore state Taken or not.

Parameters

- **SemID:** semaphore ID

Return values

- **HAL:** HSEM state

HAL_HSEM_SetClearKey

Function name

void HAL_HSEM_SetClearKey (uint32_t Key)

Function description

Set semaphore Key .

Parameters

- **Key:** Semaphore Key , value from 0 to 0xFFFF

Return values

- **None:**

HAL_HSEM_GetClearKey

Function name

uint32_t HAL_HSEM_GetClearKey (void)

Function description

Get semaphore Key .

Return values

- **Semaphore:** Key , value from 0 to 0xFFFF

HAL_HSEM_ActivateNotification

Function name

void HAL_HSEM_ActivateNotification (uint32_t SemMask)

Function description

Activate Semaphore release Notification for a given Semaphores Mask .

Parameters

- **SemMask:** Mask of Released semaphores

Return values

- **Semaphore:** Key

HAL_HSEM_DeactivateNotification

Function name

void HAL_HSEM_DeactivateNotification (uint32_t SemMask)

Function description

Deactivate Semaphore release Notification for a given Semaphores Mask .

Parameters

- **SemMask:** Mask of Released semaphores

Return values

- **Semaphore:** Key

HAL_HSEM_FreeCallback

Function name

void HAL_HSEM_FreeCallback (uint32_t SemMask)

Function description

Semaphore Released Callback.

Parameters

- **SemMask:** Mask of Released semaphores

Return values

- **None:**

HAL_HSEM_IRQHandler

Function name

void HAL_HSEM_IRQHandler (void)

Function description

This function handles HSEM interrupt request.

Return values

- **None:**

41.2 HSEM Firmware driver defines

The following section lists the various define and macros of the module.

41.2.1 HSEM

HSEM

HSEM Exported Macros

__HAL_HSEM_SEMID_TO_MASK

Description:

- SemID to mask helper Macro.

Parameters:

- __SEMID__: semaphore ID from 0 to 31

Return value:

- Semaphore: Mask.

__HAL_HSEM_ENABLE_IT

Description:

- Enables the specified HSEM interrupts.

Parameters:

- __SEM_MASK__: semaphores Mask

Return value:

- None.

__HAL_HSEM_DISABLE_IT

Description:

- Disables the specified HSEM interrupts.

Parameters:

- __SEM_MASK__: semaphores Mask

Return value:

- None.

__HAL_HSEM_GET_IT

Description:

- Checks whether interrupt has occurred or not for semaphores specified by a mask.

Parameters:

- __SEM_MASK__: semaphores Mask

Return value:

- semaphores: Mask : Semaphores where an interrupt occurred.

__HAL_HSEM_GET_FLAG

Description:

- Get the semaphores release status flags.

Parameters:

- __SEM_MASK__: semaphores Mask

Return value:

- semaphores: Mask : Semaphores where Release flags rise.

__HAL_HSEM_CLEAR_FLAG

Description:

- Clears the HSEM Interrupt flags.

Parameters:

- __SEM_MASK__: semaphores Mask

Return value:

- None.

42 HAL I2C Generic Driver

42.1 I2C Firmware driver registers structures

42.1.1 I2C_InitTypeDef

I2C_InitTypeDef is defined in the stm32h7xx_hal_i2c.h

Data Fields

- ***uint32_t Timing***
- ***uint32_t OwnAddress1***
- ***uint32_t AddressingMode***
- ***uint32_t DualAddressMode***
- ***uint32_t OwnAddress2***
- ***uint32_t OwnAddress2Masks***
- ***uint32_t GeneralCallMode***
- ***uint32_t NoStretchMode***

Field Documentation

- ***uint32_t I2C_InitTypeDef::Timing***
  Specifies the I2C_TIMINGR_register value. This parameter calculated by referring to I2C initialization section in Reference manual
- ***uint32_t I2C_InitTypeDef::OwnAddress1***
  Specifies the first device own address. This parameter can be a 7-bit or 10-bit address.
- ***uint32_t I2C_InitTypeDef::AddressingMode***
  Specifies if 7-bit or 10-bit addressing mode is selected. This parameter can be a value of ***I2C_ADDRESSING_MODE***
- ***uint32_t I2C_InitTypeDef::DualAddressMode***
  Specifies if dual addressing mode is selected. This parameter can be a value of ***I2C_DUAL_ADDRESSING_MODE***
- ***uint32_t I2C_InitTypeDef::OwnAddress2***
  Specifies the second device own address if dual addressing mode is selected This parameter can be a 7-bit address.
- ***uint32_t I2C_InitTypeDef::OwnAddress2Masks***
  Specifies the acknowledge mask address second device own address if dual addressing mode is selected. This parameter can be a value of ***I2C_OWN_ADDRESS2_MASKS***
- ***uint32_t I2C_InitTypeDef::GeneralCallMode***
  Specifies if general call mode is selected. This parameter can be a value of ***I2C_GENERAL_CALL_ADDRESSING_MODE***
- ***uint32_t I2C_InitTypeDef::NoStretchMode***
  Specifies if nostretch mode is selected. This parameter can be a value of ***I2C_NOSTRETCH_MODE***

42.1.2 __I2C_HandleTypeDef

__I2C_HandleTypeDef is defined in the stm32h7xx_hal_i2c.h

Data Fields

- ***I2C_TypeDef * Instance***
- ***I2C_InitTypeDef Init***
- ***uint8_t * pBuffPtr***
- ***uint16_t XferSize***
- ***__IO uint16_t XferCount***

- *__IO uint32_t XferOptions*
- *__IO uint32_t PreviousState*
- *HAL_StatusTypeDef(* XferISR*
- *DMA_HandleTypeDef * hdmatx*
- *DMA_HandleTypeDef * hdmarx*
- *HAL_LockTypeDef Lock*
- *__IO HAL_I2C_StateTypeDef State*
- *__IO HAL_I2C_ModeTypeDef Mode*
- *__IO uint32_t ErrorCode*
- *__IO uint32_t AddrEventCount*
- *__IO uint32_t Devaddress*
- *__IO uint32_t Memaddress*
- *void(* MasterTxCpltCallback*
- *void(* MasterRxCpltCallback*
- *void(* SlaveTxCpltCallback*
- *void(* SlaveRxCpltCallback*
- *void(* ListenCpltCallback*
- *void(* MemTxCpltCallback*
- *void(* MemRxCpltCallback*
- *void(* ErrorCallback*
- *void(* AbortCpltCallback*
- *void(* AddrCallback*
- *void(* MspInitCallback*
- *void(* MspDeInitCallback*

Field Documentation

- ***I2C_TypeDef* __I2C_HandleTypeDef::Instance***
  I2C registers base address
- ***I2C_InitTypeDef __I2C_HandleTypeDef::Init***
  I2C communication parameters
- ***uint8_t* __I2C_HandleTypeDef::pBuffPtr***
  Pointer to I2C transfer buffer
- ***uint16_t __I2C_HandleTypeDef::XferSize***
  I2C transfer size
- ***__IO uint16_t __I2C_HandleTypeDef::XferCount***
  I2C transfer counter
- ***__IO uint32_t __I2C_HandleTypeDef::XferOptions***
  I2C sequantial transfer options, this parameter can be a value of *I2C_XFEROPTIONS*
- ***__IO uint32_t __I2C_HandleTypeDef::PreviousState***
  I2C communication Previous state
- ***HAL_StatusTypeDef(* __I2C_HandleTypeDef::XferISR)(struct __I2C_HandleTypeDef *hi2c, uint32_t ITFlags, uint32_t ITSources)***
  I2C transfer IRQ handler function pointer
- ***DMA_HandleTypeDef* __I2C_HandleTypeDef::hdmatx***
  I2C Tx DMA handle parameters
- ***DMA_HandleTypeDef* __I2C_HandleTypeDef::hdmarx***
  I2C Rx DMA handle parameters
- ***HAL_LockTypeDef __I2C_HandleTypeDef::Lock***
  I2C locking object

- ***__IO HAL_I2C_StateTypeDef __I2C_HandleTypeDef::State***
  I2C communication state
- ***__IO HAL_I2C_ModeTypeDef __I2C_HandleTypeDef::Mode***
  I2C communication mode
- ***__IO uint32_t __I2C_HandleTypeDef::ErrorCode***
  I2C Error code
- ***__IO uint32_t __I2C_HandleTypeDef::AddrEventCount***
  I2C Address Event counter
- ***__IO uint32_t __I2C_HandleTypeDef::Devaddress***
  I2C Target device address
- ***__IO uint32_t __I2C_HandleTypeDef::Memaddress***
  I2C Target memory address
- ***void(* __I2C_HandleTypeDef::MasterTxCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Master Tx Transfer completed callback
- ***void(* __I2C_HandleTypeDef::MasterRxCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Master Rx Transfer completed callback
- ***void(* __I2C_HandleTypeDef::SlaveTxCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Slave Tx Transfer completed callback
- ***void(* __I2C_HandleTypeDef::SlaveRxCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Slave Rx Transfer completed callback
- ***void(* __I2C_HandleTypeDef::ListenCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Listen Complete callback
- ***void(* __I2C_HandleTypeDef::MemTxCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Memory Tx Transfer completed callback
- ***void(* __I2C_HandleTypeDef::MemRxCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Memory Rx Transfer completed callback
- ***void(* __I2C_HandleTypeDef::ErrorCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Error callback
- ***void(* __I2C_HandleTypeDef::AbortCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Abort callback
- ***void(* __I2C_HandleTypeDef::AddrCallback)(struct __I2C_HandleTypeDef *hi2c, uint8_t TransferDirection, uint16_t AddrMatchCode)***
  I2C Slave Address Match callback
- ***void(* __I2C_HandleTypeDef::MspInitCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Msp Init callback
- ***void(* __I2C_HandleTypeDef::MspDeInitCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Msp DeInit callback

42.2 I2C Firmware driver API description

The following section lists the various functions of the I2C library.

42.2.1 How to use this driver

The I2C HAL driver can be used as follows:

1. Declare a I2C_HandleTypeDef handle structure, for example: I2C_HandleTypeDef hi2c;

2. Initialize the I2C low level resources by implementing the HAL_I2C_MspInit() API:
   a. Enable the I2Cx interface clock
   b. I2C pins configuration
      - Enable the clock for the I2C GPIOs
      - Configure I2C pins as alternate function open-drain
   c. NVIC configuration if you need to use interrupt process
      - Configure the I2Cx interrupt priority
      - Enable the NVIC I2C IRQ Channel
   d. DMA Configuration if you need to use DMA process
      - Declare a DMA_HandleTypeDef handle structure for the transmit or receive stream or channel depends on Instance
      - Enable the DMAx interface clock using
      - Configure the DMA handle parameters
      - Configure the DMA Tx or Rx stream or channel depends on Instance
      - Associate the initialized DMA handle to the hi2c DMA Tx or Rx handle
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx stream or channel depends on Instance
3. Configure the Communication Clock Timing, Own Address1, Master Addressing mode, Dual Addressing mode, Own Address2, Own Address2 Mask, General call and Nostretch mode in the hi2c Init structure.
4. Initialize the I2C registers by calling the HAL_I2C_Init(), configures also the low level Hardware (GPIO, CLOCK, NVIC...etc) by calling the customized HAL_I2C_MspInit(&hi2c) API.
5. To check if target device is ready for communication, use the function HAL_I2C_IsDeviceReady()
6. For I2C IO and IO MEM operations, three operation modes are available within this driver :

Polling mode IO operation

- Transmit in master mode an amount of data in blocking mode using HAL_I2C_Master_Transmit()
- Receive in master mode an amount of data in blocking mode using HAL_I2C_Master_Receive()
- Transmit in slave mode an amount of data in blocking mode using HAL_I2C_Slave_Transmit()
- Receive in slave mode an amount of data in blocking mode using HAL_I2C_Slave_Receive()

Polling mode IO MEM operation

- Write an amount of data in blocking mode to a specific memory address using HAL_I2C_Mem_Write()
- Read an amount of data in blocking mode from a specific memory address using HAL_I2C_Mem_Read()

Interrupt mode IO operation

- Transmit in master mode an amount of data in non-blocking mode using HAL_I2C_Master_Transmit_IT()
- At transmission end of transfer, HAL_I2C_MasterTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MasterTxCpltCallback()
- Receive in master mode an amount of data in non-blocking mode using HAL_I2C_Master_Receive_IT()
- At reception end of transfer, HAL_I2C_MasterRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MasterRxCpltCallback()
- Transmit in slave mode an amount of data in non-blocking mode using HAL_I2C_Slave_Transmit_IT()
- At transmission end of transfer, HAL_I2C_SlaveTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_SlaveTxCpltCallback()
- Receive in slave mode an amount of data in non-blocking mode using HAL_I2C_Slave_Receive_IT()
- At reception end of transfer, HAL_I2C_SlaveRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_SlaveRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and users can add their own code by customization of function pointer HAL_I2C_ErrorCallback()
- Abort a master I2C process communication with Interrupt using HAL_I2C_Master_Abort_IT()
- End of abort process, HAL_I2C_AbortCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_AbortCpltCallback()

- Discard a slave I2C process communication using __HAL_I2C_GENERATE_NACK() macro. This action will inform Master to generate a Stop condition to discard the communication.

Interrupt mode or DMA mode IO sequential operation

Note: These interfaces allow to manage a sequential transfer with a repeated start condition when a direction change during transfer

- A specific option field manage the different steps of a sequential transfer
- Option field values are defined through I2C_XFEROPTIONS and are listed below:
  - I2C_FIRST_AND_LAST_FRAME: No sequential usage, functional is same as associated interfaces in no sequential mode
  - I2C_FIRST_FRAME: Sequential usage, this option allow to manage a sequence with start condition, address and data to transfer without a final stop condition
  - I2C_FIRST_AND_NEXT_FRAME: Sequential usage (Master only), this option allow to manage a sequence with start condition, address and data to transfer without a final stop condition, an then permit a call the same master sequential interface several times (like HAL_I2C_Master_Seq_Transmit_IT() then HAL_I2C_Master_Seq_Transmit_IT() or HAL_I2C_Master_Seq_Transmit_DMA() then HAL_I2C_Master_Seq_Transmit_DMA())
  - I2C_NEXT_FRAME: Sequential usage, this option allow to manage a sequence with a restart condition, address and with new data to transfer if the direction change or manage only the new data to transfer if no direction change and without a final stop condition in both cases
  - I2C_LAST_FRAME: Sequential usage, this option allow to manage a sequance with a restart condition, address and with new data to transfer if the direction change or manage only the new data to transfer if no direction change and with a final stop condition in both cases
  - I2C_LAST_FRAME_NO_STOP: Sequential usage (Master only), this option allow to manage a restart condition after several call of the same master sequential interface several times (link with option I2C_FIRST_AND_NEXT_FRAME). Usage can, transfer several bytes one by one using HAL_I2C_Master_Seq_Transmit_IT or HAL_I2C_Master_Seq_Receive_IT or HAL_I2C_Master_Seq_Transmit_DMA or HAL_I2C_Master_Seq_Receive_DMA with option I2C_FIRST_AND_NEXT_FRAME then I2C_NEXT_FRAME. Then usage of this option I2C_LAST_FRAME_NO_STOP at the last Transmit or Receive sequence permit to call the opposite interface Receive or Transmit without stopping the communication and so generate a restart condition.
  - I2C_OTHER_FRAME: Sequential usage (Master only), this option allow to manage a restart condition after each call of the same master sequential interface. Usage can, transfer several bytes one by one with a restart with slave address between each bytes using HAL_I2C_Master_Seq_Transmit_IT or HAL_I2C_Master_Seq_Receive_IT or HAL_I2C_Master_Seq_Transmit_DMA or HAL_I2C_Master_Seq_Receive_DMA with option I2C_FIRST_FRAME then I2C_OTHER_FRAME. Then usage of this option I2C_OTHER_AND_LAST_FRAME at the last frame to help automatic generation of STOP condition.

- Different sequential I2C interfaces are listed below:
  - Sequential transmit in master I2C mode an amount of data in non-blocking mode using HAL_I2C_Master_Seq_Transmit_IT() or using HAL_I2C_Master_Seq_Transmit_DMA()
    - At transmission end of current frame transfer, HAL_I2C_MasterTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MasterTxCpltCallback()
  - Sequential receive in master I2C mode an amount of data in non-blocking mode using HAL_I2C_Master_Seq_Receive_IT() or using HAL_I2C_Master_Seq_Receive_DMA()
    - At reception end of current frame transfer, HAL_I2C_MasterRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MasterRxCpltCallback()
  - Abort a master IT or DMA I2C process communication with Interrupt using HAL_I2C_Master_Abort_IT()
    - End of abort process, HAL_I2C_AbortCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_AbortCpltCallback()
  - Enable/disable the Address listen mode in slave I2C mode using HAL_I2C_EnableListen_IT() HAL_I2C_DisableListen_IT()
    - When address slave I2C match, HAL_I2C_AddrCallback() is executed and users can add their own code to check the Address Match Code and the transmission direction request by master (Write/Read).
    - At Listen mode end HAL_I2C_ListenCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_ListenCpltCallback()
  - Sequential transmit in slave I2C mode an amount of data in non-blocking mode using HAL_I2C_Slave_Seq_Transmit_IT() or using HAL_I2C_Slave_Seq_Transmit_DMA()
    - At transmission end of current frame transfer, HAL_I2C_SlaveTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_SlaveTxCpltCallback()
  - Sequential receive in slave I2C mode an amount of data in non-blocking mode using HAL_I2C_Slave_Seq_Receive_IT() or using HAL_I2C_Slave_Seq_Receive_DMA()
    - At reception end of current frame transfer, HAL_I2C_SlaveRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_SlaveRxCpltCallback()
  - In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and users can add their own code by customization of function pointer HAL_I2C_ErrorCallback()
  - Discard a slave I2C process communication using __HAL_I2C_GENERATE_NACK() macro. This action will inform Master to generate a Stop condition to discard the communication.

Interrupt mode IO MEM operation

- Write an amount of data in non-blocking mode with Interrupt to a specific memory address using HAL_I2C_Mem_Write_IT()
- At Memory end of write transfer, HAL_I2C_MemTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MemTxCpltCallback()
- Read an amount of data in non-blocking mode with Interrupt from a specific memory address using HAL_I2C_Mem_Read_IT()
- At Memory end of read transfer, HAL_I2C_MemRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MemRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and users can add their own code by customization of function pointer HAL_I2C_ErrorCallback()

DMA mode IO operation

- Transmit in master mode an amount of data in non-blocking mode (DMA) using HAL_I2C_Master_Transmit_DMA()
- At transmission end of transfer, HAL_I2C_MasterTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MasterTxCpltCallback()

- Receive in master mode an amount of data in non-blocking mode (DMA) using HAL_I2C_Master_Receive_DMA()
- At reception end of transfer, HAL_I2C_MasterRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MasterRxCpltCallback()
- Transmit in slave mode an amount of data in non-blocking mode (DMA) using HAL_I2C_Slave_Transmit_DMA()
- At transmission end of transfer, HAL_I2C_SlaveTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_SlaveTxCpltCallback()
- Receive in slave mode an amount of data in non-blocking mode (DMA) using HAL_I2C_Slave_Receive_DMA()
- At reception end of transfer, HAL_I2C_SlaveRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_SlaveRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and users can add their own code by customization of function pointer HAL_I2C_ErrorCallback()
- Abort a master I2C process communication with Interrupt using HAL_I2C_Master_Abort_IT()
- End of abort process, HAL_I2C_AbortCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_AbortCpltCallback()
- Discard a slave I2C process communication using __HAL_I2C_GENERATE_NACK() macro. This action will inform Master to generate a Stop condition to discard the communication.

DMA mode IO MEM operation

- Write an amount of data in non-blocking mode with DMA to a specific memory address using HAL_I2C_Mem_Write_DMA()
- At Memory end of write transfer, HAL_I2C_MemTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MemTxCpltCallback()
- Read an amount of data in non-blocking mode with DMA from a specific memory address using HAL_I2C_Mem_Read_DMA()
- At Memory end of read transfer, HAL_I2C_MemRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MemRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and users can add their own code by customization of function pointer HAL_I2C_ErrorCallback()

I2C HAL driver macros list

Below the list of most used macros in I2C HAL driver.

- __HAL_I2C_ENABLE: Enable the I2C peripheral
- __HAL_I2C_DISABLE: Disable the I2C peripheral
- __HAL_I2C_GENERATE_NACK: Generate a Non-Acknowledge I2C peripheral in Slave mode
- __HAL_I2C_GET_FLAG: Check whether the specified I2C flag is set or not
- __HAL_I2C_CLEAR_FLAG: Clear the specified I2C pending flag
- __HAL_I2C_ENABLE_IT: Enable the specified I2C interrupt
- __HAL_I2C_DISABLE_IT: Disable the specified I2C interrupt

Callback registration

The compilation flag USE_HAL_I2C_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_I2C_RegisterCallback() or HAL_I2C_RegisterAddrCallback() to register an interrupt callback.

Function HAL_I2C_RegisterCallback() allows to register following callbacks:

- MasterTxCpltCallback : callback for Master transmission end of transfer.
- MasterRxCpltCallback : callback for Master reception end of transfer.
- SlaveTxCpltCallback : callback for Slave transmission end of transfer.
- SlaveRxCpltCallback : callback for Slave reception end of transfer.
- ListenCpltCallback : callback for end of listen mode.
- MemTxCpltCallback : callback for Memory transmission end of transfer.

- MemRxCpltCallback : callback for Memory reception end of transfer.
- ErrorCallback : callback for error detection.
- AbortCpltCallback : callback for abort completion process.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

For specific callback AddrCallback use dedicated register callbacks : HAL_I2C_RegisterAddrCallback().

Use function HAL_I2C_UnRegisterCallback to reset a callback to the default weak function. HAL_I2C_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- MasterTxCpltCallback : callback for Master transmission end of transfer.
- MasterRxCpltCallback : callback for Master reception end of transfer.
- SlaveTxCpltCallback : callback for Slave transmission end of transfer.
- SlaveRxCpltCallback : callback for Slave reception end of transfer.
- ListenCpltCallback : callback for end of listen mode.
- MemTxCpltCallback : callback for Memory transmission end of transfer.
- MemRxCpltCallback : callback for Memory reception end of transfer.
- ErrorCallback : callback for error detection.
- AbortCpltCallback : callback for abort completion process.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit.

For callback AddrCallback use dedicated register callbacks : HAL_I2C_UnRegisterAddrCallback().

By default, after the HAL_I2C_Init() and when the state is HAL_I2C_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_I2C_MasterTxCpltCallback(), HAL_I2C_MasterRxCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_I2C_Init()/ HAL_I2C_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the HAL_I2C_Init()/ HAL_I2C_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in HAL_I2C_STATE_READY state only. Exception done MspInit/MspDeInit functions that can be registered/unregistered in HAL_I2C_STATE_READY or HAL_I2C_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_I2C_RegisterCallback() before calling HAL_I2C_DeInit() or HAL_I2C_Init() function.

When the compilation flag USE_HAL_I2C_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

Note: You can refer to the I2C HAL driver header file for more useful macros

42.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and deinitialize the I2Cx peripheral:

- User must Implement HAL_I2C_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_I2C_Init() to configure the selected device with the selected configuration:
  - Clock Timing
  - Own Address 1
  - Addressing mode (Master, Slave)
  - Dual Addressing mode
  - Own Address 2
  - Own Address 2 Mask
  - General call mode
  - Nostretch mode
- Call the function HAL_I2C_DeInit() to restore the default configuration of the selected I2Cx peripheral.

This section contains the following APIs:

- *HAL_I2C_Init()*
- *HAL_I2C_DeInit()*
- *HAL_I2C_MspInit()*
- *HAL_I2C_MspDeInit()*
- *HAL_I2C_RegisterCallback()*
- *HAL_I2C_UnRegisterCallback()*
- *HAL_I2C_RegisterAddrCallback()*
- *HAL_I2C_UnRegisterAddrCallback()*

42.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the I2C data transfers.

1. There are two modes of transfer:
   - Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
   - No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated I2C IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
2. Blocking mode functions are :
   - HAL_I2C_Master_Transmit()
   - HAL_I2C_Master_Receive()
   - HAL_I2C_Slave_Transmit()
   - HAL_I2C_Slave_Receive()
   - HAL_I2C_Mem_Write()
   - HAL_I2C_Mem_Read()
   - HAL_I2C_IsDeviceReady()
3. No-Blocking mode functions with Interrupt are :
   - HAL_I2C_Master_Transmit_IT()
   - HAL_I2C_Master_Receive_IT()
   - HAL_I2C_Slave_Transmit_IT()
   - HAL_I2C_Slave_Receive_IT()
   - HAL_I2C_Mem_Write_IT()
   - HAL_I2C_Mem_Read_IT()
   - HAL_I2C_Master_Seq_Transmit_IT()
   - HAL_I2C_Master_Seq_Receive_IT()
   - HAL_I2C_Slave_Seq_Transmit_IT()
   - HAL_I2C_Slave_Seq_Receive_IT()
   - HAL_I2C_EnableListen_IT()
   - HAL_I2C_DisableListen_IT()
   - HAL_I2C_Master_Abort_IT()

4. No-Blocking mode functions with DMA are :
   - HAL_I2C_Master_Transmit_DMA()
   - HAL_I2C_Master_Receive_DMA()
   - HAL_I2C_Slave_Transmit_DMA()
   - HAL_I2C_Slave_Receive_DMA()
   - HAL_I2C_Mem_Write_DMA()
   - HAL_I2C_Mem_Read_DMA()
   - HAL_I2C_Master_Seq_Transmit_DMA()
   - HAL_I2C_Master_Seq_Receive_DMA()
   - HAL_I2C_Slave_Seq_Transmit_DMA()
   - HAL_I2C_Slave_Seq_Receive_DMA()
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
   - HAL_I2C_MasterTxCpltCallback()
   - HAL_I2C_MasterRxCpltCallback()
   - HAL_I2C_SlaveTxCpltCallback()
   - HAL_I2C_SlaveRxCpltCallback()
   - HAL_I2C_MemTxCpltCallback()
   - HAL_I2C_MemRxCpltCallback()
   - HAL_I2C_AddrCallback()
   - HAL_I2C_ListenCpltCallback()
   - HAL_I2C_ErrorCallback()
   - HAL_I2C_AbortCpltCallback()

This section contains the following APIs:

- ***HAL_I2C_Master_Transmit()***
- ***HAL_I2C_Master_Receive()***
- ***HAL_I2C_Slave_Transmit()***
- ***HAL_I2C_Slave_Receive()***
- ***HAL_I2C_Master_Transmit_IT()***
- ***HAL_I2C_Master_Receive_IT()***
- ***HAL_I2C_Slave_Transmit_IT()***
- ***HAL_I2C_Slave_Receive_IT()***
- ***HAL_I2C_Master_Transmit_DMA()***
- ***HAL_I2C_Master_Receive_DMA()***
- ***HAL_I2C_Slave_Transmit_DMA()***
- ***HAL_I2C_Slave_Receive_DMA()***
- ***HAL_I2C_Mem_Write()***
- ***HAL_I2C_Mem_Read()***
- ***HAL_I2C_Mem_Write_IT()***
- ***HAL_I2C_Mem_Read_IT()***
- ***HAL_I2C_Mem_Write_DMA()***
- ***HAL_I2C_Mem_Read_DMA()***
- ***HAL_I2C_IsDeviceReady()***
- ***HAL_I2C_Master_Seq_Transmit_IT()***
- ***HAL_I2C_Master_Seq_Transmit_DMA()***
- ***HAL_I2C_Master_Seq_Receive_IT()***
- ***HAL_I2C_Master_Seq_Receive_DMA()***
- ***HAL_I2C_Slave_Seq_Transmit_IT()***
- ***HAL_I2C_Slave_Seq_Transmit_DMA()***
- ***HAL_I2C_Slave_Seq_Receive_IT()***
- ***HAL_I2C_Slave_Seq_Receive_DMA()***

- *HAL_I2C_EnableListen_IT()*
- *HAL_I2C_DisableListen_IT()*
- *HAL_I2C_Master_Abort_IT()*

42.2.4 Peripheral State, Mode and Error functions

This subsection permit to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- *HAL_I2C_GetState()*
- *HAL_I2C_GetMode()*
- *HAL_I2C_GetError()*

42.2.5 Detailed description of functions

HAL_I2C_Init

Function name

HAL_StatusTypeDef HAL_I2C_Init (I2C_HandleTypeDef * hi2c)

Function description

Initializes the I2C according to the specified parameters in the I2C_InitTypeDef and initialize the associated handle.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **HAL:** status

HAL_I2C_DeInit

Function name

HAL_StatusTypeDef HAL_I2C_DeInit (I2C_HandleTypeDef * hi2c)

Function description

DeInitialize the I2C peripheral.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **HAL:** status

HAL_I2C_MspInit

Function name

void HAL_I2C_MspInit (I2C_HandleTypeDef * hi2c)

Function description

Initialize the I2C MSP.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_MspDeInit

Function name

void HAL_I2C_MspDeInit (I2C_HandleTypeDef * hi2c)

Function description

DeInitialize the I2C MSP.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_RegisterCallback

Function name

HAL_StatusTypeDef HAL_I2C_RegisterCallback (I2C_HandleTypeDef * hi2c, HAL_I2C_CallbackIDTypeDef CallbackID, pI2C_CallbackTypeDef pCallback)

Function description

Register a User I2C Callback To be used instead of the weak predefined callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_I2C_MASTER_TX_COMPLETE_CB_ID Master Tx Transfer completed callback ID
  - HAL_I2C_MASTER_RX_COMPLETE_CB_ID Master Rx Transfer completed callback ID
  - HAL_I2C_SLAVE_TX_COMPLETE_CB_ID Slave Tx Transfer completed callback ID
  - HAL_I2C_SLAVE_RX_COMPLETE_CB_ID Slave Rx Transfer completed callback ID
  - HAL_I2C_LISTEN_COMPLETE_CB_ID Listen Complete callback ID
  - HAL_I2C_MEM_TX_COMPLETE_CB_ID Memory Tx Transfer callback ID
  - HAL_I2C_MEM_RX_COMPLETE_CB_ID Memory Rx Transfer completed callback ID
  - HAL_I2C_ERROR_CB_ID Error callback ID
  - HAL_I2C_ABORT_CB_ID Abort callback ID
  - HAL_I2C_MSPINIT_CB_ID MspInit callback ID
  - HAL_I2C_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

Notes

- The HAL_I2C_RegisterCallback() may be called before HAL_I2C_Init() in HAL_I2C_STATE_RESET to register callbacks for HAL_I2C_MSPINIT_CB_ID and HAL_I2C_MSPDEINIT_CB_ID.

HAL_I2C_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_I2C_UnRegisterCallback (I2C_HandleTypeDef * hi2c, HAL_I2C_CallbackIDTypeDef CallbackID)

Function description

Unregister an I2C Callback I2C callback is redirected to the weak predefined callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values: This parameter can be one of the following values:
  - HAL_I2C_MASTER_TX_COMPLETE_CB_ID Master Tx Transfer completed callback ID
  - HAL_I2C_MASTER_RX_COMPLETE_CB_ID Master Rx Transfer completed callback ID
  - HAL_I2C_SLAVE_TX_COMPLETE_CB_ID Slave Tx Transfer completed callback ID
  - HAL_I2C_SLAVE_RX_COMPLETE_CB_ID Slave Rx Transfer completed callback ID
  - HAL_I2C_LISTEN_COMPLETE_CB_ID Listen Complete callback ID
  - HAL_I2C_MEM_TX_COMPLETE_CB_ID Memory Tx Transfer callback ID
  - HAL_I2C_MEM_RX_COMPLETE_CB_ID Memory Rx Transfer completed callback ID
  - HAL_I2C_ERROR_CB_ID Error callback ID
  - HAL_I2C_ABORT_CB_ID Abort callback ID
  - HAL_I2C_MSPINIT_CB_ID MspInit callback ID
  - HAL_I2C_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

Notes

- The HAL_I2C_UnRegisterCallback() may be called before HAL_I2C_Init() in HAL_I2C_STATE_RESET to un-register callbacks for HAL_I2C_MSPINIT_CB_ID and HAL_I2C_MSPDEINIT_CB_ID.

HAL_I2C_RegisterAddrCallback

Function name

HAL_StatusTypeDef HAL_I2C_RegisterAddrCallback (I2C_HandleTypeDef * hi2c, pI2C_AddrCallbackTypeDef pCallback)

Function description

Register the Slave Address Match I2C Callback To be used instead of the weak HAL_I2C_AddrCallback() predefined callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pCallback:** pointer to the Address Match Callback function

Return values

- **HAL:** status

HAL_I2C_UnRegisterAddrCallback

Function name

HAL_StatusTypeDef HAL_I2C_UnRegisterAddrCallback (I2C_HandleTypeDef * hi2c)

Function description

UnRegister the Slave Address Match I2C Callback Info Ready I2C Callback is redirected to the weak HAL_I2C_AddrCallback() predefined callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **HAL:** status

HAL_I2C_Master_Transmit

Function name

HAL_StatusTypeDef HAL_I2C_Master_Transmit (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Transmits in master mode an amount of data in blocking mode.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_I2C_Master_Receive

Function name

HAL_StatusTypeDef HAL_I2C_Master_Receive (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receives in master mode an amount of data in blocking mode.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_I2C_Slave_Transmit

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Transmit (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Transmits in slave mode an amount of data in blocking mode.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_I2C_Slave_Receive

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Receive (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive in slave mode an amount of data in blocking mode.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_I2C_Mem_Write

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Write (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Write an amount of data in blocking mode to a specific memory address.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_I2C_Mem_Read

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Read (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Read an amount of data in blocking mode from a specific memory address.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_I2C_IsDeviceReady

Function name

HAL_StatusTypeDef HAL_I2C_IsDeviceReady (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout)

Function description

Checks if target device is ready for communication.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **Trials:** Number of trials
- **Timeout:** Timeout duration

Return values

- **HAL:** status

Notes

- This function is used with Memory devices

HAL_I2C_Master_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I2C_Master_Transmit_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)

Function description

Transmit in master mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Master_Receive_IT

Function name

HAL_StatusTypeDef HAL_I2C_Master_Receive_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)

Function description

Receive in master mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Slave_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Transmit_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)

Function description

Transmit in slave mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Slave_Receive_IT

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Receive_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)

Function description

Receive in slave mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Mem_Write_IT

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Write_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)

Function description

Write an amount of data in non-blocking mode with Interrupt to a specific memory address.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Mem_Read_IT

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Read_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)

Function description

Read an amount of data in non-blocking mode with Interrupt from a specific memory address.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Master_Seq_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I2C_Master_Seq_Transmit_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential transmit in master I2C mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Master_Seq_Receive_IT

Function name

HAL_StatusTypeDef HAL_I2C_Master_Seq_Receive_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential receive in master I2C mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Slave_Seq_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Seq_Transmit_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential transmit in slave/device I2C mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Slave_Seq_Receive_IT

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Seq_Receive_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential receive in slave/device I2C mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_EnableListen_IT

Function name

HAL_StatusTypeDef HAL_I2C_EnableListen_IT (I2C_HandleTypeDef * hi2c)

Function description

Enable the Address listen mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **HAL:** status

HAL_I2C_DisableListen_IT

Function name

HAL_StatusTypeDef HAL_I2C_DisableListen_IT (I2C_HandleTypeDef * hi2c)

Function description

Disable the Address listen mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C

Return values

- **HAL:** status

HAL_I2C_Master_Abort_IT

Function name

HAL_StatusTypeDef HAL_I2C_Master_Abort_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress)

Function description

Abort a master I2C IT or DMA process communication with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface

Return values

- **HAL:** status

HAL_I2C_Master_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Master_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)

Function description

Transmit in master mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Master_Receive_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Master_Receive_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)

Function description

Receive in master mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Slave_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)

Function description

Transmit in slave mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Slave_Receive_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Receive_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)

Function description

Receive in slave mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Mem_Write_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Write_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)

Function description

Write an amount of data in non-blocking mode with DMA to a specific memory address.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Mem_Read_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Read_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)

Function description

Reads an amount of data in non-blocking mode with DMA from a specific memory address.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be read

Return values

- **HAL:** status

HAL_I2C_Master_Seq_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Master_Seq_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential transmit in master I2C mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Master_Seq_Receive_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Master_Seq_Receive_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential receive in master I2C mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Slave_Seq_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Seq_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential transmit in slave/device I2C mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Slave_Seq_Receive_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Seq_Receive_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential receive in slave/device I2C mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_EV_IRQHandler

Function name

void HAL_I2C_EV_IRQHandler (I2C_HandleTypeDef * hi2c)

Function description

This function handles I2C event interrupt request.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_ER_IRQHandler

Function name

void HAL_I2C_ER_IRQHandler (I2C_HandleTypeDef * hi2c)

Function description

This function handles I2C error interrupt request.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_MasterTxCpltCallback

Function name

void HAL_I2C_MasterTxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Master Tx Transfer completed callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_MasterRxCpltCallback

Function name

void HAL_I2C_MasterRxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Master Rx Transfer completed callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_SlaveTxCpltCallback

Function name

void HAL_I2C_SlaveTxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Slave Tx Transfer completed callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_SlaveRxCpltCallback

Function name

void HAL_I2C_SlaveRxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Slave Rx Transfer completed callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_AddrCallback

Function name

void HAL_I2C_AddrCallback (I2C_HandleTypeDef * hi2c, uint8_t TransferDirection, uint16_t AddrMatchCode)

Function description

Slave Address Match callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **TransferDirection:** Master request Transfer Direction (Write/Read), value of I2C Transfer Direction Master Point of View
- **AddrMatchCode:** Address Match Code

Return values

- **None:**

HAL_I2C_ListenCpltCallback

Function name

void HAL_I2C_ListenCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Listen Complete callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_MemTxCpltCallback

Function name

void HAL_I2C_MemTxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Memory Tx Transfer completed callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_MemRxCpltCallback

Function name

void HAL_I2C_MemRxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Memory Rx Transfer completed callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_ErrorCallback

Function name

void HAL_I2C_ErrorCallback (I2C_HandleTypeDef * hi2c)

Function description

I2C error callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_AbortCpltCallback

Function name

void HAL_I2C_AbortCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

I2C abort callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_GetState

Function name

HAL_I2C_StateTypeDef HAL_I2C_GetState (I2C_HandleTypeDef * hi2c)

Function description

Return the I2C handle state.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **HAL:** state

HAL_I2C_GetMode

Function name

HAL_I2C_ModeTypeDef HAL_I2C_GetMode (I2C_HandleTypeDef * hi2c)

Function description

Returns the I2C Master, Slave, Memory or no mode.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module

Return values

- **HAL:** mode

HAL_I2C_GetError

Function name

uint32_t HAL_I2C_GetError (I2C_HandleTypeDef * hi2c)

Function description

Return the I2C error code.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **I2C:** Error Code

42.3 I2C Firmware driver defines

The following section lists the various define and macros of the module.

42.3.1 I2C

I2C

I2C Addressing Mode

I2C_ADDRESSINGMODE_7BIT

I2C_ADDRESSINGMODE_10BIT

I2C Dual Addressing Mode

I2C_DUALADDRESS_DISABLE

I2C_DUALADDRESS_ENABLE

I2C Error Code definition

HAL_I2C_ERROR_NONE

No error

HAL_I2C_ERROR_BERR

BERR error

HAL_I2C_ERROR_ARLO

ARLO error

HAL_I2C_ERROR_AF

ACKF error

HAL_I2C_ERROR_OVR

OVR error

HAL_I2C_ERROR_DMA

DMA transfer error

HAL_I2C_ERROR_TIMEOUT

Timeout error

HAL_I2C_ERROR_SIZE

Size Management error

HAL_I2C_ERROR_DMA_PARAM

DMA Parameter Error

HAL_I2C_ERROR_INVALID_CALLBACK

Invalid Callback error

HAL_I2C_ERROR_INVALID_PARAM

Invalid Parameters error

I2C Exported Macros

__HAL_I2C_RESET_HANDLE_STATE

Description:

- Reset I2C handle state.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None

__HAL_I2C_ENABLE_IT

Description:

- Enable the specified I2C interrupt.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __INTERRUPT__: specifies the interrupt source to enable. This parameter can be one of the following values:
  - I2C_IT_ERRI Errors interrupt enable
  - I2C_IT_TCI Transfer complete interrupt enable
  - I2C_IT_STOPI STOP detection interrupt enable
  - I2C_IT_NACKI NACK received interrupt enable
  - I2C_IT_ADDRI Address match interrupt enable
  - I2C_IT_RXI RX interrupt enable
  - I2C_IT_TXI TX interrupt enable

Return value:

- None

__HAL_I2C_DISABLE_IT

Description:

- Disable the specified I2C interrupt.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __INTERRUPT__: specifies the interrupt source to disable. This parameter can be one of the following values:
  - I2C_IT_ERRI Errors interrupt enable
  - I2C_IT_TCI Transfer complete interrupt enable
  - I2C_IT_STOPI STOP detection interrupt enable
  - I2C_IT_NACKI NACK received interrupt enable
  - I2C_IT_ADDRI Address match interrupt enable
  - I2C_IT_RXI RX interrupt enable
  - I2C_IT_TXI TX interrupt enable

Return value:

- None

__HAL_I2C_GET_IT_SOURCE

Description:

- Check whether the specified I2C interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __INTERRUPT__: specifies the I2C interrupt source to check. This parameter can be one of the following values:
  - I2C_IT_ERRI Errors interrupt enable
  - I2C_IT_TCI Transfer complete interrupt enable
  - I2C_IT_STOPI STOP detection interrupt enable
  - I2C_IT_NACKI NACK received interrupt enable
  - I2C_IT_ADDRI Address match interrupt enable
  - I2C_IT_RXI RX interrupt enable
  - I2C_IT_TXI TX interrupt enable

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

I2C_FLAG_MASK

Description:

- Check whether the specified I2C flag is set or not.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - I2C_FLAG_TXE Transmit data register empty
  - I2C_FLAG_TXIS Transmit interrupt status
  - I2C_FLAG_RXNE Receive data register not empty
  - I2C_FLAG_ADDR Address matched (slave mode)
  - I2C_FLAG_AF Acknowledge failure received flag
  - I2C_FLAG_STOPF STOP detection flag
  - I2C_FLAG_TC Transfer complete (master mode)
  - I2C_FLAG_TCR Transfer complete reload
  - I2C_FLAG_BERR Bus error
  - I2C_FLAG_ARLO Arbitration lost
  - I2C_FLAG_OVR Overrun/Underrun
  - I2C_FLAG_PECERR PEC error in reception
  - I2C_FLAG_TIMEOUT Timeout or Tlow detection flag
  - I2C_FLAG_ALERT SMBus alert
  - I2C_FLAG_BUSY Bus busy
  - I2C_FLAG_DIR Transfer direction (slave mode)

Return value:

- The: new state of __FLAG__ (SET or RESET).

__HAL_I2C_GET_FLAG

__HAL_I2C_CLEAR_FLAG

Description:

- Clear the I2C pending flags which are cleared by writing 1 in a specific bit.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - I2C_FLAG_TXE Transmit data register empty
  - I2C_FLAG_ADDR Address matched (slave mode)
  - I2C_FLAG_AF Acknowledge failure received flag
  - I2C_FLAG_STOPF STOP detection flag
  - I2C_FLAG_BERR Bus error
  - I2C_FLAG_ARLO Arbitration lost
  - I2C_FLAG_OVR Overrun/Underrun
  - I2C_FLAG_PECERR PEC error in reception
  - I2C_FLAG_TIMEOUT Timeout or Tlow detection flag
  - I2C_FLAG_ALERT SMBus alert

Return value:

- None

__HAL_I2C_ENABLE

Description:

- Enable the specified I2C peripheral.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None

__HAL_I2C_DISABLE

Description:

- Disable the specified I2C peripheral.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None

__HAL_I2C_GENERATE_NACK

Description:

- Generate a Non-Acknowledge I2C peripheral in Slave mode.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None

I2C Flag definition

I2C_FLAG_TXE

I2C_FLAG_TXIS

I2C_FLAG_RXNE

I2C_FLAG_ADDR

I2C_FLAG_AF

I2C_FLAG_STOPF

I2C_FLAG_TC

I2C_FLAG_TCR

I2C_FLAG_BERR

I2C_FLAG_ARLO

I2C_FLAG_OVR

I2C_FLAG_PECERR

I2C_FLAG_TIMEOUT

I2C_FLAG_ALERT

I2C_FLAG_BUSY

I2C_FLAG_DIR

I2C General Call Addressing Mode

I2C_GENERALCALL_DISABLE

I2C_GENERALCALL_ENABLE

I2C Interrupt configuration definition

I2C_IT_ERRI

I2C_IT_TCI

I2C_IT_STOPI

I2C_IT_NACKI

I2C_IT_ADDRI

I2C_IT_RXI

I2C_IT_TXI

I2C Memory Address Size

I2C_MEMADD_SIZE_8BIT

I2C_MEMADD_SIZE_16BIT

I2C No-Stretch Mode

I2C_NOSTRETCH_DISABLE

I2C_NOSTRETCH_ENABLE

I2C Own Address2 Masks

I2C_OA2_NOMASK

I2C_OA2_MASK01

I2C_OA2_MASK02

I2C_OA2_MASK03

I2C_OA2_MASK04

I2C_OA2_MASK05

I2C_OA2_MASK06

I2C_OA2_MASK07

I2C Reload End Mode

I2C_RELOAD_MODE

I2C_AUTOEND_MODE

I2C_SOFTEND_MODE

I2C Start or Stop Mode

I2C_NO_STARTSTOP

I2C_GENERATE_STOP

I2C_GENERATE_START_READ

I2C_GENERATE_START_WRITE

I2C Transfer Direction Master Point of View

I2C_DIRECTION_TRANSMIT

I2C_DIRECTION_RECEIVE

I2C Sequential Transfer Options

I2C_FIRST_FRAME

I2C_FIRST_AND_NEXT_FRAME

I2C_NEXT_FRAME

I2C_FIRST_AND_LAST_FRAME

I2C_LAST_FRAME

I2C_LAST_FRAME_NO_STOP

I2C_OTHER_FRAME

I2C_OTHER_AND_LAST_FRAME

43 HAL I2C Extension Driver

43.1 I2CEx Firmware driver API description

The following section lists the various functions of the I2CEx library.

43.1.1 I2C peripheral Extended features

Comparing to other previous devices, the I2C interface for STM32H7xx devices contains the following additional features

- Possibility to disable or enable Analog Noise Filter
- Use of a configured Digital Noise Filter
- Disable or enable wakeup from Stop mode(s)
- Disable or enable Fast Mode Plus

43.1.2 How to use this driver

This driver provides functions to configure Noise Filter and Wake Up Feature

1. Configure I2C Analog noise filter using the function HAL_I2CEx_ConfigAnalogFilter()
2. Configure I2C Digital noise filter using the function HAL_I2CEx_ConfigDigitalFilter()
3. Configure the enable or disable of I2C Wake Up Mode using the functions :
   - HAL_I2CEx_EnableWakeUp()
   - HAL_I2CEx_DisableWakeUp()
4. Configure the enable or disable of fast mode plus driving capability using the functions :
   - HAL_I2CEx_EnableFastModePlus()
   - HAL_I2CEx_DisableFastModePlus()

43.1.3 Filter Mode Functions

This section provides functions allowing to:

- Configure Noise Filters

This section contains the following APIs:

- ***HAL_I2CEx_ConfigAnalogFilter()***
- ***HAL_I2CEx_ConfigDigitalFilter()***

43.1.4 WakeUp Mode Functions

This section provides functions allowing to:

- Configure Wake Up Feature

This section contains the following APIs:

- ***HAL_I2CEx_EnableWakeUp()***
- ***HAL_I2CEx_DisableWakeUp()***

43.1.5 Fast Mode Plus Functions

This section provides functions allowing to:

- Configure Fast Mode Plus

This section contains the following APIs:

- ***HAL_I2CEx_EnableFastModePlus()***
- ***HAL_I2CEx_DisableFastModePlus()***

43.1.6 Detailed description of functions

HAL_I2CEx_ConfigAnalogFilter

Function name

HAL_StatusTypeDef HAL_I2CEx_ConfigAnalogFilter (I2C_HandleTypeDef * hi2c, uint32_t AnalogFilter)

Function description

Configure I2C Analog noise filter.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.
- **AnalogFilter:** New state of the Analog filter.

Return values

- **HAL:** status

HAL_I2CEx_ConfigDigitalFilter

Function name

HAL_StatusTypeDef HAL_I2CEx_ConfigDigitalFilter (I2C_HandleTypeDef * hi2c, uint32_t DigitalFilter)

Function description

Configure I2C Digital noise filter.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.
- **DigitalFilter:** Coefficient of digital noise filter between Min_Data=0x00 and Max_Data=0x0F.

Return values

- **HAL:** status

HAL_I2CEx_EnableWakeUp

Function name

HAL_StatusTypeDef HAL_I2CEx_EnableWakeUp (I2C_HandleTypeDef * hi2c)

Function description

Enable I2C wakeup from Stop mode(s).

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.

Return values

- **HAL:** status

HAL_I2CEx_DisableWakeUp

Function name

HAL_StatusTypeDef HAL_I2CEx_DisableWakeUp (I2C_HandleTypeDef * hi2c)

Function description

Disable I2C wakeup from Stop mode(s).

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.

Return values

- **HAL:** status

HAL_I2CEx_EnableFastModePlus

Function name

void HAL_I2CEx_EnableFastModePlus (uint32_t ConfigFastModePlus)

Function description

Enable the I2C fast mode plus driving capability.

Parameters

- **ConfigFastModePlus:** Selects the pin. This parameter can be one of the I2C Extended Fast Mode Plus values

Return values

- **None:**

Notes

- For I2C1, fast mode plus driving capability can be enabled on all selected I2C1 pins using I2C_FASTMODEPLUS_I2C1 parameter or independently on each one of the following pins PB6, PB7, PB8 and PB9.
- For remaining I2C1 pins (PA14, PA15...) fast mode plus driving capability can be enabled only by using I2C_FASTMODEPLUS_I2C1 parameter.
- For all I2C2 pins fast mode plus driving capability can be enabled only by using I2C_FASTMODEPLUS_I2C2 parameter.
- For all I2C3 pins fast mode plus driving capability can be enabled only by using I2C_FASTMODEPLUS_I2C3 parameter.
- For all I2C4 pins fast mode plus driving capability can be enabled only by using I2C_FASTMODEPLUS_I2C4 parameter.
- For all I2C5 pins fast mode plus driving capability can be enabled only by using I2C_FASTMODEPLUS_I2C5 parameter.

HAL_I2CEx_DisableFastModePlus

Function name

void HAL_I2CEx_DisableFastModePlus (uint32_t ConfigFastModePlus)

Function description

Disable the I2C fast mode plus driving capability.

Parameters

- **ConfigFastModePlus:** Selects the pin. This parameter can be one of the I2C Extended Fast Mode Plus values

Return values

- **None:**

Notes

- For I2C1, fast mode plus driving capability can be disabled on all selected I2C1 pins using I2C_FASTMODEPLUS_I2C1 parameter or independently on each one of the following pins PB6, PB7, PB8 and PB9.
- For remaining I2C1 pins (PA14, PA15...) fast mode plus driving capability can be disabled only by using I2C_FASTMODEPLUS_I2C1 parameter.
- For all I2C2 pins fast mode plus driving capability can be disabled only by using I2C_FASTMODEPLUS_I2C2 parameter.
- For all I2C3 pins fast mode plus driving capability can be disabled only by using I2C_FASTMODEPLUS_I2C3 parameter.
- For all I2C4 pins fast mode plus driving capability can be disabled only by using I2C_FASTMODEPLUS_I2C4 parameter.
- For all I2C5 pins fast mode plus driving capability can be disabled only by using I2C_FASTMODEPLUS_I2C5 parameter.

43.2 I2CEx Firmware driver defines

The following section lists the various define and macros of the module.

43.2.1 I2CEx

I2CEx

I2C Extended Analog Filter

I2C_ANALOGFILTER_ENABLE

I2C_ANALOGFILTER_DISABLE

I2C Extended Fast Mode Plus

I2C_FMP_NOT_SUPPORTED

Fast Mode Plus not supported

I2C_FASTMODEPLUS_PB6

Enable Fast Mode Plus on PB6

I2C_FASTMODEPLUS_PB7

Enable Fast Mode Plus on PB7

I2C_FASTMODEPLUS_PB8

Enable Fast Mode Plus on PB8

I2C_FASTMODEPLUS_PB9

Enable Fast Mode Plus on PB9

I2C_FASTMODEPLUS_I2C1

Enable Fast Mode Plus on I2C1 pins

I2C_FASTMODEPLUS_I2C2

Enable Fast Mode Plus on I2C2 pins

I2C_FASTMODEPLUS_I2C3

Enable Fast Mode Plus on I2C3 pins

I2C_FASTMODEPLUS_I2C4

Enable Fast Mode Plus on I2C4 pins

I2C_FASTMODEPLUS_I2C5

Fast Mode Plus I2C5 not supported

44 HAL I2S Generic Driver

44.1 I2S Firmware driver registers structures

44.1.1 I2S_InitTypeDef

I2S_InitTypeDef is defined in the stm32h7xx_hal_i2s.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t Standard***
- ***uint32_t DataFormat***
- ***uint32_t MCLKOutput***
- ***uint32_t AudioFreq***
- ***uint32_t CPOL***
- ***uint32_t FirstBit***
- ***uint32_t WSInversion***
- ***uint32_t Data24BitAlignment***
- ***uint32_t MasterKeepIOState***

Field Documentation

- ***uint32_t I2S_InitTypeDef::Mode***
  Specifies the I2S operating mode. This parameter can be a value of ***I2S_Mode***
- ***uint32_t I2S_InitTypeDef::Standard***
  Specifies the standard used for the I2S communication. This parameter can be a value of ***I2S_Standard***
- ***uint32_t I2S_InitTypeDef::DataFormat***
  Specifies the data format for the I2S communication. This parameter can be a value of ***I2S_Data_Format***
- ***uint32_t I2S_InitTypeDef::MCLKOutput***
  Specifies whether the I2S MCLK output is enabled or not. This parameter can be a value of ***I2S_MCLK_Output***
- ***uint32_t I2S_InitTypeDef::AudioFreq***
  Specifies the frequency selected for the I2S communication. This parameter can be a value of ***I2S_Audio_Frequency***
- ***uint32_t I2S_InitTypeDef::CPOL***
  Specifies the idle state of the I2S clock. This parameter can be a value of ***I2S_Clock_Polarity***
- ***uint32_t I2S_InitTypeDef::FirstBit***
  Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of ***I2S_MSB_LSB_Transmission***
- ***uint32_t I2S_InitTypeDef::WSInversion***
  Control the Word Select Inversion. This parameter can be a value of ***I2S_WSInversion***
- ***uint32_t I2S_InitTypeDef::Data24BitAlignment***
  Specifies the Data Padding for 24 bits data length This parameter can be a value of ***I2S_Data_24Bit_Alignment***
- ***uint32_t I2S_InitTypeDef::MasterKeepIOState***
  Control of Alternate function GPIOs state This parameter can be a value of ***I2S_Master_Keep_IO_State***

44.1.2 __I2S_HandleTypeDef

__I2S_HandleTypeDef is defined in the stm32h7xx_hal_i2s.h

Data Fields

- ***SPI_TypeDef * Instance***

- ***I2S_InitTypeDef Init***
- ***const uint16_t * pTxBuffPtr***
- ***__IO uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***
- ***uint16_t * pRxBuffPtr***
- ***__IO uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***void(* RxISR***
- ***void(* TxISR***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***__IO HAL_LockTypeDef Lock***
- ***__IO HAL_I2S_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* TxCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* TxRxCpltCallback***
- ***void(* TxHalfCpltCallback***
- ***void(* RxHalfCpltCallback***
- ***void(* TxRxHalfCpltCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***SPI_TypeDef* __I2S_HandleTypeDef::Instance***
  I2S registers base address
- ***I2S_InitTypeDef __I2S_HandleTypeDef::Init***
  I2S communication parameters
- ***const uint16_t* __I2S_HandleTypeDef::pTxBuffPtr***
  Pointer to I2S Tx transfer buffer
- ***__IO uint16_t __I2S_HandleTypeDef::TxXferSize***
  I2S Tx transfer size
- ***__IO uint16_t __I2S_HandleTypeDef::TxXferCount***
  I2S Tx transfer Counter
- ***uint16_t* __I2S_HandleTypeDef::pRxBuffPtr***
  Pointer to I2S Rx transfer buffer
- ***__IO uint16_t __I2S_HandleTypeDef::RxXferSize***
  I2S Rx transfer size
- ***__IO uint16_t __I2S_HandleTypeDef::RxXferCount***
  I2S Rx transfer counter (This field is initialized at the same value as transfer size at the beginning of the transfer and decremented when a sample is received NbSamplesReceived = RxBufferSize-RxBufferCount)
- ***void(* __I2S_HandleTypeDef::RxISR)(struct __I2S_HandleTypeDef *hi2s)***
  function pointer on Rx ISR
- ***void(* __I2S_HandleTypeDef::TxISR)(struct __I2S_HandleTypeDef *hi2s)***
  function pointer on Tx ISR
- ***DMA_HandleTypeDef* __I2S_HandleTypeDef::hdmatx***
  I2S Tx DMA handle parameters
- ***DMA_HandleTypeDef* __I2S_HandleTypeDef::hdmarx***
  I2S Rx DMA handle parameters

- ***__IO HAL_LockTypeDef __I2S_HandleTypeDef::Lock***
  I2S locking object
- ***__IO HAL_I2S_StateTypeDef __I2S_HandleTypeDef::State***
  I2S communication state
- ***__IO uint32_t __I2S_HandleTypeDef::ErrorCode***
  I2S Error code This parameter can be a value of ***I2S_Error***
- ***void(* __I2S_HandleTypeDef::TxCpltCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S Tx Completed callback
- ***void(* __I2S_HandleTypeDef::RxCpltCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S Rx Completed callback
- ***void(* __I2S_HandleTypeDef::TxRxCpltCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S TxRx Completed callback
- ***void(* __I2S_HandleTypeDef::TxHalfCpltCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S Tx Half Completed callback
- ***void(* __I2S_HandleTypeDef::RxHalfCpltCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S Rx Half Completed callback
- ***void(* __I2S_HandleTypeDef::TxRxHalfCpltCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S TxRx Half Completed callback
- ***void(* __I2S_HandleTypeDef::ErrorCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S Error callback
- ***void(* __I2S_HandleTypeDef::MspInitCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S Msp Init callback
- ***void(* __I2S_HandleTypeDef::MspDeInitCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S Msp DeInit callback

44.2 I2S Firmware driver API description

The following section lists the various functions of the I2S library.

44.2.1 How to use this driver

The I2S HAL driver can be used as follow:

1. Declare a I2S_HandleTypeDef handle structure.
2. Initialize the I2S low level resources by implement the HAL_I2S_MspInit() API:
   a. Enable the SPIx interface clock.
   b. I2S pins configuration:
      - Enable the clock for the I2S GPIOs.
      - Configure these I2S pins as alternate function pull-up.
   c. NVIC configuration if you need to use interrupt process (HAL_I2S_Transmit_IT() and HAL_I2S_Receive_IT() APIs).
      - Configure the I2Sx interrupt priority.
      - Enable the NVIC I2S IRQ handle.
   d. DMA Configuration if you need to use DMA process (HAL_I2S_Transmit_DMA() and HAL_I2S_Receive_DMA() APIs:
      - Declare a DMA handle structure for the Tx/Rx Stream/Channel.
      - Enable the DMAx interface clock.
      - Configure the declared DMA handle structure with the required Tx/Rx parameters.
      - Configure the DMA Tx/Rx Stream/Channel.
      - Associate the initialized DMA handle to the I2S DMA Tx/Rx handle.
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx Stream/Channel.

3. Program the Mode, Standard, Data Format, MCLK Output, Audio frequency and Polarity using HAL_I2S_Init() function.

Note: The specific I2S interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_I2S_ENABLE_IT() and __HAL_I2S_DISABLE_IT() inside the transmit and receive process.

– *External clock source is configured after setting correctly the define constant EXTERNAL_CLOCK_VALUE in the stm32h7xx_hal_conf.h file.*

4. Three mode of operations are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_I2S_Transmit()
- Receive an amount of data in blocking mode using HAL_I2S_Receive()

Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL_I2S_Transmit_IT()
- At transmission end of half transfer HAL_I2S_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxHalfCpltCallback
- At transmission end of transfer HAL_I2S_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL_I2S_Receive_IT()
- At reception end of half transfer HAL_I2S_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxHalfCpltCallback
- At reception end of transfer HAL_I2S_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxCpltCallback
- In case of transfer Error, HAL_I2S_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2S_ErrorCallback

DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL_I2S_Transmit_DMA()
- At transmission end of half transfer HAL_I2S_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxHalfCpltCallback
- At transmission end of transfer HAL_I2S_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL_I2S_Receive_DMA()
- At reception end of half transfer HAL_I2S_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxHalfCpltCallback
- At reception end of transfer HAL_I2S_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxCpltCallback
- In case of transfer Error, HAL_I2S_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2S_ErrorCallback
- Pause the DMA Transfer using HAL_I2S_DMAPause()
- Resume the DMA Transfer using HAL_I2S_DMAResume()
- Stop the DMA Transfer using HAL_I2S_DMAStop()

I2S HAL driver macros list

Below the list of most used macros in I2S HAL driver.

- __HAL_I2S_ENABLE: Enable the specified SPI peripheral (in I2S mode)
- __HAL_I2S_DISABLE: Disable the specified SPI peripheral (in I2S mode)
- __HAL_I2S_ENABLE_IT : Enable the specified I2S interrupts
- __HAL_I2S_DISABLE_IT : Disable the specified I2S interrupts
- __HAL_I2S_GET_FLAG: Check whether the specified I2S flag is set or not

Note: You can refer to the I2S HAL driver header file for more useful macros

I2S HAL driver macros list

Callback registration:

1. The compilation flag USE_HAL_I2S_REGISTER_CALLBACKS when set to 1UL allows the user to configure dynamically the driver callbacks. Use Functions HAL_I2S_RegisterCallback() to register an interrupt callback. Function HAL_I2S_RegisterCallback() allows to register following callbacks: (+) TxCpltCallback : I2S Tx Completed callback (+) RxCpltCallback : I2S Rx Completed callback (+) TxRxCpltCallback : I2S TxRx Completed callback (+) TxHalfCpltCallback : I2S Tx Half Completed callback (+) RxHalfCpltCallback : I2S Rx Half Completed callback (+) TxRxHalfCpltCallback : I2S TxRx Half Completed callback (+) ErrorCallback : I2S Error callback (+) MspInitCallback : I2S Msp Init callback (+) MspDeInitCallback : I2S Msp DeInit callback This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.
2. Use function HAL_I2S_UnRegisterCallback to reset a callback to the default weak function. HAL_I2S_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks: (+) TxCpltCallback : I2S Tx Completed callback (+) RxCpltCallback : I2S Rx Completed callback (+) TxRxCpltCallback : I2S TxRx Completed callback (+) TxHalfCpltCallback : I2S Tx Half Completed callback (+) RxHalfCpltCallback : I2S Rx Half Completed callback (+) TxRxHalfCpltCallback : I2S TxRx Half Completed callback (+) ErrorCallback : I2S Error callback (+) MspInitCallback : I2S Msp Init callback (+) MspDeInitCallback : I2S Msp DeInit callback By default, after the HAL_I2S_Init() and when the state is HAL_I2S_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_I2S_MasterTxCpltCallback(), HAL_I2S_MasterRxCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_I2S_Init()/ HAL_I2S_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the HAL_I2S_Init()/ HAL_I2S_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state. Callbacks can be registered/unregistered in HAL_I2S_STATE_READY state only. Exception done MspInit/MspDeInit functions that can be registered/unregistered in HAL_I2S_STATE_READY or HAL_I2S_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_I2S_RegisterCallback() before calling HAL_I2S_DeInit() or HAL_I2S_Init() function. When The compilation define USE_HAL_I2S_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

Callback registration: (#) The compilation flag USE_HAL_I2S_REGISTER_CALLBACKS when set to 1UL allows the user to configure dynamically the driver callbacks. Use Functions HAL_I2S_RegisterCallback() to register an interrupt callback. Function HAL_I2S_RegisterCallback() allows to register following callbacks:

- TxCpltCallback : I2S Tx Completed callback
- RxCpltCallback : I2S Rx Completed callback
- TxRxCpltCallback : I2S TxRx Completed callback
- TxHalfCpltCallback : I2S Tx Half Completed callback
- RxHalfCpltCallback : I2S Rx Half Completed callback
- TxRxHalfCpltCallback : I2S TxRx Half Completed callback
- ErrorCallback : I2S Error callback
- MspInitCallback : I2S Msp Init callback
- MspDeInitCallback : I2S Msp DeInit callback This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. (#) Use function HAL_I2S_UnRegisterCallback to reset a callback to the default weak function. HAL_I2S_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:
- TxCpltCallback : I2S Tx Completed callback
- RxCpltCallback : I2S Rx Completed callback
- TxRxCpltCallback : I2S TxRx Completed callback
- TxHalfCpltCallback : I2S Tx Half Completed callback
- RxHalfCpltCallback : I2S Rx Half Completed callback
- TxRxHalfCpltCallback : I2S TxRx Half Completed callback
- ErrorCallback : I2S Error callback
- MspInitCallback : I2S Msp Init callback

- MspDeInitCallback : I2S Msp DeInit callback By default, after the HAL_I2S_Init() and when the state is HAL_I2S_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_I2S_MasterTxCpltCallback(), HAL_I2S_MasterRxCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_I2S_Init()/ HAL_I2S_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the HAL_I2S_Init()/ HAL_I2S_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state. Callbacks can be registered/unregistered in HAL_I2S_STATE_READY state only. Exception done MspInit/MspDeInit functions that can be registered/unregistered in HAL_I2S_STATE_READY or HAL_I2S_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_I2S_RegisterCallback() before calling HAL_I2S_DeInit() or HAL_I2S_Init() function. When The compilation define USE_HAL_I2S_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

44.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the I2Sx peripheral in simplex mode:

- User must Implement HAL_I2S_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_I2S_Init() to configure the selected device with the selected configuration:
  - Mode
  - Standard
  - Data Format
  - MCLK Output
  - Audio frequency
  - Polarity
- Call the function HAL_I2S_DeInit() to restore the default configuration of the selected I2Sx peripheral.

This section contains the following APIs:

- ***HAL_I2S_Init()***
- ***HAL_I2S_DeInit()***
- ***HAL_I2S_MspInit()***
- ***HAL_I2S_MspDeInit()***
- ***HAL_I2S_RegisterCallback()***
- ***HAL_I2S_UnRegisterCallback()***

44.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the I2S data transfers.

1. There are two modes of transfer:
   - Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
   - No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated I2S IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
2. Blocking mode functions are :
   - HAL_I2S_Transmit()
   - HAL_I2S_Receive()
   - HAL_I2SEx_TransmitReceive()
3. No-Blocking mode functions with Interrupt are :
   - HAL_I2S_Transmit_IT()
   - HAL_I2S_Receive_IT()
   - HAL_I2SEx_TransmitReceive_IT()

4. No-Blocking mode functions with DMA are :
   – HAL_I2S_Transmit_DMA()
   – HAL_I2S_Receive_DMA()
   – HAL_I2SEx_TransmitReceive_DMA()
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
   – HAL_I2S_TxCpltCallback()
   – HAL_I2S_RxCpltCallback()
   – HAL_I2SEx_TxRxCpltCallback()
   – HAL_I2S_ErrorCallback()

This section contains the following APIs:

- ***HAL_I2S_Transmit()***
- ***HAL_I2S_Receive()***
- ***HAL_I2SEx_TransmitReceive()***
- ***HAL_I2S_Transmit_IT()***
- ***HAL_I2S_Receive_IT()***
- ***HAL_I2SEx_TransmitReceive_IT()***
- ***HAL_I2S_Transmit_DMA()***
- ***HAL_I2S_Receive_DMA()***
- ***HAL_I2SEx_TransmitReceive_DMA()***
- ***HAL_I2S_DMAPause()***
- ***HAL_I2S_DMAResume()***
- ***HAL_I2S_DMAStop()***
- ***HAL_I2S_IRQHandler()***
- ***HAL_I2S_TxHalfCpltCallback()***
- ***HAL_I2S_TxCpltCallback()***
- ***HAL_I2S_RxHalfCpltCallback()***
- ***HAL_I2S_RxCpltCallback()***
- ***HAL_I2SEx_TxRxHalfCpltCallback()***
- ***HAL_I2SEx_TxRxCpltCallback()***
- ***HAL_I2S_ErrorCallback()***

44.2.4 Peripheral State and Errors functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- ***HAL_I2S_GetState()***
- ***HAL_I2S_GetError()***

44.2.5 Detailed description of functions

HAL_I2S_Init

Function name

HAL_StatusTypeDef HAL_I2S_Init (I2S_HandleTypeDef * hi2s)

Function description

Initializes the I2S according to the specified parameters in the I2S_InitTypeDef and create the associated handle.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **HAL:** status

HAL_I2S_DeInit

Function name

HAL_StatusTypeDef HAL_I2S_DeInit (I2S_HandleTypeDef * hi2s)

Function description

DeInitializes the I2S peripheral.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **HAL:** status

HAL_I2S_MspInit

Function name

void HAL_I2S_MspInit (I2S_HandleTypeDef * hi2s)

Function description

I2S MSP Init.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_MspDeInit

Function name

void HAL_I2S_MspDeInit (I2S_HandleTypeDef * hi2s)

Function description

I2S MSP DeInit.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_RegisterCallback

Function name

HAL_StatusTypeDef HAL_I2S_RegisterCallback (I2S_HandleTypeDef * hi2s, HAL_I2S_CallbackIDTypeDef CallbackID, pI2S_CallbackTypeDef pCallback)

Function description

Register a User I2S Callback To be used instead of the weak predefined callback.

Parameters

- **hi2s:** Pointer to a I2S_HandleTypeDef structure that contains the configuration information for the specified I2S.
- **CallbackID:** ID of the callback to be registered
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_I2S_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_I2S_UnRegisterCallback (I2S_HandleTypeDef * hi2s, HAL_I2S_CallbackIDTypeDef CallbackID)

Function description

Unregister an I2S Callback I2S callback is redirected to the weak predefined callback.

Parameters

- **hi2s:** Pointer to a I2S_HandleTypeDef structure that contains the configuration information for the specified I2S.
- **CallbackID:** ID of the callback to be unregistered

Return values

- **HAL:** status

HAL_I2S_Transmit

Function name

HAL_StatusTypeDef HAL_I2S_Transmit (I2S_HandleTypeDef * hi2s, const uint16_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Transmit an amount of data in blocking mode.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to data buffer.
- **Size:** number of data sample to be sent:
- **Timeout:** Timeout duration

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

HAL_I2S_Receive

Function name

HAL_StatusTypeDef HAL_I2S_Receive (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to data buffer.
- **Size:** number of data sample to be sent:
- **Timeout:** Timeout duration

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).
- In I2S Master Receiver mode, just after enabling the peripheral the clock will be generate in continuous way and as the I2S is not disabled at the end of the I2S transaction.

HAL_I2SEx_TransmitReceive

Function name

HAL_StatusTypeDef HAL_I2SEx_TransmitReceive (I2S_HandleTypeDef * hi2s, const uint16_t * pTxData, uint16_t * pRxData, uint16_t Size, uint32_t Timeout)

Function description

Full-Duplex Transmit/Receive data in blocking mode.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pTxData:** a 16-bit pointer to the Transmit data buffer.
- **pRxData:** a 16-bit pointer to the Receive data buffer.
- **Size:** number of data sample to be sent:
- **Timeout:** Timeout duration

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

HAL_I2S_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I2S_Transmit_IT (I2S_HandleTypeDef * hi2s, const uint16_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to data buffer.
- **Size:** number of data sample to be sent:

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

HAL_I2S_Receive_IT

Function name

HAL_StatusTypeDef HAL_I2S_Receive_IT (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)

Function description

Receive an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to the Receive data buffer.
- **Size:** number of data sample to be sent:

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).
- It is recommended to use DMA for the I2S receiver to avoid de-synchronization between Master and Slave otherwise the I2S interrupt should be optimized.

HAL_I2SEx_TransmitReceive_IT

Function name

HAL_StatusTypeDef HAL_I2SEx_TransmitReceive_IT (I2S_HandleTypeDef * hi2s, const uint16_t * pTxData, uint16_t * pRxData, uint16_t Size)

Function description

Full-Duplex Transmit/Receive data in non-blocking mode using Interrupt.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pTxData:** a 16-bit pointer to the Transmit data buffer.
- **pRxData:** a 16-bit pointer to the Receive data buffer.
- **Size:** number of data sample to be sent:

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

HAL_I2S_IRQHandler

Function name

void HAL_I2S_IRQHandler (I2S_HandleTypeDef * hi2s)

Function description

This function handles I2S interrupt request.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_I2S_Transmit_DMA (I2S_HandleTypeDef * hi2s, const uint16_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with DMA.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to the Transmit data buffer.
- **Size:** number of data sample to be sent:

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

HAL_I2S_Receive_DMA

Function name

HAL_StatusTypeDef HAL_I2S_Receive_DMA (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)

Function description

Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to the Receive data buffer.
- **Size:** number of data sample to be sent:

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

HAL_I2SEx_TransmitReceive_DMA

Function name

HAL_StatusTypeDef HAL_I2SEx_TransmitReceive_DMA (I2S_HandleTypeDef * hi2s, const uint16_t * pTxData, uint16_t * pRxData, uint16_t Size)

Function description

Full-Duplex Transmit/Receive data in non-blocking mode using DMA.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pTxData:** a 16-bit pointer to the Transmit data buffer.
- **pRxData:** a 16-bit pointer to the Receive data buffer.
- **Size:** number of data sample to be sent:

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

HAL_I2S_DMAPause

Function name

HAL_StatusTypeDef HAL_I2S_DMAPause (I2S_HandleTypeDef * hi2s)

Function description

Pauses the audio DMA Stream/Channel playing from the Media.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **HAL:** status

HAL_I2S_DMAResume

Function name

HAL_StatusTypeDef HAL_I2S_DMAResume (I2S_HandleTypeDef * hi2s)

Function description

Resumes the audio DMA Stream/Channel playing from the Media.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **HAL:** status

HAL_I2S_DMAStop

Function name

HAL_StatusTypeDef HAL_I2S_DMAStop (I2S_HandleTypeDef * hi2s)

Function description

Stops the audio DMA Stream/Channel playing from the Media.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **HAL:** status

HAL_I2S_TxHalfCpltCallback

Function name

void HAL_I2S_TxHalfCpltCallback (I2S_HandleTypeDef * hi2s)

Function description

Tx Transfer Half completed callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_TxCpltCallback

Function name

void HAL_I2S_TxCpltCallback (I2S_HandleTypeDef * hi2s)

Function description

Tx Transfer completed callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_RxHalfCpltCallback

Function name

void HAL_I2S_RxHalfCpltCallback (I2S_HandleTypeDef * hi2s)

Function description

Rx Transfer half completed callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_RxCpltCallback

Function name

void HAL_I2S_RxCpltCallback (I2S_HandleTypeDef * hi2s)

Function description

Rx Transfer completed callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2SEx_TxRxHalfCpltCallback

Function name

void HAL_I2SEx_TxRxHalfCpltCallback (I2S_HandleTypeDef * hi2s)

Function description

Rx Transfer half completed callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2SEx_TxRxCpltCallback

Function name

void HAL_I2SEx_TxRxCpltCallback (I2S_HandleTypeDef * hi2s)

Function description

Rx Transfer completed callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_ErrorCallback

Function name

void HAL_I2S_ErrorCallback (I2S_HandleTypeDef * hi2s)

Function description

I2S error callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_GetState

Function name

HAL_I2S_StateTypeDef HAL_I2S_GetState (const I2S_HandleTypeDef * hi2s)

Function description

Return the I2S state.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **HAL:** state

HAL_I2S_GetError

Function name

uint32_t HAL_I2S_GetError (const I2S_HandleTypeDef * hi2s)

Function description

Return the I2S error code.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **I2S:** Error Code

44.3 I2S Firmware driver defines

The following section lists the various define and macros of the module.

44.3.1 I2S

I2S

I2S Audio Frequency

I2S_AUDIOFREQ_192K

I2S_AUDIOFREQ_96K

I2S_AUDIOFREQ_48K

I2S_AUDIOFREQ_44K

I2S_AUDIOFREQ_32K

I2S_AUDIOFREQ_22K

I2S_AUDIOFREQ_16K

I2S_AUDIOFREQ_11K

I2S_AUDIOFREQ_8K

I2S_AUDIOFREQ_DEFAULT

I2S FullDuplex Mode

I2S_CPOL_LOW

I2S_CPOL_HIGH

Data Padding 24Bit

I2S_DATA_24BIT_ALIGNMENT_RIGHT

I2S_DATA_24BIT_ALIGNMENT_LEFT

I2S Data Format

I2S_DATAFORMAT_16B

I2S_DATAFORMAT_16B_EXTENDED

I2S_DATAFORMAT_24B

I2S_DATAFORMAT_32B

I2S Error

HAL_I2S_ERROR_NONE

No error

HAL_I2S_ERROR_TIMEOUT

Timeout error

HAL_I2S_ERROR_OVR

OVR error

HAL_I2S_ERROR_UDR

UDR error

HAL_I2S_ERROR_DMA

DMA transfer error

HAL_I2S_ERROR_PRESCALER

Prescaler Calculation error

HAL_I2S_ERROR_FRE

FRE error

HAL_I2S_ERROR_NO_OGT

No On Going Transfer error

HAL_I2S_ERROR_NOT_SUPPORTED

Requested operation not supported

HAL_I2S_ERROR_INVALID_CALLBACK

Invalid Callback error

I2S Exported Macros

__HAL_I2S_RESET_HANDLE_STATE

Description:

- Reset I2S handle state.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_ENABLE

Description:

- Enable the specified SPI peripheral (in I2S mode).

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_DISABLE

Description:

- Disable the specified SPI peripheral (in I2S mode).

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_ENABLE_IT

Description:

- Enable the specified I2S interrupts.

Parameters:

- __HANDLE__: specifies the I2S Handle. This parameter can be I2S where x: 1, 2 or 3 to select the I2S peripheral.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - I2S_IT_RXP : Rx-Packet available interrupt
  - I2S_IT_TXP : Tx-Packet space available interrupt
  - I2S_IT_UDR : Underrun interrupt
  - I2S_IT_OVR : Overrun interrupt
  - I2S_IT_FRE : TI mode frame format error interrupt
  - I2S_IT_ERR : Error interrupt enable

Return value:

- None

__HAL_I2S_DISABLE_IT

Description:

- Disable the specified I2S interrupts.

Parameters:

- __HANDLE__: specifies the I2S Handle. This parameter can be I2S where x: 1, 2 or 3 to select the I2S peripheral.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - I2S_IT_RXP : Rx-Packet available interrupt
  - I2S_IT_TXP : Tx-Packet space available interrupt
  - I2S_IT_UDR : Underrun interrupt
  - I2S_IT_OVR : Overrun interrupt
  - I2S_IT_FRE : TI mode frame format error interrupt
  - I2S_IT_ERR : Error interrupt enable

Return value:

- None

__HAL_I2S_GET_IT_SOURCE

Description:

- Check if the specified I2S interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: specifies the I2S Handle. This parameter can be I2S where x: 1, 2 or 3 to select the I2S peripheral.
- __INTERRUPT__: specifies the I2S interrupt source to check. This parameter can be one of the following values:
  - I2S_IT_RXP : Rx-Packet available interrupt
  - I2S_IT_TXP : Tx-Packet space available interrupt
  - I2S_IT_DXP : Tx-Packet space available interrupt
  - I2S_IT_UDR : Underrun interrupt
  - I2S_IT_OVR : Overrun interrupt
  - I2S_IT_FRE : TI mode frame format error interrupt
  - I2S_IT_ERR : Error interrupt enable

Return value:

- The: new state of __IT__ (TRUE or FALSE).

__HAL_I2S_GET_FLAG

Description:

- Check whether the specified I2S flag is set or not.

Parameters:

- __HANDLE__: specifies the I2S Handle. This parameter can be I2S where x: 1, 2 or 3 to select the I2S peripheral.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - I2S_FLAG_RXP : Rx-Packet available flag
  - I2S_FLAG_TXP : Tx-Packet space available flag
  - I2S_FLAG_UDR : Underrun flag
  - I2S_FLAG_OVR : Overrun flag
  - I2S_FLAG_FRE : TI mode frame format error flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_I2S_CLEAR_OVRFLAG

Description:

- Clear the I2S OVR pending flag.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_CLEAR_UDRFLAG

Description:

- Clear the I2S UDR pending flag.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_CLEAR_TIFREFLAG

Description:

- Clear the I2S FRE pending flag.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

I2S Flags Definition

I2S_FLAG_RXP

I2S_FLAG_TXP

I2S_FLAG_DXP

I2S_FLAG_UDR

I2S_FLAG_OVR

I2S_FLAG_FRE

I2S_FLAG_MASK

I2S Interrupts Definition

I2S_IT_RXP

I2S_IT_TXP

I2S_IT_DXP

I2S_IT_UDR

I2S_IT_OVR

I2S_IT_FRE

I2S_IT_ERR

Keep IO State

I2S_MASTER_KEEP_IO_STATE_DISABLE

I2S_MASTER_KEEP_IO_STATE_ENABLE

I2S MCLK Output

I2S_MCLKOUTPUT_ENABLE

I2S_MCLKOUTPUT_DISABLE

I2S Mode

I2S_MODE_SLAVE_TX

I2S_MODE_SLAVE_RX

I2S_MODE_MASTER_TX

I2S_MODE_MASTER_RX

I2S_MODE_SLAVE_FULLDUPLEX

I2S_MODE_MASTER_FULLDUPLEX

I2S MSB LSB Transmission

I2S_FIRSTBIT_MSB

I2S_FIRSTBIT_LSB

I2S Standard

I2S_STANDARD_PHILIPS

I2S_STANDARD_MSB

I2S_STANDARD_LSB

I2S_STANDARD_PCM_SHORT

I2S_STANDARD_PCM_LONG

I2S Word Select Inversion

I2S_WS_INVERSION_DISABLE

I2S_WS_INVERSION_ENABLE

45 HAL IRDA Generic Driver

45.1 IRDA Firmware driver registers structures

45.1.1 IRDA_InitTypeDef

IRDA_InitTypeDef is defined in the stm32h7xx_hal_irda.h

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t Parity***
- ***uint32_t Mode***
- ***uint8_t Prescaler***
- ***uint16_t PowerMode***
- ***uint32_t ClockPrescaler***

Field Documentation

- ***uint32_t IRDA_InitTypeDef::BaudRate***

  This member configures the IRDA communication baud rate. The baud rate register is computed using the following formula: Baud Rate Register = ((usart_ker_ckpres) / ((hirda->Init.BaudRate))) where usart_ker_ckpres is the IRDA input clock divided by a prescaler
- ***uint32_t IRDA_InitTypeDef::WordLength***

  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***IRDAEx_Word_Length***
- ***uint32_t IRDA_InitTypeDef::Parity***

  Specifies the parity mode. This parameter can be a value of ***IRDA_Parity***

  Note:

  – When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32_t IRDA_InitTypeDef::Mode***

  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***IRDA_Transfer_Mode***
- ***uint8_t IRDA_InitTypeDef::Prescaler***

  Specifies the Prescaler value for dividing the UART/USART source clock to achieve low-power frequency.

  Note:

  – Prescaler value 0 is forbidden
- ***uint16_t IRDA_InitTypeDef::PowerMode***

  Specifies the IRDA power mode. This parameter can be a value of ***IRDA_Low_Power***
- ***uint32_t IRDA_InitTypeDef::ClockPrescaler***

  Specifies the prescaler value used to divide the IRDA clock source. This parameter can be a value of ***IRDA_ClockPrescaler***.

45.1.2 __IRDA_HandleTypeDef

__IRDA_HandleTypeDef is defined in the stm32h7xx_hal_irda.h

Data Fields

- ***USART_TypeDef * Instance***
- ***IRDA_InitTypeDef Init***
- ***const uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***

- ***__IO uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***uint16_t Mask***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_IRDA_StateTypeDef gState***
- ***__IO HAL_IRDA_StateTypeDef RxState***
- ***__IO uint32_t ErrorCode***
- ***void(* TxHalfCpltCallback***
- ***void(* TxCpltCallback***
- ***void(* RxHalfCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* AbortTransmitCpltCallback***
- ***void(* AbortReceiveCpltCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***USART_TypeDef* __IRDA_HandleTypeDef::Instance***
  USART registers base address
- ***IRDA_InitTypeDef __IRDA_HandleTypeDef::Init***
  IRDA communication parameters
- ***const uint8_t* __IRDA_HandleTypeDef::pTxBuffPtr***
  Pointer to IRDA Tx transfer Buffer
- ***uint16_t __IRDA_HandleTypeDef::TxXferSize***
  IRDA Tx Transfer size
- ***__IO uint16_t __IRDA_HandleTypeDef::TxXferCount***
  IRDA Tx Transfer Counter
- ***uint8_t* __IRDA_HandleTypeDef::pRxBuffPtr***
  Pointer to IRDA Rx transfer Buffer
- ***uint16_t __IRDA_HandleTypeDef::RxXferSize***
  IRDA Rx Transfer size
- ***__IO uint16_t __IRDA_HandleTypeDef::RxXferCount***
  IRDA Rx Transfer Counter
- ***uint16_t __IRDA_HandleTypeDef::Mask***
  USART RX RDR register mask
- ***DMA_HandleTypeDef* __IRDA_HandleTypeDef::hdmatx***
  IRDA Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __IRDA_HandleTypeDef::hdmarx***
  IRDA Rx DMA Handle parameters
- ***HAL_LockTypeDef __IRDA_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_IRDA_StateTypeDef __IRDA_HandleTypeDef::gState***
  IRDA state information related to global Handle management and also related to Tx operations. This parameter can be a value of **HAL_IRDA_StateTypeDef**

- ***__IO HAL_IRDA_StateTypeDef __IRDA_HandleTypeDef::RxState***
  IRDA state information related to Rx operations. This parameter can be a value of **HAL_IRDA_StateTypeDef**
- ***__IO uint32_t __IRDA_HandleTypeDef::ErrorCode***
  IRDA Error code
- ***void(* __IRDA_HandleTypeDef::TxHalfCpltCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Tx Half Complete Callback
- ***void(* __IRDA_HandleTypeDef::TxCpltCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Tx Complete Callback
- ***void(* __IRDA_HandleTypeDef::RxHalfCpltCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Rx Half Complete Callback
- ***void(* __IRDA_HandleTypeDef::RxCpltCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Rx Complete Callback
- ***void(* __IRDA_HandleTypeDef::ErrorCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Error Callback
- ***void(* __IRDA_HandleTypeDef::AbortCpltCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Abort Complete Callback
- ***void(* __IRDA_HandleTypeDef::AbortTransmitCpltCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Abort Transmit Complete Callback
- ***void(* __IRDA_HandleTypeDef::AbortReceiveCpltCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Abort Receive Complete Callback
- ***void(* __IRDA_HandleTypeDef::MspInitCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Msp Init callback
- ***void(* __IRDA_HandleTypeDef::MspDeInitCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Msp DeInit callback

45.2 IRDA Firmware driver API description

The following section lists the various functions of the IRDA library.

45.2.1 How to use this driver

The IRDA HAL driver can be used as follows:

1. Declare a IRDA_HandleTypeDef handle structure (eg. IRDA_HandleTypeDef hirda).

2. Initialize the IRDA low level resources by implementing the HAL_IRDA_MspInit() API in setting the associated USART or UART in IRDA mode:
   - Enable the USARTx/UARTx interface clock.
   - USARTx/UARTx pins configuration:
     - Enable the clock for the USARTx/UARTx GPIOs.
     - Configure these USARTx/UARTx pins (TX as alternate function pull-up, RX as alternate function Input).
   - NVIC configuration if you need to use interrupt process (HAL_IRDA_Transmit_IT() and HAL_IRDA_Receive_IT() APIs):
     - Configure the USARTx/UARTx interrupt priority.
     - Enable the NVIC USARTx/UARTx IRQ handle.
     - The specific IRDA interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_IRDA_ENABLE_IT() and __HAL_IRDA_DISABLE_IT() inside the transmit and receive process.
   - DMA Configuration if you need to use DMA process (HAL_IRDA_Transmit_DMA() and HAL_IRDA_Receive_DMA() APIs):
     - Declare a DMA handle structure for the Tx/Rx channel.
     - Enable the DMAx interface clock.
     - Configure the declared DMA handle structure with the required Tx/Rx parameters.
     - Configure the DMA Tx/Rx channel.
     - Associate the initialized DMA handle to the IRDA DMA Tx/Rx handle.
     - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
3. Program the Baud Rate, Word Length and Parity and Mode(Receiver/Transmitter), the normal or low power mode and the clock prescaler in the hirda handle Init structure.
4. Initialize the IRDA registers by calling the HAL_IRDA_Init() API:
   - This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_IRDA_MspInit() API.

Note: The specific IRDA interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_IRDA_ENABLE_IT() and __HAL_IRDA_DISABLE_IT() inside the transmit and receive process.

5. Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_IRDA_Transmit()
- Receive an amount of data in blocking mode using HAL_IRDA_Receive()

Interrupt mode IO operation

- Send an amount of data in non-blocking mode using HAL_IRDA_Transmit_IT()
- At transmission end of transfer HAL_IRDA_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_TxCpltCallback()
- Receive an amount of data in non-blocking mode using HAL_IRDA_Receive_IT()
- At reception end of transfer HAL_IRDA_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_RxCpltCallback()
- In case of transfer Error, HAL_IRDA_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_IRDA_ErrorCallback()

DMA mode IO operation

- Send an amount of data in non-blocking mode (DMA) using HAL_IRDA_Transmit_DMA()
- At transmission half of transfer HAL_IRDA_TxHalfCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_TxHalfCpltCallback()
- At transmission end of transfer HAL_IRDA_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_TxCpltCallback()

- Receive an amount of data in non-blocking mode (DMA) using HAL_IRDA_Receive_DMA()
- At reception half of transfer HAL_IRDA_RxHalfCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_RxHalfCpltCallback()
- At reception end of transfer HAL_IRDA_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_RxCpltCallback()
- In case of transfer Error, HAL_IRDA_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_IRDA_ErrorCallback()

IRDA HAL driver macros list

Below the list of most used macros in IRDA HAL driver.

- __HAL_IRDA_ENABLE: Enable the IRDA peripheral
- __HAL_IRDA_DISABLE: Disable the IRDA peripheral
- __HAL_IRDA_GET_FLAG : Check whether the specified IRDA flag is set or not
- __HAL_IRDA_CLEAR_FLAG : Clear the specified IRDA pending flag
- __HAL_IRDA_ENABLE_IT: Enable the specified IRDA interrupt
- __HAL_IRDA_DISABLE_IT: Disable the specified IRDA interrupt
- __HAL_IRDA_GET_IT_SOURCE: Check whether or not the specified IRDA interrupt is enabled

Note: You can refer to the IRDA HAL driver header file for more useful macros

45.2.2 Callback registration

The compilation define USE_HAL_IRDA_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_IRDA_RegisterCallback() to register a user callback. Function HAL_IRDA_RegisterCallback() allows to register following callbacks:

- TxHalfCpltCallback : Tx Half Complete Callback.
- TxCpltCallback : Tx Complete Callback.
- RxHalfCpltCallback : Rx Half Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- AbortTransmitCpltCallback : Abort Transmit Complete Callback.
- AbortReceiveCpltCallback : Abort Receive Complete Callback.
- MspInitCallback : IRDA MspInit.
- MspDeInitCallback : IRDA MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_IRDA_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. HAL_IRDA_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- TxHalfCpltCallback : Tx Half Complete Callback.
- TxCpltCallback : Tx Complete Callback.
- RxHalfCpltCallback : Rx Half Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- AbortTransmitCpltCallback : Abort Transmit Complete Callback.
- AbortReceiveCpltCallback : Abort Receive Complete Callback.
- MspInitCallback : IRDA MspInit.
- MspDeInitCallback : IRDA MspDeInit.

By default, after the HAL_IRDA_Init() and when the state is HAL_IRDA_STATE_RESET all callbacks are set to the corresponding weak (surcharged) functions: examples HAL_IRDA_TxCpltCallback(), HAL_IRDA_RxHalfCpltCallback(). Exception done for MspInit and MspDeInit functions that are respectively reset to the legacy weak (surcharged) functions in the HAL_IRDA_Init() and HAL_IRDA_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_IRDA_Init() and HAL_IRDA_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_IRDA_STATE_READY state only. Exception done MspInit/ MspDeInit that can be registered/unregistered in HAL_IRDA_STATE_READY or HAL_IRDA_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_IRDA_RegisterCallback() before calling HAL_IRDA_DeInit() or HAL_IRDA_Init() function.

When The compilation define USE_HAL_IRDA_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and weak (surcharged) callbacks are used.

45.2.3 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx in asynchronous IRDA mode.

- For the asynchronous mode only these parameters can be configured:
  - Baud Rate
  - Word Length
  - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit.
  - Power mode
  - Prescaler setting
  - Receiver/transmitter modes

The HAL_IRDA_Init() API follows the USART asynchronous configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- ***HAL_IRDA_Init()***
- ***HAL_IRDA_DeInit()***
- ***HAL_IRDA_MspInit()***
- ***HAL_IRDA_MspDeInit()***
- ***HAL_IRDA_RegisterCallback()***
- ***HAL_IRDA_UnRegisterCallback()***

45.2.4 IO operation functions

This subsection provides a set of functions allowing to manage the IRDA data transfers.

IrDA is a half duplex communication protocol. If the Transmitter is busy, any data on the IrDA receive line will be ignored by the IrDA decoder and if the Receiver is busy, data on the TX from the USART to IrDA will not be encoded by IrDA. While receiving data, transmission should be avoided as the data to be transmitted could be corrupted.

1. There are two modes of transfer:
   - Blocking mode: the communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
   - Non-Blocking mode: the communication is performed using Interrupts or DMA, these API's return the HAL status. The end of the data processing will be indicated through the dedicated IRDA IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_IRDA_TxCpltCallback(), HAL_IRDA_RxCpltCallback() user callbacks will be executed respectively at the end of the Transmit or Receive process The HAL_IRDA_ErrorCallback() user callback will be executed when a communication error is detected
2. Blocking mode APIs are :
   - HAL_IRDA_Transmit()
   - HAL_IRDA_Receive()

3. Non Blocking mode APIs with Interrupt are :
   - HAL_IRDA_Transmit_IT()
   - HAL_IRDA_Receive_IT()
   - HAL_IRDA_IRQHandler()
4. Non Blocking mode functions with DMA are :
   - HAL_IRDA_Transmit_DMA()
   - HAL_IRDA_Receive_DMA()
   - HAL_IRDA_DMAPause()
   - HAL_IRDA_DMAResume()
   - HAL_IRDA_DMAStop()
5. A set of Transfer Complete Callbacks are provided in Non Blocking mode:
   - HAL_IRDA_TxHalfCpltCallback()
   - HAL_IRDA_TxCpltCallback()
   - HAL_IRDA_RxHalfCpltCallback()
   - HAL_IRDA_RxCpltCallback()
   - HAL_IRDA_ErrorCallback()
6. Non-Blocking mode transfers could be aborted using Abort API's :
   - HAL_IRDA_Abort()
   - HAL_IRDA_AbortTransmit()
   - HAL_IRDA_AbortReceive()
   - HAL_IRDA_Abort_IT()
   - HAL_IRDA_AbortTransmit_IT()
   - HAL_IRDA_AbortReceive_IT()
7. For Abort services based on interrupts (HAL_IRDA_Abortxxx_IT), a set of Abort Complete Callbacks are provided:
   - HAL_IRDA_AbortCpltCallback()
   - HAL_IRDA_AbortTransmitCpltCallback()
   - HAL_IRDA_AbortReceiveCpltCallback()
8. In Non-Blocking mode transfers, possible errors are split into 2 categories. Errors are handled as follows :
   - Error is considered as Recoverable and non blocking : Transfer could go till end, but error severity is to be evaluated by user : this concerns Frame Error, Parity Error or Noise Error in Interrupt mode reception . Received character is then retrieved and stored in Rx buffer, Error code is set to allow user to identify error type, and HAL_IRDA_ErrorCallback() user callback is executed. Transfer is kept ongoing on IRDA side. If user wants to abort it, Abort services should be called by user.
   - Error is considered as Blocking : Transfer could not be completed properly and is aborted. This concerns Overrun Error In Interrupt mode reception and all errors in DMA mode. Error code is set to allow user to identify error type, and HAL_IRDA_ErrorCallback() user callback is executed.

This section contains the following APIs:

- ***HAL_IRDA_Transmit()***
- ***HAL_IRDA_Receive()***
- ***HAL_IRDA_Transmit_IT()***
- ***HAL_IRDA_Receive_IT()***
- ***HAL_IRDA_Transmit_DMA()***
- ***HAL_IRDA_Receive_DMA()***
- ***HAL_IRDA_DMAPause()***
- ***HAL_IRDA_DMAResume()***
- ***HAL_IRDA_DMAStop()***
- ***HAL_IRDA_Abort()***
- ***HAL_IRDA_AbortTransmit()***
- ***HAL_IRDA_AbortReceive()***
- ***HAL_IRDA_Abort_IT()***

- *HAL_IRDA_AbortTransmit_IT()*
- *HAL_IRDA_AbortReceive_IT()*
- *HAL_IRDA_IRQHandler()*
- *HAL_IRDA_TxCpltCallback()*
- *HAL_IRDA_TxHalfCpltCallback()*
- *HAL_IRDA_RxCpltCallback()*
- *HAL_IRDA_RxHalfCpltCallback()*
- *HAL_IRDA_ErrorCallback()*
- *HAL_IRDA_AbortCpltCallback()*
- *HAL_IRDA_AbortTransmitCpltCallback()*
- *HAL_IRDA_AbortReceiveCpltCallback()*

45.2.5 Peripheral State and Error functions

This subsection provides a set of functions allowing to return the State of IrDA communication process and also return Peripheral Errors occurred during communication process

- HAL_IRDA_GetState() API can be helpful to check in run-time the state of the IRDA peripheral handle.
- HAL_IRDA_GetError() checks in run-time errors that could occur during communication.

This section contains the following APIs:

- *HAL_IRDA_GetState()*
- *HAL_IRDA_GetError()*

45.2.6 Detailed description of functions

HAL_IRDA_Init

Function name

HAL_StatusTypeDef HAL_IRDA_Init (IRDA_HandleTypeDef * hirda)

Function description

Initialize the IRDA mode according to the specified parameters in the IRDA_InitTypeDef and initialize the associated handle.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **HAL:** status

HAL_IRDA_DeInit

Function name

HAL_StatusTypeDef HAL_IRDA_DeInit (IRDA_HandleTypeDef * hirda)

Function description

DeInitialize the IRDA peripheral.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **HAL:** status

HAL_IRDA_MspInit

Function name

void HAL_IRDA_MspInit (IRDA_HandleTypeDef * hirda)

Function description

Initialize the IRDA MSP.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_MspDeInit

Function name

void HAL_IRDA_MspDeInit (IRDA_HandleTypeDef * hirda)

Function description

DeInitialize the IRDA MSP.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_RegisterCallback

Function name

HAL_StatusTypeDef HAL_IRDA_RegisterCallback (IRDA_HandleTypeDef * hirda, HAL_IRDA_CallbackIDTypeDef CallbackID, pIRDA_CallbackTypeDef pCallback)

Function description

Register a User IRDA Callback To be used instead of the weak predefined callback.

Parameters

- **hirda:** irda handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_IRDA_TX_HALFCOMPLETE_CB_ID Tx Half Complete Callback ID
  - HAL_IRDA_TX_COMPLETE_CB_ID Tx Complete Callback ID
  - HAL_IRDA_RX_HALFCOMPLETE_CB_ID Rx Half Complete Callback ID
  - HAL_IRDA_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_IRDA_ERROR_CB_ID Error Callback ID
  - HAL_IRDA_ABORT_COMPLETE_CB_ID Abort Complete Callback ID
  - HAL_IRDA_ABORT_TRANSMIT_COMPLETE_CB_ID Abort Transmit Complete Callback ID
  - HAL_IRDA_ABORT_RECEIVE_COMPLETE_CB_ID Abort Receive Complete Callback ID
  - HAL_IRDA_MSPINIT_CB_ID MspInit Callback ID
  - HAL_IRDA_MSPDEINIT_CB_ID MspDeInit Callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

Notes

- The HAL_IRDA_RegisterCallback() may be called before HAL_IRDA_Init() in HAL_IRDA_STATE_RESET to register callbacks for HAL_IRDA_MSPINIT_CB_ID and HAL_IRDA_MSPDEINIT_CB_ID

HAL_IRDA_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_IRDA_UnRegisterCallback (IRDA_HandleTypeDef * hirda, HAL_IRDA_CallbackIDTypeDef CallbackID)

Function description

Unregister an IRDA callback IRDA callback is redirected to the weak predefined callback.

Parameters

- **hirda:** irda handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_IRDA_TX_HALFCOMPLETE_CB_ID Tx Half Complete Callback ID
  - HAL_IRDA_TX_COMPLETE_CB_ID Tx Complete Callback ID
  - HAL_IRDA_RX_HALFCOMPLETE_CB_ID Rx Half Complete Callback ID
  - HAL_IRDA_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_IRDA_ERROR_CB_ID Error Callback ID
  - HAL_IRDA_ABORT_COMPLETE_CB_ID Abort Complete Callback ID
  - HAL_IRDA_ABORT_TRANSMIT_COMPLETE_CB_ID Abort Transmit Complete Callback ID
  - HAL_IRDA_ABORT_RECEIVE_COMPLETE_CB_ID Abort Receive Complete Callback ID
  - HAL_IRDA_MSPINIT_CB_ID MspInit Callback ID
  - HAL_IRDA_MSPDEINIT_CB_ID MspDeInit Callback ID

Return values

- **HAL:** status

Notes

- The HAL_IRDA_UnRegisterCallback() may be called before HAL_IRDA_Init() in HAL_IRDA_STATE_RESET to un-register callbacks for HAL_IRDA_MSPINIT_CB_ID and HAL_IRDA_MSPDEINIT_CB_ID

HAL_IRDA_Transmit

Function name

HAL_StatusTypeDef HAL_IRDA_Transmit (IRDA_HandleTypeDef * hirda, const uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Send an amount of data in blocking mode.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be sent.
- **Timeout:** Specify timeout value.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must reflect the number of u16 available through pData.

HAL_IRDA_Receive

Function name

HAL_StatusTypeDef HAL_IRDA_Receive (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.
- **Timeout:** Specify timeout value.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must reflect the number of u16 available through pData.

HAL_IRDA_Transmit_IT

Function name

HAL_StatusTypeDef HAL_IRDA_Transmit_IT (IRDA_HandleTypeDef * hirda, const uint8_t * pData, uint16_t Size)

Function description

Send an amount of data in interrupt mode.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be sent.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must reflect the number of u16 available through pData.

HAL_IRDA_Receive_IT

Function name

HAL_StatusTypeDef HAL_IRDA_Receive_IT (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in interrupt mode.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must reflect the number of u16 available through pData.

HAL_IRDA_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_IRDA_Transmit_DMA (IRDA_HandleTypeDef * hirda, const uint8_t * pData, uint16_t Size)

Function description

Send an amount of data in DMA mode.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be sent.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must reflect the number of u16 available through pData.

HAL_IRDA_Receive_DMA

Function name

HAL_StatusTypeDef HAL_IRDA_Receive_DMA (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in DMA mode.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must reflect the number of u16 available through pData.
- When the IRDA parity is enabled (PCE = 1), the received data contains the parity bit (MSB position).

HAL_IRDA_DMAPause

Function name

HAL_StatusTypeDef HAL_IRDA_DMAPause (IRDA_HandleTypeDef * hirda)

Function description

Pause the DMA Transfer.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **HAL:** status

HAL_IRDA_DMAResume

Function name

HAL_StatusTypeDef HAL_IRDA_DMAResume (IRDA_HandleTypeDef * hirda)

Function description

Resume the DMA Transfer.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

HAL_IRDA_DMAStop

Function name

HAL_StatusTypeDef HAL_IRDA_DMAStop (IRDA_HandleTypeDef * hirda)

Function description

Stop the DMA Transfer.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

HAL_IRDA_Abort

Function name

HAL_StatusTypeDef HAL_IRDA_Abort (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing transfers (blocking mode).

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_IRDA_AbortTransmit

Function name

HAL_StatusTypeDef HAL_IRDA_AbortTransmit (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing Transmit transfer (blocking mode).

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_IRDA_AbortReceive

Function name

HAL_StatusTypeDef HAL_IRDA_AbortReceive (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing Receive transfer (blocking mode).

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_IRDA_Abort_IT

Function name

HAL_StatusTypeDef HAL_IRDA_Abort_IT (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing transfers (Interrupt mode).

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_IRDA_AbortTransmit_IT

Function name

HAL_StatusTypeDef HAL_IRDA_AbortTransmit_IT (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing Transmit transfer (Interrupt mode).

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_IRDA_AbortReceive_IT

Function name

HAL_StatusTypeDef HAL_IRDA_AbortReceive_IT (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing Receive transfer (Interrupt mode).

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_IRDA_IRQHandler

Function name

void HAL_IRDA_IRQHandler (IRDA_HandleTypeDef * hirda)

Function description

Handle IRDA interrupt request.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_TxCpltCallback

Function name

void HAL_IRDA_TxCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

Tx Transfer completed callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_RxCpltCallback

Function name

void HAL_IRDA_RxCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

Rx Transfer completed callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_TxHalfCpltCallback

Function name

void HAL_IRDA_TxHalfCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

Tx Half Transfer completed callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values

- **None:**

HAL_IRDA_RxHalfCpltCallback

Function name

void HAL_IRDA_RxHalfCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

Rx Half Transfer complete callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_ErrorCallback

Function name

void HAL_IRDA_ErrorCallback (IRDA_HandleTypeDef * hirda)

Function description

IRDA error callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_AbortCpltCallback

Function name

void HAL_IRDA_AbortCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

IRDA Abort Complete callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_AbortTransmitCpltCallback

Function name

void HAL_IRDA_AbortTransmitCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

IRDA Abort Complete callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_AbortReceiveCpltCallback

Function name

void HAL_IRDA_AbortReceiveCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

IRDA Abort Receive Complete callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_GetState

Function name

HAL_IRDA_StateTypeDef HAL_IRDA_GetState (const IRDA_HandleTypeDef * hirda)

Function description

Return the IRDA handle state.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **HAL:** state

HAL_IRDA_GetError

Function name

uint32_t HAL_IRDA_GetError (const IRDA_HandleTypeDef * hirda)

Function description

Return the IRDA handle error code.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **IRDA:** Error Code

45.3 IRDA Firmware driver defines

The following section lists the various define and macros of the module.

45.3.1 IRDA

IRDA

IRDA Clock Prescaler

IRDA_PRESCALER_DIV1

fclk_pres = fclk

IRDA_PRESCALER_DIV2

fclk_pres = fclk/2

IRDA_PRESCALER_DIV4

fclk_pres = fclk/4

IRDA_PRESCALER_DIV6

fclk_pres = fclk/6

IRDA_PRESCALER_DIV8

fclk_pres = fclk/8

IRDA_PRESCALER_DIV10

fclk_pres = fclk/10

IRDA_PRESCALER_DIV12

fclk_pres = fclk/12

IRDA_PRESCALER_DIV16

fclk_pres = fclk/16

IRDA_PRESCALER_DIV32

fclk_pres = fclk/32

IRDA_PRESCALER_DIV64

fclk_pres = fclk/64

IRDA_PRESCALER_DIV128

fclk_pres = fclk/128

IRDA_PRESCALER_DIV256

fclk_pres = fclk/256

IRDA DMA Rx

IRDA_DMA_RX_DISABLE

IRDA DMA RX disabled

IRDA_DMA_RX_ENABLE

IRDA DMA RX enabled

IRDA DMA Tx

IRDA_DMA_TX_DISABLE

IRDA DMA TX disabled

IRDA_DMA_TX_ENABLE

IRDA DMA TX enabled

IRDA Error Code Definition

HAL_IRDA_ERROR_NONE

No error

HAL_IRDA_ERROR_PE

Parity error

HAL_IRDA_ERROR_NE

Noise error

HAL_IRDA_ERROR_FE

frame error

HAL_IRDA_ERROR_ORE

Overrun error

HAL_IRDA_ERROR_DMA

DMA transfer error

HAL_IRDA_ERROR_BUSY

Busy Error

HAL_IRDA_ERROR_INVALID_CALLBACK

Invalid Callback error

IRDA Exported Macros

__HAL_IRDA_RESET_HANDLE_STATE

Description:

- Reset IRDA handle state.

Parameters:

- __HANDLE__: IRDA handle.

Return value:

- None

__HAL_IRDA_FLUSH_DRREGISTER

Description:

- Flush the IRDA DR register.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_FLAG

Description:

- Clear the specified IRDA pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - IRDA_CLEAR_PEF
  - IRDA_CLEAR_FEF
  - IRDA_CLEAR_NEF
  - IRDA_CLEAR_OREF
  - IRDA_CLEAR_TCF
  - IRDA_CLEAR_IDLEF

Return value:

- None

__HAL_IRDA_CLEAR_PEFLAG

Description:

- Clear the IRDA PE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_FEFLAG

Description:

- Clear the IRDA FE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_NEFLAG

Description:

- Clear the IRDA NE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_OREFLAG

Description:

- Clear the IRDA ORE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_IDLEFLAG

Description:

- Clear the IRDA IDLE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_GET_FLAG

Description:

- Check whether the specified IRDA flag is set or not.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - IRDA_FLAG_REACK Receive enable acknowledge flag
  - IRDA_FLAG_TEACK Transmit enable acknowledge flag
  - IRDA_FLAG_BUSY Busy flag
  - IRDA_FLAG_ABRF Auto Baud rate detection flag
  - IRDA_FLAG_ABRE Auto Baud rate detection error flag
  - IRDA_FLAG_TXE Transmit data register empty flag
  - IRDA_FLAG_TC Transmission Complete flag
  - IRDA_FLAG_RXNE Receive data register not empty flag
  - IRDA_FLAG_ORE OverRun Error flag
  - IRDA_FLAG_NE Noise Error flag
  - IRDA_FLAG_FE Framing Error flag
  - IRDA_FLAG_PE Parity Error flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_IRDA_ENABLE_IT

Description:

- Enable the specified IRDA interrupt.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __INTERRUPT__: specifies the IRDA interrupt source to enable. This parameter can be one of the following values:
  - IRDA_IT_TXE Transmit Data Register empty interrupt
  - IRDA_IT_TC Transmission complete interrupt
  - IRDA_IT_RXNE Receive Data register not empty interrupt
  - IRDA_IT_IDLE Idle line detection interrupt
  - IRDA_IT_PE Parity Error interrupt
  - IRDA_IT_ERR Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

__HAL_IRDA_DISABLE_IT

Description:

- Disable the specified IRDA interrupt.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __INTERRUPT__: specifies the IRDA interrupt source to disable. This parameter can be one of the following values:
  - IRDA_IT_TXE Transmit Data Register empty interrupt
  - IRDA_IT_TC Transmission complete interrupt
  - IRDA_IT_RXNE Receive Data register not empty interrupt
  - IRDA_IT_IDLE Idle line detection interrupt
  - IRDA_IT_PE Parity Error interrupt
  - IRDA_IT_ERR Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

__HAL_IRDA_GET_IT

Description:

- Check whether the specified IRDA interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __INTERRUPT__: specifies the IRDA interrupt source to check. This parameter can be one of the following values:
  - IRDA_IT_TXE Transmit Data Register empty interrupt
  - IRDA_IT_TC Transmission complete interrupt
  - IRDA_IT_RXNE Receive Data register not empty interrupt
  - IRDA_IT_IDLE Idle line detection interrupt
  - IRDA_IT_ORE OverRun Error interrupt
  - IRDA_IT_NE Noise Error interrupt
  - IRDA_IT_FE Framing Error interrupt
  - IRDA_IT_PE Parity Error interrupt

Return value:

- The: new state of __IT__ (SET or RESET).

__HAL_IRDA_GET_IT_SOURCE

Description:

- Check whether the specified IRDA interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __INTERRUPT__: specifies the IRDA interrupt source to check. This parameter can be one of the following values:
  - IRDA_IT_TXE Transmit Data Register empty interrupt
  - IRDA_IT_TC Transmission complete interrupt
  - IRDA_IT_RXNE Receive Data register not empty interrupt
  - IRDA_IT_IDLE Idle line detection interrupt
  - IRDA_IT_ERR Framing, overrun or noise error interrupt
  - IRDA_IT_PE Parity Error interrupt

Return value:

- The: new state of __IT__ (SET or RESET).

__HAL_IRDA_CLEAR_IT

Description:

- Clear the specified IRDA ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt This parameter can be one of the following values:
  - IRDA_CLEAR_PEF Parity Error Clear Flag
  - IRDA_CLEAR_FEF Framing Error Clear Flag
  - IRDA_CLEAR_NEF Noise detected Clear Flag
  - IRDA_CLEAR_OREF OverRun Error Clear Flag
  - IRDA_CLEAR_TCF Transmission Complete Clear Flag

Return value:

- None

__HAL_IRDA_SEND_REQ

Description:

- Set a specific IRDA request flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __REQ__: specifies the request flag to set This parameter can be one of the following values:
  - IRDA_AUTOBAUD_REQUEST Auto-Baud Rate Request
  - IRDA_RXDATA_FLUSH_REQUEST Receive Data flush Request
  - IRDA_TXDATA_FLUSH_REQUEST Transmit data flush Request

Return value:

- None

__HAL_IRDA_ONE_BIT_SAMPLE_ENABLE

Description:

- Enable the IRDA one bit sample method.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_ONE_BIT_SAMPLE_DISABLE

Description:

- Disable the IRDA one bit sample method.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_ENABLE

Description:

- Enable UART/USART associated to IRDA Handle.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_DISABLE

Description:

- Disable UART/USART associated to IRDA Handle.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

IRDA Flags

IRDA_FLAG_REACK

IRDA receive enable acknowledge flag

IRDA_FLAG_TEACK

IRDA transmit enable acknowledge flag

IRDA_FLAG_BUSY

IRDA busy flag

IRDA_FLAG_ABRF

IRDA auto Baud rate flag

IRDA_FLAG_ABRE

IRDA auto Baud rate error

IRDA_FLAG_TXE

IRDA transmit data register empty

IRDA_FLAG_TC

IRDA transmission complete

IRDA_FLAG_RXNE

IRDA read data register not empty

IRDA_FLAG_ORE

IRDA overrun error

IRDA_FLAG_NE

IRDA noise error

IRDA_FLAG_FE

IRDA frame error

IRDA_FLAG_PE

IRDA parity error

IRDA interruptions flags mask

IRDA_IT_MASK

IRDA Interruptions flags mask

IRDA_CR_MASK

IRDA control register mask

IRDA_CR_POS

IRDA control register position

IRDA_ISR_MASK

IRDA ISR register mask

IRDA_ISR_POS

IRDA ISR register position

IRDA Interrupts Definition

IRDA_IT_PE

IRDA Parity error interruption

IRDA_IT_TXE

IRDA Transmit data register empty interruption

IRDA_IT_TC

IRDA Transmission complete interruption

IRDA_IT_RXNE

IRDA Read data register not empty interruption

IRDA_IT_IDLE

IRDA Idle interruption

IRDA_IT_ERR

IRDA Error interruption

IRDA_IT_ORE

IRDA Overrun error interruption

IRDA_IT_NE

IRDA Noise error interruption

IRDA_IT_FE

IRDA Frame error interruption

IRDA Interruption Clear Flags

IRDA_CLEAR_PEF

Parity Error Clear Flag

IRDA_CLEAR_FEF

Framing Error Clear Flag

IRDA_CLEAR_NEF

Noise Error detected Clear Flag

IRDA_CLEAR_OREF

OverRun Error Clear Flag

IRDA_CLEAR_IDLEF

IDLE line detected Clear Flag

IRDA_CLEAR_TCF

Transmission Complete Clear Flag

IRDA Low Power

IRDA_POWERMODE_NORMAL

IRDA normal power mode

IRDA_POWERMODE_LOWPOWER

IRDA low power mode

IRDA Mode

IRDA_MODE_DISABLE

Associated UART disabled in IRDA mode

IRDA_MODE_ENABLE

Associated UART enabled in IRDA mode

IRDA One Bit Sampling

IRDA_ONE_BIT_SAMPLE_DISABLE

One-bit sampling disabled

IRDA_ONE_BIT_SAMPLE_ENABLE

One-bit sampling enabled

IRDA Parity

IRDA_PARITY_NONE

No parity

IRDA_PARITY_EVEN

Even parity

IRDA_PARITY_ODD

Odd parity

IRDA Request Parameters

IRDA_AUTOBAUD_REQUEST

Auto-Baud Rate Request

IRDA_RXDATA_FLUSH_REQUEST

Receive Data flush Request

IRDA_TXDATA_FLUSH_REQUEST

Transmit data flush Request

IRDA State

IRDA_STATE_DISABLE

IRDA disabled

IRDA_STATE_ENABLE

IRDA enabled

IRDA State Code Definition

HAL_IRDA_STATE_RESET

Peripheral is not initialized Value is allowed for gState and RxState

HAL_IRDA_STATE_READY

Peripheral Initialized and ready for use Value is allowed for gState and RxState

HAL_IRDA_STATE_BUSY

An internal process is ongoing Value is allowed for gState only

HAL_IRDA_STATE_BUSY_TX

Data Transmission process is ongoing Value is allowed for gState only

HAL_IRDA_STATE_BUSY_RX

Data Reception process is ongoing Value is allowed for RxState only

HAL_IRDA_STATE_BUSY_TX_RX

Data Transmission and Reception process is ongoing Not to be used for neither gState nor RxState. Value is result of combination (Or) between gState and RxState values

HAL_IRDA_STATE_TIMEOUT

Timeout state Value is allowed for gState only

HAL_IRDA_STATE_ERROR

Error Value is allowed for gState only

IRDA Transfer Mode

IRDA_MODE_RX

RX mode

IRDA_MODE_TX

TX mode

IRDA_MODE_TX_RX

RX and TX mode

46 HAL IRDA Extension Driver

46.1 IRDAEx Firmware driver defines

The following section lists the various define and macros of the module.

46.1.1 IRDAEx

IRDAEx

IRDAEx Word Length

IRDA_WORDLENGTH_7B

7-bit long frame

IRDA_WORDLENGTH_8B

8-bit long frame

IRDA_WORDLENGTH_9B

9-bit long frame

47 HAL IWDG Generic Driver

47.1 IWDG Firmware driver registers structures

47.1.1 IWDG_InitTypeDef

IWDG_InitTypeDef is defined in the stm32h7xx_hal_iwdg.h

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t Reload***
- ***uint32_t Window***

Field Documentation

- ***uint32_t IWDG_InitTypeDef::Prescaler***
  Select the prescaler of the IWDG. This parameter can be a value of ***IWDG_Prescaler***
- ***uint32_t IWDG_InitTypeDef::Reload***
  Specifies the IWDG down-counter reload value. This parameter must be a number between Min_Data = 0 and Max_Data = 0x0FFF
- ***uint32_t IWDG_InitTypeDef::Window***
  Specifies the window value to be compared to the down-counter. This parameter must be a number between Min_Data = 0 and Max_Data = 0x0FFF

47.1.2 IWDG_HandleTypeDef

IWDG_HandleTypeDef is defined in the stm32h7xx_hal_iwdg.h

Data Fields

- ***IWDG_TypeDef * Instance***
- ***IWDG_InitTypeDef Init***

Field Documentation

- ***IWDG_TypeDef* IWDG_HandleTypeDef::Instance***
  Register base address
- ***IWDG_InitTypeDef IWDG_HandleTypeDef::Init***
  IWDG required parameters

47.2 IWDG Firmware driver API description

The following section lists the various functions of the IWDG library.

47.2.1 IWDG Generic features

- The IWDG can be started by either software or hardware (configurable through option byte).
- The IWDG is clocked by the Low-Speed Internal clock (LSI) and thus stays active even if the main clock fails.
- Once the IWDG is started, the LSI is forced ON and both cannot be disabled. The counter starts counting down from the reset value (0xFFF). When it reaches the end of count value (0x000) a reset signal is generated (IWDG reset).
- Whenever the key value 0x0000 AAAA is written in the IWDG_KR register, the IWDG_RLR value is reloaded into the counter and the watchdog reset is prevented.
- The IWDG is implemented in the VDD voltage domain that is still functional in STOP and STANDBY mode (IWDG reset can wake up the CPU from STANDBY). IWDGRST flag in RCC_CSR register can be used to inform when an IWDG reset occurs.

- Debug mode: When the microcontroller enters debug mode (core halted), the IWDG counter either continues to work normally or stops, depending on DBG_IWDG_STOP configuration bit in DBG module, accessible through __HAL_DBGMCU_FREEZE_IWDG1() or __HAL_DBGMCU_FREEZE2_IWDG2() and __HAL_DBGMCU_UnFreeze_IWDG1 or __HAL_DBGMCU_UnFreeze2_IWDG2() macros.

Min-max timeout value @32KHz (LSI): ~125us / ~32.7s The IWDG timeout may vary due to LSI clock frequency dispersion. STM32H7xx devices provide the capability to measure the LSI clock frequency (LSI clock is internally connected to TIM16 CH1 input capture). The measured value can be used to have an IWDG timeout with an acceptable accuracy.

Default timeout value (necessary for IWDG_SR status register update): Constant LSI_VALUE is defined based on the nominal LSI clock frequency. This frequency being subject to variations as mentioned above, the default timeout value (defined through constant HAL_IWDG_DEFAULT_TIMEOUT below) may become too short or too long. In such cases, this default timeout value can be tuned by redefining the constant LSI_VALUE at user-application level (based, for instance, on the measured LSI clock frequency as explained above).

47.2.2 How to use this driver

1. Use IWDG using HAL_IWDG_Init() function to :
   - Enable instance by writing Start keyword in IWDG_KEY register. LSI clock is forced ON and IWDG counter starts counting down.
   - Enable write access to configuration registers: IWDG_PR, IWDG_RLR and IWDG_WINR.
   - Configure the IWDG prescaler and counter reload value. This reload value will be loaded in the IWDG counter each time the watchdog is reloaded, then the IWDG will start counting down from this value.
   - Depending on window parameter:
     - If Window Init parameter is same as Window register value, nothing more is done but reload counter value in order to exit function with exact time base.
     - Else modify Window register. This will automatically reload watchdog counter.
   - Wait for status flags to be reset.
2. Then the application program must refresh the IWDG counter at regular intervals during normal operation to prevent an MCU reset, using HAL_IWDG_Refresh() function.

IWDG HAL driver macros list

Below the list of most used macros in IWDG HAL driver:

- __HAL_IWDG_START: Enable the IWDG peripheral
- __HAL_IWDG_RELOAD_COUNTER: Reloads IWDG counter with value defined in the reload register

47.2.3 Initialization and Start functions

This section provides functions allowing to:

- Initialize the IWDG according to the specified parameters in the IWDG_InitTypeDef of associated handle.
- Manage Window option.
- Once initialization is performed in HAL_IWDG_Init function, Watchdog is reloaded in order to exit function with correct time base.

This section contains the following APIs:

- ***HAL_IWDG_Init()***

47.2.4 IO operation functions

This section provides functions allowing to:

- Refresh the IWDG.

This section contains the following APIs:

- ***HAL_IWDG_Refresh()***

47.2.5 Detailed description of functions

HAL_IWDG_Init

Function name

HAL_StatusTypeDef HAL_IWDG_Init (IWDG_HandleTypeDef * hiwdg)

Function description

Initialize the IWDG according to the specified parameters in the IWDG_InitTypeDef and start watchdog.

Parameters

- **hiwdg:** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.

Return values

- **HAL:** status

HAL_IWDG_Refresh

Function name

HAL_StatusTypeDef HAL_IWDG_Refresh (IWDG_HandleTypeDef * hiwdg)

Function description

Refresh the IWDG.

Parameters

- **hiwdg:** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.

Return values

- **HAL:** status

47.3 IWDG Firmware driver defines

The following section lists the various define and macros of the module.

47.3.1 IWDG

IWDG

IWDG Exported Macros

__HAL_IWDG_START

Description:

- Enable the IWDG peripheral.

Parameters:

- __HANDLE__: IWDG handle

Return value:

- None

__HAL_IWDG_RELOAD_COUNTER

Description:

- Reload IWDG counter with value defined in the reload register (write access to IWDG_PR, IWDG_RLR and IWDG_WINR registers disabled).

Parameters:

- __HANDLE__: IWDG handle

Return value:

- None

IWDG Prescaler

IWDG_PRESCALER_4

IWDG prescaler set to 4

IWDG_PRESCALER_8

IWDG prescaler set to 8

IWDG_PRESCALER_16

IWDG prescaler set to 16

IWDG_PRESCALER_32

IWDG prescaler set to 32

IWDG_PRESCALER_64

IWDG prescaler set to 64

IWDG_PRESCALER_128

IWDG prescaler set to 128

IWDG_PRESCALER_256

IWDG prescaler set to 256

IWDG Window option

IWDG_WINDOW_DISABLE

48 HAL JPEG Generic Driver

48.1 JPEG Firmware driver registers structures

48.1.1 JPEG_ConfTypeDef

JPEG_ConfTypeDef is defined in the stm32h7xx_hal_jpeg.h

Data Fields

- ***uint32_t ColorSpace***
- ***uint32_t ChromaSubsampling***
- ***uint32_t ImageHeight***
- ***uint32_t ImageWidth***
- ***uint32_t ImageQuality***

Field Documentation

- ***uint32_t JPEG_ConfTypeDef::ColorSpace***
  Image Color space : gray-scale, YCBCR, RGB or CMYK This parameter can be a value of ***JPEG_ColorSpace***
- ***uint32_t JPEG_ConfTypeDef::ChromaSubsampling***
  Chroma Subsampling in case of YCBCR or CMYK color space, 0-> 4:4:4 , 1-> 4:2:2, 2 -> 4:1:1, 3 -> 4:2:0 This parameter can be a value of ***JPEG_ChromaSubsampling***
- ***uint32_t JPEG_ConfTypeDef::ImageHeight***
  Image height : number of lines
- ***uint32_t JPEG_ConfTypeDef::ImageWidth***
  Image width : number of pixels per line
- ***uint32_t JPEG_ConfTypeDef::ImageQuality***
  Quality of the JPEG encoding : from 1 to 100

48.1.2 __JPEG_HandleTypeDef

__JPEG_HandleTypeDef is defined in the stm32h7xx_hal_jpeg.h

Data Fields

- ***JPEG_TypeDef * Instance***
- ***JPEG_ConfTypeDef Conf***
- ***uint8_t * pJpegInBuffPtr***
- ***uint8_t * pJpegOutBuffPtr***
- ***__IO uint32_t JpegInCount***
- ***__IO uint32_t JpegOutCount***
- ***uint32_t InDataLength***
- ***uint32_t OutDataLength***
- ***MDMA_HandleTypeDef * hdmain***
- ***MDMA_HandleTypeDef * hdmaout***
- ***uint8_t CustomQuanTable***
- ***uint8_t * QuantTable0***
- ***uint8_t * QuantTable1***
- ***uint8_t * QuantTable2***
- ***uint8_t * QuantTable3***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_JPEG_STATETypeDef State***
- ***__IO uint32_t ErrorCode***

- ***__IO uint32_t Context***
- ***void(* InfoReadyCallback***
- ***void(* EncodeCpltCallback***
- ***void(* DecodeCpltCallback***
- ***void(* ErrorCallback***
- ***void(* GetDataCallback***
- ***void(* DataReadyCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***JPEG_TypeDef* __JPEG_HandleTypeDef::Instance***
  JPEG peripheral register base address
- ***JPEG_ConfTypeDef __JPEG_HandleTypeDef::Conf***
  Current JPEG encoding/decoding parameters
- ***uint8_t* __JPEG_HandleTypeDef::pJpegInBuffPtr***
  Pointer to JPEG processing (encoding, decoding,...) input buffer
- ***uint8_t* __JPEG_HandleTypeDef::pJpegOutBuffPtr***
  Pointer to JPEG processing (encoding, decoding,...) output buffer
- ***__IO uint32_t __JPEG_HandleTypeDef::JpegInCount***
  Internal Counter of input data
- ***__IO uint32_t __JPEG_HandleTypeDef::JpegOutCount***
  Internal Counter of output data
- ***uint32_t __JPEG_HandleTypeDef::InDataLength***
  Input Buffer Length in Bytes
- ***uint32_t __JPEG_HandleTypeDef::OutDataLength***
  Output Buffer Length in Bytes
- ***MDMA_HandleTypeDef* __JPEG_HandleTypeDef::hdmain***
  JPEG In MDMA handle parameters
- ***MDMA_HandleTypeDef* __JPEG_HandleTypeDef::hdmaout***
  JPEG Out MDMA handle parameters
- ***uint8_t __JPEG_HandleTypeDef::CustomQuanTable***
  If set to 1 specify that user customized quantization tables are used
- ***uint8_t* __JPEG_HandleTypeDef::QuantTable0***
  Basic Quantization Table for component 0
- ***uint8_t* __JPEG_HandleTypeDef::QuantTable1***
  Basic Quantization Table for component 1
- ***uint8_t* __JPEG_HandleTypeDef::QuantTable2***
  Basic Quantization Table for component 2
- ***uint8_t* __JPEG_HandleTypeDef::QuantTable3***
  Basic Quantization Table for component 3
- ***HAL_LockTypeDef __JPEG_HandleTypeDef::Lock***
  JPEG locking object
- ***__IO HAL_JPEG_STATETypeDef __JPEG_HandleTypeDef::State***
  JPEG peripheral state
- ***__IO uint32_t __JPEG_HandleTypeDef::ErrorCode***
  JPEG Error code
- ***__IO uint32_t __JPEG_HandleTypeDef::Context***
  JPEG Internal context

- ***void(* __JPEG_HandleTypeDef::InfoReadyCallback)(struct __JPEG_HandleTypeDef *hjpeg, JPEG_ConfTypeDef *pInfo)***
  JPEG Info ready callback
- ***void(* __JPEG_HandleTypeDef::EncodeCpltCallback)(struct __JPEG_HandleTypeDef *hjpeg)***
  JPEG Encode complete callback
- ***void(* __JPEG_HandleTypeDef::DecodeCpltCallback)(struct __JPEG_HandleTypeDef *hjpeg)***
  JPEG Decode complete callback
- ***void(* __JPEG_HandleTypeDef::ErrorCallback)(struct __JPEG_HandleTypeDef *hjpeg)***
  JPEG Error callback
- ***void(* __JPEG_HandleTypeDef::GetDataCallback)(struct __JPEG_HandleTypeDef *hjpeg, uint32_t NbDecodedData)***
  JPEG Get Data callback
- ***void(* __JPEG_HandleTypeDef::DataReadyCallback)(struct __JPEG_HandleTypeDef *hjpeg, uint8_t *pDataOut, uint32_t OutDataLength)***
  JPEG Data ready callback
- ***void(* __JPEG_HandleTypeDef::MspInitCallback)(struct __JPEG_HandleTypeDef *hjpeg)***
  JPEG Msp Init callback
- ***void(* __JPEG_HandleTypeDef::MspDeInitCallback)(struct __JPEG_HandleTypeDef *hjpeg)***
  JPEG Msp DeInit callback

48.2 JPEG Firmware driver API description

The following section lists the various functions of the JPEG library.

48.2.1 How to use this driver

1. Initialize the JPEG peripheral using HAL_JPEG_Init : No initialization parameters are required. Only the call to HAL_JPEG_Init is necessary to initialize the JPEG peripheral.
2. If operation is JPEG encoding use function HAL_JPEG_ConfigEncoding to set the encoding parameters (mandatory before calling the encoding function). the application can change the encoding parameter ImageQuality from 1 to 100 to obtain a more or less quality (visual quality vs the original row image), and inversely more or less jpg file size.
3. Note that for decoding operation the JPEG peripheral output data are organized in YCbCr blocks called MCU (Minimum Coded Unit) as defioned in the JPEG specification ISO/IEC 10918-1 standard. It is up to the application to transform these YCbCr blocks to RGB data that can be display. Respectively, for Encoding operation the JPEG peripheral input should be organized in YCbCr MCU blocks. It is up to the application to perform the necessary RGB to YCbCr MCU blocks transformation before feeding the JPEG peripheral with data.
4. Use functions HAL_JPEG_Encode and HAL_JPEG_Decode to start respectively a JPEG encoding/decoding operation in polling method (blocking).
5. Use functions HAL_JPEG_Encode_IT and HAL_JPEG_Decode_IT to start respectively a JPEG encoding/ decoding operation with Interrupt method (not blocking).
6. Use functions HAL_JPEG_Encode_DMA and HAL_JPEG_Decode_DMA to start respectively a JPEG encoding/decoding operation with DMA method (not blocking).
7. Callback HAL_JPEG_InfoReadyCallback is asserted if the current operation is a JPEG decoding to provide the application with JPEG image parameters. This callback is asserted when the JPEG peripheral successfully parse the JPEG header.

8. Callback HAL_JPEG_GetDataCallback is asserted for both encoding and decoding operations to inform the application that the input buffer has been consumed by the peripheral and to ask for a new data chunk if the operation (encoding/decoding) has not been complete yet.
   - This CallBack should be implemented in the application side. It should call the function HAL_JPEG_ConfigInputBuffer if new input data are available, or call HAL_JPEG_Pause with parameter XferSelection set to JPEG_PAUSE_RESUME_INPUT to inform the JPEG HAL driver that the ongoing operation shall pause waiting for the application to provide a new input data chunk. Once the application succeed getting new data and if the input has been paused, the application can call the function HAL_JPEG_ConfigInputBuffer to set the new input buffer and size, then resume the JPEG HAL input by calling new function HAL_JPEG_Resume. If the application has ended feeding the HAL JPEG with input data (no more input data), the application Should call the function HAL_JPEG_ConfigInputBuffer (within the callback HAL_JPEG_GetDataCallback) with the parameter InDataLength set to zero.
   - The mechanism of HAL_JPEG_ConfigInputBuffer/HAL_JPEG_Pause/HAL_JPEG_Resume allows to the application to provide the input data (for encoding or decoding) by chunks. If the new input data chunk is not available (because data should be read from an input file for example) the application can pause the JPEG input (using function HAL_JPEG_Pause) Once the new input data chunk is available (read from a file for example), the application can call the function HAL_JPEG_ConfigInputBuffer to provide the HAL with the new chunk then resume the JPEG HAL input by calling function HAL_JPEG_Resume.
   - The application can call functions HAL_JPEG_ConfigInputBuffer then HAL_JPEG_Resume. any time (outside the HAL_JPEG_GetDataCallback) Once the new input chunk data available. However, to keep data coherency, the function HAL_JPEG_Pause must be imperatively called (if necessary) within the callback HAL_JPEG_GetDataCallback, i.e when the HAL JPEG has ended Transferring the previous chunk buffer to the JPEG peripheral.
9. Callback HAL_JPEG_DataReadyCallback is asserted when the HAL JPEG driver has filled the given output buffer with the given size.
   - This CallBack should be implemented in the application side. It should call the function HAL_JPEG_ConfigOutputBuffer to provide the HAL JPEG driver with the new output buffer location and size to be used to store next data chunk. if the application is not ready to provide the output chunk location then it can call the function HAL_JPEG_Pause with parameter XferSelection set to JPEG_PAUSE_RESUME_OUTPUT to inform the JPEG HAL driver that it shall pause output data. Once the application is ready to receive the new data chunk (output buffer location free or available) it should call the function HAL_JPEG_ConfigOutputBuffer to provide the HAL JPEG driver with the new output chunk buffer location and size, then call HAL_JPEG_Resume to inform the HAL that it shall resume outputting data in the given output buffer.
   - The mechanism of HAL_JPEG_ConfigOutputBuffer/HAL_JPEG_Pause/HAL_JPEG_Resume allows the application to receive data from the JPEG peripheral by chunks. when a chunk is received, the application can pause the HAL JPEG output data to be able to process these received data (YCbCr to RGB conversion in case of decoding or data storage in case of encoding).
   - The application can call functions HAL_JPEG_ ConfigOutputBuffer then HAL_JPEG_Resume. any time (outside the HAL_JPEG_DataReadyCallback) Once the output data buffer is free to use. However, to keep data coherency, the function HAL_JPEG_Pause must be imperatively called (if necessary) within the callback HAL_JPEG_ DataReadyCallback, i.e when the HAL JPEG has ended Transferring the previous chunk buffer from the JPEG peripheral to the application.
10. Callback HAL_JPEG_EncodeCpltCallback is asserted when the HAL JPEG driver has ended the current JPEG encoding operation, and all output data has been transmitted to the application.
11. Callback HAL_JPEG_DecodeCpltCallback is asserted when the HAL JPEG driver has ended the current JPEG decoding operation. and all output data has been transmitted to the application.
12. Callback HAL_JPEG_ErrorCallback is asserted when an error occurred during the current operation. the application can call the function HAL_JPEG_GetError() to retrieve the error codes.
13. By default the HAL JPEG driver uses the default quantization tables as provide in the JPEG specification (ISO/IEC 10918-1 standard) for encoding. User can change these default tables if necessary using the function HAL_JPEG_SetUserQuantTables Note that for decoding the quantization tables are automatically extracted from the JPEG header.
14. To control JPEG state you can use the following function: HAL_JPEG_GetState()

JPEG HAL driver macros list

Below the list of most used macros in JPEG HAL driver.

- __HAL_JPEG_RESET_HANDLE_STATE : Reset JPEG handle state.
- __HAL_JPEG_ENABLE : Enable the JPEG peripheral.
- __HAL_JPEG_DISABLE : Disable the JPEG peripheral.
- __HAL_JPEG_GET_FLAG : Check the specified JPEG status flag.
- __HAL_JPEG_CLEAR_FLAG : Clear the specified JPEG status flag.
- __HAL_JPEG_ENABLE_IT : Enable the specified JPEG Interrupt.
- __HAL_JPEG_DISABLE_IT : Disable the specified JPEG Interrupt.
- __HAL_JPEG_GET_IT_SOURCE : returns the state of the specified JPEG Interrupt (Enabled or disabled).

Callback registration

48.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the JPEG peripheral and creates the associated handle
- DeInitialize the JPEG peripheral

This section contains the following APIs:

- ***HAL_JPEG_Init()***
- ***HAL_JPEG_DeInit()***
- ***HAL_JPEG_MspInit()***
- ***HAL_JPEG_MspDeInit()***
- ***HAL_JPEG_RegisterCallback()***
- ***HAL_JPEG_UnRegisterCallback()***
- ***HAL_JPEG_RegisterInfoReadyCallback()***
- ***HAL_JPEG_UnRegisterInfoReadyCallback()***
- ***HAL_JPEG_RegisterGetDataCallback()***
- ***HAL_JPEG_UnRegisterGetDataCallback()***
- ***HAL_JPEG_RegisterDataReadyCallback()***
- ***HAL_JPEG_UnRegisterDataReadyCallback()***

48.2.3 Configuration functions

This section provides functions allowing to:

- HAL_JPEG_ConfigEncoding() : JPEG encoding configuration
- HAL_JPEG_GetInfo() : Extract the image configuration from the JPEG header during the decoding
- HAL_JPEG_EnableHeaderParsing() : Enable JPEG Header parsing for decoding
- HAL_JPEG_DisableHeaderParsing() : Disable JPEG Header parsing for decoding
- HAL_JPEG_SetUserQuantTables : Modify the default Quantization tables used for JPEG encoding.

This section contains the following APIs:

- ***HAL_JPEG_ConfigEncoding()***
- ***HAL_JPEG_GetInfo()***
- ***HAL_JPEG_EnableHeaderParsing()***
- ***HAL_JPEG_DisableHeaderParsing()***
- ***HAL_JPEG_SetUserQuantTables()***

48.2.4 JPEG processing functions

This section provides functions allowing to:

- HAL_JPEG_Encode() : JPEG encoding with polling process
- HAL_JPEG_Decode() : JPEG decoding with polling process
- HAL_JPEG_Encode_IT() : JPEG encoding with interrupt process
- HAL_JPEG_Decode_IT() : JPEG decoding with interrupt process
- HAL_JPEG_Encode_DMA() : JPEG encoding with DMA process
- HAL_JPEG_Decode_DMA() : JPEG decoding with DMA process
- HAL_JPEG_Pause() : Pause the Input/Output processing
- HAL_JPEG_Resume() : Resume the JPEG Input/Output processing
- HAL_JPEG_ConfigInputBuffer() : Config Encoding/Decoding Input Buffer
- HAL_JPEG_ConfigOutputBuffer() : Config Encoding/Decoding Output Buffer
- HAL_JPEG_Abort() : Aborts the JPEG Encoding/Decoding

This section contains the following APIs:

- ***HAL_JPEG_Encode()***
- ***HAL_JPEG_Decode()***
- ***HAL_JPEG_Encode_IT()***
- ***HAL_JPEG_Decode_IT()***
- ***HAL_JPEG_Encode_DMA()***
- ***HAL_JPEG_Decode_DMA()***
- ***HAL_JPEG_Pause()***
- ***HAL_JPEG_Resume()***
- ***HAL_JPEG_ConfigInputBuffer()***
- ***HAL_JPEG_ConfigOutputBuffer()***
- ***HAL_JPEG_Abort()***

48.2.5 JPEG Decode and Encode callback functions

This section provides callback functions:

- HAL_JPEG_InfoReadyCallback() : Decoding JPEG Info ready callback
- HAL_JPEG_EncodeCpltCallback() : Encoding complete callback.
- HAL_JPEG_DecodeCpltCallback() : Decoding complete callback.
- HAL_JPEG_ErrorCallback() : JPEG error callback.
- HAL_JPEG_GetDataCallback() : Get New Data chunk callback.
- HAL_JPEG_DataReadyCallback() : Decoded/Encoded Data ready callback.

This section contains the following APIs:

- ***HAL_JPEG_InfoReadyCallback()***
- ***HAL_JPEG_EncodeCpltCallback()***
- ***HAL_JPEG_DecodeCpltCallback()***
- ***HAL_JPEG_ErrorCallback()***
- ***HAL_JPEG_GetDataCallback()***
- ***HAL_JPEG_DataReadyCallback()***

48.2.6 JPEG IRQ handler management

This section provides JPEG IRQ handler function.

- HAL_JPEG_IRQHandler() : handles JPEG interrupt request

This section contains the following APIs:

- ***HAL_JPEG_IRQHandler()***

48.2.7 Peripheral State and Error functions

This section provides JPEG State and Errors function.

- HAL_JPEG_GetState() : permits to get in run-time the JPEG state.
- HAL_JPEG_GetError() : Returns the JPEG error code if any.

This section contains the following APIs:

- *HAL_JPEG_GetState()*
- *HAL_JPEG_GetError()*

48.2.8 Detailed description of functions

HAL_JPEG_Init

Function name

HAL_StatusTypeDef HAL_JPEG_Init (JPEG_HandleTypeDef * hjpeg)

Function description

Initializes the JPEG according to the specified parameters in the JPEG_InitTypeDef and creates the associated handle.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module

Return values

- **HAL:** status

HAL_JPEG_DeInit

Function name

HAL_StatusTypeDef HAL_JPEG_DeInit (JPEG_HandleTypeDef * hjpeg)

Function description

DeInitializes the JPEG peripheral.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module

Return values

- **HAL:** status

HAL_JPEG_MspInit

Function name

void HAL_JPEG_MspInit (JPEG_HandleTypeDef * hjpeg)

Function description

Initializes the JPEG MSP.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module

Return values

- **None:**

HAL_JPEG_MspDeInit

Function name

void HAL_JPEG_MspDeInit (JPEG_HandleTypeDef * hjpeg)

Function description

DeInitializes JPEG MSP.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module

Return values

- **None:**

HAL_JPEG_RegisterCallback

Function name

HAL_StatusTypeDef HAL_JPEG_RegisterCallback (JPEG_HandleTypeDef * hjpeg, HAL_JPEG_CallbackIDTypeDef CallbackID, pJPEG_CallbackTypeDef pCallback)

Function description

Register a User JPEG Callback To be used instead of the weak predefined callback.

Parameters

- **hjpeg:** JPEG handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_JPEG_ENCODE_CPLT_CB_ID Encode Complete callback ID
  - HAL_JPEG_DECODE_CPLT_CB_ID Decode Complete callback ID
  - HAL_JPEG_ERROR_CB_ID Error callback ID
  - HAL_JPEG_MSPINIT_CB_ID MspInit callback ID
  - HAL_JPEG_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_JPEG_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_JPEG_UnRegisterCallback (JPEG_HandleTypeDef * hjpeg, HAL_JPEG_CallbackIDTypeDef CallbackID)

Function description

Unregister a JPEG Callback JPEG callabck is redirected to the weak predefined callback.

Parameters

- **hjpeg:** JPEG handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values: This parameter can be one of the following values:
  - HAL_JPEG_ENCODE_CPLT_CB_ID Encode Complete callback ID
  - HAL_JPEG_DECODE_CPLT_CB_ID Decode Complete callback ID
  - HAL_JPEG_ERROR_CB_ID Error callback ID
  - HAL_JPEG_MSPINIT_CB_ID MspInit callback ID
  - HAL_JPEG_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_JPEG_RegisterInfoReadyCallback

Function name

HAL_StatusTypeDef HAL_JPEG_RegisterInfoReadyCallback (JPEG_HandleTypeDef * hjpeg, pJPEG_InfoReadyCallbackTypeDef pCallback)

Function description

Register Info Ready JPEG Callback To be used instead of the weak HAL_JPEG_InfoReadyCallback() predefined callback.

Parameters

- **hjpeg:** JPEG handle
- **pCallback:** pointer to the Info Ready Callback function

Return values

- **HAL:** status

HAL_JPEG_UnRegisterInfoReadyCallback

Function name

HAL_StatusTypeDef HAL_JPEG_UnRegisterInfoReadyCallback (JPEG_HandleTypeDef * hjpeg)

Function description

UnRegister the Info Ready JPEG Callback Info Ready JPEG Callback is redirected to the weak HAL_JPEG_InfoReadyCallback() predefined callback.

Parameters

- **hjpeg:** JPEG handle

Return values

- **HAL:** status

HAL_JPEG_RegisterGetDataCallback

Function name

HAL_StatusTypeDef HAL_JPEG_RegisterGetDataCallback (JPEG_HandleTypeDef * hjpeg, pJPEG_GetDataCallbackTypeDef pCallback)

Function description

Register Get Data JPEG Callback To be used instead of the weak HAL_JPEG_GetDataCallback() predefined callback.

Parameters

- **hjpeg:** JPEG handle
- **pCallback:** pointer to the Get Data Callback function

Return values

- **HAL:** status

HAL_JPEG_UnRegisterGetDataCallback

Function name

HAL_StatusTypeDef HAL_JPEG_UnRegisterGetDataCallback (JPEG_HandleTypeDef * hjpeg)

Function description

UnRegister the Get Data JPEG Callback Get Data JPEG Callback is redirected to the weak HAL_JPEG_GetDataCallback() predefined callback.

Parameters

- **hjpeg:** JPEG handle

Return values

- **HAL:** status

HAL_JPEG_RegisterDataReadyCallback

Function name

HAL_StatusTypeDef HAL_JPEG_RegisterDataReadyCallback (JPEG_HandleTypeDef * hjpeg, pJPEG_DataReadyCallbackTypeDef pCallback)

Function description

Register Data Ready JPEG Callback To be used instead of the weak HAL_JPEG_DataReadyCallback() predefined callback.

Parameters

- **hjpeg:** JPEG handle
- **pCallback:** pointer to the Get Data Callback function

Return values

- **HAL:** status

HAL_JPEG_UnRegisterDataReadyCallback

Function name

HAL_StatusTypeDef HAL_JPEG_UnRegisterDataReadyCallback (JPEG_HandleTypeDef * hjpeg)

Function description

UnRegister the Data Ready JPEG Callback Get Data Ready Callback is redirected to the weak HAL_JPEG_DataReadyCallback() predefined callback.

Parameters

- **hjpeg:** JPEG handle

Return values

- **HAL:** status

HAL_JPEG_ConfigEncoding

Function name

HAL_StatusTypeDef HAL_JPEG_ConfigEncoding (JPEG_HandleTypeDef * hjpeg, JPEG_ConfTypeDef * pConf)

Function description

Set the JPEG encoding configuration.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **pConf:** pointer to a JPEG_ConfTypeDef structure that contains the encoding configuration

Return values

- **HAL:** status

HAL_JPEG_GetInfo

Function name

HAL_StatusTypeDef HAL_JPEG_GetInfo (JPEG_HandleTypeDef * hjpeg, JPEG_ConfTypeDef * pInfo)

Function description

Extract the image configuration from the JPEG header during the decoding.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **pInfo:** pointer to a JPEG_ConfTypeDef structure that contains The JPEG decoded header information

Return values

- **HAL:** status

HAL_JPEG_EnableHeaderParsing

Function name

HAL_StatusTypeDef HAL_JPEG_EnableHeaderParsing (JPEG_HandleTypeDef * hjpeg)

Function description

Enable JPEG Header parsing for decoding.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for the JPEG.

Return values

- **HAL:** status

HAL_JPEG_DisableHeaderParsing

Function name

HAL_StatusTypeDef HAL_JPEG_DisableHeaderParsing (JPEG_HandleTypeDef * hjpeg)

Function description

Disable JPEG Header parsing for decoding.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for the JPEG.

Return values

- **HAL:** status

HAL_JPEG_SetUserQuantTables

Function name

HAL_StatusTypeDef HAL_JPEG_SetUserQuantTables (JPEG_HandleTypeDef * hjpeg, uint8_t * QTable0, uint8_t * QTable1, uint8_t * QTable2, uint8_t * QTable3)

Function description

Modify the default Quantization tables used for JPEG encoding.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **QTable0:** pointer to uint8_t , define the user quantification table for color component 1. If NULL assume no need to update the table and no error return
- **QTable1:** pointer to uint8_t , define the user quantification table for color component 2. If NULL assume no need to update the table and no error return.
- **QTable2:** pointer to uint8_t , define the user quantification table for color component 3, If NULL assume no need to update the table and no error return.
- **QTable3:** pointer to uint8_t , define the user quantification table for color component 4. If NULL assume no need to update the table and no error return.

Return values

- **HAL:** status

HAL_JPEG_Encode

Function name

HAL_StatusTypeDef HAL_JPEG_Encode (JPEG_HandleTypeDef * hjpeg, uint8_t * pDataInMCU, uint32_t InDataLength, uint8_t * pDataOut, uint32_t OutDataLength, uint32_t Timeout)

Function description

Starts JPEG encoding with polling processing.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **pDataInMCU:** Pointer to the Input buffer
- **InDataLength:** size in bytes Input buffer
- **pDataOut:** Pointer to the jpeg output data buffer
- **OutDataLength:** size in bytes of the Output buffer
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_JPEG_Decode

Function name

HAL_StatusTypeDef HAL_JPEG_Decode (JPEG_HandleTypeDef * hjpeg, uint8_t * pDataIn, uint32_t InDataLength, uint8_t * pDataOutMCU, uint32_t OutDataLength, uint32_t Timeout)

Function description

Starts JPEG decoding with polling processing.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **pDataIn:** Pointer to the input data buffer
- **InDataLength:** size in bytes Input buffer
- **pDataOutMCU:** Pointer to the Output data buffer
- **OutDataLength:** size in bytes of the Output buffer
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_JPEG_Encode_IT

Function name

HAL_StatusTypeDef HAL_JPEG_Encode_IT (JPEG_HandleTypeDef * hjpeg, uint8_t * pDataInMCU, uint32_t InDataLength, uint8_t * pDataOut, uint32_t OutDataLength)

Function description

Starts JPEG encoding with interrupt processing.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **pDataInMCU:** Pointer to the Input buffer
- **InDataLength:** size in bytes Input buffer
- **pDataOut:** Pointer to the jpeg output data buffer
- **OutDataLength:** size in bytes of the Output buffer

Return values

- **HAL:** status

HAL_JPEG_Decode_IT

Function name

HAL_StatusTypeDef HAL_JPEG_Decode_IT (JPEG_HandleTypeDef * hjpeg, uint8_t * pDataIn, uint32_t InDataLength, uint8_t * pDataOutMCU, uint32_t OutDataLength)

Function description

Starts JPEG decoding with interrupt processing.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **pDataIn:** Pointer to the input data buffer
- **InDataLength:** size in bytes Input buffer
- **pDataOutMCU:** Pointer to the Output data buffer
- **OutDataLength:** size in bytes of the Output buffer

Return values

- **HAL:** status

HAL_JPEG_Encode_DMA

Function name

HAL_StatusTypeDef HAL_JPEG_Encode_DMA (JPEG_HandleTypeDef * hjpeg, uint8_t * pDataInMCU, uint32_t InDataLength, uint8_t * pDataOut, uint32_t OutDataLength)

Function description

Starts JPEG encoding with DMA processing.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **pDataInMCU:** Pointer to the Input buffer
- **InDataLength:** size in bytes Input buffer
- **pDataOut:** Pointer to the jpeg output data buffer
- **OutDataLength:** size in bytes of the Output buffer

Return values

- **HAL:** status

HAL_JPEG_Decode_DMA

Function name

HAL_StatusTypeDef HAL_JPEG_Decode_DMA (JPEG_HandleTypeDef * hjpeg, uint8_t * pDataIn, uint32_t InDataLength, uint8_t * pDataOutMCU, uint32_t OutDataLength)

Function description

Starts JPEG decoding with DMA processing.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **pDataIn:** Pointer to the input data buffer
- **InDataLength:** size in bytes Input buffer
- **pDataOutMCU:** Pointer to the Output data buffer
- **OutDataLength:** size in bytes of the Output buffer

Return values

- **HAL:** status

HAL_JPEG_Pause

Function name

HAL_StatusTypeDef HAL_JPEG_Pause (JPEG_HandleTypeDef * hjpeg, uint32_t XferSelection)

Function description

Pause the JPEG Input/Output processing.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **XferSelection:** This parameter can be one of the following values : JPEG_PAUSE_RESUME_INPUT : Pause Input processing JPEG_PAUSE_RESUME_OUTPUT: Pause Output processing JPEG_PAUSE_RESUME_INPUT_OUTPUT: Pause Input and Output processing

Return values

- **HAL:** status

HAL_JPEG_Resume

Function name

HAL_StatusTypeDef HAL_JPEG_Resume (JPEG_HandleTypeDef * hjpeg, uint32_t XferSelection)

Function description

Resume the JPEG Input/Output processing.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **XferSelection:** This parameter can be one of the following values : JPEG_PAUSE_RESUME_INPUT : Resume Input processing JPEG_PAUSE_RESUME_OUTPUT: Resume Output processing JPEG_PAUSE_RESUME_INPUT_OUTPUT: Resume Input and Output processing

Return values

- **HAL:** status

HAL_JPEG_ConfigInputBuffer

Function name

void HAL_JPEG_ConfigInputBuffer (JPEG_HandleTypeDef * hjpeg, uint8_t * pNewInputBuffer, uint32_t InDataLength)

Function description

Config Encoding/Decoding Input Buffer.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module.
- **pNewInputBuffer:** Pointer to the new input data buffer
- **InDataLength:** Size in bytes of the new Input data buffer

Return values

- **HAL:** status

HAL_JPEG_ConfigOutputBuffer

Function name

void HAL_JPEG_ConfigOutputBuffer (JPEG_HandleTypeDef * hjpeg, uint8_t * pNewOutputBuffer, uint32_t OutDataLength)

Function description

Config Encoding/Decoding Output Buffer.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module.
- **pNewOutputBuffer:** Pointer to the new output data buffer
- **OutDataLength:** Size in bytes of the new Output data buffer

Return values

- **HAL:** status

HAL_JPEG_Abort

Function name

HAL_StatusTypeDef HAL_JPEG_Abort (JPEG_HandleTypeDef * hjpeg)

Function description

Aborts the JPEG Encoding/Decoding.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module

Return values

- **HAL:** status

HAL_JPEG_InfoReadyCallback

Function name

void HAL_JPEG_InfoReadyCallback (JPEG_HandleTypeDef * hjpeg, JPEG_ConfTypeDef * pInfo)

Function description

Decoding JPEG Info ready callback.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **pInfo:** pointer to a JPEG_ConfTypeDef structure that contains The JPEG decoded header information

Return values

- **None:**

HAL_JPEG_EncodeCpltCallback

Function name

void HAL_JPEG_EncodeCpltCallback (JPEG_HandleTypeDef * hjpeg)

Function description

Encoding complete callback.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module

Return values

- **None:**

HAL_JPEG_DecodeCpltCallback

Function name

void HAL_JPEG_DecodeCpltCallback (JPEG_HandleTypeDef * hjpeg)

Function description

Decoding complete callback.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module

Return values

- **None:**

HAL_JPEG_ErrorCallback

Function name

void HAL_JPEG_ErrorCallback (JPEG_HandleTypeDef * hjpeg)

Function description

JPEG error callback.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module

Return values

- **None:**

HAL_JPEG_GetDataCallback

Function name

void HAL_JPEG_GetDataCallback (JPEG_HandleTypeDef * hjpeg, uint32_t NbDecodedData)

Function description

Get New Data chunk callback.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **NbDecodedData:** Number of consummed data in the previous chunk in bytes

Return values

- **None:**

HAL_JPEG_DataReadyCallback

Function name

void HAL_JPEG_DataReadyCallback (JPEG_HandleTypeDef * hjpeg, uint8_t * pDataOut, uint32_t OutDataLength)

Function description

Decoded/Encoded Data ready callback.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **pDataOut:** pointer to the output data buffer
- **OutDataLength:** number in bytes of data available in the specified output buffer

Return values

- **None:**

HAL_JPEG_IRQHandler

Function name

void HAL_JPEG_IRQHandler (JPEG_HandleTypeDef * hjpeg)

Function description

This function handles JPEG interrupt request.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module

Return values

- **None:**

HAL_JPEG_GetState

Function name

HAL_JPEG_STATETypeDef HAL_JPEG_GetState (JPEG_HandleTypeDef * hjpeg)

Function description

Returns the JPEG state.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module

Return values

- **JPEG:** state

HAL_JPEG_GetError

Function name

uint32_t HAL_JPEG_GetError (JPEG_HandleTypeDef * hjpeg)

Function description

Return the JPEG error code.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for the specified JPEG.

Return values

- **JPEG:** Error Code

48.3 JPEG Firmware driver defines

The following section lists the various define and macros of the module.

48.3.1 JPEG

JPEG

JPEG Chrominance Sampling

JPEG_444_SUBSAMPLING

Chroma Subsampling 4:4:4

JPEG_420_SUBSAMPLING

Chroma Subsampling 4:2:0

JPEG_422_SUBSAMPLING

Chroma Subsampling 4:2:2

JPEG ColorSpace

JPEG_GRAYSCALE_COLORSPACE

JPEG_YCBCR_COLORSPACE

JPEG_CMYK_COLORSPACE

JPEG Error Code definition

HAL_JPEG_ERROR_NONE

No error

HAL_JPEG_ERROR_HUFF_TABLE

HUffman Table programming error

HAL_JPEG_ERROR_QUANT_TABLE

Quantization Table programming error

HAL_JPEG_ERROR_DMA

DMA transfer error

HAL_JPEG_ERROR_TIMEOUT

Timeout error

HAL_JPEG_ERROR_INVALID_CALLBACK

Invalid Callback error

JPEG Exported Macros

__HAL_JPEG_RESET_HANDLE_STATE

Description:

- Reset JPEG handle state.

Parameters:

- __HANDLE__: specifies the JPEG handle.

Return value:

- None

__HAL_JPEG_ENABLE

Description:

- Enable the JPEG peripheral.

Parameters:

- __HANDLE__: specifies the JPEG handle.

Return value:

- None

__HAL_JPEG_DISABLE

Description:

- Disable the JPEG peripheral.

Parameters:

- __HANDLE__: specifies the JPEG handle.

Return value:

- None

__HAL_JPEG_GET_FLAG

Description:

- Check the specified JPEG status flag.

Parameters:

- __HANDLE__: specifies the JPEG handle.
- __FLAG__: specifies the flag to check This parameter can be one of the following values:
  - JPEG_FLAG_IFTF : The input FIFO is not full and is bellow its threshold flag
  - JPEG_FLAG_IFNFF : The input FIFO Not Full Flag, a data can be written
  - JPEG_FLAG_OFTF : The output FIFO is not empty and has reach its threshold
  - JPEG_FLAG_OFNEF : The output FIFO is not empty, a data is available
  - JPEG_FLAG_EOCF : JPEG Codec core has finished the encoding or the decoding process and than last data has been sent to the output FIFO
  - JPEG_FLAG_HPDF : JPEG Codec has finished the parsing of the headers and the internal registers have been updated
  - JPEG_FLAG_COF : JPEG Codec operation on going flag

Return value:

- __HAL_JPEG_GET_FLAG: : returns The new state of __FLAG__ (TRUE or FALSE)

__HAL_JPEG_CLEAR_FLAG

Description:

- Clear the specified JPEG status flag.

Parameters:

- __HANDLE__: specifies the JPEG handle.
- __FLAG__: specifies the flag to clear This parameter can be one of the following values:
  - JPEG_FLAG_EOCF : JPEG Codec core has finished the encoding or the decoding process and than last data has been sent to the output FIFO
  - JPEG_FLAG_HPDF : JPEG Codec has finished the parsing of the headers

Return value:

- None

__HAL_JPEG_ENABLE_IT

Description:

- Enable Interrupt.

Parameters:

- __HANDLE__: specifies the JPEG handle.
- __INTERRUPT__: specifies the interrupt to enable This parameter can be one of the following values:
  - JPEG_IT_IFT : Input FIFO Threshold Interrupt
  - JPEG_IT_IFNF : Input FIFO Not Full Interrupt
  - JPEG_IT_OFT : Output FIFO Threshold Interrupt
  - JPEG_IT_OFNE : Output FIFO Not empty Interrupt
  - JPEG_IT_EOC : End of Conversion Interrupt
  - JPEG_IT_HPD : Header Parsing Done Interrupt

Return value:

- No: return

__HAL_JPEG_DISABLE_IT

Description:

- Disable Interrupt.

Parameters:

- __HANDLE__: specifies the JPEG handle.
- __INTERRUPT__: specifies the interrupt to disable This parameter can be one of the following values:
  - JPEG_IT_IFT : Input FIFO Threshold Interrupt
  - JPEG_IT_IFNF : Input FIFO Not Full Interrupt
  - JPEG_IT_OFT : Output FIFO Threshold Interrupt
  - JPEG_IT_OFNE : Output FIFO Not empty Interrupt
  - JPEG_IT_EOC : End of Conversion Interrupt
  - JPEG_IT_HPD : Header Parsing Done Interrupt

Return value:

- No: return

Notes:

- To disable an IT we must use MODIFY_REG macro to avoid writing "1" to the FIFO flush bits located in the same IT enable register (CR register).

__HAL_JPEG_GET_IT_SOURCE

Description:

- Get Interrupt state.

Parameters:

- __HANDLE__: specifies the JPEG handle.
- __INTERRUPT__: specifies the interrupt to check This parameter can be one of the following values:
  - JPEG_IT_IFT : Input FIFO Threshold Interrupt
  - JPEG_IT_IFNF : Input FIFO Not Full Interrupt
  - JPEG_IT_OFT : Output FIFO Threshold Interrupt
  - JPEG_IT_OFNE : Output FIFO Not empty Interrupt
  - JPEG_IT_EOC : End of Conversion Interrupt
  - JPEG_IT_HPD : Header Parsing Done Interrupt

Return value:

- returns: The new state of __INTERRUPT__ (Enabled or disabled)

JPEG Flag definition

JPEG_FLAG_IFTF

Input FIFO is not full and is bellow its threshold flag

JPEG_FLAG_IFNFF

Input FIFO Not Full Flag, a data can be written

JPEG_FLAG_OFTF

Output FIFO is not empty and has reach its threshold

JPEG_FLAG_OFNEF

Output FIFO is not empty, a data is available

JPEG_FLAG_EOCF

JPEG Codec core has finished the encoding or the decoding process and than last data has been sent to the output FIFO

JPEG_FLAG_HPDF

JPEG Codec has finished the parsing of the headers and the internal registers have been updated

JPEG_FLAG_COF

JPEG Codec operation on going flag

JPEG_FLAG_ALL

JPEG Codec All previous flag

JPEG Image Quality

JPEG_IMAGE_QUALITY_MIN

Minimum JPEG quality

JPEG_IMAGE_QUALITY_MAX

Maximum JPEG quality

JPEG Interrupt configuration definition

JPEG_IT_IFT

Input FIFO Threshold Interrupt

JPEG_IT_IFNF

Input FIFO Not Full Interrupt

JPEG_IT_OFT

Output FIFO Threshold Interrupt

JPEG_IT_OFNE

Output FIFO Not Empty Interrupt

JPEG_IT_EOC

End of Conversion Interrupt

JPEG_IT_HPD

Header Parsing Done Interrupt

JPEG Process Pause Resume definition

JPEG_PAUSE_RESUME_INPUT

Pause/Resume Input FIFO Xfer

JPEG_PAUSE_RESUME_OUTPUT

Pause/Resume Output FIFO Xfer

JPEG_PAUSE_RESUME_INPUT_OUTPUT

Pause/Resume Input and Output FIFO Xfer

JPEG Quantization Table Size

JPEG_QUANT_TABLE_SIZE

JPEG Quantization Table Size in bytes

49 HAL LPTIM Generic Driver

49.1 LPTIM Firmware driver registers structures

49.1.1 LPTIM_ClockConfigTypeDef

LPTIM_ClockConfigTypeDef is defined in the stm32h7xx_hal_lptim.h

Data Fields

- ***uint32_t Source***
- ***uint32_t Prescaler***

Field Documentation

- ***uint32_t LPTIM_ClockConfigTypeDef::Source***
  Selects the clock source. This parameter can be a value of ***LPTIM_Clock_Source***
- ***uint32_t LPTIM_ClockConfigTypeDef::Prescaler***
  Specifies the counter clock Prescaler. This parameter can be a value of ***LPTIM_Clock_Prescaler***

49.1.2 LPTIM_ULPClockConfigTypeDef

LPTIM_ULPClockConfigTypeDef is defined in the stm32h7xx_hal_lptim.h

Data Fields

- ***uint32_t Polarity***
- ***uint32_t SampleTime***

Field Documentation

- ***uint32_t LPTIM_ULPClockConfigTypeDef::Polarity***
  Selects the polarity of the active edge for the counter unit if the ULPTIM input is selected. Note: This parameter is used only when Ultra low power clock source is used. Note: If the polarity is configured on 'both edges', an auxiliary clock (one of the Low power oscillator) must be active. This parameter can be a value of ***LPTIM_Clock_Polarity***
- ***uint32_t LPTIM_ULPClockConfigTypeDef::SampleTime***
  Selects the clock sampling time to configure the clock glitch filter. Note: This parameter is used only when Ultra low power clock source is used. This parameter can be a value of ***LPTIM_Clock_Sample_Time***

49.1.3 LPTIM_TriggerConfigTypeDef

LPTIM_TriggerConfigTypeDef is defined in the stm32h7xx_hal_lptim.h

Data Fields

- ***uint32_t Source***
- ***uint32_t ActiveEdge***
- ***uint32_t SampleTime***

Field Documentation

- ***uint32_t LPTIM_TriggerConfigTypeDef::Source***
  Selects the Trigger source. This parameter can be a value of ***LPTIM_Trigger_Source***
- ***uint32_t LPTIM_TriggerConfigTypeDef::ActiveEdge***
  Selects the Trigger active edge. Note: This parameter is used only when an external trigger is used. This parameter can be a value of ***LPTIM_External_Trigger_Polarity***
- ***uint32_t LPTIM_TriggerConfigTypeDef::SampleTime***
  Selects the trigger sampling time to configure the clock glitch filter. Note: This parameter is used only when an external trigger is used. This parameter can be a value of ***LPTIM_Trigger_Sample_Time***

49.1.4 LPTIM_InitTypeDef

LPTIM_InitTypeDef is defined in the stm32h7xx_hal_lptim.h

Data Fields

- ***LPTIM_ClockConfigTypeDef Clock***
- ***LPTIM_ULPClockConfigTypeDef UltraLowPowerClock***
- ***LPTIM_TriggerConfigTypeDef Trigger***
- ***uint32_t OutputPolarity***
- ***uint32_t UpdateMode***
- ***uint32_t CounterSource***
- ***uint32_t Input1Source***
- ***uint32_t Input2Source***

Field Documentation

- ***LPTIM_ClockConfigTypeDef LPTIM_InitTypeDef::Clock***
  Specifies the clock parameters
- ***LPTIM_ULPClockConfigTypeDef LPTIM_InitTypeDef::UltraLowPowerClock***
  Specifies the Ultra Low Power clock parameters
- ***LPTIM_TriggerConfigTypeDef LPTIM_InitTypeDef::Trigger***
  Specifies the Trigger parameters
- ***uint32_t LPTIM_InitTypeDef::OutputPolarity***
  Specifies the Output polarity. This parameter can be a value of ***LPTIM_Output_Polarity***
- ***uint32_t LPTIM_InitTypeDef::UpdateMode***
  Specifies whether the update of the autoreload and the compare values is done immediately or after the end of current period. This parameter can be a value of ***LPTIM_Updating_Mode***
- ***uint32_t LPTIM_InitTypeDef::CounterSource***
  Specifies whether the counter is incremented each internal event or each external event. This parameter can be a value of ***LPTIM_Counter_Source***
- ***uint32_t LPTIM_InitTypeDef::Input1Source***
  Specifies source selected for input1 (GPIO or comparator output). This parameter can be a value of ***LPTIM_Input1_Source***
- ***uint32_t LPTIM_InitTypeDef::Input2Source***
  Specifies source selected for input2 (GPIO or comparator output). Note: This parameter is used only for encoder feature so is used only for LPTIM1 instance. This parameter can be a value of ***LPTIM_Input2_Source***

49.1.5 __LPTIM_HandleTypeDef

__LPTIM_HandleTypeDef is defined in the stm32h7xx_hal_lptim.h

Data Fields

- ***LPTIM_TypeDef * Instance***
- ***LPTIM_InitTypeDef Init***
- ***HAL_StatusTypeDef Status***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_LPTIM_StateTypeDef State***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***void(* CompareMatchCallback***
- ***void(* AutoReloadMatchCallback***
- ***void(* TriggerCallback***
- ***void(* CompareWriteCallback***
- ***void(* AutoReloadWriteCallback***
- ***void(* DirectionUpCallback***

- ***void(* DirectionDownCallback***

Field Documentation

- ***LPTIM_TypeDef* __LPTIM_HandleTypeDef::Instance***
  Register base address
- ***LPTIM_InitTypeDef __LPTIM_HandleTypeDef::Init***
  LPTIM required parameters
- ***HAL_StatusTypeDef __LPTIM_HandleTypeDef::Status***
  LPTIM peripheral status
- ***HAL_LockTypeDef __LPTIM_HandleTypeDef::Lock***
  LPTIM locking object
- ***__IO HAL_LPTIM_StateTypeDef __LPTIM_HandleTypeDef::State***
  LPTIM peripheral state
- ***void(* __LPTIM_HandleTypeDef::MspInitCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  LPTIM Base Msp Init Callback
- ***void(* __LPTIM_HandleTypeDef::MspDeInitCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  LPTIM Base Msp DeInit Callback
- ***void(* __LPTIM_HandleTypeDef::CompareMatchCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Compare match Callback
- ***void(* __LPTIM_HandleTypeDef::AutoReloadMatchCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Auto-reload match Callback
- ***void(* __LPTIM_HandleTypeDef::TriggerCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  External trigger event detection Callback
- ***void(* __LPTIM_HandleTypeDef::CompareWriteCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Compare register write complete Callback
- ***void(* __LPTIM_HandleTypeDef::AutoReloadWriteCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Auto-reload register write complete Callback
- ***void(* __LPTIM_HandleTypeDef::DirectionUpCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Up-counting direction change Callback
- ***void(* __LPTIM_HandleTypeDef::DirectionDownCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Down-counting direction change Callback

49.2 LPTIM Firmware driver API description

The following section lists the various functions of the LPTIM library.

49.2.1 How to use this driver

The LPTIM HAL driver can be used as follows:

1. Initialize the LPTIM low level resources by implementing the HAL_LPTIM_MspInit():
   - Enable the LPTIM interface clock using __HAL_RCC_LPTIMx_CLK_ENABLE().
   - In case of using interrupts (e.g. HAL_LPTIM_PWM_Start_IT()):
     - Configure the LPTIM interrupt priority using HAL_NVIC_SetPriority().
     - Enable the LPTIM IRQ handler using HAL_NVIC_EnableIRQ().
     - In LPTIM IRQ handler, call HAL_LPTIM_IRQHandler().

2. Initialize the LPTIM HAL using HAL_LPTIM_Init(). This function configures mainly:
   - The instance: LPTIM1 or LPTIM2.
   - Clock: the counter clock.
     - Source : it can be either the ULPTIM input (IN1) or one of the internal clock; (APB, LSE, LSI or MSI).
     - Prescaler: select the clock divider.
   - UltraLowPowerClock : To be used only if the ULPTIM is selected as counter clock source.
     - Polarity: polarity of the active edge for the counter unit if the ULPTIM input is selected.
     - SampleTime: clock sampling time to configure the clock glitch filter.
   - Trigger: How the counter start.
     - Source: trigger can be software or one of the hardware triggers.
     - ActiveEdge : only for hardware trigger.
     - SampleTime : trigger sampling time to configure the trigger glitch filter.
   - OutputPolarity : 2 opposite polarities are possible.
   - UpdateMode: specifies whether the update of the autoreload and the compare values is done immediately or after the end of current period.
   - Input1Source: Source selected for input1 (GPIO or comparator output).
   - Input2Source: Source selected for input2 (GPIO or comparator output). Input2 is used only for encoder feature so is used only for LPTIM1 instance.
3. Six modes are available:
   - PWM Mode: To generate a PWM signal with specified period and pulse, call HAL_LPTIM_PWM_Start() or HAL_LPTIM_PWM_Start_IT() for interruption mode.
   - One Pulse Mode: To generate pulse with specified width in response to a stimulus, call HAL_LPTIM_OnePulse_Start() or HAL_LPTIM_OnePulse_Start_IT() for interruption mode.
   - Set once Mode: In this mode, the output changes the level (from low level to high level if the output polarity is configured high, else the opposite) when a compare match occurs. To start this mode, call HAL_LPTIM_SetOnce_Start() or HAL_LPTIM_SetOnce_Start_IT() for interruption mode.
   - Encoder Mode: To use the encoder interface call HAL_LPTIM_Encoder_Start() or HAL_LPTIM_Encoder_Start_IT() for interruption mode. Only available for LPTIM1 instance.
   - Time out Mode: an active edge on one selected trigger input rests the counter. The first trigger event will start the timer, any successive trigger event will reset the counter and the timer will restart. To start this mode call HAL_LPTIM_TimeOut_Start_IT() or HAL_LPTIM_TimeOut_Start_IT() for interruption mode.
   - Counter Mode: counter can be used to count external events on the LPTIM Input1 or it can be used to count internal clock cycles. To start this mode, call HAL_LPTIM_Counter_Start() or HAL_LPTIM_Counter_Start_IT() for interruption mode.
4. User can stop any process by calling the corresponding API: HAL_LPTIM_Xxx_Stop() or HAL_LPTIM_Xxx_Stop_IT() if the process is already started in interruption mode.
5. De-initialize the LPTIM peripheral using HAL_LPTIM_DeInit().

Callback registration

The compilation define USE_HAL_LPTIM_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_LPTIM_RegisterCallback() to register a callback. HAL_LPTIM_RegisterCallback() takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_LPTIM_UnRegisterCallback() to reset a callback to the default weak function. HAL_LPTIM_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID.

These functions allow to register/unregister following callbacks:

- MspInitCallback : LPTIM Base Msp Init Callback.
- MspDeInitCallback : LPTIM Base Msp DeInit Callback.
- CompareMatchCallback : Compare match Callback.
- AutoReloadMatchCallback : Auto-reload match Callback.
- TriggerCallback : External trigger event detection Callback.

- CompareWriteCallback : Compare register write complete Callback.
- AutoReloadWriteCallback : Auto-reload register write complete Callback.
- DirectionUpCallback : Up-counting direction change Callback.
- DirectionDownCallback : Down-counting direction change Callback.

By default, after the Init and when the state is HAL_LPTIM_STATE_RESET all interrupt callbacks are set to the corresponding weak functions: examples HAL_LPTIM_TriggerCallback(), HAL_LPTIM_CompareMatchCallback().

Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functionalities in the Init/DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the Init/DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand)

Callbacks can be registered/unregistered in HAL_LPTIM_STATE_READY state only. Exception done MspInit/MspDeInit that can be registered/unregistered in HAL_LPTIM_STATE_READY or HAL_LPTIM_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_LPTIM_RegisterCallback() before calling DeInit or Init function.

When The compilation define USE_HAL_LPTIM_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

49.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the LPTIM according to the specified parameters in the LPTIM_InitTypeDef and initialize the associated handle.
- DeInitialize the LPTIM peripheral.
- Initialize the LPTIM MSP.
- DeInitialize the LPTIM MSP.

This section contains the following APIs:

- ***HAL_LPTIM_Init()***
- ***HAL_LPTIM_DeInit()***
- ***HAL_LPTIM_MspInit()***
- ***HAL_LPTIM_MspDeInit()***

49.2.3 LPTIM Start Stop operation functions

This section provides functions allowing to:

- Start the PWM mode.
- Stop the PWM mode.
- Start the One pulse mode.
- Stop the One pulse mode.
- Start the Set once mode.
- Stop the Set once mode.
- Start the Encoder mode.
- Stop the Encoder mode.
- Start the Timeout mode.
- Stop the Timeout mode.
- Start the Counter mode.
- Stop the Counter mode.

This section contains the following APIs:

- ***HAL_LPTIM_PWM_Start()***
- ***HAL_LPTIM_PWM_Stop()***
- ***HAL_LPTIM_PWM_Start_IT()***
- ***HAL_LPTIM_PWM_Stop_IT()***
- ***HAL_LPTIM_OnePulse_Start()***
- ***HAL_LPTIM_OnePulse_Stop()***

- *HAL_LPTIM_OnePulse_Start_IT()*
- *HAL_LPTIM_OnePulse_Stop_IT()*
- *HAL_LPTIM_SetOnce_Start()*
- *HAL_LPTIM_SetOnce_Stop()*
- *HAL_LPTIM_SetOnce_Start_IT()*
- *HAL_LPTIM_SetOnce_Stop_IT()*
- *HAL_LPTIM_Encoder_Start()*
- *HAL_LPTIM_Encoder_Stop()*
- *HAL_LPTIM_Encoder_Start_IT()*
- *HAL_LPTIM_Encoder_Stop_IT()*
- *HAL_LPTIM_TimeOut_Start()*
- *HAL_LPTIM_TimeOut_Stop()*
- *HAL_LPTIM_TimeOut_Start_IT()*
- *HAL_LPTIM_TimeOut_Stop_IT()*
- *HAL_LPTIM_Counter_Start()*
- *HAL_LPTIM_Counter_Stop()*
- *HAL_LPTIM_Counter_Start_IT()*
- *HAL_LPTIM_Counter_Stop_IT()*

49.2.4 LPTIM Read operation functions

This section provides LPTIM Reading functions.

- Read the counter value.
- Read the period (Auto-reload) value.
- Read the pulse (Compare)value.

This section contains the following APIs:

- *HAL_LPTIM_ReadCounter()*
- *HAL_LPTIM_ReadAutoReload()*
- *HAL_LPTIM_ReadCompare()*

49.2.5 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral.

This section contains the following APIs:

- *HAL_LPTIM_GetState()*

49.2.6 Detailed description of functions

HAL_LPTIM_Init

Function name

HAL_StatusTypeDef HAL_LPTIM_Init (LPTIM_HandleTypeDef * hlptim)

Function description

Initialize the LPTIM according to the specified parameters in the LPTIM_InitTypeDef and initialize the associated handle.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_DeInit

Function name

HAL_StatusTypeDef HAL_LPTIM_DeInit (LPTIM_HandleTypeDef * hlptim)

Function description

DeInitialize the LPTIM peripheral.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_MspInit

Function name

void HAL_LPTIM_MspInit (LPTIM_HandleTypeDef * hlptim)

Function description

Initialize the LPTIM MSP.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_MspDeInit

Function name

void HAL_LPTIM_MspDeInit (LPTIM_HandleTypeDef * hlptim)

Function description

DeInitialize LPTIM MSP.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_PWM_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)

Function description

Start the LPTIM PWM generation.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0001 and 0xFFFF.
- **Pulse:** Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_PWM_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Stop (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the LPTIM PWM generation.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_PWM_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)

Function description

Start the LPTIM PWM generation in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0001 and 0xFFFF
- **Pulse:** Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF

Return values

- **HAL:** status

HAL_LPTIM_PWM_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Stop_IT (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the LPTIM PWM generation in interrupt mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_OnePulse_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_OnePulse_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)

Function description

Start the LPTIM One pulse generation.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0001 and 0xFFFF.
- **Pulse:** Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_OnePulse_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_OnePulse_Stop (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the LPTIM One pulse generation.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_OnePulse_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_OnePulse_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)

Function description

Start the LPTIM One pulse generation in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0001 and 0xFFFF.
- **Pulse:** Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_OnePulse_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_OnePulse_Stop_IT (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the LPTIM One pulse generation in interrupt mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_SetOnce_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_SetOnce_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)

Function description

Start the LPTIM in Set once mode.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0001 and 0xFFFF.
- **Pulse:** Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_SetOnce_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_SetOnce_Stop (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the LPTIM Set once mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_SetOnce_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_SetOnce_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)

Function description

Start the LPTIM Set once mode in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
- **Pulse:** Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_SetOnce_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_SetOnce_Stop_IT (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the LPTIM Set once mode in interrupt mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_Encoder_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_Encoder_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period)

Function description

Start the Encoder interface.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0001 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_Encoder_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_Encoder_Stop (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Encoder interface.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_Encoder_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_Encoder_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period)

Function description

Start the Encoder interface in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_Encoder_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_Encoder_Stop_IT (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Encoder interface in interrupt mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_TimeOut_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_TimeOut_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Timeout)

Function description

Start the Timeout function.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0001 and 0xFFFF.
- **Timeout:** Specifies the TimeOut value to reset the counter. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

Notes

- The first trigger event will start the timer, any successive trigger event will reset the counter and the timer restarts.

HAL_LPTIM_TimeOut_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_TimeOut_Stop (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Timeout function.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_TimeOut_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_TimeOut_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Timeout)

Function description

Start the Timeout function in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0001 and 0xFFFF.
- **Timeout:** Specifies the TimeOut value to reset the counter. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

Notes

- The first trigger event will start the timer, any successive trigger event will reset the counter and the timer restarts.

HAL_LPTIM_TimeOut_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_TimeOut_Stop_IT (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Timeout function in interrupt mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_Counter_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_Counter_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period)

Function description

Start the Counter mode.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0001 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_Counter_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_Counter_Stop (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Counter mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_Counter_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_Counter_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period)

Function description

Start the Counter mode in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Period:** Specifies the Autoreload value. This parameter must be a value between 0x0001 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_Counter_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_Counter_Stop_IT (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Counter mode in interrupt mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_ReadCounter

Function name

uint32_t HAL_LPTIM_ReadCounter (const LPTIM_HandleTypeDef * hlptim)

Function description

Return the current counter value.

Parameters

- **hlptim:** LPTIM handle

Return values

- **Counter:** value.

HAL_LPTIM_ReadAutoReload

Function name

uint32_t HAL_LPTIM_ReadAutoReload (const LPTIM_HandleTypeDef * hlptim)

Function description

Return the current Autoreload (Period) value.

Parameters

- **hlptim:** LPTIM handle

Return values

- **Autoreload:** value.

HAL_LPTIM_ReadCompare

Function name

uint32_t HAL_LPTIM_ReadCompare (const LPTIM_HandleTypeDef * hlptim)

Function description

Return the current Compare (Pulse) value.

Parameters

- **hlptim:** LPTIM handle

Return values

- **Compare:** value.

HAL_LPTIM_IRQHandler

Function name

void HAL_LPTIM_IRQHandler (LPTIM_HandleTypeDef * hlptim)

Function description

Handle LPTIM interrupt request.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_CompareMatchCallback

Function name

void HAL_LPTIM_CompareMatchCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Compare match callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_AutoReloadMatchCallback

Function name

void HAL_LPTIM_AutoReloadMatchCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Autoreload match callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_TriggerCallback

Function name

void HAL_LPTIM_TriggerCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Trigger detected callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_CompareWriteCallback

Function name

void HAL_LPTIM_CompareWriteCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Compare write callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_AutoReloadWriteCallback

Function name

void HAL_LPTIM_AutoReloadWriteCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Autoreload write callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_DirectionUpCallback

Function name

void HAL_LPTIM_DirectionUpCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Direction counter changed from Down to Up callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_DirectionDownCallback

Function name

void HAL_LPTIM_DirectionDownCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Direction counter changed from Up to Down callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_RegisterCallback

Function name

HAL_StatusTypeDef HAL_LPTIM_RegisterCallback (LPTIM_HandleTypeDef * hlptim, HAL_LPTIM_CallbackIDTypeDef CallbackID, pLPTIM_CallbackTypeDef pCallback)

Function description

Register a User LPTIM callback to be used instead of the weak predefined callback.

Parameters

- **hlptim:** LPTIM handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_LPTIM_MSPINIT_CB_ID LPTIM Base Msp Init Callback ID
  - HAL_LPTIM_MSPDEINIT_CB_ID LPTIM Base Msp DeInit Callback ID
  - HAL_LPTIM_COMPARE_MATCH_CB_ID Compare match Callback ID
  - HAL_LPTIM_AUTORELOAD_MATCH_CB_ID Auto-reload match Callback ID
  - HAL_LPTIM_TRIGGER_CB_ID External trigger event detection Callback ID
  - HAL_LPTIM_COMPARE_WRITE_CB_ID Compare register write complete Callback ID
  - HAL_LPTIM_AUTORELOAD_WRITE_CB_ID Auto-reload register write complete Callback ID
  - HAL_LPTIM_DIRECTION_UP_CB_ID Up-counting direction change Callback ID
  - HAL_LPTIM_DIRECTION_DOWN_CB_ID Down-counting direction change Callback ID
- **pCallback:** pointer to the callback function

Return values

- **status:**

HAL_LPTIM_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_LPTIM_UnRegisterCallback (LPTIM_HandleTypeDef * hlptim, HAL_LPTIM_CallbackIDTypeDef CallbackID)

Function description

Unregister a LPTIM callback LLPTIM callback is redirected to the weak predefined callback.

Parameters

- **hlptim:** LPTIM handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_LPTIM_MSPINIT_CB_ID LPTIM Base Msp Init Callback ID
  - HAL_LPTIM_MSPDEINIT_CB_ID LPTIM Base Msp DeInit Callback ID
  - HAL_LPTIM_COMPARE_MATCH_CB_ID Compare match Callback ID
  - HAL_LPTIM_AUTORELOAD_MATCH_CB_ID Auto-reload match Callback ID
  - HAL_LPTIM_TRIGGER_CB_ID External trigger event detection Callback ID
  - HAL_LPTIM_COMPARE_WRITE_CB_ID Compare register write complete Callback ID
  - HAL_LPTIM_AUTORELOAD_WRITE_CB_ID Auto-reload register write complete Callback ID
  - HAL_LPTIM_DIRECTION_UP_CB_ID Up-counting direction change Callback ID
  - HAL_LPTIM_DIRECTION_DOWN_CB_ID Down-counting direction change Callback ID

Return values

- **status:**

HAL_LPTIM_GetState

Function name

HAL_LPTIM_StateTypeDef HAL_LPTIM_GetState (LPTIM_HandleTypeDef * hlptim)

Function description

Return the LPTIM handle state.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** state

LPTIM_Disable

Function name

void LPTIM_Disable (LPTIM_HandleTypeDef * hlptim)

Function description

Disable LPTIM HW instance.

Parameters

- **hlptim:** pointer to a LPTIM_HandleTypeDef structure that contains the configuration information for LPTIM module.

Return values

- **None:**

Notes

- The following sequence is required to solve LPTIM disable HW limitation. Please check Errata Sheet ES0335 for more details under "MCU may remain stuck in LPTIM interrupt when entering Stop mode" section.

49.3 LPTIM Firmware driver defines

The following section lists the various define and macros of the module.

49.3.1 LPTIM

LPTIM

LPTIM Clock Polarity

LPTIM_CLOCKPOLARITY_RISING

LPTIM_CLOCKPOLARITY_FALLING

LPTIM_CLOCKPOLARITY_RISING_FALLING

LPTIM Clock Prescaler

LPTIM_PRESCALER_DIV1

LPTIM_PRESCALER_DIV2

LPTIM_PRESCALER_DIV4

LPTIM_PRESCALER_DIV8

LPTIM_PRESCALER_DIV16

LPTIM_PRESCALER_DIV32

LPTIM_PRESCALER_DIV64

LPTIM_PRESCALER_DIV128

LPTIM Clock Sample Time

LPTIM_CLOCKSAMPLETIME_DIRECTTRANSITION

LPTIM_CLOCKSAMPLETIME_2TRANSITIONS

LPTIM_CLOCKSAMPLETIME_4TRANSITIONS

LPTIM_CLOCKSAMPLETIME_8TRANSITIONS

LPTIM Clock Source

LPTIM_CLOCKSOURCE_APBCLOCK_LPOSC

LPTIM_CLOCKSOURCE_ULPTIM

LPTIM Counter Source

LPTIM_COUNTERSOURCE_INTERNAL

LPTIM_COUNTERSOURCE_EXTERNAL

LPTIM Exported Macros

__HAL_LPTIM_RESET_HANDLE_STATE

Description:

- Reset LPTIM handle state.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_ENABLE

Description:

- Enable the LPTIM peripheral.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_DISABLE

Description:

- Disable the LPTIM peripheral.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

Notes:

- The following sequence is required to solve LPTIM disable HW limitation. Please check Errata Sheet ES0335 for more details under "MCU may remain stuck in LPTIM interrupt when entering Stop mode" section. Please call HAL_LPTIM_GetState() after a call to __HAL_LPTIM_DISABLE to check for TIMEOUT.

__HAL_LPTIM_START_CONTINUOUS

Description:

- Start the LPTIM peripheral in Continuous mode.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_START_SINGLE

Description:

- Start the LPTIM peripheral in single mode.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_RESET_COUNTER

Description:

- Reset the LPTIM Counter register in synchronous mode.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_RESET_COUNTER_AFTERREAD

Description:

- Reset after read of the LPTIM Counter register in asynchronous mode.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_AUTORELOAD_SET

Description:

- Write the passed parameter in the Autoreload register.

Parameters:

- __HANDLE__: LPTIM handle
- __VALUE__: Autoreload value

Return value:

- None

Notes:

- The ARR register can only be modified when the LPTIM instance is enabled.

__HAL_LPTIM_COMPARE_SET

Description:

- Write the passed parameter in the Compare register.

Parameters:

- __HANDLE__: LPTIM handle
- __VALUE__: Compare value

Return value:

- None

Notes:

- The CMP register can only be modified when the LPTIM instance is enabled.

__HAL_LPTIM_GET_FLAG

Description:

- Check whether the specified LPTIM flag is set or not.

Parameters:

- __HANDLE__: LPTIM handle
- __FLAG__: LPTIM flag to check This parameter can be a value of:
  - LPTIM_FLAG_DOWN : Counter direction change up Flag.
  - LPTIM_FLAG_UP : Counter direction change down to up Flag.
  - LPTIM_FLAG_ARROK : Autoreload register update OK Flag.
  - LPTIM_FLAG_CMPOK : Compare register update OK Flag.
  - LPTIM_FLAG_EXTTRIG : External trigger edge event Flag.
  - LPTIM_FLAG_ARRM : Autoreload match Flag.
  - LPTIM_FLAG_CMPM : Compare match Flag.

Return value:

- The: state of the specified flag (SET or RESET).

__HAL_LPTIM_CLEAR_FLAG

Description:

- Clear the specified LPTIM flag.

Parameters:

- __HANDLE__: LPTIM handle.
- __FLAG__: LPTIM flag to clear. This parameter can be a value of:
  - LPTIM_FLAG_DOWN : Counter direction change up Flag.
  - LPTIM_FLAG_UP : Counter direction change down to up Flag.
  - LPTIM_FLAG_ARROK : Autoreload register update OK Flag.
  - LPTIM_FLAG_CMPOK : Compare register update OK Flag.
  - LPTIM_FLAG_EXTTRIG : External trigger edge event Flag.
  - LPTIM_FLAG_ARRM : Autoreload match Flag.
  - LPTIM_FLAG_CMPM : Compare match Flag.

Return value:

- None.

__HAL_LPTIM_ENABLE_IT

Description:

- Enable the specified LPTIM interrupt.

Parameters:

- __HANDLE__: LPTIM handle.
- __INTERRUPT__: LPTIM interrupt to set. This parameter can be a value of:
  - LPTIM_IT_DOWN : Counter direction change up Interrupt.
  - LPTIM_IT_UP : Counter direction change down to up Interrupt.
  - LPTIM_IT_ARROK : Autoreload register update OK Interrupt.
  - LPTIM_IT_CMPOK : Compare register update OK Interrupt.
  - LPTIM_IT_EXTTRIG : External trigger edge event Interrupt.
  - LPTIM_IT_ARRM : Autoreload match Interrupt.
  - LPTIM_IT_CMPM : Compare match Interrupt.

Return value:

- None.

Notes:

- The LPTIM interrupts can only be enabled when the LPTIM instance is disabled.

__HAL_LPTIM_DISABLE_IT

Description:

- Disable the specified LPTIM interrupt.

Parameters:

- __HANDLE__: LPTIM handle.
- __INTERRUPT__: LPTIM interrupt to set. This parameter can be a value of:
  - LPTIM_IT_DOWN : Counter direction change up Interrupt.
  - LPTIM_IT_UP : Counter direction change down to up Interrupt.
  - LPTIM_IT_ARROK : Autoreload register update OK Interrupt.
  - LPTIM_IT_CMPOK : Compare register update OK Interrupt.
  - LPTIM_IT_EXTTRIG : External trigger edge event Interrupt.
  - LPTIM_IT_ARRM : Autoreload match Interrupt.
  - LPTIM_IT_CMPM : Compare match Interrupt.

Return value:

- None.

Notes:

- The LPTIM interrupts can only be disabled when the LPTIM instance is disabled.

__HAL_LPTIM_GET_IT_SOURCE

Description:

- Check whether the specified LPTIM interrupt source is enabled or not.

Parameters:

- __HANDLE__: LPTIM handle.
- __INTERRUPT__: LPTIM interrupt to check. This parameter can be a value of:
  - LPTIM_IT_DOWN : Counter direction change up Interrupt.
  - LPTIM_IT_UP : Counter direction change down to up Interrupt.
  - LPTIM_IT_ARROK : Autoreload register update OK Interrupt.
  - LPTIM_IT_CMPOK : Compare register update OK Interrupt.
  - LPTIM_IT_EXTTRIG : External trigger edge event Interrupt.
  - LPTIM_IT_ARRM : Autoreload match Interrupt.
  - LPTIM_IT_CMPM : Compare match Interrupt.

Return value:

- Interrupt: status.

LPTIM External Trigger Polarity

LPTIM_ACTIVEEDGE_RISING

LPTIM_ACTIVEEDGE_FALLING

LPTIM_ACTIVEEDGE_RISING_FALLING

LPTIM Flags Definition

LPTIM_FLAG_DOWN

LPTIM_FLAG_UP

LPTIM_FLAG_ARROK

LPTIM_FLAG_CMPOK

LPTIM_FLAG_EXTTRIG

LPTIM_FLAG_ARRM

LPTIM_FLAG_CMPM

LPTIM Input1 Source

LPTIM_INPUT1SOURCE_GPIO

For LPTIM1 and LPTIM2

LPTIM_INPUT1SOURCE_COMP1

For LPTIM1 and LPTIM2

LPTIM_INPUT1SOURCE_COMP2

For LPTIM2

LPTIM_INPUT1SOURCE_COMP1_COMP2

For LPTIM2

LPTIM_INPUT1SOURCE_NOT_CONNECTED

For LPTIM3

LPTIM_INPUT1SOURCE_SAI4_FSA

For LPTIM3

LPTIM_INPUT1SOURCE_SAI4_FSB

For LPTIM3

LPTIM Input2 Source

LPTIM_INPUT2SOURCE_GPIO

For LPTIM1

LPTIM_INPUT2SOURCE_COMP2

For LPTIM1

LPTIM Interrupts Definition

LPTIM_IT_DOWN

LPTIM_IT_UP

LPTIM_IT_ARROK

LPTIM_IT_CMPOK

LPTIM_IT_EXTTRIG

LPTIM_IT_ARRM

LPTIM_IT_CMPM

LPTIM Output Polarity

LPTIM_OUTPUTPOLARITY_HIGH

LPTIM_OUTPUTPOLARITY_LOW

LPTIM Trigger Sample Time

LPTIM_TRIGSAMPLETIME_DIRECTTRANSITION

LPTIM_TRIGSAMPLETIME_2TRANSITIONS

LPTIM_TRIGSAMPLETIME_4TRANSITIONS

LPTIM_TRIGSAMPLETIME_8TRANSITIONS

LPTIM Trigger Source

LPTIM_TRIGSOURCE_SOFTWARE

LPTIM_TRIGSOURCE_0

LPTIM_TRIGSOURCE_1

LPTIM_TRIGSOURCE_2

LPTIM_TRIGSOURCE_3

LPTIM_TRIGSOURCE_4

LPTIM_TRIGSOURCE_5

LPTIM_TRIGSOURCE_6

LPTIM_TRIGSOURCE_7

LPTIM Updating Mode

LPTIM_UPDATE_IMMEDIATE

LPTIM_UPDATE_ENDOFPERIOD

50 HAL LTDC Generic Driver

50.1 LTDC Firmware driver registers structures

50.1.1 LTDC_ColorTypeDef

LTDC_ColorTypeDef is defined in the stm32h7xx_hal_ltdc.h

Data Fields

- ***uint8_t Blue***
- ***uint8_t Green***
- ***uint8_t Red***
- ***uint8_t Reserved***

Field Documentation

- ***uint8_t LTDC_ColorTypeDef::Blue***
  Configures the blue value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- ***uint8_t LTDC_ColorTypeDef::Green***
  Configures the green value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- ***uint8_t LTDC_ColorTypeDef::Red***
  Configures the red value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- ***uint8_t LTDC_ColorTypeDef::Reserved***
  Reserved 0xFF

50.1.2 LTDC_InitTypeDef

LTDC_InitTypeDef is defined in the stm32h7xx_hal_ltdc.h

Data Fields

- ***uint32_t HSPolarity***
- ***uint32_t VSPolarity***
- ***uint32_t DEPolarity***
- ***uint32_t PCPolarity***
- ***uint32_t HorizontalSync***
- ***uint32_t VerticalSync***
- ***uint32_t AccumulatedHBP***
- ***uint32_t AccumulatedVBP***
- ***uint32_t AccumulatedActiveW***
- ***uint32_t AccumulatedActiveH***
- ***uint32_t TotalWidth***
- ***uint32_t TotalHeigh***
- ***LTDC_ColorTypeDef Backcolor***

Field Documentation

- ***uint32_t LTDC_InitTypeDef::HSPolarity***
  configures the horizontal synchronization polarity. This parameter can be one value of ***LTDC_HS_POLARITY***
- ***uint32_t LTDC_InitTypeDef::VSPolarity***
  configures the vertical synchronization polarity. This parameter can be one value of ***LTDC_VS_POLARITY***

- ***uint32_t LTDC_InitTypeDef::DEPolarity***
  configures the data enable polarity. This parameter can be one of value of ***LTDC_DE_POLARITY***
- ***uint32_t LTDC_InitTypeDef::PCPolarity***
  configures the pixel clock polarity. This parameter can be one of value of ***LTDC_PC_POLARITY***
- ***uint32_t LTDC_InitTypeDef::HorizontalSync***
  configures the number of Horizontal synchronization width. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF.
- ***uint32_t LTDC_InitTypeDef::VerticalSync***
  configures the number of Vertical synchronization height. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0x7FF.
- ***uint32_t LTDC_InitTypeDef::AccumulatedHBP***
  configures the accumulated horizontal back porch width. This parameter must be a number between Min_Data = LTDC_HorizontalSync and Max_Data = 0xFFF.
- ***uint32_t LTDC_InitTypeDef::AccumulatedVBP***
  configures the accumulated vertical back porch height. This parameter must be a number between Min_Data = LTDC_VerticalSync and Max_Data = 0x7FF.
- ***uint32_t LTDC_InitTypeDef::AccumulatedActiveW***
  configures the accumulated active width. This parameter must be a number between Min_Data = LTDC_AccumulatedHBP and Max_Data = 0xFFF.
- ***uint32_t LTDC_InitTypeDef::AccumulatedActiveH***
  configures the accumulated active height. This parameter must be a number between Min_Data = LTDC_AccumulatedVBP and Max_Data = 0x7FF.
- ***uint32_t LTDC_InitTypeDef::TotalWidth***
  configures the total width. This parameter must be a number between Min_Data = LTDC_AccumulatedActiveW and Max_Data = 0xFFF.
- ***uint32_t LTDC_InitTypeDef::TotalHeigh***
  configures the total height. This parameter must be a number between Min_Data = LTDC_AccumulatedActiveH and Max_Data = 0x7FF.
- ***LTDC_ColorTypeDef LTDC_InitTypeDef::Backcolor***
  Configures the background color.

50.1.3 LTDC_LayerCfgTypeDef

LTDC_LayerCfgTypeDef is defined in the stm32h7xx_hal_ltdc.h

Data Fields

- ***uint32_t WindowX0***
- ***uint32_t WindowX1***
- ***uint32_t WindowY0***
- ***uint32_t WindowY1***
- ***uint32_t PixelFormat***
- ***uint32_t Alpha***
- ***uint32_t Alpha0***
- ***uint32_t BlendingFactor1***
- ***uint32_t BlendingFactor2***
- ***uint32_t FBStartAdress***
- ***uint32_t ImageWidth***
- ***uint32_t ImageHeight***
- ***LTDC_ColorTypeDef Backcolor***

Field Documentation

- ***uint32_t LTDC_LayerCfgTypeDef::WindowX0***
  Configures the Window Horizontal Start Position. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF.

- ***uint32_t LTDC_LayerCfgTypeDef::WindowX1***
  Configures the Window Horizontal Stop Position. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF.
- ***uint32_t LTDC_LayerCfgTypeDef::WindowY0***
  Configures the Window vertical Start Position. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0x7FF.
- ***uint32_t LTDC_LayerCfgTypeDef::WindowY1***
  Configures the Window vertical Stop Position. This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x7FF.
- ***uint32_t LTDC_LayerCfgTypeDef::PixelFormat***
  Specifies the pixel format. This parameter can be one of value of ***LTDC_Pixelformat***
- ***uint32_t LTDC_LayerCfgTypeDef::Alpha***
  Specifies the constant alpha used for blending. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- ***uint32_t LTDC_LayerCfgTypeDef::Alpha0***
  Configures the default alpha value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- ***uint32_t LTDC_LayerCfgTypeDef::BlendingFactor1***
  Select the blending factor 1. This parameter can be one of value of ***LTDC_BlendingFactor1***
- ***uint32_t LTDC_LayerCfgTypeDef::BlendingFactor2***
  Select the blending factor 2. This parameter can be one of value of ***LTDC_BlendingFactor2***
- ***uint32_t LTDC_LayerCfgTypeDef::FBStartAdress***
  Configures the color frame buffer address
- ***uint32_t LTDC_LayerCfgTypeDef::ImageWidth***
  Configures the color frame buffer line length. This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x1FFF.
- ***uint32_t LTDC_LayerCfgTypeDef::ImageHeight***
  Specifies the number of line in frame buffer. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0x7FF.
- ***LTDC_ColorTypeDef LTDC_LayerCfgTypeDef::Backcolor***
  Configures the layer background color.

50.1.4 __LTDC_HandleTypeDef

__LTDC_HandleTypeDef is defined in the stm32h7xx_hal_ltdc.h

Data Fields

- ***LTDC_TypeDef * Instance***
- ***LTDC_InitTypeDef Init***
- ***LTDC_LayerCfgTypeDef LayerCfg***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_LTDC_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* LineEventCallback***
- ***void(* ReloadEventCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***LTDC_TypeDef* __LTDC_HandleTypeDef::Instance***
  LTDC Register base address
- ***LTDC_InitTypeDef __LTDC_HandleTypeDef::Init***
  LTDC parameters

- ***LTDC_LayerCfgTypeDef __LTDC_HandleTypeDef::LayerCfg[MAX_LAYER]***
  LTDC Layers parameters
- ***HAL_LockTypeDef __LTDC_HandleTypeDef::Lock***
  LTDC Lock
- ***__IO HAL_LTDC_StateTypeDef __LTDC_HandleTypeDef::State***
  LTDC state
- ***__IO uint32_t __LTDC_HandleTypeDef::ErrorCode***
  LTDC Error code
- ***void(* __LTDC_HandleTypeDef::LineEventCallback)(struct __LTDC_HandleTypeDef *hltdc)***
  LTDC Line Event Callback
- ***void(* __LTDC_HandleTypeDef::ReloadEventCallback)(struct __LTDC_HandleTypeDef *hltdc)***
  LTDC Reload Event Callback
- ***void(* __LTDC_HandleTypeDef::ErrorCallback)(struct __LTDC_HandleTypeDef *hltdc)***
  LTDC Error Callback
- ***void(* __LTDC_HandleTypeDef::MspInitCallback)(struct __LTDC_HandleTypeDef *hltdc)***
  LTDC Msp Init callback
- ***void(* __LTDC_HandleTypeDef::MspDeInitCallback)(struct __LTDC_HandleTypeDef *hltdc)***
  LTDC Msp DeInit callback

50.2 LTDC Firmware driver API description

The following section lists the various functions of the LTDC library.

50.2.1 How to use this driver

The LTDC HAL driver can be used as follows:

1. Declare a LTDC_HandleTypeDef handle structure, for example: LTDC_HandleTypeDef hltdc;
2. Initialize the LTDC low level resources by implementing the HAL_LTDC_MspInit() API:
   a. Enable the LTDC interface clock
   b. NVIC configuration if you need to use interrupt process
      - Configure the LTDC interrupt priority
      - Enable the NVIC LTDC IRQ Channel
3. Initialize the required configuration through the following parameters: the LTDC timing, the horizontal and vertical polarity, the pixel clock polarity, Data Enable polarity and the LTDC background color value using HAL_LTDC_Init() function

Configuration

1. Program the required configuration through the following parameters: the pixel format, the blending factors, input alpha value, the window size and the image size using HAL_LTDC_ConfigLayer() function for foreground or/and background layer.
2. Optionally, configure and enable the CLUT using HAL_LTDC_ConfigCLUT() and HAL_LTDC_EnableCLUT functions.
3. Optionally, enable the Dither using HAL_LTDC_EnableDither().
4. Optionally, configure and enable the Color keying using HAL_LTDC_ConfigColorKeying() and HAL_LTDC_EnableColorKeying functions.
5. Optionally, configure LineInterrupt using HAL_LTDC_ProgramLineEvent() function
6. If needed, reconfigure and change the pixel format value, the alpha value value, the window size, the window position and the layer start address for foreground or/and background layer using respectively the following functions: HAL_LTDC_SetPixelFormat(), HAL_LTDC_SetAlpha(), HAL_LTDC_SetWindowSize(), HAL_LTDC_SetWindowPosition() and HAL_LTDC_SetAddress().

7. Variant functions with _NoReload suffix allows to set the LTDC configuration/settings without immediate reload. This is useful in case when the program requires to modify serval LTDC settings (on one or both layers) then applying(reload) these settings in one shot by calling the function HAL_LTDC_Reload(). After calling the _NoReload functions to set different color/format/layer settings, the program shall call the function HAL_LTDC_Reload() to apply(reload) these settings. Function HAL_LTDC_Reload() can be called with the parameter ReloadType set to LTDC_RELOAD_IMMEDIATE if an immediate reload is required. Function HAL_LTDC_Reload() can be called with the parameter ReloadType set to LTDC_RELOAD_VERTICAL_BLANKING if the reload should be done in the next vertical blanking period, this option allows to avoid display flicker by applying the new settings during the vertical blanking period.
8. To control LTDC state you can use the following function: HAL_LTDC_GetState()

LTDC HAL driver macros list

Below the list of most used macros in LTDC HAL driver.

- __HAL_LTDC_ENABLE: Enable the LTDC.
- __HAL_LTDC_DISABLE: Disable the LTDC.
- __HAL_LTDC_LAYER_ENABLE: Enable an LTDC Layer.
- __HAL_LTDC_LAYER_DISABLE: Disable an LTDC Layer.
- __HAL_LTDC_RELOAD_IMMEDIATE_CONFIG: Reload Layer Configuration.
- __HAL_LTDC_GET_FLAG: Get the LTDC pending flags.
- __HAL_LTDC_CLEAR_FLAG: Clear the LTDC pending flags.
- __HAL_LTDC_ENABLE_IT: Enable the specified LTDC interrupts.
- __HAL_LTDC_DISABLE_IT: Disable the specified LTDC interrupts.
- __HAL_LTDC_GET_IT_SOURCE: Check whether the specified LTDC interrupt has occurred or not.

Note: You can refer to the LTDC HAL driver header file for more useful macros

Callback registration

The compilation define USE_HAL_LTDC_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use function HAL_LTDC_RegisterCallback() to register a callback.

Function HAL_LTDC_RegisterCallback() allows to register following callbacks:

- LineEventCallback : LTDC Line Event Callback.
- ReloadEventCallback : LTDC Reload Event Callback.
- ErrorCallback : LTDC Error Callback
- MspInitCallback : LTDC MspInit.
- MspDeInitCallback : LTDC MspDeInit.

This function takes as parameters the HAL peripheral handle, the callback ID and a pointer to the user callback function.

Use function HAL_LTDC_UnRegisterCallback() to reset a callback to the default weak function. HAL_LTDC_UnRegisterCallback() takes as parameters the HAL peripheral handle and the callback ID.

This function allows to reset following callbacks:

- LineEventCallback : LTDC Line Event Callback
- ReloadEventCallback : LTDC Reload Event Callback
- ErrorCallback : LTDC Error Callback
- MspInitCallback : LTDC MspInit
- MspDeInitCallback : LTDC MspDeInit.

By default, after the HAL_LTDC_Init and when the state is HAL_LTDC_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_LTDC_LineEventCallback(), HAL_LTDC_ErrorCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak (surcharged) functions in the HAL_LTDC_Init() and HAL_LTDC_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_LTDC_Init() and HAL_LTDC_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_LTDC_STATE_READY state only. Exception done MspInit/MspDeInit that can be registered/unregistered in HAL_LTDC_STATE_READY or HAL_LTDC_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_LTDC_RegisterCallback() before calling HAL_LTDC_DeInit() or HAL_LTDC_Init() function.

When the compilation define USE_HAL_LTDC_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

50.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the LTDC
- De-initialize the LTDC

This section contains the following APIs:

- ***HAL_LTDC_Init()***
- ***HAL_LTDC_DeInit()***
- ***HAL_LTDC_MspInit()***
- ***HAL_LTDC_MspDeInit()***
- ***HAL_LTDC_RegisterCallback()***
- ***HAL_LTDC_UnRegisterCallback()***
- ***HAL_LTDC_ErrorCallback()***
- ***HAL_LTDC_LineEventCallback()***
- ***HAL_LTDC_ReloadEventCallback()***

50.2.3 IO operation functions

This section provides function allowing to:

- Handle LTDC interrupt request

This section contains the following APIs:

- ***HAL_LTDC_IRQHandler()***
- ***HAL_LTDC_ErrorCallback()***
- ***HAL_LTDC_LineEventCallback()***
- ***HAL_LTDC_ReloadEventCallback()***

50.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure the LTDC foreground or/and background parameters.
- Set the active layer.
- Configure the color keying.
- Configure the C-LUT.
- Enable / Disable the color keying.
- Enable / Disable the C-LUT.
- Update the layer position.
- Update the layer size.
- Update pixel format on the fly.
- Update transparency on the fly.
- Update address on the fly.

This section contains the following APIs:

- ***HAL_LTDC_ConfigLayer()***
- ***HAL_LTDC_ConfigColorKeying()***
- ***HAL_LTDC_ConfigCLUT()***

- *HAL_LTDC_EnableColorKeying()*
- *HAL_LTDC_DisableColorKeying()*
- *HAL_LTDC_EnableCLUT()*
- *HAL_LTDC_DisableCLUT()*
- *HAL_LTDC_EnableDither()*
- *HAL_LTDC_DisableDither()*
- *HAL_LTDC_SetWindowSize()*
- *HAL_LTDC_SetWindowPosition()*
- *HAL_LTDC_SetPixelFormat()*
- *HAL_LTDC_SetAlpha()*
- *HAL_LTDC_SetAddress()*
- *HAL_LTDC_SetPitch()*
- *HAL_LTDC_ProgramLineEvent()*
- *HAL_LTDC_Reload()*
- *HAL_LTDC_ConfigLayer_NoReload()*
- *HAL_LTDC_SetWindowSize_NoReload()*
- *HAL_LTDC_SetWindowPosition_NoReload()*
- *HAL_LTDC_SetPixelFormat_NoReload()*
- *HAL_LTDC_SetAlpha_NoReload()*
- *HAL_LTDC_SetAddress_NoReload()*
- *HAL_LTDC_SetPitch_NoReload()*
- *HAL_LTDC_ConfigColorKeying_NoReload()*
- *HAL_LTDC_EnableColorKeying_NoReload()*
- *HAL_LTDC_DisableColorKeying_NoReload()*
- *HAL_LTDC_EnableCLUT_NoReload()*
- *HAL_LTDC_DisableCLUT_NoReload()*

50.2.5 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the LTDC handle state.
- Get the LTDC handle error code.

This section contains the following APIs:

- *HAL_LTDC_GetState()*
- *HAL_LTDC_GetError()*

50.2.6 Detailed description of functions

HAL_LTDC_Init

Function name

HAL_StatusTypeDef HAL_LTDC_Init (LTDC_HandleTypeDef * hltdc)

Function description

Initialize the LTDC according to the specified parameters in the LTDC_InitTypeDef.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **HAL:** status

HAL_LTDC_DeInit

Function name

HAL_StatusTypeDef HAL_LTDC_DeInit (LTDC_HandleTypeDef * hltdc)

Function description

De-initialize the LTDC peripheral.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **None:**

HAL_LTDC_MspInit

Function name

void HAL_LTDC_MspInit (LTDC_HandleTypeDef * hltdc)

Function description

Initialize the LTDC MSP.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **None:**

HAL_LTDC_MspDeInit

Function name

void HAL_LTDC_MspDeInit (LTDC_HandleTypeDef * hltdc)

Function description

De-initialize the LTDC MSP.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **None:**

HAL_LTDC_ErrorCallback

Function name

void HAL_LTDC_ErrorCallback (LTDC_HandleTypeDef * hltdc)

Function description

Error LTDC callback.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **None:**

HAL_LTDC_LineEventCallback

Function name

void HAL_LTDC_LineEventCallback (LTDC_HandleTypeDef * hltdc)

Function description

Line Event callback.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **None:**

HAL_LTDC_ReloadEventCallback

Function name

void HAL_LTDC_ReloadEventCallback (LTDC_HandleTypeDef * hltdc)

Function description

Reload Event callback.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **None:**

HAL_LTDC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_LTDC_RegisterCallback (LTDC_HandleTypeDef * hltdc, HAL_LTDC_CallbackIDTypeDef CallbackID, pLTDC_CallbackTypeDef pCallback)

Function description

Register a User LTDC Callback To be used instead of the weak predefined callback.

Parameters

- **hltdc:** ltdc handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_LTDC_LINE_EVENT_CB_ID Line Event Callback ID
  - HAL_LTDC_RELOAD_EVENT_CB_ID Reload Event Callback ID
  - HAL_LTDC_ERROR_CB_ID Error Callback ID
  - HAL_LTDC_MSPINIT_CB_ID MspInit callback ID
  - HAL_LTDC_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **status:**

HAL_LTDC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_LTDC_UnRegisterCallback (LTDC_HandleTypeDef * hltdc, HAL_LTDC_CallbackIDTypeDef CallbackID)

Function description

Unregister an LTDC Callback LTDC callback is redirected to the weak predefined callback.

Parameters

- **hltdc:** ltdc handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_LTDC_LINE_EVENT_CB_ID Line Event Callback ID
  - HAL_LTDC_RELOAD_EVENT_CB_ID Reload Event Callback ID
  - HAL_LTDC_ERROR_CB_ID Error Callback ID
  - HAL_LTDC_MSPINIT_CB_ID MspInit callback ID
  - HAL_LTDC_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **status:**

HAL_LTDC_IRQHandler

Function name

void HAL_LTDC_IRQHandler (LTDC_HandleTypeDef * hltdc)

Function description

Handle LTDC interrupt request.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **HAL:** status

HAL_LTDC_ConfigLayer

Function name

HAL_StatusTypeDef HAL_LTDC_ConfigLayer (LTDC_HandleTypeDef * hltdc, LTDC_LayerCfgTypeDef * pLayerCfg, uint32_t LayerIdx)

Function description

Configure the LTDC Layer according to the specified parameters in the LTDC_InitTypeDef and create the associated handle.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **pLayerCfg:** pointer to a LTDC_LayerCfgTypeDef structure that contains the configuration information for the Layer.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_SetWindowSize

Function name

HAL_StatusTypeDef HAL_LTDC_SetWindowSize (LTDC_HandleTypeDef * hltdc, uint32_t XSize, uint32_t YSize, uint32_t LayerIdx)

Function description

Set the LTDC window size.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **XSize:** LTDC Pixel per line
- **YSize:** LTDC Line number
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_SetWindowPosition

Function name

HAL_StatusTypeDef HAL_LTDC_SetWindowPosition (LTDC_HandleTypeDef * hltdc, uint32_t X0, uint32_t Y0, uint32_t LayerIdx)

Function description

Set the LTDC window position.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **X0:** LTDC window X offset
- **Y0:** LTDC window Y offset
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_SetPixelFormat

Function name

HAL_StatusTypeDef HAL_LTDC_SetPixelFormat (LTDC_HandleTypeDef * hltdc, uint32_t Pixelformat, uint32_t LayerIdx)

Function description

Reconfigure the pixel format.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **Pixelformat:** new pixel format value.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1).

Return values

- **HAL:** status

HAL_LTDC_SetAlpha

Function name

HAL_StatusTypeDef HAL_LTDC_SetAlpha (LTDC_HandleTypeDef * hltdc, uint32_t Alpha, uint32_t LayerIdx)

Function description

Reconfigure the layer alpha value.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **Alpha:** new alpha value.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_SetAddress

Function name

HAL_StatusTypeDef HAL_LTDC_SetAddress (LTDC_HandleTypeDef * hltdc, uint32_t Address, uint32_t LayerIdx)

Function description

Reconfigure the frame buffer Address.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **Address:** new address value.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1).

Return values

- **HAL:** status

HAL_LTDC_SetPitch

Function name

HAL_StatusTypeDef HAL_LTDC_SetPitch (LTDC_HandleTypeDef * hltdc, uint32_t LinePitchInPixels, uint32_t LayerIdx)

Function description

Function used to reconfigure the pitch for specific cases where the attached LayerIdx buffer have a width that is larger than the one intended to be displayed on screen.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LinePitchInPixels:** New line pitch in pixels to configure for LTDC layer 'LayerIdx'.
- **LayerIdx:** LTDC layer index concerned by the modification of line pitch.

Return values

- **HAL:** status

Notes

- This function should be called only after a previous call to HAL_LTDC_ConfigLayer() to modify the default pitch configured by HAL_LTDC_ConfigLayer() when required (refer to example described just above).

HAL_LTDC_ConfigColorKeying

Function name

HAL_StatusTypeDef HAL_LTDC_ConfigColorKeying (LTDC_HandleTypeDef * hltdc, uint32_t RGBValue, uint32_t LayerIdx)

Function description

Configure the color keying.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **RGBValue:** the color key value
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_ConfigCLUT

Function name

HAL_StatusTypeDef HAL_LTDC_ConfigCLUT (LTDC_HandleTypeDef * hltdc, uint32_t * pCLUT, uint32_t CLUTSize, uint32_t LayerIdx)

Function description

Load the color lookup table.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **pCLUT:** pointer to the color lookup table address.
- **CLUTSize:** the color lookup table size.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_EnableColorKeying

Function name

HAL_StatusTypeDef HAL_LTDC_EnableColorKeying (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)

Function description

Enable the color keying.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_DisableColorKeying

Function name

HAL_StatusTypeDef HAL_LTDC_DisableColorKeying (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)

Function description

Disable the color keying.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_EnableCLUT

Function name

HAL_StatusTypeDef HAL_LTDC_EnableCLUT (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)

Function description

Enable the color lookup table.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_DisableCLUT

Function name

HAL_StatusTypeDef HAL_LTDC_DisableCLUT (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)

Function description

Disable the color lookup table.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_ProgramLineEvent

Function name

HAL_StatusTypeDef HAL_LTDC_ProgramLineEvent (LTDC_HandleTypeDef * hltdc, uint32_t Line)

Function description

Define the position of the line interrupt.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **Line:** Line Interrupt Position.

Return values

- **HAL:** status

Notes

- User application may resort to HAL_LTDC_LineEventCallback() at line interrupt generation.

HAL_LTDC_EnableDither

Function name

HAL_StatusTypeDef HAL_LTDC_EnableDither (LTDC_HandleTypeDef * hltdc)

Function description

Enable Dither.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **HAL:** status

HAL_LTDC_DisableDither

Function name

HAL_StatusTypeDef HAL_LTDC_DisableDither (LTDC_HandleTypeDef * hltdc)

Function description

Disable Dither.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **HAL:** status

HAL_LTDC_Reload

Function name

HAL_StatusTypeDef HAL_LTDC_Reload (LTDC_HandleTypeDef * hltdc, uint32_t ReloadType)

Function description

Reload LTDC Layers configuration.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **ReloadType:** This parameter can be one of the following values : LTDC_RELOAD_IMMEDIATE : Immediate Reload LTDC_RELOAD_VERTICAL_BLANKING : Reload in the next Vertical Blanking

Return values

- **HAL:** status

Notes

- User application may resort to HAL_LTDC_ReloadEventCallback() at reload interrupt generation.

HAL_LTDC_ConfigLayer_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_ConfigLayer_NoReload (LTDC_HandleTypeDef * hltdc, LTDC_LayerCfgTypeDef * pLayerCfg, uint32_t LayerIdx)

Function description

Configure the LTDC Layer according to the specified without reloading parameters in the LTDC_InitTypeDef and create the associated handle.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **pLayerCfg:** pointer to a LTDC_LayerCfgTypeDef structure that contains the configuration information for the Layer.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_SetWindowSize_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_SetWindowSize_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t XSize, uint32_t YSize, uint32_t LayerIdx)

Function description

Set the LTDC window size without reloading.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **XSize:** LTDC Pixel per line
- **YSize:** LTDC Line number
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_SetWindowPosition_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_SetWindowPosition_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t X0, uint32_t Y0, uint32_t LayerIdx)

Function description

Set the LTDC window position without reloading.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **X0:** LTDC window X offset
- **Y0:** LTDC window Y offset
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_SetPixelFormat_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_SetPixelFormat_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t Pixelformat, uint32_t LayerIdx)

Function description

Reconfigure the pixel format without reloading.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDfef structure that contains the configuration information for the LTDC.
- **Pixelformat:** new pixel format value.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1).

Return values

- **HAL:** status

HAL_LTDC_SetAlpha_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_SetAlpha_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t Alpha, uint32_t LayerIdx)

Function description

Reconfigure the layer alpha value without reloading.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **Alpha:** new alpha value.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_SetAddress_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_SetAddress_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t Address, uint32_t LayerIdx)

Function description

Reconfigure the frame buffer Address without reloading.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **Address:** new address value.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1).

Return values

- **HAL:** status

HAL_LTDC_SetPitch_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_SetPitch_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t LinePitchInPixels, uint32_t LayerIdx)

Function description

Function used to reconfigure the pitch for specific cases where the attached LayerIdx buffer have a width that is larger than the one intended to be displayed on screen.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LinePitchInPixels:** New line pitch in pixels to configure for LTDC layer 'LayerIdx'.
- **LayerIdx:** LTDC layer index concerned by the modification of line pitch.

Return values

- **HAL:** status

Notes

- This function should be called only after a previous call to HAL_LTDC_ConfigLayer() to modify the default pitch configured by HAL_LTDC_ConfigLayer() when required (refer to example described just above). Variant of the function HAL_LTDC_SetPitch without immediate reload.

HAL_LTDC_ConfigColorKeying_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_ConfigColorKeying_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t RGBValue, uint32_t LayerIdx)

Function description

Configure the color keying without reloading.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **RGBValue:** the color key value
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_EnableColorKeying_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_EnableColorKeying_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)

Function description

Enable the color keying without reloading.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_DisableColorKeying_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_DisableColorKeying_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)

Function description

Disable the color keying without reloading.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_EnableCLUT_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_EnableCLUT_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)

Function description

Enable the color lookup table without reloading.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_DisableCLUT_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_DisableCLUT_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)

Function description

Disable the color lookup table without reloading.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_GetState

Function name

HAL_LTDC_StateTypeDef HAL_LTDC_GetState (LTDC_HandleTypeDef * hltdc)

Function description

Return the LTDC handle state.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **HAL:** state

HAL_LTDC_GetError

Function name

uint32_t HAL_LTDC_GetError (LTDC_HandleTypeDef * hltdc)

Function description

Return the LTDC handle error code.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **LTDC:** Error Code

50.3 LTDC Firmware driver defines

The following section lists the various define and macros of the module.

50.3.1 LTDC

LTDC

LTDC Alpha

LTDC_ALPHA

LTDC Constant Alpha mask

LTDC BACK COLOR

LTDC_COLOR

Color mask

LTDC Blending Factor1

LTDC_BLENDING_FACTOR1_CA

Blending factor : Cte Alpha

LTDC_BLENDING_FACTOR1_PAxCA

Blending factor : Cte Alpha x Pixel Alpha

LTDC Blending Factor2

LTDC_BLENDING_FACTOR2_CA

Blending factor : Cte Alpha

LTDC_BLENDING_FACTOR2_PAxCA

Blending factor : Cte Alpha x Pixel Alpha

LTDC DE POLARITY

LTDC_DEPOLARITY_AL

Data Enable, is active low.

LTDC_DEPOLARITY_AH

Data Enable, is active high.

LTDC Error Code

HAL_LTDC_ERROR_NONE

LTDC No error

HAL_LTDC_ERROR_TE

LTDC Transfer error

HAL_LTDC_ERROR_FU

LTDC FIFO Underrun

HAL_LTDC_ERROR_TIMEOUT

LTDC Timeout error

HAL_LTDC_ERROR_INVALID_CALLBACK

LTDC Invalid Callback error

LTDC Exported Macros

__HAL_LTDC_RESET_HANDLE_STATE

Description:

- Reset LTDC handle state.

Parameters:

- __HANDLE__: LTDC handle

Return value:

- None

__HAL_LTDC_ENABLE

Description:

- Enable the LTDC.

Parameters:

- __HANDLE__: LTDC handle

Return value:

- None.

__HAL_LTDC_DISABLE

Description:

- Disable the LTDC.

Parameters:

- __HANDLE__: LTDC handle

Return value:

- None.

__HAL_LTDC_LAYER_ENABLE

Description:

- Enable the LTDC Layer.

Parameters:

- __HANDLE__: LTDC handle
- __LAYER__: Specify the layer to be enabled. This parameter can be LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1).

Return value:

- None.

__HAL_LTDC_LAYER_DISABLE

Description:

- Disable the LTDC Layer.

Parameters:

- __HANDLE__: LTDC handle
- __LAYER__: Specify the layer to be disabled. This parameter can be LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1).

Return value:

- None.

__HAL_LTDC_RELOAD_IMMEDIATE_CONFIG

Description:

- Reload immediately all LTDC Layers.

Parameters:

- __HANDLE__: LTDC handle

Return value:

- None.

__HAL_LTDC_VERTICAL_BLANKING_RELOAD_CONFIG

Description:

- Reload during vertical blanking period all LTDC Layers.

Parameters:

- __HANDLE__: LTDC handle

Return value:

- None.

__HAL_LTDC_GET_FLAG

Description:

- Get the LTDC pending flags.

Parameters:

- __HANDLE__: LTDC handle
- __FLAG__: Get the specified flag. This parameter can be any combination of the following values:
  - LTDC_FLAG_LI: Line Interrupt flag
  - LTDC_FLAG_FU: FIFO Underrun Interrupt flag
  - LTDC_FLAG_TE: Transfer Error interrupt flag
  - LTDC_FLAG_RR: Register Reload Interrupt Flag

Return value:

- The: state of FLAG (SET or RESET).

__HAL_LTDC_CLEAR_FLAG

Description:

- Clears the LTDC pending flags.

Parameters:

- __HANDLE__: LTDC handle
- __FLAG__: Specify the flag to clear. This parameter can be any combination of the following values:
  - LTDC_FLAG_LI: Line Interrupt flag
  - LTDC_FLAG_FU: FIFO Underrun Interrupt flag
  - LTDC_FLAG_TE: Transfer Error interrupt flag
  - LTDC_FLAG_RR: Register Reload Interrupt Flag

Return value:

- None

__HAL_LTDC_ENABLE_IT

Description:

- Enables the specified LTDC interrupts.

Parameters:

- __HANDLE__: LTDC handle
- __INTERRUPT__: Specify the LTDC interrupt sources to be enabled. This parameter can be any combination of the following values:
  - LTDC_IT_LI: Line Interrupt flag
  - LTDC_IT_FU: FIFO Underrun Interrupt flag
  - LTDC_IT_TE: Transfer Error interrupt flag
  - LTDC_IT_RR: Register Reload Interrupt Flag

Return value:

- None

__HAL_LTDC_DISABLE_IT

Description:

- Disables the specified LTDC interrupts.

Parameters:

- __HANDLE__: LTDC handle
- __INTERRUPT__: Specify the LTDC interrupt sources to be disabled. This parameter can be any combination of the following values:
  - LTDC_IT_LI: Line Interrupt flag
  - LTDC_IT_FU: FIFO Underrun Interrupt flag
  - LTDC_IT_TE: Transfer Error interrupt flag
  - LTDC_IT_RR: Register Reload Interrupt Flag

Return value:

- None

__HAL_LTDC_GET_IT_SOURCE

Description:

- Check whether the specified LTDC interrupt has occurred or not.

Parameters:

- __HANDLE__: LTDC handle
- __INTERRUPT__: Specify the LTDC interrupt source to check. This parameter can be one of the following values:
  - LTDC_IT_LI: Line Interrupt flag
  - LTDC_IT_FU: FIFO Underrun Interrupt flag
  - LTDC_IT_TE: Transfer Error interrupt flag
  - LTDC_IT_RR: Register Reload Interrupt Flag

Return value:

- The: state of INTERRUPT (SET or RESET).

LTDC Exported Types

MAX_LAYER

LTDC Flags

LTDC_FLAG_LI

LTDC Line Interrupt Flag

LTDC_FLAG_FU

LTDC FIFO Underrun interrupt Flag

LTDC_FLAG_TE

LTDC Transfer Error interrupt Flag

LTDC_FLAG_RR

LTDC Register Reload interrupt Flag

LTDC HS POLARITY

LTDC_HSPOLARITY_AL

Horizontal Synchronization is active low.

LTDC_HSPOLARITY_AH

Horizontal Synchronization is active high.

LTDC Interrupts

LTDC_IT_LI

LTDC Line Interrupt

LTDC_IT_FU

LTDC FIFO Underrun Interrupt

LTDC_IT_TE

LTDC Transfer Error Interrupt

LTDC_IT_RR

LTDC Register Reload Interrupt

LTDC Layer

LTDC_LAYER_1

LTDC Layer 1

LTDC_LAYER_2

LTDC Layer 2

LTDC LAYER Config

LTDC_STOPPOSITION

LTDC Layer stop position

LTDC_STARTPOSITION

LTDC Layer start position

LTDC_COLOR_FRAME_BUFFER

LTDC Layer Line length

LTDC_LINE_NUMBER

LTDC Layer Line number

LTDC PC POLARITY

LTDC_PCPOLARITY_IPC

input pixel clock.

LTDC_PCPOLARITY_IIPC

inverted input pixel clock.

LTDC Pixel format

LTDC_PIXEL_FORMAT_ARGB8888

ARGB8888 LTDC pixel format

LTDC_PIXEL_FORMAT_RGB888

RGB888 LTDC pixel format

LTDC_PIXEL_FORMAT_RGB565

RGB565 LTDC pixel format

LTDC_PIXEL_FORMAT_ARGB1555

ARGB1555 LTDC pixel format

LTDC_PIXEL_FORMAT_ARGB4444

ARGB4444 LTDC pixel format

LTDC_PIXEL_FORMAT_L8

L8 LTDC pixel format

LTDC_PIXEL_FORMAT_AL44

AL44 LTDC pixel format

LTDC_PIXEL_FORMAT_AL88

AL88 LTDC pixel format

LTDC Reload Type

LTDC_RELOAD_IMMEDIATE

Immediate Reload

LTDC_RELOAD_VERTICAL_BLANKING

Vertical Blanking Reload

LTDC SYNC

LTDC_HORIZONTALSYNC

Horizontal synchronization width.

LTDC_VERTICALSYNC

Vertical synchronization height.

LTDC VS POLARITY

LTDC_VSPOLARITY_AL

Vertical Synchronization is active low.

LTDC_VSPOLARITY_AH

Vertical Synchronization is active high.

51 HAL MDIOS Generic Driver

51.1 MDIOS Firmware driver registers structures

51.1.1 MDIOS_InitTypeDef

MDIOS_InitTypeDef is defined in the stm32h7xx_hal_mdios.h

Data Fields

- ***uint32_t PortAddress***
- ***uint32_t PreambleCheck***

Field Documentation

- ***uint32_t MDIOS_InitTypeDef::PortAddress***
  Specifies the MDIOS port address. This parameter can be a value from 0 to 31
- ***uint32_t MDIOS_InitTypeDef::PreambleCheck***
  Specifies whether the preamble check is enabled or disabled. This parameter can be a value of ***MDIOS_Preamble_Check***

51.1.2 __MDIOS_HandleTypeDef

__MDIOS_HandleTypeDef is defined in the stm32h7xx_hal_mdios.h

Data Fields

- ***MDIOS_TypeDef * Instance***
- ***MDIOS_InitTypeDef Init***
- ***__IO HAL_MDIOS_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***HAL_LockTypeDef Lock***
- ***void(* WriteCpltCallback***
- ***void(* ReadCpltCallback***
- ***void(* ErrorCallback***
- ***void(* WakeUpCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***MDIOS_TypeDef* __MDIOS_HandleTypeDef::Instance***
  Register base address
- ***MDIOS_InitTypeDef __MDIOS_HandleTypeDef::Init***
  MDIOS Init Structure
- ***__IO HAL_MDIOS_StateTypeDef __MDIOS_HandleTypeDef::State***
  MDIOS communication state This parameter can be a value of of **HAL_MDIOS_StateTypeDef**
- ***__IO uint32_t __MDIOS_HandleTypeDef::ErrorCode***
  Holds the global Error code of the MDIOS HAL status machine This parameter can be a value of of ***MDIOS_Error_Code***
- ***HAL_LockTypeDef __MDIOS_HandleTypeDef::Lock***
  MDIOS Lock
- ***void(* __MDIOS_HandleTypeDef::WriteCpltCallback)(struct __MDIOS_HandleTypeDef *hmdios)***
  MDIOS Write Complete Callback
- ***void(* __MDIOS_HandleTypeDef::ReadCpltCallback)(struct __MDIOS_HandleTypeDef *hmdios)***
  MDIOS Read Complete Callback

- ***void(* __MDIOS_HandleTypeDef::ErrorCallback)(struct __MDIOS_HandleTypeDef *hmdios)***
  MDIOS Error Callback
- ***void(* __MDIOS_HandleTypeDef::WakeUpCallback)(struct __MDIOS_HandleTypeDef *hmdios)***
  MDIOS Wake UP Callback
- ***void(* __MDIOS_HandleTypeDef::MspInitCallback)(struct __MDIOS_HandleTypeDef *hmdios)***
  MDIOS Msp Init callback
- ***void(* __MDIOS_HandleTypeDef::MspDeInitCallback)(struct __MDIOS_HandleTypeDef *hmdios)***
  MDIOS Msp DeInit callback

51.2 MDIOS Firmware driver API description

The following section lists the various functions of the MDIOS library.

51.2.1 How to use this driver

Note: A callback is executed for each genereted interrupt, so the driver provide the following HAL_MDIOS_WriteCpltCallback(), HAL_MDIOS_ReadCpltCallback() and HAL_MDIOS_ErrorCallback()

Note: HAL_MDIOS_IRQHandler() must be called from the MDIOS IRQ Handler, to handle the interrupt and execute the previous callbacks (#) Reset the MDIOS peripheral and all related resources by calling the HAL_MDIOS_DeInit() API. (##) HAL_MDIOS_MspDeInit() must be implemented to reset low level resources (GPIO, Clocks, NVIC configuration ...)

Callback registration

51.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the MDIOS

- The following parameters can be configured:
  - Port Address
  - Preamble Check

This section contains the following APIs:

- ***HAL_MDIOS_Init()***
- ***HAL_MDIOS_DeInit()***
- ***HAL_MDIOS_MspInit()***
- ***HAL_MDIOS_MspDeInit()***
- ***HAL_MDIOS_RegisterCallback()***
- ***HAL_MDIOS_UnRegisterCallback()***

51.2.3 IO operation functions

This section contains the following APIs:

- ***HAL_MDIOS_WriteReg()***
- ***HAL_MDIOS_ReadReg()***
- ***HAL_MDIOS_GetWrittenRegAddress()***
- ***HAL_MDIOS_GetReadRegAddress()***
- ***HAL_MDIOS_ClearWriteRegAddress()***
- ***HAL_MDIOS_ClearReadRegAddress()***
- ***HAL_MDIOS_EnableEvents()***
- ***HAL_MDIOS_IRQHandler()***
- ***HAL_MDIOS_WriteCpltCallback()***
- ***HAL_MDIOS_ReadCpltCallback()***
- ***HAL_MDIOS_ErrorCallback()***
- ***HAL_MDIOS_WakeUpCallback()***

51.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the MDIOS.

- HAL_MDIOS_GetState() API, helpful to check in run-time the state.
- HAL_MDIOS_GetError() API, returns the errors code of the HAL state machine.

This section contains the following APIs:

- ***HAL_MDIOS_GetError()***
- ***HAL_MDIOS_GetState()***

51.2.5 Detailed description of functions

HAL_MDIOS_Init

Function name

HAL_StatusTypeDef HAL_MDIOS_Init (MDIOS_HandleTypeDef * hmdios)

Function description

Initializes the MDIOS according to the specified parameters in the MDIOS_InitTypeDef and creates the associated handle .

Parameters

- **hmdios:** pointer to a MDIOS_HandleTypeDef structure that contains the configuration information for MDIOS module

Return values

- **HAL:** status

HAL_MDIOS_DeInit

Function name

HAL_StatusTypeDef HAL_MDIOS_DeInit (MDIOS_HandleTypeDef * hmdios)

Function description

DeInitializes the MDIOS peripheral.

Parameters

- **hmdios:** MDIOS handle

Return values

- **HAL:** status

HAL_MDIOS_MspInit

Function name

void HAL_MDIOS_MspInit (MDIOS_HandleTypeDef * hmdios)

Function description

MDIOS MSP Init.

Parameters

- **hmdios:** mdios handle

Return values

- **None:**

HAL_MDIOS_MspDeInit

Function name

void HAL_MDIOS_MspDeInit (MDIOS_HandleTypeDef * hmdios)

Function description

MDIOS MSP DeInit.

Parameters

- **hmdios:** mdios handle

Return values

- **None:**

HAL_MDIOS_RegisterCallback

Function name

HAL_StatusTypeDef HAL_MDIOS_RegisterCallback (MDIOS_HandleTypeDef * hmdios, HAL_MDIOS_CallbackIDTypeDef CallbackID, pMDIOS_CallbackTypeDef pCallback)

Function description

Register a User MDIOS Callback To be used instead of the weak predefined callback.

Parameters

- **hmdios:** mdios handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_MDIOS_WRITE_COMPLETE_CB_ID Write Complete Callback ID
  - HAL_MDIOS_READ_COMPLETE_CB_ID Read Complete Callback ID
  - HAL_MDIOS_ERROR_CB_ID Error Callback ID
  - HAL_MDIOS_WAKEUP_CB_ID Wake Up Callback ID
  - HAL_MDIOS_MSPINIT_CB_ID MspInit callback ID
  - HAL_MDIOS_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **status:**

HAL_MDIOS_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_MDIOS_UnRegisterCallback (MDIOS_HandleTypeDef * hmdios, HAL_MDIOS_CallbackIDTypeDef CallbackID)

Function description

Unregister an MDIOS Callback MDIOS callback is redirected to the weak predefined callback.

Parameters

- **hmdios:** mdios handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_MDIOS_WRITE_COMPLETE_CB_ID Write Complete Callback ID
  - HAL_MDIOS_READ_COMPLETE_CB_ID Read Complete Callback ID
  - HAL_MDIOS_ERROR_CB_ID Error Callback ID
  - HAL_MDIOS_WAKEUP_CB_ID Wake Up Callback ID
  - HAL_MDIOS_MSPINIT_CB_ID MspInit callback ID
  - HAL_MDIOS_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **status:**

HAL_MDIOS_WriteReg

Function name

HAL_StatusTypeDef HAL_MDIOS_WriteReg (MDIOS_HandleTypeDef * hmdios, uint32_t RegNum, uint16_t Data)

Function description

Writes to an MDIOS output register.

Parameters

- **hmdios:** mdios handle
- **RegNum:** MDIOS output register address
- **Data:** Data to write

Return values

- **HAL:** status

HAL_MDIOS_ReadReg

Function name

HAL_StatusTypeDef HAL_MDIOS_ReadReg (MDIOS_HandleTypeDef * hmdios, uint32_t RegNum, uint16_t * pData)

Function description

Reads an MDIOS input register.

Parameters

- **hmdios:** mdios handle
- **RegNum:** MDIOS input register address
- **pData:** pointer to Data

Return values

- **HAL:** status

HAL_MDIOS_GetWrittenRegAddress

Function name

uint32_t HAL_MDIOS_GetWrittenRegAddress (MDIOS_HandleTypeDef * hmdios)

Function description

Gets Written registers by MDIO master.

Parameters

- **hmdios:** mdios handle

Return values

- **bit:** map of written registers addresses

HAL_MDIOS_GetReadRegAddress

Function name

uint32_t HAL_MDIOS_GetReadRegAddress (MDIOS_HandleTypeDef * hmdios)

Function description

Gets Read registers by MDIO master.

Parameters

- **hmdios:** mdios handle

Return values

- **bit:** map of read registers addresses

HAL_MDIOS_ClearWriteRegAddress

Function name

HAL_StatusTypeDef HAL_MDIOS_ClearWriteRegAddress (MDIOS_HandleTypeDef * hmdios, uint32_t RegNum)

Function description

Clears Write registers flag.

Parameters

- **hmdios:** mdios handle
- **RegNum:** registers addresses to be cleared

Return values

- **HAL:** status

HAL_MDIOS_ClearReadRegAddress

Function name

HAL_StatusTypeDef HAL_MDIOS_ClearReadRegAddress (MDIOS_HandleTypeDef * hmdios, uint32_t RegNum)

Function description

Clears Read register flag.

Parameters

- **hmdios:** mdios handle
- **RegNum:** registers addresses to be cleared

Return values

- **HAL:** status

HAL_MDIOS_EnableEvents

Function name

HAL_StatusTypeDef HAL_MDIOS_EnableEvents (MDIOS_HandleTypeDef * hmdios)

Function description

Enables Events for MDIOS peripheral.

Parameters

- **hmdios:** mdios handle

Return values

- **HAL:** status

HAL_MDIOS_IRQHandler

Function name

void HAL_MDIOS_IRQHandler (MDIOS_HandleTypeDef * hmdios)

Function description

This function handles MDIOS interrupt request.

Parameters

- **hmdios:** MDIOS handle

Return values

- **None:**

HAL_MDIOS_WriteCpltCallback

Function name

void HAL_MDIOS_WriteCpltCallback (MDIOS_HandleTypeDef * hmdios)

Function description

Write Complete Callback.

Parameters

- **hmdios:** mdios handle

Return values

- **None:**

HAL_MDIOS_ReadCpltCallback

Function name

void HAL_MDIOS_ReadCpltCallback (MDIOS_HandleTypeDef * hmdios)

Function description

Read Complete Callback.

Parameters

- **hmdios:** mdios handle

Return values

- **None:**

HAL_MDIOS_ErrorCallback

Function name

void HAL_MDIOS_ErrorCallback (MDIOS_HandleTypeDef * hmdios)

Function description

Error Callback.

Parameters

- **hmdios:** mdios handle

Return values

- **None:**

HAL_MDIOS_WakeUpCallback

Function name

void HAL_MDIOS_WakeUpCallback (MDIOS_HandleTypeDef * hmdios)

Function description

MDIOS WAKEUP interrupt callback.

Parameters

- **hmdios:** mdios handle

Return values

- **None:**

HAL_MDIOS_GetError

Function name

uint32_t HAL_MDIOS_GetError (MDIOS_HandleTypeDef * hmdios)

Function description

Gets MDIOS error code.

Parameters

- **hmdios:** mdios handle

Return values

- **mdios:** error code

HAL_MDIOS_GetState

Function name

HAL_MDIOS_StateTypeDef HAL_MDIOS_GetState (MDIOS_HandleTypeDef * hmdios)

Function description

Return the MDIOS HAL state.

Parameters

- **hmdios:** mdios handle

Return values

- **HAL:** state

51.3 MDIOS Firmware driver defines

The following section lists the various define and macros of the module.

51.3.1 MDIOS

MDIOS

MDIOS Error Code

HAL_MDIOS_ERROR_NONE

No error

HAL_MDIOS_ERROR_PARAM

Busy error

HAL_MDIOS_ERROR_BUSY

Parameter error

HAL_MDIOS_ERROR_TIMEOUT

Timeout error

HAL_MDIOS_ERROR_DATA

Data transfer error

HAL_MDIOS_ERROR_INVALID_CALLBACK

Invalid Callback error

MDIOS Exported Macros

__HAL_MDIOS_RESET_HANDLE_STATE

Description:

- Reset MDIOS handle state.

Parameters:

- __HANDLE__: MDIOS handle.

Return value:

- None

__HAL_MDIOS_ENABLE

Description:

- Enable/Disable the MDIOS peripheral.

Parameters:

- __HANDLE__: specifies the MDIOS handle.

Return value:

- None

__HAL_MDIOS_DISABLE

__HAL_MDIOS_ENABLE_IT

Description:

- Enable the MDIOS device interrupt.

Parameters:

- __HANDLE__: specifies the MDIOS handle.
- __INTERRUPT__: : specifies the MDIOS interrupt sources to be enabled. This parameter can be one or a combination of the following values:
  - MDIOS_IT_WRITE: Register write interrupt
  - MDIOS_IT_READ: Register read interrupt
  - MDIOS_IT_ERROR: Error interrupt

Return value:

- None

__HAL_MDIOS_DISABLE_IT

Description:

- Disable the MDIOS device interrupt.

Parameters:

- __HANDLE__: specifies the MDIOS handle.
- __INTERRUPT__: : specifies the MDIOS interrupt sources to be disabled. This parameter can be one or a combination of the following values:
  - MDIOS_IT_WRITE: Register write interrupt
  - MDIOS_IT_READ: Register read interrupt
  - MDIOS_IT_ERROR: Error interrupt

Return value:

- None

__HAL_MDIOS_GET_WRITE_FLAG

Description:

- Set MDIOS slave get write register flag.

Parameters:

- __HANDLE__: specifies the MDIOS handle.
- __FLAG__: specifies the write register flag

Return value:

- The: state of write flag

__HAL_MDIOS_GET_READ_FLAG

Description:

- MDIOS slave get read register flag.

Parameters:

- __HANDLE__: specifies the MDIOS handle.
- __FLAG__: specifies the read register flag

Return value:

- The: state of read flag

__HAL_MDIOS_GET_ERROR_FLAG

Description:

- MDIOS slave get interrupt.

Parameters:

- __HANDLE__: specifies the MDIOS handle.
- __FLAG__: : specifies the Error flag. This parameter can be one or a combination of the following values:
  - MDIOS_TURNARROUND_ERROR_FLAG: Register write interrupt
  - MDIOS_START_ERROR_FLAG: Register read interrupt
  - MDIOS_PREAMBLE_ERROR_FLAG: Error interrupt

Return value:

- The: state of the error flag

__HAL_MDIOS_CLEAR_ERROR_FLAG

Description:

- MDIOS slave clear interrupt.

Parameters:

- __HANDLE__: specifies the MDIOS handle.
- __FLAG__: : specifies the Error flag. This parameter can be one or a combination of the following values:
  - MDIOS_TURNARROUND_ERROR_FLAG: Register write interrupt
  - MDIOS_START_ERROR_FLAG: Register read interrupt
  - MDIOS_PREAMBLE_ERROR_FLAG: Error interrupt

Return value:

- none

__HAL_MDIOS_GET_IT_SOURCE

Description:

- Checks whether the specified MDIOS interrupt is set or not.

Parameters:

- __HANDLE__: specifies the MDIOS handle.
- __INTERRUPT__: : specifies the MDIOS interrupt sources This parameter can be one or a combination of the following values:
  - MDIOS_IT_WRITE: Register write interrupt
  - MDIOS_IT_READ: Register read interrupt
  - MDIOS_IT_ERROR: Error interrupt

Return value:

- The: state of the interrupt source

__HAL_MDIOS_WAKEUP_EXTI_ENABLE_IT

Description:

- Enable the MDIOS WAKEUP Exti Line.

Parameters:

- __EXTI_LINE__: specifies the MDIOS WAKEUP Exti sources to be enabled. This parameter can be:
  - MDIOS_WAKEUP_EXTI_LINE

Return value:

- None.

__HAL_MDIOS_WAKEUP_EXTID2_ENABLE_IT

Description:

- Enable the MDIOS WAKEUP Exti Line by Domain2.

Parameters:

- __EXTI_LINE__: specifies the MDIOS WAKEUP Exti sources to be enabled. This parameter can be:
  - MDIOS_WAKEUP_EXTI_LINE

Return value:

- None.

__HAL_MDIOS_WAKEUP_EXTI_GET_FLAG

Description:

- checks whether the specified MDIOS WAKEUP Exti interrupt flag is set or not.

Parameters:

- __EXTI_LINE__: specifies the MDIOS WAKEUP Exti sources to be cleared. This parameter can be:
  - MDIOS_WAKEUP_EXTI_LINE

Return value:

- EXTI: MDIOS WAKEUP Line Status.

__HAL_MDIOS_WAKEUP_EXTID2_GET_FLAG

Description:

- checks whether the specified MDIOS WAKEUP Exti interrupt flag is set or not.

Parameters:

- __EXTI_LINE__: specifies the MDIOS WAKEUP Exti sources to be cleared. This parameter can be:
  - MDIOS_WAKEUP_EXTI_LINE

Return value:

- EXTI: MDIOS WAKEUP Line Status.

__HAL_MDIOS_WAKEUP_EXTI_CLEAR_FLAG

Description:

- Clear the MDIOS WAKEUP Exti flag.

Parameters:

- __EXTI_LINE__: specifies the MDIOS WAKEUP Exti sources to be cleared. This parameter can be:
  - MDIOS_WAKEUP_EXTI_LINE

Return value:

- None.

__HAL_MDIOS_WAKEUP_EXTID2_CLEAR_FLAG

Description:

- Clear the MDIOS WAKEUP Exti flag.

Parameters:

- __EXTI_LINE__: specifies the MDIOS WAKEUP Exti sources to be cleared. This parameter can be:
  - MDIOS_WAKEUP_EXTI_LINE

Return value:

- None.

__HAL_MDIOS_WAKEUP_EXTI_ENABLE_RISING_EDGE

Description:

- enable rising edge interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the MDIOS WAKEUP EXTI sources to be disabled. This parameter can be:
  - MDIOS_WAKEUP_EXTI_LINE

Return value:

- None

__HAL_MDIOS_WAKEUP_EXTI_ENABLE_FALLING_EDGE

Description:

- enable falling edge interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the MDIOS WAKEUP EXTI sources to be disabled. This parameter can be:
  - MDIOS_WAKEUP_EXTI_LINE

Return value:

- None

__HAL_MDIOS_WAKEUP_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- enable falling edge interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the MDIOS WAKEUP EXTI sources to be disabled. This parameter can be:
  - MDIOS_WAKEUP_EXTI_LINE

Return value:

- None

__HAL_MDIOS_WAKEUP_EXTI_GENERATE_SWIT

Description:

- Generates a Software interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the MDIOS WAKEUP EXTI sources to be disabled. This parameter can be:
  - MDIOS_WAKEUP_EXTI_LINE

Return value:

- None

MDIOS Input Output Registers Definitions

MDIOS_REG0

MDIOS_REG1

MDIOS_REG2

MDIOS_REG3

MDIOS_REG4

MDIOS_REG5

MDIOS_REG6

MDIOS_REG7

MDIOS_REG8

MDIOS_REG9

MDIOS_REG10

MDIOS_REG11

MDIOS_REG12

MDIOS_REG13

MDIOS_REG14

MDIOS_REG15

MDIOS_REG16

MDIOS_REG17

MDIOS_REG18

MDIOS_REG19

MDIOS_REG20

MDIOS_REG21

MDIOS_REG22

MDIOS_REG23

MDIOS_REG24

MDIOS_REG25

MDIOS_REG26

MDIOS_REG27

MDIOS_REG28

MDIOS_REG29

MDIOS_REG30

MDIOS_REG31

MDIOS Interrupt Flags

MDIOS_TURNAROUND_ERROR_FLAG

MDIOS_START_ERROR_FLAG

MDIOS_PREAMBLE_ERROR_FLAG

Interrupt Sources

MDIOS_IT_WRITE

MDIOS_IT_READ

MDIOS_IT_ERROR

MDIOS Preamble Check

MDIOS_PREAMBLE_CHECK_ENABLE

MDIOS_PREAMBLE_CHECK_DISABLE

MDIOS Registers Flags

MDIOS_REG0_FLAG

MDIOS_REG1_FLAG

MDIOS_REG2_FLAG

MDIOS_REG3_FLAG

MDIOS_REG4_FLAG

MDIOS_REG5_FLAG

MDIOS_REG6_FLAG

MDIOS_REG7_FLAG

MDIOS_REG8_FLAG

MDIOS_REG9_FLAG

MDIOS_REG10_FLAG

MDIOS_REG11_FLAG

MDIOS_REG12_FLAG

MDIOS_REG13_FLAG

MDIOS_REG14_FLAG

MDIOS_REG15_FLAG

MDIOS_REG16_FLAG

MDIOS_REG17_FLAG

MDIOS_REG18_FLAG

MDIOS_REG19_FLAG

MDIOS_REG20_FLAG

MDIOS_REG21_FLAG

MDIOS_REG22_FLAG

MDIOS_REG23_FLAG

MDIOS_REG24_FLAG

MDIOS_REG25_FLAG

MDIOS_REG26_FLAG

MDIOS_REG27_FLAG

MDIOS_REG28_FLAG

MDIOS_REG29_FLAG

MDIOS_REG30_FLAG

MDIOS_REG31_FLAG

MDIOS_ALLREG_FLAG

MDIOS Wakeup Line

MDIOS_WAKEUP_EXTI_LINE

52 HAL MDMA Generic Driver

52.1 MDMA Firmware driver registers structures

52.1.1 MDMA_InitTypeDef

MDMA_InitTypeDef is defined in the stm32h7xx_hal_mdma.h

Data Fields

- ***uint32_t Request***
- ***uint32_t TransferTriggerMode***
- ***uint32_t Priority***
- ***uint32_t Endianness***
- ***uint32_t SourceInc***
- ***uint32_t DestinationInc***
- ***uint32_t SourceDataSize***
- ***uint32_t DestDataSize***
- ***uint32_t DataAlignment***
- ***uint32_t BufferTransferLength***
- ***uint32_t SourceBurst***
- ***uint32_t DestBurst***
- ***int32_t SourceBlockAddressOffset***
- ***int32_t DestBlockAddressOffset***

Field Documentation

- ***uint32_t MDMA_InitTypeDef::Request***

  Specifies the MDMA request. This parameter can be a value of ***MDMA_Request_selection***
- ***uint32_t MDMA_InitTypeDef::TransferTriggerMode***

  Specifies the Trigger Transfer mode : each request triggers a : a buffer transfer, a block transfer, a repeated block transfer or a linked list transfer This parameter can be a value of ***MDMA_Transfer_TriggerMode***
- ***uint32_t MDMA_InitTypeDef::Priority***

  Specifies the software priority for the MDMAy channelx. This parameter can be a value of ***MDMA_Priority_level***
- ***uint32_t MDMA_InitTypeDef::Endianness***

  Specifies if the MDMA transactions preserve the Little endianness. This parameter can be a value of ***MDMA_Endianness***
- ***uint32_t MDMA_InitTypeDef::SourceInc***

  Specifies if the Source increment mode . This parameter can be a value of ***MDMA_Source_increment_mode***
- ***uint32_t MDMA_InitTypeDef::DestinationInc***

  Specifies if the Destination increment mode . This parameter can be a value of ***MDMA_Destination_increment_mode***
- ***uint32_t MDMA_InitTypeDef::SourceDataSize***

  Specifies the source data size. This parameter can be a value of ***MDMA_Source_data_size***
- ***uint32_t MDMA_InitTypeDef::DestDataSize***

  Specifies the destination data size. This parameter can be a value of ***MDMA_Destination_data_size***
- ***uint32_t MDMA_InitTypeDef::DataAlignment***

  Specifies the source to destination Memory data packing/padding mode. This parameter can be a value of ***MDMA_data_Alignment***

- ***uint32_t MDMA_InitTypeDef::BufferTransferLength***

  Specifies the buffer Transfer Length (number of bytes), this is the number of bytes to be transferred in a single transfer (1 byte to 128 bytes)
- ***uint32_t MDMA_InitTypeDef::SourceBurst***

  Specifies the Burst transfer configuration for the source memory transfers. It specifies the amount of data to be transferred in a single non interruptible transaction. This parameter can be a value of ***MDMA_Source_burst***

  Note:

  – : the burst may be FIXED/INCR based on SourceInc value , the BURST must be programmed as to ensure that the burst size will be lower than than BufferTransferLength
- ***uint32_t MDMA_InitTypeDef::DestBurst***

  Specifies the Burst transfer configuration for the destination memory transfers. It specifies the amount of data to be transferred in a single non interruptible transaction. This parameter can be a value of ***MDMA_Destination_burst***

  Note:

  – : the burst may be FIXED/INCR based on DestinationInc value , the BURST must be programmed as to ensure that the burst size will be lower than than BufferTransferLength
- ***int32_t MDMA_InitTypeDef::SourceBlockAddressOffset***

  this field specifies the Next block source address offset signed value : if > 0 then increment the next block source Address by offset from where the last block ends if < 0 then decrement the next block source Address by offset from where the last block ends if == 0, the next block source address starts from where the last block ends
- ***int32_t MDMA_InitTypeDef::DestBlockAddressOffset***

  this field specifies the Next block destination address offset signed value : if > 0 then increment the next block destination Address by offset from where the last block ends if < 0 then decrement the next block destination Address by offset from where the last block ends if == 0, the next block destination address starts from where the last block ends

52.1.2 MDMA_LinkNodeTypeDef

MDMA_LinkNodeTypeDef is defined in the stm32h7xx_hal_mdma.h

Data Fields

- ***__IO uint32_t CTCR***
- ***__IO uint32_t CBNDTR***
- ***__IO uint32_t CSAR***
- ***__IO uint32_t CDAR***
- ***__IO uint32_t CBRUR***
- ***__IO uint32_t CLAR***
- ***__IO uint32_t CTBR***
- ***__IO uint32_t Reserved***
- ***__IO uint32_t CMAR***
- ***__IO uint32_t CMDR***

Field Documentation

- ***__IO uint32_t MDMA_LinkNodeTypeDef::CTCR***

  New CTCR register configuration for the given MDMA linked list node
- ***__IO uint32_t MDMA_LinkNodeTypeDef::CBNDTR***

  New CBNDTR register configuration for the given MDMA linked list node
- ***__IO uint32_t MDMA_LinkNodeTypeDef::CSAR***

  New CSAR register configuration for the given MDMA linked list node
- ***__IO uint32_t MDMA_LinkNodeTypeDef::CDAR***

  New CDAR register configuration for the given MDMA linked list node
- ***__IO uint32_t MDMA_LinkNodeTypeDef::CBRUR***

  New CBRUR register configuration for the given MDMA linked list node

- ***__IO uint32_t MDMA_LinkNodeTypeDef::CLAR***
  New CLAR register configuration for the given MDMA linked list node
- ***__IO uint32_t MDMA_LinkNodeTypeDef::CTBR***
  New CTBR register configuration for the given MDMA linked list node
- ***__IO uint32_t MDMA_LinkNodeTypeDef::Reserved***
  Reserved register
- ***__IO uint32_t MDMA_LinkNodeTypeDef::CMAR***
  New CMAR register configuration for the given MDMA linked list node
- ***__IO uint32_t MDMA_LinkNodeTypeDef::CMDR***
  New CMDR register configuration for the given MDMA linked list node

52.1.3 MDMA_LinkNodeConfTypeDef

MDMA_LinkNodeConfTypeDef is defined in the stm32h7xx_hal_mdma.h

Data Fields

- ***MDMA_InitTypeDef Init***
- ***uint32_t SrcAddress***
- ***uint32_t DstAddress***
- ***uint32_t BlockDataLength***
- ***uint32_t BlockCount***
- ***uint32_t PostRequestMaskAddress***
- ***uint32_t PostRequestMaskData***

Field Documentation

- ***MDMA_InitTypeDef MDMA_LinkNodeConfTypeDef::Init***
  configuration of the specified MDMA Linked List Node
- ***uint32_t MDMA_LinkNodeConfTypeDef::SrcAddress***
  The source memory address for the Linked list Node
- ***uint32_t MDMA_LinkNodeConfTypeDef::DstAddress***
  The destination memory address for the Linked list Node
- ***uint32_t MDMA_LinkNodeConfTypeDef::BlockDataLength***
  The data length of a block in bytes
- ***uint32_t MDMA_LinkNodeConfTypeDef::BlockCount***
  The number of blocks to be transferred
- ***uint32_t MDMA_LinkNodeConfTypeDef::PostRequestMaskAddress***
  specifies the address to be updated (written) with PostRequestMaskData after a request is served. PostRequestMaskAddress and PostRequestMaskData could be used to automatically clear a peripheral flag when the request is served
- ***uint32_t MDMA_LinkNodeConfTypeDef::PostRequestMaskData***
  specifies the value to be written to PostRequestMaskAddress after a request is served. PostRequestMaskAddress and PostRequestMaskData could be used to automatically clear a peripheral flag when the request is served

52.1.4 __MDMA_HandleTypeDef

__MDMA_HandleTypeDef is defined in the stm32h7xx_hal_mdma.h

Data Fields

- ***MDMA_Channel_TypeDef * Instance***
- ***MDMA_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_MDMA_StateTypeDef State***
- ***void * Parent***
- ***void(* XferCpltCallback***

- ***void(* XferBufferCpltCallback***
- ***void(* XferBlockCpltCallback***
- ***void(* XferRepeatBlockCpltCallback***
- ***void(* XferErrorCallback***
- ***void(* XferAbortCallback***
- ***MDMA_LinkNodeTypeDef * FirstLinkedListNodeAddress***
- ***MDMA_LinkNodeTypeDef * LastLinkedListNodeAddress***
- ***uint32_t LinkedListNodeCounter***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***MDMA_Channel_TypeDef* __MDMA_HandleTypeDef::Instance***
  Register base address
- ***MDMA_InitTypeDef __MDMA_HandleTypeDef::Init***
  MDMA communication parameters
- ***HAL_LockTypeDef __MDMA_HandleTypeDef::Lock***
  MDMA locking object
- ***__IO HAL_MDMA_StateTypeDef __MDMA_HandleTypeDef::State***
  MDMA transfer state
- ***void* __MDMA_HandleTypeDef::Parent***
  Parent object state
- ***void(* __MDMA_HandleTypeDef::XferCpltCallback)(struct __MDMA_HandleTypeDef *hmdma)***
  MDMA transfer complete callback
- ***void(* __MDMA_HandleTypeDef::XferBufferCpltCallback)(struct __MDMA_HandleTypeDef *hmdma)***
  MDMA buffer transfer complete callback
- ***void(* __MDMA_HandleTypeDef::XferBlockCpltCallback)(struct __MDMA_HandleTypeDef *hmdma)***
  MDMA block transfer complete callback
- ***void(* __MDMA_HandleTypeDef::XferRepeatBlockCpltCallback)(struct __MDMA_HandleTypeDef *hmdma)***
  MDMA block transfer repeat callback
- ***void(* __MDMA_HandleTypeDef::XferErrorCallback)(struct __MDMA_HandleTypeDef *hmdma)***
  MDMA transfer error callback
- ***void(* __MDMA_HandleTypeDef::XferAbortCallback)(struct __MDMA_HandleTypeDef *hmdma)***
  MDMA transfer Abort callback
- ***MDMA_LinkNodeTypeDef* __MDMA_HandleTypeDef::FirstLinkedListNodeAddress***
  specifies the first node address of the transfer list (after the initial node defined by the Init struct) this parameter is used internally by the MDMA driver to construct the linked list node
- ***MDMA_LinkNodeTypeDef* __MDMA_HandleTypeDef::LastLinkedListNodeAddress***
  specifies the last node address of the transfer list this parameter is used internally by the MDMA driver to construct the linked list node
- ***uint32_t __MDMA_HandleTypeDef::LinkedListNodeCounter***
  Number of nodes in the MDMA linked list
- ***__IO uint32_t __MDMA_HandleTypeDef::ErrorCode***
  MDMA Error code

52.2 MDMA Firmware driver API description

The following section lists the various functions of the MDMA library.

52.2.1 How to use this driver

1. Enable and configure the peripheral to be connected to the MDMA Channel (except for internal SRAM/ FLASH memories: no initialization is necessary) please refer to Reference manual for connection between peripherals and MDMA requests.
2. For a given Channel use HAL_MDMA_Init function to program the required configuration through the following parameters: transfer request , channel priority, data endianness, Source increment, destination increment , source data size, destination data size, data alignment, source Burst, destination Burst , buffer Transfer Length, Transfer Trigger Mode (buffer transfer, block transfer, repeated block transfer or full transfer) source and destination block address offset, mask address and data. If using the MDMA in linked list mode then use function HAL_MDMA_LinkedList_CreateNode to fill a transfer node. Note that parameters given to the function HAL_MDMA_Init corresponds always to the node zero. Use function HAL_MDMA_LinkedList_AddNode to connect the created node to the linked list at a given position. User can make a linked list circular using function HAL_MDMA_LinkedList_EnableCircularMode , this function will automatically connect the last node of the list to the first one in order to make the list circular. In this case the linked list will loop on node 1 : first node connected after the initial transfer defined by the HAL_MDMA_Init

Note: The initial transfer itself (node 0 corresponding to the Init). User can disable the circular mode using function HAL_MDMA_LinkedList_DisableCircularMode, this function will then remove the connection between last node and first one. Function HAL_MDMA_LinkedList_RemoveNode can be used to remove (disconnect) a node from the transfer linked list. When a linked list is circular (last node connected to first one), if removing node1 (node where the linked list loops), the linked list remains circular and node 2 becomes the first one. Note that if the linked list is made circular the transfer will loop infinitely (or until aborted by the user).

- User can select the transfer trigger mode (parameter TransferTriggerMode) to define the amount of data to be transfer upon a request :
  - MDMA_BUFFER_TRANSFER : each request triggers a transfer of BufferTransferLength data with BufferTransferLength defined within the HAL_MDMA_Init.
  - MDMA_BLOCK_TRANSFER : each request triggers a transfer of a block with block size defined within the function HAL_MDMA_Start/HAL_MDMA_Start_IT or within the current linked list node parameters.
  - MDMA_REPEAT_BLOCK_TRANSFER : each request triggers a transfer of a number of blocks with block size and number of blocks defined within the function HAL_MDMA_Start/HAL_MDMA_Start_IT or within the current linked list node parameters.
  - MDMA_FULL_TRANSFER : each request triggers a full transfer all blocks and all nodes(if a linked list has been created using HAL_MDMA_LinkedList_CreateNode \ HAL_MDMA_LinkedList_AddNode).

Polling mode IO operation

- Use HAL_MDMA_Start() to start MDMA transfer after the configuration of Source address and destination address and the Length of data to be transferred.
- Use HAL_MDMA_PollForTransfer() to poll for the end of current transfer or a transfer level In this case a fixed Timeout can be configured by User depending from his application.
- Use HAL_MDMA_Abort() function to abort the current transfer : blocking method this API returns when the abort ends or timeout (should not be called from an interrupt service routine).

Interrupt mode IO operation

- Configure the MDMA interrupt priority using HAL_NVIC_SetPriority()
- Enable the MDMA IRQ handler using HAL_NVIC_EnableIRQ()
- Use HAL_MDMA_Start_IT() to start MDMA transfer after the configuration of Source address and destination address and the Length of data to be transferred. In this case the MDMA interrupt is configured.
- Use HAL_MDMA_IRQHandler() called under MDMA_IRQHandler() Interrupt subroutine
- At the end of data transfer HAL_MDMA_IRQHandler() function is executed and user can add his own function by customization of function pointer XferCpltCallback and XferErrorCallback (i.e a member of MDMA handle structure).
- Use HAL_MDMA_Abort_IT() function to abort the current transfer : non-blocking method. This API will finish the execution immediately then the callback XferAbortCallback (if specified by the user) is asserted once the MDMA channel has effectively aborted. (could be called from an interrupt service routine).

- Use functions HAL_MDMA_RegisterCallback and HAL_MDMA_UnRegisterCallback respectevely to register unregister user callbacks from the following list :
  - XferCpltCallback : transfer complete callback.
  - XferBufferCpltCallback : buffer transfer complete callback.
  - XferBlockCpltCallback : block transfer complete callback.
  - XferRepeatBlockCpltCallback : repeated block transfer complete callback.
  - XferErrorCallback : transfer error callback.
  - XferAbortCallback : transfer abort complete callback.
- If the transfer Request corresponds to SW request (MDMA_REQUEST_SW) User can use function HAL_MDMA_GenerateSWRequest to trigger requests manually. Function HAL_MDMA_GenerateSWRequest must be used with the following precautions:
  - This function returns an error if used while the Transfer has ended or not started.
  - If used while the current request has not been served yet (current request transfer on going) this function returns an error and the new request is ignored. Generally this function should be used in conjunctions with the MDMA callbacks:
  - example 1:
    - Configure a transfer with request set to MDMA_REQUEST_SW and trigger mode set to MDMA_BUFFER_TRANSFER
    - Register a callback for buffer transfer complete (using callback ID set to HAL_MDMA_XFER_BUFFERCPLT_CB_ID)
    - After calling HAL_MDMA_Start_IT the MDMA will issue the transfer of a first BufferTransferLength data.
    - When the buffer transfer complete callback is asserted first buffer has been transferred and user can ask for a new buffer transfer request using HAL_MDMA_GenerateSWRequest.
  - example 2:
    - Configure a transfer with request set to MDMA_REQUEST_SW and trigger mode set to MDMA_BLOCK_TRANSFER
    - Register a callback for block transfer complete (using callback ID HAL_MDMA_XFER_BLOCKCPLT_CB_ID)
    - After calling HAL_MDMA_Start_IT the MDMA will issue the transfer of a first block of data.
    - When the block transfer complete callback is asserted the first block has been transferred and user can ask for a new block transfer request using HAL_MDMA_GenerateSWRequest.

Use HAL_MDMA_GetState() function to return the MDMA state and HAL_MDMA_GetError() in case of error detection.

MDMA HAL driver macros list

Below the list of most used macros in MDMA HAL driver.

- __HAL_MDMA_ENABLE: Enable the specified MDMA Channel.
- __HAL_MDMA_DISABLE: Disable the specified MDMA Channel.
- __HAL_MDMA_GET_FLAG: Get the MDMA Channel pending flags.
- __HAL_MDMA_CLEAR_FLAG: Clear the MDMA Channel pending flags.
- __HAL_MDMA_ENABLE_IT: Enable the specified MDMA Channel interrupts.
- __HAL_MDMA_DISABLE_IT: Disable the specified MDMA Channel interrupts.
- __HAL_MDMA_GET_IT_SOURCE: Check whether the specified MDMA Channel interrupt has occurred or not.

Note: You can refer to the header file of the MDMA HAL driver for more useful macros.

52.2.2 Initialization and de-initialization functions

This section provides functions allowing to : Initialize and de-initialize the MDMA channel. Register and Unregister MDMA callbacks

The HAL_MDMA_Init() function follows the MDMA channel configuration procedures as described in reference manual. The HAL_MDMA_DeInit function allows to deinitialize the MDMA channel. HAL_MDMA_RegisterCallback and HAL_MDMA_UnRegisterCallback functions allows respectevely to register/unregister an MDMA callback function.

This section contains the following APIs:

- ***HAL_MDMA_Init()***
- ***HAL_MDMA_DeInit()***
- ***HAL_MDMA_ConfigPostRequestMask()***
- ***HAL_MDMA_RegisterCallback()***
- ***HAL_MDMA_UnRegisterCallback()***

52.2.3 Linked list operation functions

This section provides functions allowing to:

- Create a linked list node
- Add a node to the MDMA linked list
- Remove a node from the MDMA linked list
- Enable/Disable linked list circular mode

This section contains the following APIs:

- ***HAL_MDMA_LinkedList_CreateNode()***
- ***HAL_MDMA_LinkedList_AddNode()***
- ***HAL_MDMA_LinkedList_RemoveNode()***
- ***HAL_MDMA_LinkedList_EnableCircularMode()***
- ***HAL_MDMA_LinkedList_DisableCircularMode()***

52.2.4 IO operation functions

This section provides functions allowing to:

- Configure the source, destination address and data length and Start MDMA transfer
- Configure the source, destination address and data length and Start MDMA transfer with interrupt
- Abort MDMA transfer
- Poll for transfer complete
- Generate a SW request (when Request is set to MDMA_REQUEST_SW)
- Handle MDMA interrupt request

This section contains the following APIs:

- ***HAL_MDMA_Start()***
- ***HAL_MDMA_Start_IT()***
- ***HAL_MDMA_Abort()***
- ***HAL_MDMA_Abort_IT()***
- ***HAL_MDMA_PollForTransfer()***
- ***HAL_MDMA_GenerateSWRequest()***
- ***HAL_MDMA_IRQHandler()***

52.2.5 State and Errors functions

This subsection provides functions allowing to

- Check the MDMA state
- Get error code

This section contains the following APIs:

- ***HAL_MDMA_GetState()***
- ***HAL_MDMA_GetError()***

52.2.6 Detailed description of functions

HAL_MDMA_Init

Function name

HAL_StatusTypeDef HAL_MDMA_Init (MDMA_HandleTypeDef * hmdma)

Function description

Initializes the MDMA according to the specified parameters in the MDMA_InitTypeDef and create the associated handle.

Parameters

- **hmdma:** Pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Channel.

Return values

- **HAL:** status

HAL_MDMA_DeInit

Function name

HAL_StatusTypeDef HAL_MDMA_DeInit (MDMA_HandleTypeDef * hmdma)

Function description

DeInitializes the MDMA peripheral.

Parameters

- **hmdma:** pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Channel.

Return values

- **HAL:** status

HAL_MDMA_ConfigPostRequestMask

Function name

HAL_StatusTypeDef HAL_MDMA_ConfigPostRequestMask (MDMA_HandleTypeDef * hmdma, uint32_t MaskAddress, uint32_t MaskData)

Function description

Config the Post request Mask address and Mask data.

Parameters

- **hmdma:** : pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Channel.
- **MaskAddress:** specifies the address to be updated (written) with MaskData after a request is served.
- **MaskData:** specifies the value to be written to MaskAddress after a request is served. MaskAddress and MaskData could be used to automatically clear a peripheral flag when the request is served.

Return values

- **HAL:** status

HAL_MDMA_RegisterCallback

Function name

HAL_StatusTypeDef HAL_MDMA_RegisterCallback (MDMA_HandleTypeDef * hmdma, HAL_MDMA_CallbackIDTypeDef CallbackID, void(*)(MDMA_HandleTypeDef *_hmdma) pCallback)

Function description

Register callbacks.

Parameters

- **hmdma:** pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Channel.
- **CallbackID:** User Callback identifier
- **pCallback:** pointer to callbacsk function.

Return values

- **HAL:** status

HAL_MDMA_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_MDMA_UnRegisterCallback (MDMA_HandleTypeDef * hmdma, HAL_MDMA_CallbackIDTypeDef CallbackID)

Function description

UnRegister callbacks.

Parameters

- **hmdma:** pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Channel.
- **CallbackID:** User Callback identifier a HAL_MDMA_CallbackIDTypeDef ENUM as parameter.

Return values

- **HAL:** status

HAL_MDMA_LinkedList_CreateNode

Function name

HAL_StatusTypeDef HAL_MDMA_LinkedList_CreateNode (MDMA_LinkNodeTypeDef * pNode, MDMA_LinkNodeConfTypeDef * pNodeConfig)

Function description

Initializes an MDMA Link Node according to the specified parameters in the pMDMA_LinkedListNodeConfig .

Parameters

- **pNode:** Pointer to a MDMA_LinkNodeTypeDef structure that contains Linked list node registers configurations.
- **pNodeConfig:** Pointer to a MDMA_LinkNodeConfTypeDef structure that contains the configuration information for the specified MDMA Linked List Node.

Return values

- **HAL:** status

HAL_MDMA_LinkedList_AddNode

Function name

HAL_StatusTypeDef HAL_MDMA_LinkedList_AddNode (MDMA_HandleTypeDef * hmdma, MDMA_LinkNodeTypeDef * pNewNode, MDMA_LinkNodeTypeDef * pPrevNode)

Function description

Connect a node to the linked list.

Parameters

- **hmdma:** : Pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Channel.
- **pNewNode:** : Pointer to a MDMA_LinkNodeTypeDef structure that contains Linked list node to be add to the list.
- **pPrevNode:** : Pointer to the new node position in the linked list or zero to insert the new node at the end of the list

Return values

- **HAL:** status

HAL_MDMA_LinkedList_RemoveNode

Function name

HAL_StatusTypeDef HAL_MDMA_LinkedList_RemoveNode (MDMA_HandleTypeDef * hmdma, MDMA_LinkNodeTypeDef * pNode)

Function description

Disconnect/Remove a node from the transfer linked list.

Parameters

- **hmdma:** : Pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Channel.
- **pNode:** : Pointer to a MDMA_LinkNodeTypeDef structure that contains Linked list node to be removed from the list.

Return values

- **HAL:** status

HAL_MDMA_LinkedList_EnableCircularMode

Function name

HAL_StatusTypeDef HAL_MDMA_LinkedList_EnableCircularMode (MDMA_HandleTypeDef * hmdma)

Function description

Make the linked list circular by connecting the last node to the first.

Parameters

- **hmdma:** : Pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Channel.

Return values

- **HAL:** status

HAL_MDMA_LinkedList_DisableCircularMode

Function name

HAL_StatusTypeDef HAL_MDMA_LinkedList_DisableCircularMode (MDMA_HandleTypeDef * hmdma)

Function description

Disable the linked list circular mode by setting the last node connection to null.

Parameters

- **hmdma:** : Pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Channel.

Return values

- **HAL:** status

HAL_MDMA_Start

Function name

HAL_StatusTypeDef HAL_MDMA_Start (MDMA_HandleTypeDef * hmdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t BlockDataLength, uint32_t BlockCount)

Function description

Starts the MDMA Transfer.

Parameters

- **hmdma:** : pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Channel.
- **SrcAddress:** : The source memory Buffer address
- **DstAddress:** : The destination memory Buffer address
- **BlockDataLength:** : The length of a block transfer in bytes
- **BlockCount:** : The number of a blocks to be transfer

Return values

- **HAL:** status

HAL_MDMA_Start_IT

Function name

HAL_StatusTypeDef HAL_MDMA_Start_IT (MDMA_HandleTypeDef * hmdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t BlockDataLength, uint32_t BlockCount)

Function description

Starts the MDMA Transfer with interrupts enabled.

Parameters

- **hmdma:** : pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Channel.
- **SrcAddress:** : The source memory Buffer address
- **DstAddress:** : The destination memory Buffer address
- **BlockDataLength:** : The length of a block transfer in bytes
- **BlockCount:** : The number of a blocks to be transfer

Return values

- **HAL:** status

HAL_MDMA_Abort

Function name

HAL_StatusTypeDef HAL_MDMA_Abort (MDMA_HandleTypeDef * hmdma)

Function description

Aborts the MDMA Transfer.

Parameters

- **hmdma:** : pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Channel.

Return values

- **HAL:** status

Notes

- After disabling a MDMA Channel, a check for wait until the MDMA Channel is effectively disabled is added. If a Channel is disabled while a data transfer is ongoing, the current data will be transferred and the Channel will be effectively disabled only after the transfer of this single data is finished.

HAL_MDMA_Abort_IT

Function name

HAL_StatusTypeDef HAL_MDMA_Abort_IT (MDMA_HandleTypeDef * hmdma)

Function description

Aborts the MDMA Transfer in Interrupt mode.

Parameters

- **hmdma:** : pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Channel.

Return values

- **HAL:** status

HAL_MDMA_PollForTransfer

Function name

HAL_StatusTypeDef HAL_MDMA_PollForTransfer (MDMA_HandleTypeDef * hmdma, HAL_MDMA_LevelCompleteTypeDef CompleteLevel, uint32_t Timeout)

Function description

Polling for transfer complete.

Parameters

- **hmdma:** pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Channel.
- **CompleteLevel:** Specifies the MDMA level complete.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

HAL_MDMA_GenerateSWRequest

Function name

HAL_StatusTypeDef HAL_MDMA_GenerateSWRequest (MDMA_HandleTypeDef * hmdma)

Function description

Generate an MDMA SW request trigger to activate the request on the given Channel.

Parameters

- **hmdma:** pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Stream.

Return values

- **HAL:** status

HAL_MDMA_IRQHandler

Function name

void HAL_MDMA_IRQHandler (MDMA_HandleTypeDef * hmdma)

Function description

Handles MDMA interrupt request.

Parameters

- **hmdma:** pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Channel.

Return values

- **None:**

HAL_MDMA_GetState

Function name

HAL_MDMA_StateTypeDef HAL_MDMA_GetState (MDMA_HandleTypeDef * hmdma)

Function description

Returns the MDMA state.

Parameters

- **hmdma:** pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Channel.

Return values

- **HAL:** state

HAL_MDMA_GetError

Function name

uint32_t HAL_MDMA_GetError (MDMA_HandleTypeDef * hmdma)

Function description

Return the MDMA error code.

Parameters

- **hmdma:** : pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Channel.

Return values

- **MDMA:** Error Code

52.3 MDMA Firmware driver defines

The following section lists the various define and macros of the module.

52.3.1 MDMA

MDMA

MDMA data alignment

MDMA_DATAALIGN_PACKENABLE

The source data is packed/un-packed into the destination data size All data are right aligned, in Little Endien mode.

MDMA_DATAALIGN_RIGHT

Right Aligned, padded w/ 0s (default)

MDMA_DATAALIGN_RIGHT_SIGNED

Right Aligned, Sign extended , Note : this mode is allowed only if the Source data size is smaller than Destination data size

MDMA_DATAALIGN_LEFT

Left Aligned (padded with 0s)

MDMA Destination burst

MDMA_DEST_BURST_SINGLE

single transfer

MDMA_DEST_BURST_2BEATS

Burst 2 beats

MDMA_DEST_BURST_4BEATS

Burst 4 beats

MDMA_DEST_BURST_8BEATS

Burst 8 beats

MDMA_DEST_BURST_16BEATS

Burst 16 beats

MDMA_DEST_BURST_32BEATS

Burst 32 beats

MDMA_DEST_BURST_64BEATS

Burst 64 beats

MDMA_DEST_BURST_128BEATS

Burst 128 beats

MDMA Destination data size

MDMA_DEST_DATASIZE_BYTE

Destination data size is Byte

MDMA_DEST_DATASIZE_HALFWORD

Destination data size is half word

MDMA_DEST_DATASIZE_WORD

Destination data size is word

MDMA_DEST_DATASIZE_DOUBLEWORD

Destination data size is double word

MDMA Destination increment mode

MDMA_DEST_INC_DISABLE

Source address pointer is fixed

MDMA_DEST_INC_BYTE

Source address pointer is incremented by a BYTE (8 bits)

MDMA_DEST_INC_HALFWORD

Source address pointer is incremented by a half Word (16 bits)

MDMA_DEST_INC_WORD

Source address pointer is incremented by a Word (32 bits)

MDMA_DEST_INC_DOUBLEWORD

Source address pointer is incremented by a double Word (64 bits))

MDMA_DEST_DEC_BYTE

Source address pointer is decremented by a BYTE (8 bits)

MDMA_DEST_DEC_HALFWORD

Source address pointer is decremented by a half Word (16 bits)

MDMA_DEST_DEC_WORD

Source address pointer is decremented by a Word (32 bits)

MDMA_DEST_DEC_DOUBLEWORD

Source address pointer is decremented by a double Word (64 bits))

MDMA Endianness

MDMA_LITTLE_ENDIANNESS_PRESERVE

little endianness preserve

MDMA_LITTLE_BYTE_ENDIANNESS_EXCHANGE

BYTEs endianness exchange when destination data size is > Byte

MDMA_LITTLE_HALFWORD_ENDIANNESS_EXCHANGE

HALF WORDs endianness exchange when destination data size is > HALF WORD

MDMA_LITTLE_WORD_ENDIANNESS_EXCHANGE

WORDs endianness exchange when destination data size is > DOUBLE WORD

MDMA Error Codes

HAL_MDMA_ERROR_NONE

No error

HAL_MDMA_ERROR_READ_XFER

Read Transfer error

HAL_MDMA_ERROR_WRITE_XFER

Write Transfer error

HAL_MDMA_ERROR_MASK_DATA

Error Mask Data error

HAL_MDMA_ERROR_LINKED_LIST

Linked list Data error

HAL_MDMA_ERROR_ALIGNMENT

Address/Size alignment error

HAL_MDMA_ERROR_BLOCK_SIZE

Block Size error

HAL_MDMA_ERROR_TIMEOUT

Timeout error

HAL_MDMA_ERROR_NO_XFER

Abort or SW trigger requested with no Xfer ongoing

HAL_MDMA_ERROR_BUSY

DeInit or SW trigger requested with Xfer ongoing

MDMA Exported Macros

__HAL_MDMA_ENABLE

Description:

- Enable the specified MDMA Channel.

Parameters:

- __HANDLE__: MDMA handle

Return value:

- None

__HAL_MDMA_DISABLE

Description:

- Disable the specified MDMA Channel.

Parameters:

- __HANDLE__: MDMA handle

Return value:

- None

__HAL_MDMA_GET_FLAG

Description:

- Get the MDMA Channel pending flags.

Parameters:

- __HANDLE__: MDMA handle
- __FLAG__: Get the specified flag. This parameter can be any combination of the following values:
  - MDMA_FLAG_TE : Transfer Error flag.
  - MDMA_FLAG_CTC : Channel Transfer Complete flag.
  - MDMA_FLAG_BRT : Block Repeat Transfer flag.
  - MDMA_FLAG_BT : Block Transfer complete flag.
  - MDMA_FLAG_BFTC : BuFfer Transfer Complete flag.
  - MDMA_FLAG_CRQA : Channel request Active flag.

Return value:

- The: state of FLAG (SET or RESET).

__HAL_MDMA_CLEAR_FLAG

Description:

- Clear the MDMA Stream pending flags.

Parameters:

- __HANDLE__: MDMA handle
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - MDMA_FLAG_TE : Transfer Error flag.
  - MDMA_FLAG_CTC : Channel Transfer Complete flag.
  - MDMA_FLAG_BRT : Block Repeat Transfer flag.
  - MDMA_FLAG_BT : Block Transfer complete flag.
  - MDMA_FLAG_BFTC : BuFfer Transfer Complete flag.

Return value:

- None

__HAL_MDMA_ENABLE_IT

Description:

- Enables the specified MDMA Channel interrupts.

Parameters:

- __HANDLE__: MDMA handle
- __INTERRUPT__: specifies the DMA interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - MDMA_IT_TE : Transfer Error interrupt mask
  - MDMA_IT_CTC : Channel Transfer Complete interrupt mask
  - MDMA_IT_BRT : Block Repeat Transfer interrupt mask
  - MDMA_IT_BT : Block Transfer interrupt mask
  - MDMA_IT_BFTC : BuFfer Transfer Complete interrupt mask

Return value:

- None

__HAL_MDMA_DISABLE_IT

Description:

- Disables the specified MDMA Channel interrupts.

Parameters:

- __HANDLE__: MDMA handle
- __INTERRUPT__: specifies the MDMA interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - MDMA_IT_TE : Transfer Error interrupt mask
  - MDMA_IT_CTC : Channel Transfer Complete interrupt mask
  - MDMA_IT_BRT : Block Repeat Transfer interrupt mask
  - MDMA_IT_BT : Block Transfer interrupt mask
  - MDMA_IT_BFTC : BuFfer Transfer Complete interrupt mask

Return value:

- None

__HAL_MDMA_GET_IT_SOURCE

Description:

- Checks whether the specified MDMA Channel interrupt is enabled or not.

Parameters:

- __HANDLE__: MDMA handle
- __INTERRUPT__: specifies the MDMA interrupt source to check.
  - MDMA_IT_TE : Transfer Error interrupt mask
  - MDMA_IT_CTC : Channel Transfer Complete interrupt mask
  - MDMA_IT_BRT : Block Repeat Transfer interrupt mask
  - MDMA_IT_BT : Block Transfer interrupt mask
  - MDMA_IT_BFTC : BuFfer Transfer Complete interrupt mask

Return value:

- The: state of MDMA_IT (SET or RESET).

__HAL_MDMA_SET_COUNTER

Description:

- Writes the number of data in bytes to be transferred on the MDMA Channelx.

Parameters:

- __HANDLE__: : MDMA handle
- __COUNTER__: Number of data in bytes to be transferred.

Return value:

- None

__HAL_MDMA_GET_COUNTER

Description:

- Returns the number of remaining data in bytes in the current MDMA Channelx transfer.

Parameters:

- __HANDLE__: : MDMA handle

Return value:

- The: number of remaining data in bytes in the current MDMA Channelx transfer.

MDMA flag definitions

MDMA_FLAG_TE

Transfer Error flag

MDMA_FLAG_CTC

Channel Transfer Complete flag

MDMA_FLAG_BRT

Block Repeat Transfer complete flag

MDMA_FLAG_BT

Block Transfer complete flag

MDMA_FLAG_BFTC

BuFfer Transfer complete flag

MDMA_FLAG_CRQA

Channel request Active flag

MDMA interrupt enable definitions

MDMA_IT_TE

Transfer Error interrupt

MDMA_IT_CTC

Channel Transfer Complete interrupt

MDMA_IT_BRT

Block Repeat Transfer interrupt

MDMA_IT_BT

Block Transfer interrupt

MDMA_IT_BFTC

Buffer Transfer Complete interrupt

MDMA Priority level

MDMA_PRIORITY_LOW

Priority level: Low

MDMA_PRIORITY_MEDIUM

Priority level: Medium

MDMA_PRIORITY_HIGH

Priority level: High

MDMA_PRIORITY_VERY_HIGH

Priority level: Very High

MDMA Request selection

MDMA_REQUEST_DMA1_Stream0_TC

MDMA HW request is DMA1 Stream 0 Transfer Complete Flag

MDMA_REQUEST_DMA1_Stream1_TC

MDMA HW request is DMA1 Stream 1 Transfer Complete Flag

MDMA_REQUEST_DMA1_Stream2_TC

MDMA HW request is DMA1 Stream 2 Transfer Complete Flag

MDMA_REQUEST_DMA1_Stream3_TC

MDMA HW request is DMA1 Stream 3 Transfer Complete Flag

MDMA_REQUEST_DMA1_Stream4_TC

MDMA HW request is DMA1 Stream 4 Transfer Complete Flag

MDMA_REQUEST_DMA1_Stream5_TC

MDMA HW request is DMA1 Stream 5 Transfer Complete Flag

MDMA_REQUEST_DMA1_Stream6_TC

MDMA HW request is DMA1 Stream 6 Transfer Complete Flag

MDMA_REQUEST_DMA1_Stream7_TC

MDMA HW request is DMA1 Stream 7 Transfer Complete Flag

MDMA_REQUEST_DMA2_Stream0_TC

MDMA HW request is DMA2 Stream 0 Transfer Complete Flag

MDMA_REQUEST_DMA2_Stream1_TC

MDMA HW request is DMA2 Stream 1 Transfer Complete Flag

MDMA_REQUEST_DMA2_Stream2_TC

MDMA HW request is DMA2 Stream 2 Transfer Complete Flag

MDMA_REQUEST_DMA2_Stream3_TC

MDMA HW request is DMA2 Stream 3 Transfer Complete Flag

MDMA_REQUEST_DMA2_Stream4_TC

MDMA HW request is DMA2 Stream 4 Transfer Complete Flag

MDMA_REQUEST_DMA2_Stream5_TC

MDMA HW request is DMA2 Stream 5 Transfer Complete Flag

MDMA_REQUEST_DMA2_Stream6_TC

MDMA HW request is DMA2 Stream 6 Transfer Complete Flag

MDMA_REQUEST_DMA2_Stream7_TC

MDMA HW request is DMA2 Stream 7 Transfer Complete Flag

MDMA_REQUEST_LTDC_LINE_IT

MDMA HW request is LTDC Line interrupt Flag

MDMA_REQUEST_JPEG_INFIFO_TH

MDMA HW request is JPEG Input FIFO threshold Flag

MDMA_REQUEST_JPEG_INFIFO_NF

MDMA HW request is JPEG Input FIFO not full Flag

MDMA_REQUEST_JPEG_OUTFIFO_TH

MDMA HW request is JPEG Output FIFO threshold Flag

MDMA_REQUEST_JPEG_OUTFIFO_NE

MDMA HW request is JPEG Output FIFO not empty Flag

MDMA_REQUEST_JPEG_END_CONVERSION

MDMA HW request is JPEG End of conversion Flag

MDMA_REQUEST_QUADSPI_FIFO_TH

MDMA HW request is QSPI FIFO threshold Flag

MDMA_REQUEST_QUADSPI_TC

MDMA HW request is QSPI Transfer complete Flag

MDMA_REQUEST_DMA2D_CLUT_TC

MDMA HW request is DMA2D CLUT Transfer Complete Flag

MDMA_REQUEST_DMA2D_TC

MDMA HW request is DMA2D Transfer Complete Flag

MDMA_REQUEST_DMA2D_TW

MDMA HW request is DMA2D Transfer Watermark Flag

MDMA_REQUEST_SDMMC1_END_DATA

MDMA HW request is SDMMC1 End of Data Flag

MDMA_REQUEST_SDMMC1_DMA_ENDBUFFER

MDMA HW request is SDMMC1 Internal DMA buffer End Flag

MDMA_REQUEST_SDMMC1_COMMAND_END

MDMA HW request is SDMMC1 Command End Flag

MDMA_REQUEST_SW

MDMA SW request

MDMA Source burst

MDMA_SOURCE_BURST_SINGLE

single transfer

MDMA_SOURCE_BURST_2BEATS

Burst 2 beats

MDMA_SOURCE_BURST_4BEATS

Burst 4 beats

MDMA_SOURCE_BURST_8BEATS

Burst 8 beats

MDMA_SOURCE_BURST_16BEATS

Burst 16 beats

MDMA_SOURCE_BURST_32BEATS

Burst 32 beats

MDMA_SOURCE_BURST_64BEATS

Burst 64 beats

MDMA_SOURCE_BURST_128BEATS

Burst 128 beats

MDMA Source data size

MDMA_SRC_DATASIZE_BYTE

Source data size is Byte

MDMA_SRC_DATASIZE_HALFWORD

Source data size is half word

MDMA_SRC_DATASIZE_WORD

Source data size is word

MDMA_SRC_DATASIZE_DOUBLEWORD

Source data size is double word

MDMA Source increment mode

MDMA_SRC_INC_DISABLE

Source address pointer is fixed

MDMA_SRC_INC_BYTE

Source address pointer is incremented by a BYTE (8 bits)

MDMA_SRC_INC_HALFWORD

Source address pointer is incremented by a half Word (16 bits)

MDMA_SRC_INC_WORD

Source address pointer is incremented by a Word (32 bits)

MDMA_SRC_INC_DOUBLEWORD

Source address pointer is incremented by a double Word (64 bits))

MDMA_SRC_DEC_BYTE

Source address pointer is decremented by a BYTE (8 bits)

MDMA_SRC_DEC_HALFWORD

Source address pointer is decremented by a half Word (16 bits)

MDMA_SRC_DEC_WORD

Source address pointer is decremented by a Word (32 bits)

MDMA_SRC_DEC_DOUBLEWORD

Source address pointer is decremented by a double Word (64 bits))

MDMA Transfer Trigger Mode

MDMA_BUFFER_TRANSFER

Each MDMA request (SW or HW) triggers a buffer transfer

MDMA_BLOCK_TRANSFER

Each MDMA request (SW or HW) triggers a block transfer

MDMA_REPEAT_BLOCK_TRANSFER

Each MDMA request (SW or HW) triggers a repeated block transfer

MDMA_FULL_TRANSFER

Each MDMA request (SW or HW) triggers a Full transfer or a linked list transfer if any

53 HAL MMC Generic Driver

53.1 MMC Firmware driver registers structures

53.1.1 HAL_MMC_CardInfoTypeDef

HAL_MMC_CardInfoTypeDef is defined in the stm32h7xx_hal_mmc.h

Data Fields

- ***uint32_t CardType***
- ***uint32_t Class***
- ***uint32_t RelCardAdd***
- ***uint32_t BlockNbr***
- ***uint32_t BlockSize***
- ***uint32_t LogBlockNbr***
- ***uint32_t LogBlockSize***

Field Documentation

- ***uint32_t HAL_MMC_CardInfoTypeDef::CardType***
  Specifies the card Type
- ***uint32_t HAL_MMC_CardInfoTypeDef::Class***
  Specifies the class of the card class
- ***uint32_t HAL_MMC_CardInfoTypeDef::RelCardAdd***
  Specifies the Relative Card Address
- ***uint32_t HAL_MMC_CardInfoTypeDef::BlockNbr***
  Specifies the Card Capacity in blocks
- ***uint32_t HAL_MMC_CardInfoTypeDef::BlockSize***
  Specifies one block size in bytes
- ***uint32_t HAL_MMC_CardInfoTypeDef::LogBlockNbr***
  Specifies the Card logical Capacity in blocks
- ***uint32_t HAL_MMC_CardInfoTypeDef::LogBlockSize***
  Specifies logical block size in bytes

53.1.2 __MMC_HandleTypeDef

__MMC_HandleTypeDef is defined in the stm32h7xx_hal_mmc.h

Data Fields

- ***MMC_TypeDef * Instance***
- ***MMC_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***const uint8_t * pTxBuffPtr***
- ***uint32_t TxXferSize***
- ***uint8_t * pRxBuffPtr***
- ***uint32_t RxXferSize***
- ***__IO uint32_t Context***
- ***__IO HAL_MMC_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***HAL_MMC_CardInfoTypeDef MmcCard***
- ***uint32_t CSD***
- ***uint32_t CID***

- ***uint32_t Ext_CSD***
- ***void(* TxCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* Read_DMADblBuf0CpltCallback***
- ***void(* Read_DMADblBuf1CpltCallback***
- ***void(* Write_DMADblBuf0CpltCallback***
- ***void(* Write_DMADblBuf1CpltCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***MMC_TypeDef* __MMC_HandleTypeDef::Instance***
  MMC registers base address
- ***MMC_InitTypeDef __MMC_HandleTypeDef::Init***
  MMC required parameters
- ***HAL_LockTypeDef __MMC_HandleTypeDef::Lock***
  MMC locking object
- ***const uint8_t* __MMC_HandleTypeDef::pTxBuffPtr***
  Pointer to MMC Tx transfer Buffer
- ***uint32_t __MMC_HandleTypeDef::TxXferSize***
  MMC Tx Transfer size
- ***uint8_t* __MMC_HandleTypeDef::pRxBuffPtr***
  Pointer to MMC Rx transfer Buffer
- ***uint32_t __MMC_HandleTypeDef::RxXferSize***
  MMC Rx Transfer size
- ***__IO uint32_t __MMC_HandleTypeDef::Context***
  MMC transfer context
- ***__IO HAL_MMC_StateTypeDef __MMC_HandleTypeDef::State***
  MMC card State
- ***__IO uint32_t __MMC_HandleTypeDef::ErrorCode***
  MMC Card Error codes
- ***HAL_MMC_CardInfoTypeDef __MMC_HandleTypeDef::MmcCard***
  MMC Card information
- ***uint32_t __MMC_HandleTypeDef::CSD[4U]***
  MMC card specific data table
- ***uint32_t __MMC_HandleTypeDef::CID[4U]***
  MMC card identification number table
- ***uint32_t __MMC_HandleTypeDef::Ext_CSD[128]***
- ***void(* __MMC_HandleTypeDef::TxCpltCallback)(struct __MMC_HandleTypeDef *hmmc)***
- ***void(* __MMC_HandleTypeDef::RxCpltCallback)(struct __MMC_HandleTypeDef *hmmc)***
- ***void(* __MMC_HandleTypeDef::ErrorCallback)(struct __MMC_HandleTypeDef *hmmc)***
- ***void(* __MMC_HandleTypeDef::AbortCpltCallback)(struct __MMC_HandleTypeDef *hmmc)***
- ***void(* __MMC_HandleTypeDef::Read_DMADblBuf0CpltCallback)(struct __MMC_HandleTypeDef *hmmc)***
- ***void(* __MMC_HandleTypeDef::Read_DMADblBuf1CpltCallback)(struct __MMC_HandleTypeDef *hmmc)***

- *void(* __MMC_HandleTypeDef::Write_DMADblBuf0CpltCallback)(struct __MMC_HandleTypeDef *hmmc)*
- *void(* __MMC_HandleTypeDef::Write_DMADblBuf1CpltCallback)(struct __MMC_HandleTypeDef *hmmc)*
- *void(* __MMC_HandleTypeDef::MspInitCallback)(struct __MMC_HandleTypeDef *hmmc)*
- *void(* __MMC_HandleTypeDef::MspDeInitCallback)(struct __MMC_HandleTypeDef *hmmc)*

53.1.3 HAL_MMC_CardCSDTypeDef

HAL_MMC_CardCSDTypeDef is defined in the stm32h7xx_hal_mmc.h

Data Fields

- *__IO uint8_t CSDStruct*
- *__IO uint8_t SysSpecVersion*
- *__IO uint8_t Reserved1*
- *__IO uint8_t TAAC*
- *__IO uint8_t NSAC*
- *__IO uint8_t MaxBusClkFrec*
- *__IO uint16_t CardComdClasses*
- *__IO uint8_t RdBlockLen*
- *__IO uint8_t PartBlockRead*
- *__IO uint8_t WrBlockMisalign*
- *__IO uint8_t RdBlockMisalign*
- *__IO uint8_t DSRImpl*
- *__IO uint8_t Reserved2*
- *__IO uint32_t DeviceSize*
- *__IO uint8_t MaxRdCurrentVDDMin*
- *__IO uint8_t MaxRdCurrentVDDMax*
- *__IO uint8_t MaxWrCurrentVDDMin*
- *__IO uint8_t MaxWrCurrentVDDMax*
- *__IO uint8_t DeviceSizeMul*
- *__IO uint8_t EraseGrSize*
- *__IO uint8_t EraseGrMul*
- *__IO uint8_t WrProtectGrSize*
- *__IO uint8_t WrProtectGrEnable*
- *__IO uint8_t ManDeflECC*
- *__IO uint8_t WrSpeedFact*
- *__IO uint8_t MaxWrBlockLen*
- *__IO uint8_t WriteBlockPaPartial*
- *__IO uint8_t Reserved3*
- *__IO uint8_t ContentProtectAppli*
- *__IO uint8_t FileFormatGroup*
- *__IO uint8_t CopyFlag*
- *__IO uint8_t PermWrProtect*
- *__IO uint8_t TempWrProtect*
- *__IO uint8_t FileFormat*
- *__IO uint8_t ECC*
- *__IO uint8_t CSD_CRC*
- *__IO uint8_t Reserved4*

Field Documentation

- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::CSDStruct***
  CSD structure
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::SysSpecVersion***
  System specification version
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::Reserved1***
  Reserved
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::TAAC***
  Data read access time 1
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::NSAC***
  Data read access time 2 in CLK cycles
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxBusClkFrec***
  Max. bus clock frequency
- ***__IO uint16_t HAL_MMC_CardCSDTypeDef::CardComdClasses***
  Card command classes
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::RdBlockLen***
  Max. read data block length
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::PartBlockRead***
  Partial blocks for read allowed
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::WrBlockMisalign***
  Write block misalignment
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::RdBlockMisalign***
  Read block misalignment
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::DSRImpl***
  DSR implemented
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::Reserved2***
  Reserved
- ***__IO uint32_t HAL_MMC_CardCSDTypeDef::DeviceSize***
  Device Size
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxRdCurrentVDDMin***
  Max. read current @ VDD min
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxRdCurrentVDDMax***
  Max. read current @ VDD max
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxWrCurrentVDDMin***
  Max. write current @ VDD min
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxWrCurrentVDDMax***
  Max. write current @ VDD max
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::DeviceSizeMul***
  Device size multiplier
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::EraseGrSize***
  Erase group size
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::EraseGrMul***
  Erase group size multiplier
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::WrProtectGrSize***
  Write protect group size
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::WrProtectGrEnable***
  Write protect group enable
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::ManDeflECC***
  Manufacturer default ECC
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::WrSpeedFact***
  Write speed factor

- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxWrBlockLen***
  Max. write data block length
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::WriteBlockPaPartial***
  Partial blocks for write allowed
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::Reserved3***
  Reserved
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::ContentProtectAppli***
  Content protection application
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::FileFormatGroup***
  File format group
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::CopyFlag***
  Copy flag (OTP)
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::PermWrProtect***
  Permanent write protection
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::TempWrProtect***
  Temporary write protection
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::FileFormat***
  File format
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::ECC***
  ECC code
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::CSD_CRC***
  CSD CRC
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::Reserved4***
  Always 1

53.1.4 HAL_MMC_CardCIDTypeDef

HAL_MMC_CardCIDTypeDef is defined in the stm32h7xx_hal_mmc.h

Data Fields

- ***__IO uint8_t ManufacturerID***
- ***__IO uint16_t OEM_AppliID***
- ***__IO uint32_t ProdName1***
- ***__IO uint8_t ProdName2***
- ***__IO uint8_t ProdRev***
- ***__IO uint32_t ProdSN***
- ***__IO uint8_t Reserved1***
- ***__IO uint16_t ManufactDate***
- ***__IO uint8_t CID_CRC***
- ***__IO uint8_t Reserved2***

Field Documentation

- ***__IO uint8_t HAL_MMC_CardCIDTypeDef::ManufacturerID***
  Manufacturer ID
- ***__IO uint16_t HAL_MMC_CardCIDTypeDef::OEM_AppliID***
  OEM/Application ID
- ***__IO uint32_t HAL_MMC_CardCIDTypeDef::ProdName1***
  Product Name part1
- ***__IO uint8_t HAL_MMC_CardCIDTypeDef::ProdName2***
  Product Name part2
- ***__IO uint8_t HAL_MMC_CardCIDTypeDef::ProdRev***
  Product Revision

- ***__IO uint32_t HAL_MMC_CardCIDTypeDef::ProdSN***
  Product Serial Number
- ***__IO uint8_t HAL_MMC_CardCIDTypeDef::Reserved1***
  Reserved1
- ***__IO uint16_t HAL_MMC_CardCIDTypeDef::ManufactDate***
  Manufacturing Date
- ***__IO uint8_t HAL_MMC_CardCIDTypeDef::CID_CRC***
  CID CRC
- ***__IO uint8_t HAL_MMC_CardCIDTypeDef::Reserved2***
  Always 1

53.2 MMC Firmware driver API description

The following section lists the various functions of the MMC library.

53.2.1 How to use this driver

This driver implements a high level communication layer for read and write from/to this memory. The needed STM32 hardware resources (SDMMC and GPIO) are performed by the user in HAL_MMC_MspInit() function (MSP layer). Basically, the MSP layer configuration should be the same as we provide in the examples. You can easily tailor this configuration according to hardware resources.

This driver is a generic layered driver for SDMMC memories which uses the HAL SDMMC driver functions to interface with MMC and eMMC cards devices. It is used as follows:

1. Initialize the SDMMC low level resources by implement the HAL_MMC_MspInit() API:
   a. Enable the SDMMC interface clock using __HAL_RCC_SDMMC_CLK_ENABLE();
   b. SDMMC pins configuration for MMC card
      - Enable the clock for the SDMMC GPIOs using the functions __HAL_RCC_GPIOx_CLK_ENABLE();
      - Configure these SDMMC pins as alternate function pull-up using HAL_GPIO_Init() and according to your pin assignment;
   c. NVIC configuration if you need to use interrupt process (HAL_MMC_ReadBlocks_IT() and HAL_MMC_WriteBlocks_IT() APIs).
      - Configure the SDMMC interrupt priorities using function HAL_NVIC_SetPriority();
      - Enable the NVIC SDMMC IRQs using function HAL_NVIC_EnableIRQ()
      - SDMMC interrupts are managed using the macros __HAL_MMC_ENABLE_IT() and __HAL_MMC_DISABLE_IT() inside the communication process.
      - SDMMC interrupts pending bits are managed using the macros __HAL_MMC_GET_IT() and __HAL_MMC_CLEAR_IT()
   d. No general propose DMA Configuration is needed, an Internal DMA for SDMMC Peripheral are used.
2. At this stage, you can perform MMC read/write/erase operations after MMC card initialization

MMC Card Initialization and configuration

To initialize the MMC Card, use the HAL_MMC_Init() function. It Initializes SDMMC Peripheral (STM32 side) and the MMC Card, and put it into StandBy State (Ready for data transfer). This function provide the following operations:

1. Initialize the SDMMC peripheral interface with default configuration. The initialization process is done at 400KHz. You can change or adapt this frequency by adjusting the "ClockDiv" field. The MMC Card frequency (SDMMC_CK) is computed as follows: SDMMC_CK = SDMMCCLK / (2 * ClockDiv) In initialization mode and according to the MMC Card standard, make sure that the SDMMC_CK frequency doesn't exceed 400KHz. This phase of initialization is done through SDMMC_Init() and SDMMC_PowerState_ON() SDMMC low level APIs.

2. Initialize the MMC card. The API used is HAL_MMC_InitCard(). This phase allows the card initialization and identification and check the MMC Card type (Standard Capacity or High Capacity) The initialization flow is compatible with MMC standard. This API (HAL_MMC_InitCard()) could be used also to reinitialize the card in case of plug-off plug-in.
3. Configure the MMC Card Data transfer frequency. By Default, the card transfer frequency by adjusting the "ClockDiv" field. In transfer mode and according to the MMC Card standard, make sure that the SDMMC_CK frequency doesn't exceed 25MHz and 100MHz in High-speed mode switch.
4. Select the corresponding MMC Card according to the address read with the step 2.
5. Configure the MMC Card in wide bus mode: 4-bits data.

MMC Card Read operation

- You can read from MMC card in polling mode by using function HAL_MMC_ReadBlocks(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetCardState() function for MMC card state.
- You can read from MMC card in DMA mode by using function HAL_MMC_ReadBlocks_DMA(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetCardState() function for MMC card state. You could also check the DMA transfer process through the MMC Rx interrupt event.
- You can read from MMC card in Interrupt mode by using function HAL_MMC_ReadBlocks_IT(). This function allows the read of 512 bytes blocks. You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetCardState() function for MMC card state. You could also check the IT transfer process through the MMC Rx interrupt event.

MMC Card Write operation

- You can write to MMC card in polling mode by using function HAL_MMC_WriteBlocks(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetCardState() function for MMC card state.
- You can write to MMC card in DMA mode by using function HAL_MMC_WriteBlocks_DMA(). This function support only 512-bytes block length (the block size should be chosen as 512 byte). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetCardState() function for MMC card state. You could also check the DMA transfer process through the MMC Tx interrupt event.
- You can write to MMC card in Interrupt mode by using function HAL_MMC_WriteBlocks_IT(). This function allows the read of 512 bytes blocks. You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetCardState() function for MMC card state. You could also check the IT transfer process through the MMC Tx interrupt event.

MMC card information

- To get MMC card information, you can use the function HAL_MMC_GetCardInfo(). It returns useful information about the MMC card such as block size, card type, block number ...

MMC card CSD register

- The HAL_MMC_GetCardCSD() API allows to get the parameters of the CSD register. Some of the CSD parameters are useful for card initialization and identification.

MMC card CID register

- The HAL_MMC_GetCardCID() API allows to get the parameters of the CID register. Some of the CID parameters are useful for card initialization and identification.

MMC HAL driver macros list

Below the list of most used macros in MMC HAL driver.

- __HAL_MMC_ENABLE_IT: Enable the MMC device interrupt
- __HAL_MMC_DISABLE_IT: Disable the MMC device interrupt
- __HAL_MMC_GET_FLAG:Check whether the specified MMC flag is set or not
- __HAL_MMC_CLEAR_FLAG: Clear the MMC's pending flags

Note: You can refer to the MMC HAL driver header file for more useful macros

Callback registration

The compilation define USE_HAL_MMC_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_MMC_RegisterCallback() to register a user callback, it allows to register following callbacks:

- TxCpltCallback : callback when a transmission transfer is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- Read_DMADblBuf0CpltCallback : callback when the DMA reception of first buffer is completed.
- Read_DMADblBuf1CpltCallback : callback when the DMA reception of second buffer is completed.
- Write_DMADblBuf0CpltCallback : callback when the DMA transmission of first buffer is completed.
- Write_DMADblBuf1CpltCallback : callback when the DMA transmission of second buffer is completed.
- MspInitCallback : MMC MspInit.
- MspDeInitCallback : MMC MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. Use function HAL_MMC_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. It allows to reset following callbacks:
- TxCpltCallback : callback when a transmission transfer is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- Read_DMADblBuf0CpltCallback : callback when the DMA reception of first buffer is completed.
- Read_DMADblBuf1CpltCallback : callback when the DMA reception of second buffer is completed.
- Write_DMADblBuf0CpltCallback : callback when the DMA transmission of first buffer is completed.
- Write_DMADblBuf1CpltCallback : callback when the DMA transmission of second buffer is completed.
- MspInitCallback : MMC MspInit.
- MspDeInitCallback : MMC MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the HAL_MMC_Init and if the state is HAL_MMC_STATE_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the HAL_MMC_Init and HAL_MMC_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_MMC_Init and HAL_MMC_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/ unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_MMC_RegisterCallback before calling HAL_MMC_DeInit or HAL_MMC_Init function. When The compilation define USE_HAL_MMC_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

53.2.2 Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the MMC card device to be ready for use.

This section contains the following APIs:

- *HAL_MMC_Init()*
- *HAL_MMC_InitCard()*
- *HAL_MMC_DeInit()*
- *HAL_MMC_MspInit()*
- *HAL_MMC_MspDeInit()*

53.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the data transfer from/to MMC card.

This section contains the following APIs:

- *HAL_MMC_ReadBlocks()*
- *HAL_MMC_WriteBlocks()*
- *HAL_MMC_ReadBlocks_IT()*
- *HAL_MMC_WriteBlocks_IT()*
- *HAL_MMC_ReadBlocks_DMA()*
- *HAL_MMC_WriteBlocks_DMA()*
- *HAL_MMC_Erase()*
- *HAL_MMC_IRQHandler()*
- *HAL_MMC_GetState()*
- *HAL_MMC_GetError()*
- *HAL_MMC_TxCpltCallback()*
- *HAL_MMC_RxCpltCallback()*
- *HAL_MMC_ErrorCallback()*
- *HAL_MMC_AbortCallback()*
- *HAL_MMC_RegisterCallback()*
- *HAL_MMC_UnRegisterCallback()*

53.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the MMC card operations and get the related information

This section contains the following APIs:

- *HAL_MMC_GetCardCID()*
- *HAL_MMC_GetCardCSD()*
- *HAL_MMC_GetCardInfo()*
- *HAL_MMC_GetCardExtCSD()*
- *HAL_MMC_ConfigWideBusOperation()*
- *HAL_MMC_ConfigSpeedBusOperation()*
- *HAL_MMC_GetCardState()*
- *HAL_MMC_Abort()*
- *HAL_MMC_Abort_IT()*
- *HAL_MMC_EraseSequence()*
- *HAL_MMC_Sanitize()*
- *HAL_MMC_ConfigSecRemovalType()*
- *HAL_MMC_GetSupportedSecRemovalType()*
- *HAL_MMC_SleepDevice()*
- *HAL_MMC_AwakeDevice()*

53.2.5 Detailed description of functions

HAL_MMC_Init

Function name

HAL_StatusTypeDef HAL_MMC_Init (MMC_HandleTypeDef * hmmc)

Function description

Initializes the MMC according to the specified parameters in the MMC_HandleTypeDef and create the associated handle.

Parameters

- **hmmc:** Pointer to the MMC handle

Return values

- **HAL:** status

HAL_MMC_InitCard

Function name

HAL_StatusTypeDef HAL_MMC_InitCard (MMC_HandleTypeDef * hmmc)

Function description

Initializes the MMC Card.

Parameters

- **hmmc:** Pointer to MMC handle

Return values

- **HAL:** status

Notes

- This function initializes the MMC card. It could be used when a card re-initialization is needed.

HAL_MMC_DeInit

Function name

HAL_StatusTypeDef HAL_MMC_DeInit (MMC_HandleTypeDef * hmmc)

Function description

De-Initializes the MMC card.

Parameters

- **hmmc:** Pointer to MMC handle

Return values

- **HAL:** status

HAL_MMC_MspInit

Function name

void HAL_MMC_MspInit (MMC_HandleTypeDef * hmmc)

Function description

Initializes the MMC MSP.

Parameters

- **hmmc:** Pointer to MMC handle

Return values

- **None:**

HAL_MMC_MspDeInit

Function name

void HAL_MMC_MspDeInit (MMC_HandleTypeDef * hmmc)

Function description

De-Initialize MMC MSP.

Parameters

- **hmmc:** Pointer to MMC handle

Return values

- **None:**

HAL_MMC_ReadBlocks

Function name

HAL_StatusTypeDef HAL_MMC_ReadBlocks (MMC_HandleTypeDef * hmmc, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks, uint32_t Timeout)

Function description

Reads block(s) from a specified address in a card.

Parameters

- **hmmc:** Pointer to MMC handle
- **pData:** pointer to the buffer that will contain the received data
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Number of MMC blocks to read
- **Timeout:** Specify timeout value

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().

HAL_MMC_WriteBlocks

Function name

HAL_StatusTypeDef HAL_MMC_WriteBlocks (MMC_HandleTypeDef * hmmc, const uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks, uint32_t Timeout)

Function description

Allows to write block(s) to a specified address in a card.

Parameters

- **hmmc:** Pointer to MMC handle
- **pData:** pointer to the buffer that will contain the data to transmit
- **BlockAdd:** Block Address where data will be written
- **NumberOfBlocks:** Number of MMC blocks to write
- **Timeout:** Specify timeout value

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().

HAL_MMC_Erase

Function name

HAL_StatusTypeDef HAL_MMC_Erase (MMC_HandleTypeDef * hmmc, uint32_t BlockStartAdd, uint32_t BlockEndAdd)

Function description

Erases the specified memory area of the given MMC card.

Parameters

- **hmmc:** Pointer to MMC handle
- **BlockStartAdd:** Start Block address
- **BlockEndAdd:** End Block address

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().

HAL_MMC_ReadBlocks_IT

Function name

HAL_StatusTypeDef HAL_MMC_ReadBlocks_IT (MMC_HandleTypeDef * hmmc, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Reads block(s) from a specified address in a card.

Parameters

- **hmmc:** Pointer to MMC handle
- **pData:** Pointer to the buffer that will contain the received data
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Number of blocks to read.

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().
- You could also check the IT transfer process through the MMC Rx interrupt event.

HAL_MMC_WriteBlocks_IT

Function name

HAL_StatusTypeDef HAL_MMC_WriteBlocks_IT (MMC_HandleTypeDef * hmmc, const uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Writes block(s) to a specified address in a card.

Parameters

- **hmmc:** Pointer to MMC handle
- **pData:** Pointer to the buffer that will contain the data to transmit
- **BlockAdd:** Block Address where data will be written
- **NumberOfBlocks:** Number of blocks to write

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().
- You could also check the IT transfer process through the MMC Tx interrupt event.

HAL_MMC_ReadBlocks_DMA

Function name

HAL_StatusTypeDef HAL_MMC_ReadBlocks_DMA (MMC_HandleTypeDef * hmmc, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Reads block(s) from a specified address in a card.

Parameters

- **hmmc:** Pointer MMC handle
- **pData:** Pointer to the buffer that will contain the received data
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Number of blocks to read.

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().
- You could also check the DMA transfer process through the MMC Rx interrupt event.

HAL_MMC_WriteBlocks_DMA

Function name

HAL_StatusTypeDef HAL_MMC_WriteBlocks_DMA (MMC_HandleTypeDef * hmmc, const uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Writes block(s) to a specified address in a card.

Parameters

- **hmmc:** Pointer to MMC handle
- **pData:** Pointer to the buffer that will contain the data to transmit
- **BlockAdd:** Block Address where data will be written
- **NumberOfBlocks:** Number of blocks to write

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().
- You could also check the DMA transfer process through the MMC Tx interrupt event.

HAL_MMC_IRQHandler

Function name

void HAL_MMC_IRQHandler (MMC_HandleTypeDef * hmmc)

Function description

This function handles MMC card interrupt request.

Parameters

- **hmmc:** Pointer to MMC handle

Return values

- **None:**

HAL_MMC_TxCpltCallback

Function name

void HAL_MMC_TxCpltCallback (MMC_HandleTypeDef * hmmc)

Function description

Tx Transfer completed callbacks.

Parameters

- **hmmc:** Pointer to MMC handle

Return values

- **None:**

HAL_MMC_RxCpltCallback

Function name

void HAL_MMC_RxCpltCallback (MMC_HandleTypeDef * hmmc)

Function description

Rx Transfer completed callbacks.

Parameters

- **hmmc:** Pointer MMC handle

Return values

- **None:**

HAL_MMC_ErrorCallback

Function name

void HAL_MMC_ErrorCallback (MMC_HandleTypeDef * hmmc)

Function description

MMC error callbacks.

Parameters

- **hmmc:** Pointer MMC handle

Return values

- **None:**

HAL_MMC_AbortCallback

Function name

void HAL_MMC_AbortCallback (MMC_HandleTypeDef * hmmc)

Function description

MMC Abort callbacks.

Parameters

- **hmmc:** Pointer MMC handle

Return values

- **None:**

HAL_MMC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_MMC_RegisterCallback (MMC_HandleTypeDef * hmmc, HAL_MMC_CallbackIDTypeDef CallbackId, pMMC_CallbackTypeDef pCallback)

Function description

Register a User MMC Callback To be used instead of the weak (surcharged) predefined callback.

Parameters

- **hmmc:** : MMC handle
- **CallbackId:** : ID of the callback to be registered This parameter can be one of the following values:
  - HAL_MMC_TX_CPLT_CB_ID MMC Tx Complete Callback ID
  - HAL_MMC_RX_CPLT_CB_ID MMC Rx Complete Callback ID
  - HAL_MMC_ERROR_CB_ID MMC Error Callback ID
  - HAL_MMC_ABORT_CB_ID MMC Abort Callback ID
  - HAL_MMC_READ_DMA_DBL_BUF0_CPLT_CB_ID MMC DMA Rx Double buffer 0 Callback ID
  - HAL_MMC_READ_DMA_DBL_BUF1_CPLT_CB_ID MMC DMA Rx Double buffer 1 Callback ID
  - HAL_MMC_WRITE_DMA_DBL_BUF0_CPLT_CB_ID MMC DMA Tx Double buffer 0 Callback ID
  - HAL_MMC_WRITE_DMA_DBL_BUF1_CPLT_CB_ID MMC DMA Tx Double buffer 1 Callback ID
  - HAL_MMC_MSP_INIT_CB_ID MMC MspInit Callback ID
  - HAL_MMC_MSP_DEINIT_CB_ID MMC MspDeInit Callback ID
- **pCallback:** : pointer to the Callback function

Return values

- **status:**

Notes

- The HAL_MMC_RegisterCallback() may be called before HAL_MMC_Init() in HAL_MMC_STATE_RESET to register callbacks for HAL_MMC_MSP_INIT_CB_ID and HAL_MMC_MSP_DEINIT_CB_ID.

HAL_MMC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_MMC_UnRegisterCallback (MMC_HandleTypeDef * hmmc, HAL_MMC_CallbackIDTypeDef CallbackId)

Function description

Unregister a User MMC Callback MMC Callback is redirected to the weak (surcharged) predefined callback.

Parameters

- **hmmc:** : MMC handle
- **CallbackId:** : ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_MMC_TX_CPLT_CB_ID MMC Tx Complete Callback ID
  - HAL_MMC_RX_CPLT_CB_ID MMC Rx Complete Callback ID
  - HAL_MMC_ERROR_CB_ID MMC Error Callback ID
  - HAL_MMC_ABORT_CB_ID MMC Abort Callback ID
  - HAL_MMC_READ_DMA_DBL_BUF0_CPLT_CB_ID MMC DMA Rx Double buffer 0 Callback ID
  - HAL_MMC_READ_DMA_DBL_BUF1_CPLT_CB_ID MMC DMA Rx Double buffer 1 Callback ID
  - HAL_MMC_WRITE_DMA_DBL_BUF0_CPLT_CB_ID MMC DMA Tx Double buffer 0 Callback ID
  - HAL_MMC_WRITE_DMA_DBL_BUF1_CPLT_CB_ID MMC DMA Tx Double buffer 1 Callback ID
  - HAL_MMC_MSP_INIT_CB_ID MMC MspInit Callback ID
  - HAL_MMC_MSP_DEINIT_CB_ID MMC MspDeInit Callback ID

Return values

- **status:**

Notes

- The HAL_MMC_UnRegisterCallback() may be called before HAL_MMC_Init() in HAL_MMC_STATE_RESET to register callbacks for HAL_MMC_MSP_INIT_CB_ID and HAL_MMC_MSP_DEINIT_CB_ID.

HAL_MMC_ConfigWideBusOperation

Function name

HAL_StatusTypeDef HAL_MMC_ConfigWideBusOperation (MMC_HandleTypeDef * hmmc, uint32_t WideMode)

Function description

Enables wide bus operation for the requested card if supported by card.

Parameters

- **hmmc:** Pointer to MMC handle
- **WideMode:** Specifies the MMC card wide bus mode This parameter can be one of the following values:
  - SDMMC_BUS_WIDE_8B: 8-bit data transfer
  - SDMMC_BUS_WIDE_4B: 4-bit data transfer
  - SDMMC_BUS_WIDE_1B: 1-bit data transfer

Return values

- **HAL:** status

HAL_MMC_ConfigSpeedBusOperation

Function name

HAL_StatusTypeDef HAL_MMC_ConfigSpeedBusOperation (MMC_HandleTypeDef * hmmc, uint32_t SpeedMode)

Function description

Configure the speed bus mode.

Parameters

- **hmmc:** Pointer to the MMC handle
- **SpeedMode:** Specifies the MMC card speed bus mode This parameter can be one of the following values:
  - SDMMC_SPEED_MODE_AUTO: Max speed mode supported by the card
  - SDMMC_SPEED_MODE_DEFAULT: Default Speed (MMC @ 26MHz)
  - SDMMC_SPEED_MODE_HIGH: High Speed (MMC @ 52 MHz)
  - SDMMC_SPEED_MODE_DDR: High Speed DDR (MMC DDR @ 52 MHz)

Return values

- **HAL:** status

HAL_MMC_GetCardState

Function name

HAL_MMC_CardStateTypeDef HAL_MMC_GetCardState (MMC_HandleTypeDef * hmmc)

Function description

Gets the current mmc card data state.

Parameters

- **hmmc:** pointer to MMC handle

Return values

- **Card:** state

HAL_MMC_GetCardCID

Function name

HAL_StatusTypeDef HAL_MMC_GetCardCID (MMC_HandleTypeDef * hmmc, HAL_MMC_CardCIDTypeDef * pCID)

Function description

Returns information the information of the card which are stored on the CID register.

Parameters

- **hmmc:** Pointer to MMC handle
- **pCID:** Pointer to a HAL_MMC_CIDTypedef structure that contains all CID register parameters

Return values

- **HAL:** status

HAL_MMC_GetCardCSD

Function name

HAL_StatusTypeDef HAL_MMC_GetCardCSD (MMC_HandleTypeDef * hmmc, HAL_MMC_CardCSDTypeDef * pCSD)

Function description

Returns information the information of the card which are stored on the CSD register.

Parameters

- **hmmc:** Pointer to MMC handle
- **pCSD:** Pointer to a HAL_MMC_CardCSDTypeDef structure that contains all CSD register parameters

Return values

- **HAL:** status

HAL_MMC_GetCardInfo

Function name

HAL_StatusTypeDef HAL_MMC_GetCardInfo (MMC_HandleTypeDef * hmmc, HAL_MMC_CardInfoTypeDef * pCardInfo)

Function description

Gets the MMC card info.

Parameters

- **hmmc:** Pointer to MMC handle
- **pCardInfo:** Pointer to the HAL_MMC_CardInfoTypeDef structure that will contain the MMC card status information

Return values

- **HAL:** status

HAL_MMC_GetCardExtCSD

Function name

HAL_StatusTypeDef HAL_MMC_GetCardExtCSD (MMC_HandleTypeDef * hmmc, uint32_t * pExtCSD, uint32_t Timeout)

Function description

Returns information the information of the card which are stored on the Extended CSD register.

Parameters

- **hmmc:** Pointer to MMC handle
- **pExtCSD:** Pointer to a memory area (512 bytes) that contains all Extended CSD register parameters
- **Timeout:** Specify timeout value

Return values

- **HAL:** status

HAL_MMC_GetState

Function name

HAL_MMC_StateTypeDef HAL_MMC_GetState (MMC_HandleTypeDef * hmmc)

Function description

return the MMC state

Parameters

- **hmmc:** Pointer to mmc handle

Return values

- **HAL:** state

HAL_MMC_GetError

Function name

uint32_t HAL_MMC_GetError (MMC_HandleTypeDef * hmmc)

Function description

Return the MMC error code.

Parameters

- **hmmc:** : Pointer to a MMC_HandleTypeDef structure that contains the configuration information.

Return values

- **MMC:** Error Code

HAL_MMC_Abort

Function name

HAL_StatusTypeDef HAL_MMC_Abort (MMC_HandleTypeDef * hmmc)

Function description

Abort the current transfer and disable the MMC.

Parameters

- **hmmc:** pointer to a MMC_HandleTypeDef structure that contains the configuration information for MMC module.

Return values

- **HAL:** status

HAL_MMC_Abort_IT

Function name

HAL_StatusTypeDef HAL_MMC_Abort_IT (MMC_HandleTypeDef * hmmc)

Function description

Abort the current transfer and disable the MMC (IT mode).

Parameters

- **hmmc:** pointer to a MMC_HandleTypeDef structure that contains the configuration information for MMC module.

Return values

- **HAL:** status

HAL_MMC_EraseSequence

Function name

HAL_StatusTypeDef HAL_MMC_EraseSequence (MMC_HandleTypeDef * hmmc, uint32_t EraseType, uint32_t BlockStartAdd, uint32_t BlockEndAdd)

Function description

Perform specific commands sequence for the different type of erase.

Parameters

- **hmmc:** Pointer to MMC handle
- **EraseType:** Specifies the type of erase to be performed This parameter can be one of the following values:
  - HAL_MMC_TRIM Erase the write blocks identified by CMD35 & 36
  - HAL_MMC_ERASE Erase the erase groups identified by CMD35 & 36
  - HAL_MMC_DISCARD Discard the write blocks identified by CMD35 & 36
  - HAL_MMC_SECURE_ERASE Perform a secure purge according SRT on the erase groups identified by CMD35 & 36
  - HAL_MMC_SECURE_TRIM_STEP1 Mark the write blocks identified by CMD35 & 36 for secure erase
  - HAL_MMC_SECURE_TRIM_STEP2 Perform a secure purge according SRT on the write blocks previously identified
- **BlockStartAdd:** Start Block address
- **BlockEndAdd:** End Block address

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().

HAL_MMC_Sanitize

Function name

HAL_StatusTypeDef HAL_MMC_Sanitize (MMC_HandleTypeDef * hmmc)

Function description

Perform sanitize operation on the device.

Parameters

- **hmmc:** Pointer to MMC handle

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().

HAL_MMC_ConfigSecRemovalType

Function name

HAL_StatusTypeDef HAL_MMC_ConfigSecRemovalType (MMC_HandleTypeDef * hmmc, uint32_t SRTMode)

Function description

Configure the Secure Removal Type (SRT) in the Extended CSD register.

Parameters

- **hmmc:** Pointer to MMC handle
- **SRTMode:** Specifies the type of erase to be performed This parameter can be one of the following values:
  - HAL_MMC_SRT_ERASE Information removed by an erase
  - HAL_MMC_SRT_WRITE_CHAR_ERASE Information removed by an overwriting with a character followed by an erase
  - HAL_MMC_SRT_WRITE_CHAR_COMPL_RANDOM Information removed by an overwriting with a character, its complement then a random character
  - HAL_MMC_SRT_VENDOR_DEFINED Information removed using a vendor defined

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().

HAL_MMC_GetSupportedSecRemovalType

Function name

HAL_StatusTypeDef HAL_MMC_GetSupportedSecRemovalType (MMC_HandleTypeDef * hmmc, uint32_t * SupportedSRT)

Function description

Gets the supported values of the the Secure Removal Type (SRT).

Parameters

- **hmmc:** pointer to MMC handle
- **SupportedSRT:** pointer for supported SRT value This parameter is a bit field of the following values:
  - HAL_MMC_SRT_ERASE Information removed by an erase
  - HAL_MMC_SRT_WRITE_CHAR_ERASE Information removed by an overwriting with a character followed by an erase
  - HAL_MMC_SRT_WRITE_CHAR_COMPL_RANDOM Information removed by an overwriting with a character, its complement then a random character
  - HAL_MMC_SRT_VENDOR_DEFINED Information removed using a vendor defined

Return values

- **HAL:** status

HAL_MMC_SleepDevice

Function name

HAL_StatusTypeDef HAL_MMC_SleepDevice (MMC_HandleTypeDef * hmmc)

Function description

Switch the device from Standby State to Sleep State.

Parameters

- **hmmc:** pointer to MMC handle

Return values

- **HAL:** status

HAL_MMC_AwakeDevice

Function name

HAL_StatusTypeDef HAL_MMC_AwakeDevice (MMC_HandleTypeDef * hmmc)

Function description

Switch the device from Sleep State to Standby State.

Parameters

- **hmmc:** pointer to MMC handle

Return values

- **HAL:** status

53.3 MMC Firmware driver defines

The following section lists the various define and macros of the module.

53.3.1 MMC

MMC

MMC Error status enumeration Structure definition

HAL_MMC_ERROR_NONE

No error

HAL_MMC_ERROR_CMD_CRC_FAIL

Command response received (but CRC check failed)

HAL_MMC_ERROR_DATA_CRC_FAIL

Data block sent/received (CRC check failed)

HAL_MMC_ERROR_CMD_RSP_TIMEOUT

Command response timeout

HAL_MMC_ERROR_DATA_TIMEOUT

Data timeout

HAL_MMC_ERROR_TX_UNDERRUN

Transmit FIFO underrun

HAL_MMC_ERROR_RX_OVERRUN

Receive FIFO overrun

HAL_MMC_ERROR_ADDR_MISALIGNED

Misaligned address

HAL_MMC_ERROR_BLOCK_LEN_ERR

Transferred block length is not allowed for the card or the number of transferred bytes does not match the block length

HAL_MMC_ERROR_ERASE_SEQ_ERR

An error in the sequence of erase command occurs

HAL_MMC_ERROR_BAD_ERASE_PARAM

An invalid selection for erase groups

HAL_MMC_ERROR_WRITE_PROT_VIOLATION

Attempt to program a write protect block

HAL_MMC_ERROR_LOCK_UNLOCK_FAILED

Sequence or password error has been detected in unlock command or if there was an attempt to access a locked card

HAL_MMC_ERROR_COM_CRC_FAILED

CRC check of the previous command failed

HAL_MMC_ERROR_ILLEGAL_CMD

Command is not legal for the card state

HAL_MMC_ERROR_CARD_ECC_FAILED

Card internal ECC was applied but failed to correct the data

HAL_MMC_ERROR_CC_ERR

Internal card controller error

HAL_MMC_ERROR_GENERAL_UNKNOWN_ERR

General or unknown error

HAL_MMC_ERROR_STREAM_READ_UNDERRUN

The card could not sustain data reading in stream rmode

HAL_MMC_ERROR_STREAM_WRITE_OVERRUN

The card could not sustain data programming in stream mode

HAL_MMC_ERROR_CID_CSD_OVERWRITE

CID/CSD overwrite error

HAL_MMC_ERROR_WP_ERASE_SKIP

Only partial address space was erased

HAL_MMC_ERROR_CARD_ECC_DISABLED

Command has been executed without using internal ECC

HAL_MMC_ERROR_ERASE_RESET

Erase sequence was cleared before executing because an out of erase sequence command was received

HAL_MMC_ERROR_AKE_SEQ_ERR

Error in sequence of authentication

HAL_MMC_ERROR_INVALID_VOLTRANGE

Error in case of invalid voltage range

HAL_MMC_ERROR_ADDR_OUT_OF_RANGE

Error when addressed block is out of range

HAL_MMC_ERROR_REQUEST_NOT_APPLICABLE

Error when command request is not applicable

HAL_MMC_ERROR_PARAM

the used parameter is not valid

HAL_MMC_ERROR_UNSUPPORTED_FEATURE

Error when feature is not insupported

HAL_MMC_ERROR_BUSY

Error when transfer process is busy

HAL_MMC_ERROR_DMA

Error while DMA transfer

HAL_MMC_ERROR_TIMEOUT

Timeout error

HAL_MMC_ERROR_INVALID_CALLBACK

Invalid callback error

MMC context enumeration

MMC_CONTEXT_NONE

None

MMC_CONTEXT_READ_SINGLE_BLOCK

Read single block operation

MMC_CONTEXT_READ_MULTIPLE_BLOCK

Read multiple blocks operation

MMC_CONTEXT_WRITE_SINGLE_BLOCK

Write single block operation

MMC_CONTEXT_WRITE_MULTIPLE_BLOCK

Write multiple blocks operation

MMC_CONTEXT_IT

Process in Interrupt mode

MMC_CONTEXT_DMA

Process in DMA mode

MMC Voltage mode

MMC_HIGH_VOLTAGE_RANGE

High voltage in byte mode

MMC_DUAL_VOLTAGE_RANGE

Dual voltage in byte mode

MMC_LOW_VOLTAGE_RANGE

Low voltage in byte mode

EMMC_HIGH_VOLTAGE_RANGE

High voltage in sector mode

EMMC_DUAL_VOLTAGE_RANGE

Dual voltage in sector mode

EMMC_LOW_VOLTAGE_RANGE

Low voltage in sector mode

MMC_INVALID_VOLTAGE_RANGE

MMC Memory Cards

MMC_LOW_CAPACITY_CARD

MMC Card Capacity <=2Gbytes

MMC_HIGH_CAPACITY_CARD

MMC Card Capacity >2Gbytes and <2Tbytes

MMC Erase Type

HAL_MMC_ERASE

Erase the erase groups identified by CMD35 & 36

HAL_MMC_TRIM

Erase the write blocks identified by CMD35 & 36

HAL_MMC_DISCARD

Discard the write blocks identified by CMD35 & 36

HAL_MMC_SECURE_ERASE

Perform a secure purge according SRT on the erase groups identified by CMD35 & 36

HAL_MMC_SECURE_TRIM_STEP1

Mark the write blocks identified by CMD35 & 36 for secure erase

HAL_MMC_SECURE_TRIM_STEP2

Perform a secure purge according SRT on the write blocks previously identified

IS_MMC_ERASE_TYPE

MMC Secure Removal Type

HAL_MMC_SRT_ERASE

Information removed by an erase

HAL_MMC_SRT_WRITE_CHAR_ERASE

Information removed by an overwriting with a character followed by an erase

HAL_MMC_SRT_WRITE_CHAR_COMPL_RANDOM

Information removed by an overwriting with a character, its complement then a random character

HAL_MMC_SRT_VENDOR_DEFINED

Information removed using a vendor defined

IS_MMC_SRT_TYPE

Exported Constants

MMC_BLOCKSIZE

Block size is 512 bytes

MMC Exported Macros

__HAL_MMC_RESET_HANDLE_STATE

Description:

- Reset MMC handle state.

Parameters:

- __HANDLE__: MMC Handle.

Return value:

- None

__HAL_MMC_ENABLE_IT

Description:

- Enable the MMC device interrupt.

Parameters:

- __HANDLE__: MMC Handle.
- __INTERRUPT__: specifies the SDMMC interrupt sources to be enabled. This parameter can be one or a combination of the following values:
  - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  - SDMMC_IT_DTIMEOUT: Data timeout interrupt
  - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDMMC_IT_DHOLD: Data transfer Hold interrupt
  - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  - SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
  - SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
  - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  - SDMMC_IT_SDIOIT: SD I/O interrupt received interrupt
  - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- None

__HAL_MMC_DISABLE_IT

Description:

- Disable the MMC device interrupt.

Parameters:

- __HANDLE__: MMC Handle.
- __INTERRUPT__: specifies the SDMMC interrupt sources to be disabled. This parameter can be one or a combination of the following values:
  - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  - SDMMC_IT_DTIMEOUT: Data timeout interrupt
  - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDMMC_IT_DHOLD: Data transfer Hold interrupt
  - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  - SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
  - SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
  - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  - SDMMC_IT_SDIOIT: SD I/O interrupt received interrupt
  - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- None

__HAL_MMC_GET_FLAG

Description:

- Check whether the specified MMC flag is set or not.

Parameters:

- __HANDLE__: MMC Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SDMMC_FLAG_CCRCFAIL: Command response received (CRC check failed)
  - SDMMC_FLAG_DCRCFAIL: Data block sent/received (CRC check failed)
  - SDMMC_FLAG_CTIMEOUT: Command response timeout
  - SDMMC_FLAG_DTIMEOUT: Data timeout
  - SDMMC_FLAG_TXUNDERR: Transmit FIFO underrun error
  - SDMMC_FLAG_RXOVERR: Received FIFO overrun error
  - SDMMC_FLAG_CMDREND: Command response received (CRC check passed)
  - SDMMC_FLAG_CMDSENT: Command sent (no response required)
  - SDMMC_FLAG_DATAEND: Data end (data counter, DATACOUNT, is zero)
  - SDMMC_FLAG_DHOLD: Data transfer Hold
  - SDMMC_FLAG_DBCKEND: Data block sent/received (CRC check passed)
  - SDMMC_FLAG_DABORT: Data transfer aborted by CMD12
  - SDMMC_FLAG_DPSMACT: Data path state machine active
  - SDMMC_FLAG_CPSMACT: Command path state machine active
  - SDMMC_FLAG_TXFIFOHE: Transmit FIFO Half Empty
  - SDMMC_FLAG_RXFIFOHF: Receive FIFO Half Full
  - SDMMC_FLAG_TXFIFOF: Transmit FIFO full
  - SDMMC_FLAG_RXFIFOF: Receive FIFO full
  - SDMMC_FLAG_TXFIFOE: Transmit FIFO empty
  - SDMMC_FLAG_RXFIFOE: Receive FIFO empty
  - SDMMC_FLAG_BUSYD0: Inverted value of SDMMC_D0 line (Busy)
  - SDMMC_FLAG_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected
  - SDMMC_FLAG_SDIOIT: SD I/O interrupt received
  - SDMMC_FLAG_ACKFAIL: Boot Acknowledgment received
  - SDMMC_FLAG_ACKTIMEOUT: Boot Acknowledgment timeout
  - SDMMC_FLAG_VSWEND: Voltage switch critical timing section completion
  - SDMMC_FLAG_CKSTOP: SDMMC_CK stopped in Voltage switch procedure
  - SDMMC_FLAG_IDMATE: IDMA transfer error
  - SDMMC_FLAG_IDMABTC: IDMA buffer transfer complete

Return value:

- The: new state of MMC FLAG (SET or RESET).

__HAL_MMC_CLEAR_FLAG

Description:

- Clear the MMC's pending flags.

Parameters:

- __HANDLE__: MMC Handle.
- __FLAG__: specifies the flag to clear. This parameter can be one or a combination of the following values:
  - SDMMC_FLAG_CCRCFAIL: Command response received (CRC check failed)
  - SDMMC_FLAG_DCRCFAIL: Data block sent/received (CRC check failed)
  - SDMMC_FLAG_CTIMEOUT: Command response timeout
  - SDMMC_FLAG_DTIMEOUT: Data timeout
  - SDMMC_FLAG_TXUNDERR: Transmit FIFO underrun error
  - SDMMC_FLAG_RXOVERR: Received FIFO overrun error
  - SDMMC_FLAG_CMDREND: Command response received (CRC check passed)
  - SDMMC_FLAG_CMDSENT: Command sent (no response required)
  - SDMMC_FLAG_DATAEND: Data end (data counter, DATACOUNT, is zero)
  - SDMMC_FLAG_DHOLD: Data transfer Hold
  - SDMMC_FLAG_DBCKEND: Data block sent/received (CRC check passed)
  - SDMMC_FLAG_DABORT: Data transfer aborted by CMD12
  - SDMMC_FLAG_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected
  - SDMMC_FLAG_SDIOIT: SD I/O interrupt received
  - SDMMC_FLAG_ACKFAIL: Boot Acknowledgment received
  - SDMMC_FLAG_ACKTIMEOUT: Boot Acknowledgment timeout
  - SDMMC_FLAG_VSWEND: Voltage switch critical timing section completion
  - SDMMC_FLAG_CKSTOP: SDMMC_CK stopped in Voltage switch procedure
  - SDMMC_FLAG_IDMATE: IDMA transfer error
  - SDMMC_FLAG_IDMABTC: IDMA buffer transfer complete

Return value:

- None

__HAL_MMC_GET_IT

Description:

- Check whether the specified MMC interrupt has occurred or not.

Parameters:

- __HANDLE__: MMC Handle.
- __INTERRUPT__: specifies the SDMMC interrupt source to check. This parameter can be one of the following values:
  - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  - SDMMC_IT_DTIMEOUT: Data timeout interrupt
  - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDMMC_IT_DHOLD: Data transfer Hold interrupt
  - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  - SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
  - SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
  - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  - SDMMC_IT_SDIOIT: SD I/O interrupt received interrupt
  - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- The: new state of MMC IT (SET or RESET).

__HAL_MMC_CLEAR_IT

Description:

- Clear the MMC's interrupt pending bits.

Parameters:

- __HANDLE__: MMC Handle.
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one or a combination of the following values:
  - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  - SDMMC_IT_DTIMEOUT: Data timeout interrupt
  - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDMMC_IT_DHOLD: Data transfer Hold interrupt
  - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  - SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
  - SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
  - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  - SDMMC_IT_SDIOIT: SD I/O interrupt received interrupt
  - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- None

MMC Card State enumeration structure

HAL_MMC_CARD_IDLE

Card is in idle state (can't be checked by CMD13)

HAL_MMC_CARD_READY

Card state is ready (can't be checked by CMD13)

HAL_MMC_CARD_IDENTIFICATION

Card is in identification state (can't be checked by CMD13)

HAL_MMC_CARD_STANDBY

Card is in standby state

HAL_MMC_CARD_TRANSFER

Card is in transfer state

HAL_MMC_CARD_SENDING

Card is sending an operation

HAL_MMC_CARD_RECEIVING

Card is receiving operation information

HAL_MMC_CARD_PROGRAMMING

Card is in programming state

HAL_MMC_CARD_DISCONNECTED

Card is disconnected

HAL_MMC_CARD_BUSTEST

Card is in bus test state

HAL_MMC_CARD_SLEEP

Card is in sleep state (can't be checked by CMD13)

HAL_MMC_CARD_ERROR

Card response Error (can't be checked by CMD13)

MMC Handle Structure definition

MMC_InitTypeDef

MMC_TypeDef

54 HAL MMC Extension Driver

54.1 MMCEx Firmware driver API description

The following section lists the various functions of the MMCEx library.

54.1.1 How to use this driver

The MMC Extension HAL driver can be used as follows:

- Configure Buffer0 and Buffer1 start address and Buffer size using HAL_MMCEx_ConfigDMAMultiBuffer() function.
- Start Read and Write for multibuffer mode using HAL_MMCEx_ReadBlocksDMAMultiBuffer() and HAL_MMCEx_WriteBlocksDMAMultiBuffer() functions.

54.1.2 Multibuffer functions

This section provides functions allowing to configure the multibuffer mode and start read and write multibuffer mode for MMC HAL driver.

This section contains the following APIs:

- *HAL_MMCEx_ConfigDMAMultiBuffer()*
- *HAL_MMCEx_ReadBlocksDMAMultiBuffer()*
- *HAL_MMCEx_WriteBlocksDMAMultiBuffer()*
- *HAL_MMCEx_ChangeDMABuffer()*
- *HAL_MMCEx_Read_DMADoubleBuf0CpltCallback()*
- *HAL_MMCEx_Read_DMADoubleBuf1CpltCallback()*
- *HAL_MMCEx_Write_DMADoubleBuf0CpltCallback()*
- *HAL_MMCEx_Write_DMADoubleBuf1CpltCallback()*

54.1.3 Detailed description of functions

HAL_MMCEx_ConfigDMAMultiBuffer

Function name

HAL_StatusTypeDef HAL_MMCEx_ConfigDMAMultiBuffer (MMC_HandleTypeDef * hmmc, uint32_t * pDataBuffer0, uint32_t * pDataBuffer1, uint32_t BufferSize)

Function description

Configure DMA Dual Buffer mode.

Parameters

- **hmmc:** MMC handle
- **pDataBuffer0:** Pointer to the buffer0 that will contain/receive the transferred data
- **pDataBuffer1:** Pointer to the buffer1 that will contain/receive the transferred data
- **BufferSize:** Size of Buffer0 in Blocks. Buffer0 and Buffer1 must have the same size.

Return values

- **HAL:** status

HAL_MMCEx_ReadBlocksDMAMultiBuffer

Function name

HAL_StatusTypeDef HAL_MMCEx_ReadBlocksDMAMultiBuffer (MMC_HandleTypeDef * hmmc, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Reads block(s) from a specified address in a card.

Parameters

- **hmmc:** MMC handle
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Total number of blocks to read

Return values

- **HAL:** status

HAL_MMCEx_WriteBlocksDMAMultiBuffer

Function name

HAL_StatusTypeDef HAL_MMCEx_WriteBlocksDMAMultiBuffer (MMC_HandleTypeDef * hmmc, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Write block(s) to a specified address in a card.

Parameters

- **hmmc:** MMC handle
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Total number of blocks to read

Return values

- **HAL:** status

HAL_MMCEx_ChangeDMABuffer

Function name

HAL_StatusTypeDef HAL_MMCEx_ChangeDMABuffer (MMC_HandleTypeDef * hmmc, HAL_MMCEx_DMABuffer_MemoryTypeDef Buffer, uint32_t * pDataBuffer)

Function description

Change the DMA Buffer0 or Buffer1 address on the fly.

Parameters

- **hmmc:** pointer to a MMC_HandleTypeDef structure.
- **Buffer:** the buffer to be changed, This parameter can be one of the following values: MMC_DMA_BUFFER0 or MMC_DMA_BUFFER1
- **pDataBuffer:** The new address

Return values

- **HAL:** status

Notes

- The BUFFER0 address can be changed only when the current transfer use BUFFER1 and the BUFFER1 address can be changed only when the current transfer use BUFFER0.

HAL_MMCEx_Read_DMADoubleBuf0CpltCallback

Function name

void HAL_MMCEx_Read_DMADoubleBuf0CpltCallback (MMC_HandleTypeDef * hmmc)

Function description

Read DMA Buffer 0 Transfer completed callbacks.

Parameters

- **hmmc:** MMC handle

Return values

- **None:**

HAL_MMCEx_Read_DMADoubleBuf1CpltCallback

Function name

void HAL_MMCEx_Read_DMADoubleBuf1CpltCallback (MMC_HandleTypeDef * hmmc)

Function description

Read DMA Buffer 1 Transfer completed callbacks.

Parameters

- **hmmc:** MMC handle

Return values

- **None:**

HAL_MMCEx_Write_DMADoubleBuf0CpltCallback

Function name

void HAL_MMCEx_Write_DMADoubleBuf0CpltCallback (MMC_HandleTypeDef * hmmc)

Function description

Write DMA Buffer 0 Transfer completed callbacks.

Parameters

- **hmmc:** MMC handle

Return values

- **None:**

HAL_MMCEx_Write_DMADoubleBuf1CpltCallback

Function name

void HAL_MMCEx_Write_DMADoubleBuf1CpltCallback (MMC_HandleTypeDef * hmmc)

Function description

Write DMA Buffer 1 Transfer completed callbacks.

Parameters

- **hmmc:** MMC handle

Return values

- **None:**

55 HAL NAND Generic Driver

55.1 NAND Firmware driver registers structures

55.1.1 NAND_IDTypeDef

NAND_IDTypeDef is defined in the stm32h7xx_hal_nand.h

Data Fields

- ***uint8_t Maker_Id***
- ***uint8_t Device_Id***
- ***uint8_t Third_Id***
- ***uint8_t Fourth_Id***

Field Documentation

- ***uint8_t NAND_IDTypeDef::Maker_Id***
- ***uint8_t NAND_IDTypeDef::Device_Id***
- ***uint8_t NAND_IDTypeDef::Third_Id***
- ***uint8_t NAND_IDTypeDef::Fourth_Id***

55.1.2 NAND_AddressTypeDef

NAND_AddressTypeDef is defined in the stm32h7xx_hal_nand.h

Data Fields

- ***uint16_t Page***
- ***uint16_t Plane***
- ***uint16_t Block***

Field Documentation

- ***uint16_t NAND_AddressTypeDef::Page***
  NAND memory Page address
- ***uint16_t NAND_AddressTypeDef::Plane***
  NAND memory Zone address
- ***uint16_t NAND_AddressTypeDef::Block***
  NAND memory Block address

55.1.3 NAND_DeviceConfigTypeDef

NAND_DeviceConfigTypeDef is defined in the stm32h7xx_hal_nand.h

Data Fields

- ***uint32_t PageSize***
- ***uint32_t SpareAreaSize***
- ***uint32_t BlockSize***
- ***uint32_t BlockNbr***
- ***uint32_t PlaneNbr***
- ***uint32_t PlaneSize***
- ***FunctionalState ExtraCommandEnable***

Field Documentation

- ***uint32_t NAND_DeviceConfigTypeDef::PageSize***
  NAND memory page (without spare area) size measured in bytes for 8 bits addressing or words for 16 bits addressing

- ***uint32_t NAND_DeviceConfigTypeDef::SpareAreaSize***
  NAND memory spare area size measured in bytes for 8 bits addressing or words for 16 bits addressing
- ***uint32_t NAND_DeviceConfigTypeDef::BlockSize***
  NAND memory block size measured in number of pages
- ***uint32_t NAND_DeviceConfigTypeDef::BlockNbr***
  NAND memory number of total blocks
- ***uint32_t NAND_DeviceConfigTypeDef::PlaneNbr***
  NAND memory number of planes
- ***uint32_t NAND_DeviceConfigTypeDef::PlaneSize***
  NAND memory zone size measured in number of blocks
- ***FunctionalState NAND_DeviceConfigTypeDef::ExtraCommandEnable***
  NAND extra command needed for Page reading mode. This parameter is mandatory for some NAND parts after the read command (NAND_CMD_AREA_TRUE1) and before DATA reading sequence. Example: Toshiba THTH58BYG3S0HBAI6. This parameter could be ENABLE or DISABLE Please check the Read Mode sequnece in the NAND device datasheet

55.1.4 __NAND_HandleTypeDef

__NAND_HandleTypeDef is defined in the stm32h7xx_hal_nand.h

Data Fields

- ***FMC_NAND_TypeDef * Instance***
- ***FMC_NAND_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_NAND_StateTypeDef State***
- ***NAND_DeviceConfigTypeDef Config***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***void(* ItCallback***

Field Documentation

- ***FMC_NAND_TypeDef* __NAND_HandleTypeDef::Instance***
  Register base address
- ***FMC_NAND_InitTypeDef __NAND_HandleTypeDef::Init***
  NAND device control configuration parameters
- ***HAL_LockTypeDef __NAND_HandleTypeDef::Lock***
  NAND locking object
- ***__IO HAL_NAND_StateTypeDef __NAND_HandleTypeDef::State***
  NAND device access state
- ***NAND_DeviceConfigTypeDef __NAND_HandleTypeDef::Config***
  NAND phusical characteristic information structure
- ***void(* __NAND_HandleTypeDef::MspInitCallback)(struct __NAND_HandleTypeDef *hnand)***
  NAND Msp Init callback
- ***void(* __NAND_HandleTypeDef::MspDeInitCallback)(struct __NAND_HandleTypeDef *hnand)***
  NAND Msp DeInit callback
- ***void(* __NAND_HandleTypeDef::ItCallback)(struct __NAND_HandleTypeDef *hnand)***
  NAND IT callback

55.2 NAND Firmware driver API description

The following section lists the various functions of the NAND library.

55.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control NAND flash memories. It uses the FMC layer functions to interface with NAND devices. This driver is used as follows:

- NAND flash memory configuration sequence using the function HAL_NAND_Init() with control and timing parameters for both common and attribute spaces.
- Read NAND flash memory maker and device IDs using the function HAL_NAND_Read_ID(). The read information is stored in the NAND_ID_TypeDef structure declared by the function caller.
- Access NAND flash memory by read/write operations using the functions HAL_NAND_Read_Page_8b()/ HAL_NAND_Read_SpareArea_8b(), HAL_NAND_Write_Page_8b()/HAL_NAND_Write_SpareArea_8b(), HAL_NAND_Read_Page_16b()/HAL_NAND_Read_SpareArea_16b(), HAL_NAND_Write_Page_16b()/ HAL_NAND_Write_SpareArea_16b() to read/write page(s)/spare area(s). These functions use specific device information (Block, page size..) predefined by the user in the NAND_DeviceConfigTypeDef structure. The read/write address information is contained by the Nand_Address_Typedef structure passed as parameter.
- Perform NAND flash Reset chip operation using the function HAL_NAND_Reset().
- Perform NAND flash erase block operation using the function HAL_NAND_Erase_Block(). The erase block address information is contained in the Nand_Address_Typedef structure passed as parameter.
- Read the NAND flash status operation using the function HAL_NAND_Read_Status().
- You can also control the NAND device by calling the control APIs HAL_NAND_ECC_Enable()/ HAL_NAND_ECC_Disable() to respectively enable/disable the ECC code correction feature or the function HAL_NAND_GetECC() to get the ECC correction code.
- You can monitor the NAND device HAL state by calling the function HAL_NAND_GetState()

Note: This driver is a set of generic APIs which handle standard NAND flash operations. If a NAND flash device contains different operations and/or implementations, it should be implemented separately.

Callback registration

The compilation define USE_HAL_NAND_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_NAND_RegisterCallback() to register a user callback, it allows to register following callbacks:

- MspInitCallback : NAND MspInit.
- MspDeInitCallback : NAND MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. Use function HAL_NAND_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. It allows to reset following callbacks:
- MspInitCallback : NAND MspInit.
- MspDeInitCallback : NAND MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the HAL_NAND_Init and if the state is HAL_NAND_STATE_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the HAL_NAND_Init and HAL_NAND_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_NAND_Init and HAL_NAND_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/ unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_NAND_RegisterCallback before calling HAL_NAND_DeInit or HAL_NAND_Init function. When The compilation define USE_HAL_NAND_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

55.2.2 NAND Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the NAND memory

This section contains the following APIs:

- ***HAL_NAND_Init()***
- ***HAL_NAND_DeInit()***
- ***HAL_NAND_MspInit()***
- ***HAL_NAND_MspDeInit()***
- ***HAL_NAND_IRQHandler()***

- *HAL_NAND_ITCallback()*
- *HAL_NAND_ConfigDevice()*
- *HAL_NAND_Read_ID()*

55.2.3 NAND Input and Output functions

This section provides functions allowing to use and control the NAND memory

This section contains the following APIs:

- *HAL_NAND_Read_ID()*
- *HAL_NAND_Reset()*
- *HAL_NAND_ConfigDevice()*
- *HAL_NAND_Read_Page_8b()*
- *HAL_NAND_Read_Page_16b()*
- *HAL_NAND_Write_Page_8b()*
- *HAL_NAND_Write_Page_16b()*
- *HAL_NAND_Read_SpareArea_8b()*
- *HAL_NAND_Read_SpareArea_16b()*
- *HAL_NAND_Write_SpareArea_8b()*
- *HAL_NAND_Write_SpareArea_16b()*
- *HAL_NAND_Erase_Block()*
- *HAL_NAND_Address_Inc()*
- *HAL_NAND_RegisterCallback()*
- *HAL_NAND_UnRegisterCallback()*

55.2.4 NAND Control functions

This subsection provides a set of functions allowing to control dynamically the NAND interface.

This section contains the following APIs:

- *HAL_NAND_ECC_Enable()*
- *HAL_NAND_ECC_Disable()*
- *HAL_NAND_GetECC()*

55.2.5 NAND State functions

This subsection permits to get in run-time the status of the NAND controller and the data flow.

This section contains the following APIs:

- *HAL_NAND_GetState()*
- *HAL_NAND_Read_Status()*

55.2.6 Detailed description of functions

HAL_NAND_Init

Function name

HAL_StatusTypeDef HAL_NAND_Init (NAND_HandleTypeDef * hnand, FMC_NAND_PCC_TimingTypeDef * ComSpace_Timing, FMC_NAND_PCC_TimingTypeDef * AttSpace_Timing)

Function description

Perform NAND memory Initialization sequence.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **ComSpace_Timing:** pointer to Common space timing structure
- **AttSpace_Timing:** pointer to Attribute space timing structure

Return values

- **HAL:** status

HAL_NAND_DeInit

Function name

HAL_StatusTypeDef HAL_NAND_DeInit (NAND_HandleTypeDef * hnand)

Function description

Perform NAND memory De-Initialization sequence.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** status

HAL_NAND_ConfigDevice

Function name

HAL_StatusTypeDef HAL_NAND_ConfigDevice (NAND_HandleTypeDef * hnand, NAND_DeviceConfigTypeDef * pDeviceConfig)

Function description

Configure the device: Enter the physical parameters of the device.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pDeviceConfig:** pointer to NAND_DeviceConfigTypeDef structure

Return values

- **HAL:** status

HAL_NAND_Read_ID

Function name

HAL_StatusTypeDef HAL_NAND_Read_ID (NAND_HandleTypeDef * hnand, NAND_IDTypeDef * pNAND_ID)

Function description

Read the NAND memory electronic signature.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pNAND_ID:** NAND ID structure

Return values

- **HAL:** status

HAL_NAND_MspInit

Function name

void HAL_NAND_MspInit (NAND_HandleTypeDef * hnand)

Function description

NAND MSP Init.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **None:**

HAL_NAND_MspDeInit

Function name

void HAL_NAND_MspDeInit (NAND_HandleTypeDef * hnand)

Function description

NAND MSP DeInit.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **None:**

HAL_NAND_IRQHandler

Function name

void HAL_NAND_IRQHandler (NAND_HandleTypeDef * hnand)

Function description

This function handles NAND device interrupt request.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** status

HAL_NAND_ITCallback

Function name

void HAL_NAND_ITCallback (NAND_HandleTypeDef * hnand)

Function description

NAND interrupt feature callback.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **None:**

HAL_NAND_Reset

Function name

HAL_StatusTypeDef HAL_NAND_Reset (NAND_HandleTypeDef * hnand)

Function description

NAND memory reset.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** status

HAL_NAND_Read_Page_8b

Function name

HAL_StatusTypeDef HAL_NAND_Read_Page_8b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumPageToRead)

Function description

Read Page(s) from NAND memory block (8-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to destination read buffer
- **NumPageToRead:** number of pages to read from block

Return values

- **HAL:** status

HAL_NAND_Write_Page_8b

Function name

HAL_StatusTypeDef HAL_NAND_Write_Page_8b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumPageToWrite)

Function description

Write Page(s) to NAND memory block (8-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write
- **NumPageToWrite:** number of pages to write to block

Return values

- **HAL:** status

HAL_NAND_Read_SpareArea_8b

Function name

HAL_StatusTypeDef HAL_NAND_Read_SpareArea_8b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumSpareAreaToRead)

Function description

Read Spare area(s) from NAND memory (8-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write
- **NumSpareAreaToRead:** Number of spare area to read

Return values

- **HAL:** status

HAL_NAND_Write_SpareArea_8b

Function name

HAL_StatusTypeDef HAL_NAND_Write_SpareArea_8b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumSpareAreaTowrite)

Function description

Write Spare area(s) to NAND memory (8-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write
- **NumSpareAreaTowrite:** number of spare areas to write to block

Return values

- **HAL:** status

HAL_NAND_Read_Page_16b

Function name

HAL_StatusTypeDef HAL_NAND_Read_Page_16b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint16_t * pBuffer, uint32_t NumPageToRead)

Function description

Read Page(s) from NAND memory block (16-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to destination read buffer. pBuffer should be 16bits aligned
- **NumPageToRead:** number of pages to read from block

Return values

- **HAL:** status

HAL_NAND_Write_Page_16b

Function name

HAL_StatusTypeDef HAL_NAND_Write_Page_16b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint16_t * pBuffer, uint32_t NumPageToWrite)

Function description

Write Page(s) to NAND memory block (16-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write. pBuffer should be 16bits aligned
- **NumPageToWrite:** number of pages to write to block

Return values

- **HAL:** status

HAL_NAND_Read_SpareArea_16b

Function name

HAL_StatusTypeDef HAL_NAND_Read_SpareArea_16b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint16_t * pBuffer, uint32_t NumSpareAreaToRead)

Function description

Read Spare area(s) from NAND memory (16-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write. pBuffer should be 16bits aligned.
- **NumSpareAreaToRead:** Number of spare area to read

Return values

- **HAL:** status

HAL_NAND_Write_SpareArea_16b

Function name

HAL_StatusTypeDef HAL_NAND_Write_SpareArea_16b (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress, uint16_t * pBuffer, uint32_t NumSpareAreaTowrite)

Function description

Write Spare area(s) to NAND memory (16-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write. pBuffer should be 16bits aligned.
- **NumSpareAreaTowrite:** number of spare areas to write to block

Return values

- **HAL:** status

HAL_NAND_Erase_Block

Function name

HAL_StatusTypeDef HAL_NAND_Erase_Block (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress)

Function description

NAND memory Block erase.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure

Return values

- **HAL:** status

HAL_NAND_Address_Inc

Function name

uint32_t HAL_NAND_Address_Inc (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress)

Function description

Increment the NAND memory address.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure

Return values

- **The:** new status of the increment address operation. It can be:
  - NAND_VALID_ADDRESS: When the new address is valid address
  - NAND_INVALID_ADDRESS: When the new address is invalid address

HAL_NAND_RegisterCallback

Function name

HAL_StatusTypeDef HAL_NAND_RegisterCallback (NAND_HandleTypeDef * hnand, HAL_NAND_CallbackIDTypeDef CallbackId, pNAND_CallbackTypeDef pCallback)

Function description

Register a User NAND Callback To be used instead of the weak (surcharged) predefined callback.

Parameters

- **hnand:** : NAND handle
- **CallbackId:** : ID of the callback to be registered This parameter can be one of the following values:
  - HAL_NAND_MSP_INIT_CB_ID NAND MspInit callback ID
  - HAL_NAND_MSP_DEINIT_CB_ID NAND MspDeInit callback ID
  - HAL_NAND_IT_CB_ID NAND IT callback ID
- **pCallback:** : pointer to the Callback function

Return values

- **status:**

HAL_NAND_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_NAND_UnRegisterCallback (NAND_HandleTypeDef * hnand, HAL_NAND_CallbackIDTypeDef CallbackId)

Function description

Unregister a User NAND Callback NAND Callback is redirected to the weak (surcharged) predefined callback.

Parameters

- **hnand:** : NAND handle
- **CallbackId:** : ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_NAND_MSP_INIT_CB_ID NAND MspInit callback ID
  - HAL_NAND_MSP_DEINIT_CB_ID NAND MspDeInit callback ID
  - HAL_NAND_IT_CB_ID NAND IT callback ID

Return values

- **status:**

HAL_NAND_ECC_Enable

Function name

HAL_StatusTypeDef HAL_NAND_ECC_Enable (NAND_HandleTypeDef * hnand)

Function description

Enables dynamically NAND ECC feature.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** status

HAL_NAND_ECC_Disable

Function name

HAL_StatusTypeDef HAL_NAND_ECC_Disable (NAND_HandleTypeDef * hnand)

Function description

Disables dynamically FMC_NAND ECC feature.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** status

HAL_NAND_GetECC

Function name

HAL_StatusTypeDef HAL_NAND_GetECC (NAND_HandleTypeDef * hnand, uint32_t * ECCval, uint32_t Timeout)

Function description

Disables dynamically NAND ECC feature.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **ECCval:** pointer to ECC value
- **Timeout:** maximum timeout to wait

Return values

- **HAL:** status

HAL_NAND_GetState

Function name

HAL_NAND_StateTypeDef HAL_NAND_GetState (NAND_HandleTypeDef * hnand)

Function description

return the NAND state

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** state

HAL_NAND_Read_Status

Function name

uint32_t HAL_NAND_Read_Status (NAND_HandleTypeDef * hnand)

Function description

NAND memory read status.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **NAND:** status

55.3 NAND Firmware driver defines

The following section lists the various define and macros of the module.

55.3.1 NAND

NAND

NAND Exported Macros

__HAL_NAND_RESET_HANDLE_STATE

Description:

- Reset NAND handle state.

Parameters:

- __HANDLE__: specifies the NAND handle.

Return value:

- None

56 HAL NOR Generic Driver

56.1 NOR Firmware driver registers structures

56.1.1 NOR_IDTypeDef

NOR_IDTypeDef is defined in the stm32h7xx_hal_nor.h

Data Fields

- ***uint16_t Manufacturer_Code***
- ***uint16_t Device_Code1***
- ***uint16_t Device_Code2***
- ***uint16_t Device_Code3***

Field Documentation

- ***uint16_t NOR_IDTypeDef::Manufacturer_Code***
  Defines the device's manufacturer code used to identify the memory
- ***uint16_t NOR_IDTypeDef::Device_Code1***
- ***uint16_t NOR_IDTypeDef::Device_Code2***
- ***uint16_t NOR_IDTypeDef::Device_Code3***
  Defines the device's codes used to identify the memory. These codes can be accessed by performing read operations with specific control signals and addresses set.They can also be accessed by issuing an Auto Select command

56.1.2 NOR_CFITypeDef

NOR_CFITypeDef is defined in the stm32h7xx_hal_nor.h

Data Fields

- ***uint16_t CFI_1***
- ***uint16_t CFI_2***
- ***uint16_t CFI_3***
- ***uint16_t CFI_4***

Field Documentation

- ***uint16_t NOR_CFITypeDef::CFI_1***
  < Defines the information stored in the memory's Common flash interface which contains a description of various electrical and timing parameters, density information and functions supported by the memory
- ***uint16_t NOR_CFITypeDef::CFI_2***
- ***uint16_t NOR_CFITypeDef::CFI_3***
- ***uint16_t NOR_CFITypeDef::CFI_4***

56.1.3 __NOR_HandleTypeDef

__NOR_HandleTypeDef is defined in the stm32h7xx_hal_nor.h

Data Fields

- ***FMC_NORSRAM_TypeDef * Instance***
- ***FMC_NORSRAM_EXTENDED_TypeDef * Extended***
- ***FMC_NORSRAM_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_NOR_StateTypeDef State***
- ***uint32_t CommandSet***
- ***void(* MspInitCallback***

- ***void(* MspDeInitCallback***

Field Documentation

- ***FMC_NORSRAM_TypeDef* __NOR_HandleTypeDef::Instance***
  Register base address
- ***FMC_NORSRAM_EXTENDED_TypeDef* __NOR_HandleTypeDef::Extended***
  Extended mode register base address
- ***FMC_NORSRAM_InitTypeDef __NOR_HandleTypeDef::Init***
  NOR device control configuration parameters
- ***HAL_LockTypeDef __NOR_HandleTypeDef::Lock***
  NOR locking object
- ***__IO HAL_NOR_StateTypeDef __NOR_HandleTypeDef::State***
  NOR device access state
- ***uint32_t __NOR_HandleTypeDef::CommandSet***
  NOR algorithm command set and control
- ***void(* __NOR_HandleTypeDef::MspInitCallback)(struct __NOR_HandleTypeDef *hnor)***
  NOR Msp Init callback
- ***void(* __NOR_HandleTypeDef::MspDeInitCallback)(struct __NOR_HandleTypeDef *hnor)***
  NOR Msp DeInit callback

56.2 NOR Firmware driver API description

The following section lists the various functions of the NOR library.

56.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control NOR flash memories. It uses the FMC layer functions to interface with NOR devices. This driver is used as follows:

- NOR flash memory configuration sequence using the function HAL_NOR_Init() with control and timing parameters for both normal and extended mode.
- Read NOR flash memory manufacturer code and device IDs using the function HAL_NOR_Read_ID(). The read information is stored in the NOR_ID_TypeDef structure declared by the function caller.
- Access NOR flash memory by read/write data unit operations using the functions HAL_NOR_Read(), HAL_NOR_Program().
- Perform NOR flash erase block/chip operations using the functions HAL_NOR_Erase_Block() and HAL_NOR_Erase_Chip().
- Read the NOR flash CFI (common flash interface) IDs using the function HAL_NOR_Read_CFI(). The read information is stored in the NOR_CFI_TypeDef structure declared by the function caller.
- You can also control the NOR device by calling the control APIs HAL_NOR_WriteOperation_Enable()/ HAL_NOR_WriteOperation_Disable() to respectively enable/disable the NOR write operation
- You can monitor the NOR device HAL state by calling the function HAL_NOR_GetState()

Note: This driver is a set of generic APIs which handle standard NOR flash operations. If a NOR flash device contains different operations and/or implementations, it should be implemented separately.

NOR HAL driver macros list

Below the list of most used macros in NOR HAL driver.

- NOR_WRITE : NOR memory write data to specified address

Callback registration

The compilation define USE_HAL_NOR_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_NOR_RegisterCallback() to register a user callback, it allows to register following callbacks:

- MspInitCallback : NOR MspInit.

- MspDeInitCallback : NOR MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. Use function HAL_NOR_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. It allows to reset following callbacks:
- MspInitCallback : NOR MspInit.
- MspDeInitCallback : NOR MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the HAL_NOR_Init and if the state is HAL_NOR_STATE_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the HAL_NOR_Init and HAL_NOR_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_NOR_Init and HAL_NOR_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/ unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_NOR_RegisterCallback before calling HAL_NOR_DeInit or HAL_NOR_Init function. When The compilation define USE_HAL_NOR_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

56.2.2 NOR Initialization and de_initialization functions

This section provides functions allowing to initialize/de-initialize the NOR memory

This section contains the following APIs:

- ***HAL_NOR_Init()***
- ***HAL_NOR_DeInit()***
- ***HAL_NOR_MspInit()***
- ***HAL_NOR_MspDeInit()***
- ***HAL_NOR_MspWait()***

56.2.3 NOR Input and Output functions

This section provides functions allowing to use and control the NOR memory

This section contains the following APIs:

- ***HAL_NOR_Read_ID()***
- ***HAL_NOR_ReturnToReadMode()***
- ***HAL_NOR_Read()***
- ***HAL_NOR_Program()***
- ***HAL_NOR_ReadBuffer()***
- ***HAL_NOR_ProgramBuffer()***
- ***HAL_NOR_Erase_Block()***
- ***HAL_NOR_Erase_Chip()***
- ***HAL_NOR_Read_CFI()***
- ***HAL_NOR_RegisterCallback()***
- ***HAL_NOR_UnRegisterCallback()***

56.2.4 NOR Control functions

This subsection provides a set of functions allowing to control dynamically the NOR interface.

This section contains the following APIs:

- ***HAL_NOR_WriteOperation_Enable()***
- ***HAL_NOR_WriteOperation_Disable()***

56.2.5 NOR State functions

This subsection permits to get in run-time the status of the NOR controller and the data flow.

This section contains the following APIs:

- *HAL_NOR_GetState()*
- *HAL_NOR_GetStatus()*

56.2.6 Detailed description of functions

HAL_NOR_Init

Function name

HAL_StatusTypeDef HAL_NOR_Init (NOR_HandleTypeDef * hnor, FMC_NORSRAM_TimingTypeDef * Timing, FMC_NORSRAM_TimingTypeDef * ExtTiming)

Function description

Perform the NOR memory Initialization sequence.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **Timing:** pointer to NOR control timing structure
- **ExtTiming:** pointer to NOR extended mode timing structure

Return values

- **HAL:** status

HAL_NOR_DeInit

Function name

HAL_StatusTypeDef HAL_NOR_DeInit (NOR_HandleTypeDef * hnor)

Function description

Perform NOR memory De-Initialization sequence.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **HAL:** status

HAL_NOR_MspInit

Function name

void HAL_NOR_MspInit (NOR_HandleTypeDef * hnor)

Function description

NOR MSP Init.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **None:**

HAL_NOR_MspDeInit

Function name

void HAL_NOR_MspDeInit (NOR_HandleTypeDef * hnor)

Function description

NOR MSP DeInit.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **None:**

HAL_NOR_MspWait

Function name

void HAL_NOR_MspWait (NOR_HandleTypeDef * hnor, uint32_t Timeout)

Function description

NOR MSP Wait for Ready/Busy signal.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **Timeout:** Maximum timeout value

Return values

- **None:**

HAL_NOR_Read_ID

Function name

HAL_StatusTypeDef HAL_NOR_Read_ID (NOR_HandleTypeDef * hnor, NOR_IDTypeDef * pNOR_ID)

Function description

Read NOR flash IDs.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **pNOR_ID:** pointer to NOR ID structure

Return values

- **HAL:** status

HAL_NOR_ReturnToReadMode

Function name

HAL_StatusTypeDef HAL_NOR_ReturnToReadMode (NOR_HandleTypeDef * hnor)

Function description

Returns the NOR memory to Read mode.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **HAL:** status

HAL_NOR_Read

Function name

HAL_StatusTypeDef HAL_NOR_Read (NOR_HandleTypeDef * hnor, uint32_t * pAddress, uint16_t * pData)

Function description

Read data from NOR memory.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **pAddress:** pointer to Device address
- **pData:** pointer to read data

Return values

- **HAL:** status

HAL_NOR_Program

Function name

HAL_StatusTypeDef HAL_NOR_Program (NOR_HandleTypeDef * hnor, uint32_t * pAddress, uint16_t * pData)

Function description

Program data to NOR memory.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **pAddress:** Device address
- **pData:** pointer to the data to write

Return values

- **HAL:** status

HAL_NOR_ReadBuffer

Function name

HAL_StatusTypeDef HAL_NOR_ReadBuffer (NOR_HandleTypeDef * hnor, uint32_t uwAddress, uint16_t * pData, uint32_t uwBufferSize)

Function description

Reads a half-word buffer from the NOR memory.

Parameters

- **hnor:** pointer to the NOR handle
- **uwAddress:** NOR memory internal address to read from.
- **pData:** pointer to the buffer that receives the data read from the NOR memory.
- **uwBufferSize:** number of Half word to read.

Return values

- **HAL:** status

HAL_NOR_ProgramBuffer

Function name

HAL_StatusTypeDef HAL_NOR_ProgramBuffer (NOR_HandleTypeDef * hnor, uint32_t uwAddress, uint16_t * pData, uint32_t uwBufferSize)

Function description

Writes a half-word buffer to the NOR memory.

Parameters

- **hnor:** pointer to the NOR handle
- **uwAddress:** NOR memory internal start write address
- **pData:** pointer to source data buffer.
- **uwBufferSize:** Size of the buffer to write

Return values

- **HAL:** status

HAL_NOR_Erase_Block

Function name

HAL_StatusTypeDef HAL_NOR_Erase_Block (NOR_HandleTypeDef * hnor, uint32_t BlockAddress, uint32_t Address)

Function description

Erase the specified block of the NOR memory.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **BlockAddress:** Block to erase address
- **Address:** Device address

Return values

- **HAL:** status

HAL_NOR_Erase_Chip

Function name

HAL_StatusTypeDef HAL_NOR_Erase_Chip (NOR_HandleTypeDef * hnor, uint32_t Address)

Function description

Erase the entire NOR chip.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **Address:** Device address

Return values

- **HAL:** status

HAL_NOR_Read_CFI

Function name

HAL_StatusTypeDef HAL_NOR_Read_CFI (NOR_HandleTypeDef * hnor, NOR_CFITypeDef * pNOR_CFI)

Function description

Read NOR flash CFI IDs.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **pNOR_CFI:** pointer to NOR CFI IDs structure

Return values

- **HAL:** status

HAL_NOR_RegisterCallback

Function name

HAL_StatusTypeDef HAL_NOR_RegisterCallback (NOR_HandleTypeDef * hnor, HAL_NOR_CallbackIDTypeDef CallbackId, pNOR_CallbackTypeDef pCallback)

Function description

Register a User NOR Callback To be used instead of the weak (surcharged) predefined callback.

Parameters

- **hnor:** : NOR handle
- **CallbackId:** : ID of the callback to be registered This parameter can be one of the following values:
  - HAL_NOR_MSP_INIT_CB_ID NOR MspInit callback ID
  - HAL_NOR_MSP_DEINIT_CB_ID NOR MspDeInit callback ID
- **pCallback:** : pointer to the Callback function

Return values

- **status:**

HAL_NOR_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_NOR_UnRegisterCallback (NOR_HandleTypeDef * hnor, HAL_NOR_CallbackIDTypeDef CallbackId)

Function description

Unregister a User NOR Callback NOR Callback is redirected to the weak (surcharged) predefined callback.

Parameters

- **hnor:** : NOR handle
- **CallbackId:** : ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_NOR_MSP_INIT_CB_ID NOR MspInit callback ID
  - HAL_NOR_MSP_DEINIT_CB_ID NOR MspDeInit callback ID

Return values

- **status:**

HAL_NOR_WriteOperation_Enable

Function name

HAL_StatusTypeDef HAL_NOR_WriteOperation_Enable (NOR_HandleTypeDef * hnor)

Function description

Enables dynamically NOR write operation.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **HAL:** status

HAL_NOR_WriteOperation_Disable

Function name

HAL_StatusTypeDef HAL_NOR_WriteOperation_Disable (NOR_HandleTypeDef * hnor)

Function description

Disables dynamically NOR write operation.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **HAL:** status

HAL_NOR_GetState

Function name

HAL_NOR_StateTypeDef HAL_NOR_GetState (NOR_HandleTypeDef * hnor)

Function description

return the NOR controller state

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **NOR:** controller state

HAL_NOR_GetStatus

Function name

HAL_NOR_StatusTypeDef HAL_NOR_GetStatus (NOR_HandleTypeDef * hnor, uint32_t Address, uint32_t Timeout)

Function description

Returns the NOR operation status.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **Address:** Device address
- **Timeout:** NOR programming Timeout

Return values

- **NOR_Status:** The returned value can be: HAL_NOR_STATUS_SUCCESS, HAL_NOR_STATUS_ERROR or HAL_NOR_STATUS_TIMEOUT

56.3 NOR Firmware driver defines

The following section lists the various define and macros of the module.

56.3.1 NOR

NOR

NOR Exported Macros

__HAL_NOR_RESET_HANDLE_STATE

Description:

- Reset NOR handle state.

Parameters:

- __HANDLE__: specifies the NOR handle.

Return value:

- None

57 HAL OPAMP Generic Driver

57.1 OPAMP Firmware driver registers structures

57.1.1 OPAMP_InitTypeDef

OPAMP_InitTypeDef is defined in the stm32h7xx_hal_opamp.h

Data Fields

- ***uint32_t PowerMode***
- ***uint32_t Mode***
- ***uint32_t InvertingInput***
- ***uint32_t NonInvertingInput***
- ***uint32_t PgaGain***
- ***uint32_t PgaConnect***
- ***uint32_t UserTrimming***
- ***uint32_t TrimmingValueP***
- ***uint32_t TrimmingValueN***
- ***uint32_t TrimmingValuePHighSpeed***
- ***uint32_t TrimmingValueNHighSpeed***

Field Documentation

- ***uint32_t OPAMP_InitTypeDef::PowerMode***

  Specifies the power mode Normal or High Speed. This parameter must be a value of ***OPAMP_PowerMode***
- ***uint32_t OPAMP_InitTypeDef::Mode***

  Specifies the OPAMP mode This parameter must be a value of ***OPAMP_Mode*** mode is either Standalone, - Follower or PGA
- ***uint32_t OPAMP_InitTypeDef::InvertingInput***

  Specifies the inverting input in Standalone & PGA modes
  - In Standalone mode i.e when mode is OPAMP_STANDALONE_MODE This parameter must be a value of ***OPAMP_InvertingInput***
  - In Follower mode i.e when mode is OPAMP_FOLLOWER_MODE & In PGA mode i.e when mode is OPAMP_PGA_MODE This parameter is Not Applicable
- ***uint32_t OPAMP_InitTypeDef::NonInvertingInput***

  Specifies the non inverting input of the opamp: This parameter must be a value of ***OPAMP_NonInvertingInput***
- ***uint32_t OPAMP_InitTypeDef::PgaGain***

  Specifies the gain in PGA mode i.e. when mode is OPAMP_PGA_MODE. This parameter must be a value of ***OPAMP_PgaGain***
- ***uint32_t OPAMP_InitTypeDef::PgaConnect***

  Specifies the inverting pin in PGA mode i.e. when mode is OPAMP_PGA_MODE This parameter must be a value of ***OPAMP_PgaConnect*** Either: not connected, connected to VINM0, connected to VINM1 (VINM0 or VINM1 are typically used for external filtering)
- ***uint32_t OPAMP_InitTypeDef::UserTrimming***

  Specifies the trimming mode This parameter must be a value of ***OPAMP_UserTrimming*** UserTrimming is either factory or user trimming.
- ***uint32_t OPAMP_InitTypeDef::TrimmingValueP***

  Specifies the offset trimming value (PMOS) in Normal Mode i.e. when UserTrimming is OPAMP_TRIMMING_USER. This parameter must be a number between Min_Data = 0 and Max_Data = 31. 16 is typical default value

- ***uint32_t OPAMP_InitTypeDef::TrimmingValueN***

  Specifies the offset trimming value (NMOS) in Normal Mode i.e. when UserTrimming is OPAMP_TRIMMING_USER. This parameter must be a number between Min_Data = 0 and Max_Data = 31. 16 is typical default value
- ***uint32_t OPAMP_InitTypeDef::TrimmingValuePHighSpeed***

  Specifies the offset trimming value (PMOS) in High Speed Mode i.e. when UserTrimming is OPAMP_TRIMMING_USER. This parameter must be a number between Min_Data = 0 and Max_Data = 31. 16 is typical default value
- ***uint32_t OPAMP_InitTypeDef::TrimmingValueNHighSpeed***

  Specifies the offset trimming value (NMOS) in High Speed Mode i.e. when UserTrimming is OPAMP_TRIMMING_USER. This parameter must be a number between Min_Data = 0 and Max_Data = 31. 16 is typical default value

57.1.2 __OPAMP_HandleTypeDef

__OPAMP_HandleTypeDef is defined in the stm32h7xx_hal_opamp.h

Data Fields

- ***OPAMP_TypeDef * Instance***
- ***OPAMP_InitTypeDef Init***
- ***HAL_StatusTypeDef Status***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_OPAMP_StateTypeDef State***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***OPAMP_TypeDef* __OPAMP_HandleTypeDef::Instance***

  OPAMP instance's registers base address
- ***OPAMP_InitTypeDef __OPAMP_HandleTypeDef::Init***

  OPAMP required parameters
- ***HAL_StatusTypeDef __OPAMP_HandleTypeDef::Status***

  OPAMP peripheral status
- ***HAL_LockTypeDef __OPAMP_HandleTypeDef::Lock***

  Locking object
- ***__IO HAL_OPAMP_StateTypeDef __OPAMP_HandleTypeDef::State***

  OPAMP communication state
- ***void(* __OPAMP_HandleTypeDef::MspInitCallback)(struct __OPAMP_HandleTypeDef *hopamp)***
- ***void(* __OPAMP_HandleTypeDef::MspDeInitCallback)(struct __OPAMP_HandleTypeDef *hopamp)***

57.2 OPAMP Firmware driver API description

The following section lists the various functions of the OPAMP library.

57.2.1 OPAMP Peripheral Features

The device integrates 2 operational amplifiers OPAMP1 & OPAMP2

1. The OPAMP(s) provides several exclusive running modes.
   - Standalone mode
   - Programmable Gain Amplifier (PGA) modes
   - Follower mode
2. Each OPAMP(s) can be configured in normal and high speed mode.

3. The OPAMP(s) provide(s) calibration capabilities.
   - Calibration aims at correcting some offset for running mode.
   - The OPAMP uses either factory calibration settings OR user defined calibration (trimming) settings (i.e. trimming mode).
   - The user defined settings can be figured out using self calibration handled by HAL_OPAMP_SelfCalibrate, HAL_OPAMPEx_SelfCalibrateAll
   - HAL_OPAMP_SelfCalibrate:
     - Runs automatically the calibration in 2 steps. (90% of VDDA for NMOS transistors, 10% of VDDA for PMOS transistors). (As OPAMP is Rail-to-rail input/output, these 2 steps calibration is appropriate and enough in most cases).
     - Runs automatically the calibration.
     - Enables the user trimming mode
     - Updates the init structure with trimming values with fresh calibration results. The user may store the calibration results for larger (ex monitoring the trimming as a function of temperature for instance)
     - HAL_OPAMPEx_SelfCalibrateAll runs calibration of all OPAMPs in parallel to save search time.
4. Running mode: Standalone mode
   - Gain is set externally (gain depends on external loads).
   - Follower mode also possible externally by connecting the inverting input to the output.
5. Running mode: Follower mode
   - No Inverting Input is connected.
6. Running mode: Programmable Gain Amplifier (PGA) mode (Resistor feedback output)
7. The OPAMP(s) output(s) can be internally connected to resistor feedback output.
8. OPAMP gain can be selected as :
   a. Gain of x2, x4, x8 or x16 for non inverting mode with:
      - VREF- referenced.
      - Filtering on VINM0, VREF- referenced.
      - VINM0 node for bias voltage and VINP0 for input signal.
      - VINM0 node for bias voltage and VINP0 for input signal, VINM1 node for filtering.
   b. Gain of x-1, x-3, x-7 or x-15 for inverting mode with:
      - VINM0 node for input signal and VINP0 for bias.
      - VINM0 node for input signal and VINP0 for bias voltage, VINM1 node for filtering.
9. The OPAMPs inverting input can be selected according to the Reference Manual "OPAMP functional description" chapter.
10. The OPAMPs non inverting input can be selected according to the Reference Manual "OPAMP functional description" chapter.

57.2.2 How to use this driver

High speed / normal power mode

To run in high speed mode:

1. Configure the OPAMP using HAL_OPAMP_Init() function:
   - Select OPAMP_POWERMODE_HIGHSPEED
   - Otherwise select OPAMP_POWERMODE_NORMAL

Calibration

To run the OPAMP calibration self calibration:

1. Start calibration using HAL_OPAMP_SelfCalibrate. Store the calibration results.

Running mode

To use the OPAMP, perform the following steps:

1. Fill in the HAL_OPAMP_MspInit() to
   – Enable the OPAMP Peripheral clock using macro __HAL_RCC_OPAMP_CLK_ENABLE()
   – Configure the OPAMP input AND output in analog mode using HAL_GPIO_Init() to map the OPAMP output to the GPIO pin.
2. Registrate Callbacks
   – The compilation define USE_HAL_OPAMP_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.
   – Use Functions HAL_OPAMP_RegisterCallback() to register a user callback, it allows to register following callbacks:
     ◦ MspInitCallback : OPAMP MspInit.
     ◦ MspDeInitCallback : OPAMP MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.
   – Use function HAL_OPAMP_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. It allows to reset following callbacks:
     ◦ MspInitCallback : OPAMP MspInit.
     ◦ MspDeInitCallback : OPAMP MspDeInit.
     ◦ All Callbacks
3. Configure the OPAMP using HAL_OPAMP_Init() function:
   – Select the mode
   – Select the inverting input
   – Select the non-inverting input
   – If PGA mode is enabled, Select if inverting input is connected.
   – Select either factory or user defined trimming mode.
   – If the user-defined trimming mode is enabled, select PMOS & NMOS trimming values (typically values set by HAL_OPAMP_SelfCalibrate function).
4. Enable the OPAMP using HAL_OPAMP_Start() function.
5. Disable the OPAMP using HAL_OPAMP_Stop() function.
6. Lock the OPAMP in running mode using HAL_OPAMP_Lock() function. Caution: On STM32H7, HAL OPAMP lock is software lock only (not hardware lock as on some other STM32 devices)
7. If needed, unlock the OPAMP using HAL_OPAMPEx_Unlock() function.

Running mode: change of configuration while OPAMP ON

To Re-configure OPAMP when OPAMP is ON (change on the fly)

1. If needed, fill in the HAL_OPAMP_MspInit()
   – This is the case for instance if you wish to use new OPAMP I/O
2. Configure the OPAMP using HAL_OPAMP_Init() function:
   – As in configure case, select first the parameters you wish to modify.
3. Change from high speed mode to normal power mode (& vice versa) requires first HAL_OPAMP_DeInit() (force OPAMP OFF) and then HAL_OPAMP_Init(). In other words, of OPAMP is ON, HAL_OPAMP_Init can NOT change power mode alone.

57.2.3 Initialization and de-initialization functions

This section contains the following APIs:

- ***HAL_OPAMP_Init()***
- ***HAL_OPAMP_DeInit()***
- ***HAL_OPAMP_MspInit()***
- ***HAL_OPAMP_MspDeInit()***

57.2.4 IO operation functions

This subsection provides a set of functions allowing to manage the OPAMP start, stop and calibration actions.

This section contains the following APIs:

- *HAL_OPAMP_Start()*
- *HAL_OPAMP_Stop()*
- *HAL_OPAMP_SelfCalibrate()*

57.2.5 Peripheral Control functions

This subsection provides a set of functions allowing to control the OPAMP data transfers.

This section contains the following APIs:

- *HAL_OPAMP_Lock()*
- *HAL_OPAMP_GetTrimOffset()*
- *HAL_OPAMP_RegisterCallback()*
- *HAL_OPAMP_UnRegisterCallback()*

57.2.6 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral.

This section contains the following APIs:

- *HAL_OPAMP_GetState()*

57.2.7 Detailed description of functions

HAL_OPAMP_Init

Function name

HAL_StatusTypeDef HAL_OPAMP_Init (OPAMP_HandleTypeDef * hopamp)

Function description

Initialize the OPAMP according to the specified parameters in the OPAMP_InitTypeDef and initialize the associated handle.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** status

Notes

- If the selected opamp is locked, initialization can't be performed. To unlock the configuration, perform a system reset.

HAL_OPAMP_DeInit

Function name

HAL_StatusTypeDef HAL_OPAMP_DeInit (OPAMP_HandleTypeDef * hopamp)

Function description

DeInitialize the OPAMP peripheral.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** status

Notes

- Deinitialization can be performed if the OPAMP configuration is locked. (the lock is SW in H7)

HAL_OPAMP_MspInit

Function name

void HAL_OPAMP_MspInit (OPAMP_HandleTypeDef * hopamp)

Function description

Initialize the OPAMP MSP.

Parameters

- **hopamp:** OPAMP handle

Return values

- **None:**

HAL_OPAMP_MspDeInit

Function name

void HAL_OPAMP_MspDeInit (OPAMP_HandleTypeDef * hopamp)

Function description

DeInitialize OPAMP MSP.

Parameters

- **hopamp:** OPAMP handle

Return values

- **None:**

HAL_OPAMP_Start

Function name

HAL_StatusTypeDef HAL_OPAMP_Start (OPAMP_HandleTypeDef * hopamp)

Function description

Start the OPAMP.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** status

HAL_OPAMP_Stop

Function name

HAL_StatusTypeDef HAL_OPAMP_Stop (OPAMP_HandleTypeDef * hopamp)

Function description

Stop the OPAMP.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** status

HAL_OPAMP_SelfCalibrate

Function name

HAL_StatusTypeDef HAL_OPAMP_SelfCalibrate (OPAMP_HandleTypeDef * hopamp)

Function description

Run the self calibration of one OPAMP.

Parameters

- **hopamp:** handle

Return values

- **Updated:** offset trimming values (PMOS & NMOS), user trimming is enabled
- **HAL:** status

Notes

- Calibration is performed in the mode specified in OPAMP init structure (mode normal or high-speed). To perform calibration for both modes, repeat this function twice after OPAMP init structure accordingly updated.

HAL_OPAMP_RegisterCallback

Function name

HAL_StatusTypeDef HAL_OPAMP_RegisterCallback (OPAMP_HandleTypeDef * hopamp, HAL_OPAMP_CallbackIDTypeDef CallbackId, pOPAMP_CallbackTypeDef pCallback)

Function description

Register a User OPAMP Callback To be used instead of the weak (surcharged) predefined callback.

Parameters

- **hopamp:** OPAMP handle
- **CallbackId:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_OPAMP_MSPINIT_CB_ID OPAMP MspInit callback ID
  - HAL_OPAMP_MSPDEINIT_CB_ID OPAMP MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **status:**

HAL_OPAMP_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_OPAMP_UnRegisterCallback (OPAMP_HandleTypeDef * hopamp, HAL_OPAMP_CallbackIDTypeDef CallbackId)

Function description

Unregister a User OPAMP Callback OPAMP Callback is redirected to the weak (surcharged) predefined callback.

Parameters

- **hopamp:** OPAMP handle
- **CallbackId:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_OPAMP_MSPINIT_CB_ID OPAMP MSP Init Callback ID
  - HAL_OPAMP_MSPDEINIT_CB_ID OPAMP MSP DeInit Callback ID
  - HAL_OPAMP_ALL_CB_ID OPAMP All Callbacks

Return values

- **status:**

HAL_OPAMP_Lock

Function name

HAL_StatusTypeDef HAL_OPAMP_Lock (OPAMP_HandleTypeDef * hopamp)

Function description

Lock the selected OPAMP configuration.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** status

Notes

- On STM32H7, HAL OPAMP lock is software lock only (in contrast of hardware lock available on some other STM32 devices)

HAL_OPAMP_GetTrimOffset

Function name

HAL_OPAMP_TrimmingValueTypeDef HAL_OPAMP_GetTrimOffset (OPAMP_HandleTypeDef * hopamp, uint32_t trimmingoffset)

Function description

Return the OPAMP factory trimming value.

Parameters

- **hopamp:** OPAMP handle
- **trimmingoffset:** Trimming offset (P or N) This parameter must be a value of OPAMP Factory Trimming

Return values

- **Trimming:** value (P or N): range: 0->31 or OPAMP_FACTORYTRIMMING_DUMMY if trimming value is not available

Notes

- On STM32H7 OPAMP, user can retrieve factory trimming if OPAMP has never been set to user trimming before. Therefore, this function must be called when OPAMP init parameter "UserTrimming" is set to trimming factory, and before OPAMP calibration (function "HAL_OPAMP_SelfCalibrate()"). Otherwise, factory trimming value cannot be retrieved and error status is returned.
- Calibration parameter retrieved is corresponding to the mode specified in OPAMP init structure (mode normal or high-speed). To retrieve calibration parameters for both modes, repeat this function after OPAMP init structure accordingly updated.

HAL_OPAMP_GetState

Function name

HAL_OPAMP_StateTypeDef HAL_OPAMP_GetState (OPAMP_HandleTypeDef * hopamp)

Function description

Return the OPAMP handle state.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** state

57.3 OPAMP Firmware driver defines

The following section lists the various define and macros of the module.

57.3.1 OPAMP

OPAMP

OPAMP Exported Macros

__HAL_OPAMP_RESET_HANDLE_STATE

Description:

- Reset OPAMP handle state.

Parameters:

- __HANDLE__: OPAMP handle.

Return value:

- None

OPAMP Factory Trimming

OPAMP_FACTORYTRIMMING_DUMMY

Dummy value if trimming value could not be retrieved

OPAMP_FACTORYTRIMMING_N

Offset trimming N

OPAMP_FACTORYTRIMMING_P

Offset trimming P

OPAMP Inverting Input

OPAMP_INVERTINGINPUT_IO0

OPAMP inverting input connected to dedicated IO pin

OPAMP_INVERTINGINPUT_IO1

OPAMP inverting input connected to dedicated IO pin

OPAMP Mode

OPAMP_STANDALONE_MODE

standalone mode

OPAMP_PGA_MODE

PGA mode

OPAMP_FOLLOWER_MODE

follower mode

OPAMP Non Inverting Input

OPAMP_NONINVERTINGINPUT_IO0

OPAMP non-inverting input connected to dedicated IO pin

OPAMP_NONINVERTINGINPUT_DAC_CH

OPAMP non-inverting input connected internally to DAC channel

OPAMP Pga Connect

OPAMP_PGA_CONNECT_INVERTINGINPUT_NO

In PGA mode, the inverting input is not connected

OPAMP_PGA_CONNECT_INVERTINGINPUT_IO0

In PGA mode, the inverting input is connected to VINM0

OPAMP_PGA_CONNECT_INVERTINGINPUT_IO0_BIAS

In PGA mode, the inverting input is connected to VINM0 or bias

OPAMP_PGA_CONNECT_INVERTINGINPUT_IO0_IO1_BIAS

In PGA mode, the inverting input is connected to VINM0 or bias , VINM1 connected for filtering

OPAMP Pga Gain

OPAMP_PGA_GAIN_2_OR_MINUS_1

PGA gain could be 2 or -1

OPAMP_PGA_GAIN_4_OR_MINUS_3

PGA gain could be 4 or -3

OPAMP_PGA_GAIN_8_OR_MINUS_7

PGA gain could be 8 or -7

OPAMP_PGA_GAIN_16_OR_MINUS_15

PGA gain could be 16 or -15

OPAMP PowerMode

OPAMP_POWERMODE_NORMAL

OPAMP_POWERMODE_HIGHSPEED

OPAMP User Trimming

OPAMP_TRIMMING_FACTORY

Factory trimming

OPAMP_TRIMMING_USER

User trimming

OPAMP VREF

OPAMP_VREF_3VDDA

OPAMP Vref = 3.3% VDDA

OPAMP_VREF_10VDDA

OPAMP Vref = 10% VDDA

OPAMP_VREF_50VDDA

OPAMP Vref = 50% VDDA

OPAMP_VREF_90VDDA

OPAMP Vref = 90% VDDA

58 HAL OPAMP Extension Driver

58.1 OPAMPEx Firmware driver API description

The following section lists the various functions of the OPAMPEx library.

58.1.1 Extended IO operation functions

- OPAMP Self calibration.

58.1.2 Peripheral Control functions

- OPAMP unlock.

This section contains the following APIs:

- ***HAL_OPAMPEx_Unlock()***

58.1.3 Detailed description of functions

HAL_OPAMPEx_SelfCalibrateAll

Function name

HAL_StatusTypeDef HAL_OPAMPEx_SelfCalibrateAll (OPAMP_HandleTypeDef * hopamp1, OPAMP_HandleTypeDef * hopamp2)

Function description

Run the self calibration of 2 OPAMPs in parallel.

Parameters

- **hopamp1:** handle
- **hopamp2:** handle

Return values

- **HAL:** status

Notes

- Trimming values (PMOS & NMOS) are updated and user trimming is enabled is calibration is successful.
- Calibration is performed in the mode specified in OPAMP init structure (mode normal or low power). To perform calibration for both modes, repeat this function twice after OPAMP init structure accordingly updated.

HAL_OPAMPEx_Unlock

Function name

HAL_StatusTypeDef HAL_OPAMPEx_Unlock (OPAMP_HandleTypeDef * hopamp)

Function description

Unlock the selected OPAMP configuration.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** status

Notes

- This function must be called only when OPAMP is in state "locked".

59 HAL OSPI Generic Driver

59.1 OSPI Firmware driver registers structures

59.1.1 OSPI_InitTypeDef

OSPI_InitTypeDef is defined in the stm32h7xx_hal_ospi.h

Data Fields

- ***uint32_t FifoThreshold***
- ***uint32_t DualQuad***
- ***uint32_t MemoryType***
- ***uint32_t DeviceSize***
- ***uint32_t ChipSelectHighTime***
- ***uint32_t FreeRunningClock***
- ***uint32_t ClockMode***
- ***uint32_t WrapSize***
- ***uint32_t ClockPrescaler***
- ***uint32_t SampleShifting***
- ***uint32_t DelayHoldQuarterCycle***
- ***uint32_t ChipSelectBoundary***
- ***uint32_t DelayBlockBypass***
- ***uint32_t MaxTran***
- ***uint32_t Refresh***

Field Documentation

- ***uint32_t OSPI_InitTypeDef::FifoThreshold***

  This is the threshold used by the Peripheral to generate the interrupt indicating that data are available in reception or free place is available in transmission. This parameter can be a value between 1 and 32
- ***uint32_t OSPI_InitTypeDef::DualQuad***

  It enables or not the dual-quad mode which allow to access up to quad mode on two different devices to increase the throughput. This parameter can be a value of ***OSPI_DualQuad***
- ***uint32_t OSPI_InitTypeDef::MemoryType***

  It indicates the external device type connected to the OSPI. This parameter can be a value of ***OSPI_MemoryType***
- ***uint32_t OSPI_InitTypeDef::DeviceSize***

  It defines the size of the external device connected to the OSPI, it corresponds to the number of address bits required to access the external device. This parameter can be a value between 1 and 32
- ***uint32_t OSPI_InitTypeDef::ChipSelectHighTime***

  It defines the minimum number of clocks which the chip select must remain high between commands. This parameter can be a value between 1 and 8
- ***uint32_t OSPI_InitTypeDef::FreeRunningClock***

  It enables or not the free running clock. This parameter can be a value of ***OSPI_FreeRunningClock***
- ***uint32_t OSPI_InitTypeDef::ClockMode***

  It indicates the level of clock when the chip select is released. This parameter can be a value of ***OSPI_ClockMode***
- ***uint32_t OSPI_InitTypeDef::WrapSize***

  It indicates the wrap-size corresponding the external device configuration. This parameter can be a value of ***OSPI_WrapSize***

- ***uint32_t OSPI_InitTypeDef::ClockPrescaler***
  It specifies the prescaler factor used for generating the external clock based on the AHB clock. This parameter can be a value between 1 and 256
- ***uint32_t OSPI_InitTypeDef::SampleShifting***
  It allows to delay to 1/2 cycle the data sampling in order to take in account external signal delays. This parameter can be a value of ***OSPI_SampleShifting***
- ***uint32_t OSPI_InitTypeDef::DelayHoldQuarterCycle***
  It allows to hold to 1/4 cycle the data. This parameter can be a value of ***OSPI_DelayHoldQuarterCycle***
- ***uint32_t OSPI_InitTypeDef::ChipSelectBoundary***
  It enables the transaction boundary feature and defines the boundary of bytes to release the chip select. This parameter can be a value between 0 and 31
- ***uint32_t OSPI_InitTypeDef::DelayBlockBypass***
  It enables the delay block bypass, so the sampling is not affected by the delay block. This parameter can be a value of ***OSPI_DelayBlockBypass***
- ***uint32_t OSPI_InitTypeDef::MaxTran***
  It enables the communication regulation feature. The chip select is released every MaxTran+1 bytes when the other OctoSPI request the access to the bus. This parameter can be a value between 0 and 255
- ***uint32_t OSPI_InitTypeDef::Refresh***
  It enables the refresh rate feature. The chip select is released every Refresh+1 clock cycles. This parameter can be a value between 0 and 0xFFFFFFFF

59.1.2 __OSPI_HandleTypeDef

__OSPI_HandleTypeDef is defined in the stm32h7xx_hal_ospi.h

Data Fields

- ***OCTOSPI_TypeDef * Instance***
- ***OSPI_InitTypeDef Init***
- ***uint8_t * pBuffPtr***
- ***__IO uint32_t XferSize***
- ***__IO uint32_t XferCount***
- ***MDMA_HandleTypeDef * hmdma***
- ***__IO uint32_t State***
- ***__IO uint32_t ErrorCode***
- ***uint32_t Timeout***
- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* FifoThresholdCallback***
- ***void(* CmdCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* TxCpltCallback***
- ***void(* RxHalfCpltCallback***
- ***void(* TxHalfCpltCallback***
- ***void(* StatusMatchCallback***
- ***void(* TimeOutCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***OCTOSPI_TypeDef* __OSPI_HandleTypeDef::Instance***
  OSPI registers base address
- ***OSPI_InitTypeDef __OSPI_HandleTypeDef::Init***
  OSPI initialization parameters

- ***uint8_t* __OSPI_HandleTypeDef::pBuffPtr***
  Address of the OSPI buffer for transfer
- ***__IO uint32_t __OSPI_HandleTypeDef::XferSize***
  Number of data to transfer
- ***__IO uint32_t __OSPI_HandleTypeDef::XferCount***
  Counter of data transferred
- ***MDMA_HandleTypeDef* __OSPI_HandleTypeDef::hmdma***
  Handle of the MDMA channel used for the transfer
- ***__IO uint32_t __OSPI_HandleTypeDef::State***
  Internal state of the OSPI HAL driver
- ***__IO uint32_t __OSPI_HandleTypeDef::ErrorCode***
  Error code in case of HAL driver internal error
- ***uint32_t __OSPI_HandleTypeDef::Timeout***
  Timeout used for the OSPI external device access
- ***void(* __OSPI_HandleTypeDef::ErrorCallback)(struct __OSPI_HandleTypeDef *hospi)***
- ***void(* __OSPI_HandleTypeDef::AbortCpltCallback)(struct __OSPI_HandleTypeDef *hospi)***
- ***void(* __OSPI_HandleTypeDef::FifoThresholdCallback)(struct __OSPI_HandleTypeDef *hospi)***
- ***void(* __OSPI_HandleTypeDef::CmdCpltCallback)(struct __OSPI_HandleTypeDef *hospi)***
- ***void(* __OSPI_HandleTypeDef::RxCpltCallback)(struct __OSPI_HandleTypeDef *hospi)***
- ***void(* __OSPI_HandleTypeDef::TxCpltCallback)(struct __OSPI_HandleTypeDef *hospi)***
- ***void(* __OSPI_HandleTypeDef::RxHalfCpltCallback)(struct __OSPI_HandleTypeDef *hospi)***
- ***void(* __OSPI_HandleTypeDef::TxHalfCpltCallback)(struct __OSPI_HandleTypeDef *hospi)***
- ***void(* __OSPI_HandleTypeDef::StatusMatchCallback)(struct __OSPI_HandleTypeDef *hospi)***
- ***void(* __OSPI_HandleTypeDef::TimeOutCallback)(struct __OSPI_HandleTypeDef *hospi)***
- ***void(* __OSPI_HandleTypeDef::MspInitCallback)(struct __OSPI_HandleTypeDef *hospi)***
- ***void(* __OSPI_HandleTypeDef::MspDeInitCallback)(struct __OSPI_HandleTypeDef *hospi)***

59.1.3 OSPI_RegularCmdTypeDef

OSPI_RegularCmdTypeDef is defined in the stm32h7xx_hal_ospi.h

Data Fields

- ***uint32_t OperationType***
- ***uint32_t FlashId***
- ***uint32_t Instruction***
- ***uint32_t InstructionMode***
- ***uint32_t InstructionSize***
- ***uint32_t InstructionDtrMode***
- ***uint32_t Address***
- ***uint32_t AddressMode***
- ***uint32_t AddressSize***
- ***uint32_t AddressDtrMode***
- ***uint32_t AlternateBytes***
- ***uint32_t AlternateBytesMode***
- ***uint32_t AlternateBytesSize***
- ***uint32_t AlternateBytesDtrMode***
- ***uint32_t DataMode***
- ***uint32_t NbData***
- ***uint32_t DataDtrMode***

- ***uint32_t DummyCycles***
- ***uint32_t DQSMode***
- ***uint32_t SIOOMode***

Field Documentation

- ***uint32_t OSPI_RegularCmdTypeDef::OperationType***

  It indicates if the configuration applies to the common registers or to the registers for the write operation (these registers are only used for memory-mapped mode). This parameter can be a value of ***OSPI_OperationType***
- ***uint32_t OSPI_RegularCmdTypeDef::FlashId***

  It indicates which external device is selected for this command (it applies only if Dualquad is disabled in the initialization structure). This parameter can be a value of ***OSPI_FlashID***
- ***uint32_t OSPI_RegularCmdTypeDef::Instruction***

  It contains the instruction to be sent to the device. This parameter can be a value between 0 and 0xFFFFFFFF
- ***uint32_t OSPI_RegularCmdTypeDef::InstructionMode***

  It indicates the mode of the instruction. This parameter can be a value of ***OSPI_InstructionMode***
- ***uint32_t OSPI_RegularCmdTypeDef::InstructionSize***

  It indicates the size of the instruction. This parameter can be a value of ***OSPI_InstructionSize***
- ***uint32_t OSPI_RegularCmdTypeDef::InstructionDtrMode***

  It enables or not the DTR mode for the instruction phase. This parameter can be a value of ***OSPI_InstructionDtrMode***
- ***uint32_t OSPI_RegularCmdTypeDef::Address***

  It contains the address to be sent to the device. This parameter can be a value between 0 and 0xFFFFFFFF
- ***uint32_t OSPI_RegularCmdTypeDef::AddressMode***

  It indicates the mode of the address. This parameter can be a value of ***OSPI_AddressMode***
- ***uint32_t OSPI_RegularCmdTypeDef::AddressSize***

  It indicates the size of the address. This parameter can be a value of ***OSPI_AddressSize***
- ***uint32_t OSPI_RegularCmdTypeDef::AddressDtrMode***

  It enables or not the DTR mode for the address phase. This parameter can be a value of ***OSPI_AddressDtrMode***
- ***uint32_t OSPI_RegularCmdTypeDef::AlternateBytes***

  It contains the alternate bytes to be sent to the device. This parameter can be a value between 0 and 0xFFFFFFFF
- ***uint32_t OSPI_RegularCmdTypeDef::AlternateBytesMode***

  It indicates the mode of the alternate bytes. This parameter can be a value of ***OSPI_AlternateBytesMode***
- ***uint32_t OSPI_RegularCmdTypeDef::AlternateBytesSize***

  It indicates the size of the alternate bytes. This parameter can be a value of ***OSPI_AlternateBytesSize***
- ***uint32_t OSPI_RegularCmdTypeDef::AlternateBytesDtrMode***

  It enables or not the DTR mode for the alternate bytes phase. This parameter can be a value of ***OSPI_AlternateBytesDtrMode***
- ***uint32_t OSPI_RegularCmdTypeDef::DataMode***

  It indicates the mode of the data. This parameter can be a value of ***OSPI_DataMode***
- ***uint32_t OSPI_RegularCmdTypeDef::NbData***

  It indicates the number of data transferred with this command. This field is only used for indirect mode. This parameter can be a value between 1 and 0xFFFFFFFF
- ***uint32_t OSPI_RegularCmdTypeDef::DataDtrMode***

  It enables or not the DTR mode for the data phase. This parameter can be a value of ***OSPI_DataDtrMode***
- ***uint32_t OSPI_RegularCmdTypeDef::DummyCycles***

  It indicates the number of dummy cycles inserted before data phase. This parameter can be a value between 0 and 31

- ***uint32_t OSPI_RegularCmdTypeDef::DQSMode***
  It enables or not the data strobe management. This parameter can be a value of ***OSPI_DQSMode***
- ***uint32_t OSPI_RegularCmdTypeDef::SIOOMode***
  It enables or not the SIOO mode. This parameter can be a value of ***OSPI_SIOOMode***

59.1.4 OSPI_HyperbusCfgTypeDef

OSPI_HyperbusCfgTypeDef is defined in the stm32h7xx_hal_ospi.h

Data Fields

- ***uint32_t RWRecoveryTime***
- ***uint32_t AccessTime***
- ***uint32_t WriteZeroLatency***
- ***uint32_t LatencyMode***

Field Documentation

- ***uint32_t OSPI_HyperbusCfgTypeDef::RWRecoveryTime***
  It indicates the number of cycles for the device read write recovery time. This parameter can be a value between 0 and 255
- ***uint32_t OSPI_HyperbusCfgTypeDef::AccessTime***
  It indicates the number of cycles for the device access time. This parameter can be a value between 0 and 255
- ***uint32_t OSPI_HyperbusCfgTypeDef::WriteZeroLatency***
  It enables or not the latency for the write access. This parameter can be a value of ***OSPI_WriteZeroLatency***
- ***uint32_t OSPI_HyperbusCfgTypeDef::LatencyMode***
  It configures the latency mode. This parameter can be a value of ***OSPI_LatencyMode***

59.1.5 OSPI_HyperbusCmdTypeDef

OSPI_HyperbusCmdTypeDef is defined in the stm32h7xx_hal_ospi.h

Data Fields

- ***uint32_t AddressSpace***
- ***uint32_t Address***
- ***uint32_t AddressSize***
- ***uint32_t NbData***
- ***uint32_t DQSMode***

Field Documentation

- ***uint32_t OSPI_HyperbusCmdTypeDef::AddressSpace***
  It indicates the address space accessed by the command. This parameter can be a value of ***OSPI_AddressSpace***
- ***uint32_t OSPI_HyperbusCmdTypeDef::Address***
  It contains the address to be sent tot he device. This parameter can be a value between 0 and 0xFFFFFFFF
- ***uint32_t OSPI_HyperbusCmdTypeDef::AddressSize***
  It indicates the size of the address. This parameter can be a value of ***OSPI_AddressSize***
- ***uint32_t OSPI_HyperbusCmdTypeDef::NbData***
  It indicates the number of data transferred with this command. This field is only used for indirect mode. This parameter can be a value between 1 and 0xFFFFFFFF In case of autopolling mode, this parameter can be any value between 1 and 4
- ***uint32_t OSPI_HyperbusCmdTypeDef::DQSMode***
  It enables or not the data strobe management. This parameter can be a value of ***OSPI_DQSMode***

59.1.6 OSPI_AutoPollingTypeDef

OSPI_AutoPollingTypeDef is defined in the stm32h7xx_hal_ospi.h

Data Fields

- ***uint32_t Match***
- ***uint32_t Mask***
- ***uint32_t MatchMode***
- ***uint32_t AutomaticStop***
- ***uint32_t Interval***

Field Documentation

- ***uint32_t OSPI_AutoPollingTypeDef::Match***

  Specifies the value to be compared with the masked status register to get a match. This parameter can be any value between 0 and 0xFFFFFFFF
- ***uint32_t OSPI_AutoPollingTypeDef::Mask***

  Specifies the mask to be applied to the status bytes received. This parameter can be any value between 0 and 0xFFFFFFFF
- ***uint32_t OSPI_AutoPollingTypeDef::MatchMode***

  Specifies the method used for determining a match. This parameter can be a value of ***OSPI_MatchMode***
- ***uint32_t OSPI_AutoPollingTypeDef::AutomaticStop***

  Specifies if automatic polling is stopped after a match. This parameter can be a value of ***OSPI_AutomaticStop***
- ***uint32_t OSPI_AutoPollingTypeDef::Interval***

  Specifies the number of clock cycles between two read during automatic polling phases. This parameter can be any value between 0 and 0xFFFF

59.1.7 OSPI_MemoryMappedTypeDef

OSPI_MemoryMappedTypeDef is defined in the stm32h7xx_hal_ospi.h

Data Fields

- ***uint32_t TimeOutActivation***
- ***uint32_t TimeOutPeriod***

Field Documentation

- ***uint32_t OSPI_MemoryMappedTypeDef::TimeOutActivation***

  Specifies if the timeout counter is enabled to release the chip select. This parameter can be a value of ***OSPI_TimeOutActivation***
- ***uint32_t OSPI_MemoryMappedTypeDef::TimeOutPeriod***

  Specifies the number of clock to wait when the FIFO is full before to release the chip select. This parameter can be any value between 0 and 0xFFFF

59.1.8 OSPIM_CfgTypeDef

OSPIM_CfgTypeDef is defined in the stm32h7xx_hal_ospi.h

Data Fields

- ***uint32_t ClkPort***
- ***uint32_t DQSPort***
- ***uint32_t NCSPort***
- ***uint32_t IOLowPort***
- ***uint32_t IOHighPort***
- ***uint32_t Req2AckTime***

Field Documentation

- ***uint32_t OSPIM_CfgTypeDef::ClkPort***

  It indicates which port of the OSPI IO Manager is used for the CLK pins. This parameter can be a value between 1 and 8
- ***uint32_t OSPIM_CfgTypeDef::DQSPort***

  It indicates which port of the OSPI IO Manager is used for the DQS pin. This parameter can be a value between 0 and 8, 0 means that signal not used

- ***uint32_t OSPIM_CfgTypeDef::NCSPort***
  It indicates which port of the OSPI IO Manager is used for the NCS pin. This parameter can be a value between 1 and 8
- ***uint32_t OSPIM_CfgTypeDef::IOLowPort***
  It indicates which port of the OSPI IO Manager is used for the IO[3:0] pins. This parameter can be a value of ***OSPIM_IOPort***
- ***uint32_t OSPIM_CfgTypeDef::IOHighPort***
  It indicates which port of the OSPI IO Manager is used for the IO[7:4] pins. This parameter can be a value of ***OSPIM_IOPort***
- ***uint32_t OSPIM_CfgTypeDef::Req2AckTime***
  It indicates the minimum switching duration (in number of clock cycles) expected if some signals are multiplexed in the OSPI IO Manager with the other OSPI. This parameter can be a value between 1 and 256

59.2 OSPI Firmware driver API description

The following section lists the various functions of the OSPI library.

59.2.1 How to use this driver

Initialization

As prerequisite, fill in the HAL_OSPI_MspInit() :

- Enable OctoSPI and OctoSPIM clocks interface with __HAL_RCC_OSPIx_CLK_ENABLE().
- Reset OctoSPI Peripheral with __HAL_RCC_OSPIx_FORCE_RESET() and __HAL_RCC_OSPIx_RELEASE_RESET().
- Enable the clocks for the OctoSPI GPIOS with __HAL_RCC_GPIOx_CLK_ENABLE().
- Configure these OctoSPI pins in alternate mode using HAL_GPIO_Init().
- If interrupt or DMA mode is used, enable and configure OctoSPI global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().
- If DMA mode is used, enable the clocks for the OctoSPI DMA channel with __HAL_RCC_DMAx_CLK_ENABLE(), configure DMA with HAL_DMA_Init(), link it with OctoSPI handle using __HAL_LINKDMA(), enable and configure DMA channel global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().

Configure the fifo threshold, the dual-quad mode, the memory type, the device size, the CS high time, the free running clock, the clock mode, the wrap size, the clock prescaler, the sample shifting, the hold delay and the CS boundary using the HAL_OSPI_Init() function.

When using Hyperbus, configure the RW recovery time, the access time, the write latency and the latency mode unsing the HAL_OSPI_HyperbusCfg() function.

Indirect functional mode

In regular mode, configure the command sequence using the HAL_OSPI_Command() or HAL_OSPI_Command_IT() functions :

- Instruction phase : the mode used and if present the size, the instruction opcode and the DTR mode.
- Address phase : the mode used and if present the size, the address value and the DTR mode.
- Alternate-bytes phase : the mode used and if present the size, the alternate bytes values and the DTR mode.
- Dummy-cycles phase : the number of dummy cycles (mode used is same as data phase).
- Data phase : the mode used and if present the number of bytes and the DTR mode.
- Data strobe (DQS) mode : the activation (or not) of this mode
- Sending Instruction Only Once (SIOO) mode : the activation (or not) of this mode.
- Flash identifier : in dual-quad mode, indicates which flash is concerned
- Operation type : always common configuration

In Hyperbus mode, configure the command sequence using the HAL_OSPI_HyperbusCmd() function :

- Address space : indicate if the access will be done in register or memory
- Address size
- Number of data
- Data strobe (DQS) mode : the activation (or not) of this mode

If no data is required for the command (only for regular mode, not for Hyperbus mode), it is sent directly to the memory :

- In polling mode, the output of the function is done when the transfer is complete.
- In interrupt mode, HAL_OSPI_CmdCpltCallback() will be called when the transfer is complete.

For the indirect write mode, use HAL_OSPI_Transmit(), HAL_OSPI_Transmit_DMA() or HAL_OSPI_Transmit_IT() after the command configuration :

- In polling mode, the output of the function is done when the transfer is complete.
- In interrupt mode, HAL_OSPI_FifoThresholdCallback() will be called when the fifo threshold is reached and HAL_OSPI_TxCpltCallback() will be called when the transfer is complete.
- In DMA mode, HAL_OSPI_TxHalfCpltCallback() will be called at the half transfer and HAL_OSPI_TxCpltCallback() will be called when the transfer is complete.

For the indirect read mode, use HAL_OSPI_Receive(), HAL_OSPI_Receive_DMA() or HAL_OSPI_Receive_IT() after the command configuration :

- In polling mode, the output of the function is done when the transfer is complete.
- In interrupt mode, HAL_OSPI_FifoThresholdCallback() will be called when the fifo threshold is reached and HAL_OSPI_RxCpltCallback() will be called when the transfer is complete.
- In DMA mode, HAL_OSPI_RxHalfCpltCallback() will be called at the half transfer and HAL_OSPI_RxCpltCallback() will be called when the transfer is complete.

Auto-polling functional mode

Configure the command sequence by the same way than the indirect mode

Configure the auto-polling functional mode using the HAL_OSPI_AutoPolling() or HAL_OSPI_AutoPolling_IT() functions :

- The size of the status bytes, the match value, the mask used, the match mode (OR/AND), the polling interval and the automatic stop activation.

After the configuration :

- In polling mode, the output of the function is done when the status match is reached. The automatic stop is activated to avoid an infinite loop.
- In interrupt mode, HAL_OSPI_StatusMatchCallback() will be called each time the status match is reached.

MDMA functional mode

Configure the SourceInc and DestinationInc of MDMA parameters in the HAL_OSPI_MspInit() function :

- MDMA settings for write operation :
  - The DestinationInc should be MDMA_DEST_INC_DISABLE
  - The SourceInc must be a value of @ref MDMA_Source_increment_mode (Except the MDMA_SRC_INC_DOUBLEWORD).
  - The SourceDataSize must be a value of @ref MDMA Source data size (Except the MDMA_SRC_DATASIZE_DOUBLEWORD) aligned with @ref MDMA_Source_increment_mode .
  - The DestDataSize must be a value of @ref MDMA Destination data size (Except the MDMA_DEST_DATASIZE_DOUBLEWORD)
- MDMA settings for read operation :
  - The SourceInc should be MDMA_SRC_INC_DISABLE
  - The DestinationInc must be a value of @ref MDMA_Destination_increment_mode (Except the MDMA_DEST_INC_DOUBLEWORD).
  - The SourceDataSize must be a value of @ref MDMA Source data size (Except the MDMA_SRC_DATASIZE_DOUBLEWORD) .
  - The DestDataSize must be a value of @ref MDMA Destination data size (Except the MDMA_DEST_DATASIZE_DOUBLEWORD) aligned with @ref MDMA_Destination_increment_mode.

- The buffer Transfer Length (BufferTransferLength) = number of bytes in the FIFO (FifoThreshold) of the Octospi.

In case of wrong MDMA setting

- For write operation :
  - If the DestinationInc is different to MDMA_DEST_INC_DISABLE , it will be disabled by the HAL_OSPI_Transmit_DMA().
- For read operation :
  - If the SourceInc is not set to MDMA_SRC_INC_DISABLE , it will be disabled by the HAL_OSPI_Receive_DMA().

Memory-mapped functional mode

Configure the command sequence by the same way than the indirect mode except for the operation type in regular mode :

- Operation type equals to read configuration : the command configuration applies to read access in memory-mapped mode
- Operation type equals to write configuration : the command configuration applies to write access in memory-mapped mode
- Both read and write configuration should be performed before activating memory-mapped mode

Configure the memory-mapped functional mode using the HAL_OSPI_MemoryMapped() functions :

- The timeout activation and the timeout period.

After the configuration, the OctoSPI will be used as soon as an access on the AHB is done on the address range. HAL_OSPI_TimeOutCallback() will be called when the timeout expires.

Errors management and abort functionality

HAL_OSPI_GetError() function gives the error raised during the last operation.

HAL_OSPI_Abort() and HAL_OSPI_AbortIT() functions aborts any on-going operation and flushes the fifo :

- In polling mode, the output of the function is done when the transfer complete bit is set and the busy bit cleared.
- In interrupt mode, HAL_OSPI_AbortCpltCallback() will be called when the transfer complete bit is set.

Control functions

HAL_OSPI_GetState() function gives the current state of the HAL OctoSPI driver.

HAL_OSPI_SetTimeout() function configures the timeout value used in the driver.

HAL_OSPI_SetFifoThreshold() function configures the threshold on the Fifo of the OSPI Peripheral.

HAL_OSPI_GetFifoThreshold() function gives the current of the Fifo's threshold

IO manager configuration functions

HAL_OSPIM_Config() function configures the IO manager for the OctoSPI instance.

Callback registration

The compilation define USE_HAL_OSPI_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use function HAL_OSPI_RegisterCallback() to register a user callback, it allows to register following callbacks:

- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- FifoThresholdCallback : callback when the fifo threshold is reached.
- CmdCpltCallback : callback when a command without data is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- TxCpltCallback : callback when a transmission transfer is completed.
- RxHalfCpltCallback : callback when half of the reception transfer is completed.
- TxHalfCpltCallback : callback when half of the transmission transfer is completed.

- StatusMatchCallback : callback when a status match occurs.
- TimeOutCallback : callback when the timeout perioed expires.
- MspInitCallback : OSPI MspInit.
- MspDeInitCallback : OSPI MspDeInit.

This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_OSPI_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. It allows to reset following callbacks:

- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- FifoThresholdCallback : callback when the fifo threshold is reached.
- CmdCpltCallback : callback when a command without data is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- TxCpltCallback : callback when a transmission transfer is completed.
- RxHalfCpltCallback : callback when half of the reception transfer is completed.
- TxHalfCpltCallback : callback when half of the transmission transfer is completed.
- StatusMatchCallback : callback when a status match occurs.
- TimeOutCallback : callback when the timeout perioed expires.
- MspInitCallback : OSPI MspInit.
- MspDeInitCallback : OSPI MspDeInit.

This function) takes as parameters the HAL peripheral handle and the Callback ID.

By default, after the HAL_OSPI_Init() and if the state is HAL_OSPI_STATE_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the HAL_OSPI_Init() and HAL_OSPI_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_OSPI_Init() and HAL_OSPI_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand)

Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_OSPI_RegisterCallback() before calling HAL_OSPI_DeInit() or HAL_OSPI_Init() function.

When The compilation define USE_HAL_OSPI_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

59.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to :

- Initialize the OctoSPI.
- De-initialize the OctoSPI.

This section contains the following APIs:

- ***HAL_OSPI_Init()***
- ***HAL_OSPI_MspInit()***
- ***HAL_OSPI_DeInit()***
- ***HAL_OSPI_MspDeInit()***

59.2.3 IO operation functions

This subsection provides a set of functions allowing to :

- Handle the interrupts.
- Handle the command sequence (regular and Hyperbus).
- Handle the Hyperbus configuration.
- Transmit data in blocking, interrupt or DMA mode.

- Receive data in blocking, interrupt or DMA mode.
- Manage the auto-polling functional mode.
- Manage the memory-mapped functional mode.

This section contains the following APIs:

- *HAL_OSPI_IRQHandler()*
- *HAL_OSPI_Command()*
- *HAL_OSPI_Command_IT()*
- *HAL_OSPI_HyperbusCfg()*
- *HAL_OSPI_HyperbusCmd()*
- *HAL_OSPI_Transmit()*
- *HAL_OSPI_Receive()*
- *HAL_OSPI_Transmit_IT()*
- *HAL_OSPI_Receive_IT()*
- *HAL_OSPI_Transmit_DMA()*
- *HAL_OSPI_Receive_DMA()*
- *HAL_OSPI_AutoPolling()*
- *HAL_OSPI_AutoPolling_IT()*
- *HAL_OSPI_MemoryMapped()*
- *HAL_OSPI_ErrorCallback()*
- *HAL_OSPI_AbortCpltCallback()*
- *HAL_OSPI_FifoThresholdCallback()*
- *HAL_OSPI_CmdCpltCallback()*
- *HAL_OSPI_RxCpltCallback()*
- *HAL_OSPI_TxCpltCallback()*
- *HAL_OSPI_RxHalfCpltCallback()*
- *HAL_OSPI_TxHalfCpltCallback()*
- *HAL_OSPI_StatusMatchCallback()*
- *HAL_OSPI_TimeOutCallback()*
- *HAL_OSPI_RegisterCallback()*
- *HAL_OSPI_UnRegisterCallback()*

59.2.4 Peripheral Control and State functions

This subsection provides a set of functions allowing to :

- Check in run-time the state of the driver.
- Check the error code set during last operation.
- Abort any operation.
- Manage the Fifo threshold.
- Configure the timeout duration used in the driver.

This section contains the following APIs:

- *HAL_OSPI_Abort()*
- *HAL_OSPI_Abort_IT()*
- *HAL_OSPI_SetFifoThreshold()*
- *HAL_OSPI_GetFifoThreshold()*
- *HAL_OSPI_SetTimeout()*
- *HAL_OSPI_GetError()*
- *HAL_OSPI_GetState()*

59.2.5 IO Manager configuration function

This subsection provides a set of functions allowing to :

- Configure the IO manager.

This section contains the following APIs:

- *HAL_OSPIM_Config()*

59.2.6 Detailed description of functions

HAL_OSPI_Init

Function name

HAL_StatusTypeDef HAL_OSPI_Init (OSPI_HandleTypeDef * hospi)

Function description

Initialize the OSPI mode according to the specified parameters in the OSPI_InitTypeDef and initialize the associated handle.

Parameters

- **hospi:** : OSPI handle

Return values

- **HAL:** status

HAL_OSPI_MspInit

Function name

void HAL_OSPI_MspInit (OSPI_HandleTypeDef * hospi)

Function description

Initialize the OSPI MSP.

Parameters

- **hospi:** : OSPI handle

Return values

- **None:**

HAL_OSPI_DeInit

Function name

HAL_StatusTypeDef HAL_OSPI_DeInit (OSPI_HandleTypeDef * hospi)

Function description

De-Initialize the OSPI peripheral.

Parameters

- **hospi:** : OSPI handle

Return values

- **HAL:** status

HAL_OSPI_MspDeInit

Function name

void HAL_OSPI_MspDeInit (OSPI_HandleTypeDef * hospi)

Function description

DeInitialize the OSPI MSP.

Parameters

- **hospi:** : OSPI handle

Return values

- **None:**

HAL_OSPI_IRQHandler

Function name

void HAL_OSPI_IRQHandler (OSPI_HandleTypeDef * hospi)

Function description

Handle OSPI interrupt request.

Parameters

- **hospi:** : OSPI handle

Return values

- **None:**

HAL_OSPI_Command

Function name

HAL_StatusTypeDef HAL_OSPI_Command (OSPI_HandleTypeDef * hospi, OSPI_RegularCmdTypeDef * cmd, uint32_t Timeout)

Function description

Set the command configuration.

Parameters

- **hospi:** : OSPI handle
- **cmd:** : structure that contains the command configuration information
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

HAL_OSPI_Command_IT

Function name

HAL_StatusTypeDef HAL_OSPI_Command_IT (OSPI_HandleTypeDef * hospi, OSPI_RegularCmdTypeDef * cmd)

Function description

Set the command configuration in interrupt mode.

Parameters

- **hospi:** : OSPI handle
- **cmd:** : structure that contains the command configuration information

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read or Write Modes

HAL_OSPI_HyperbusCfg

Function name

HAL_StatusTypeDef HAL_OSPI_HyperbusCfg (OSPI_HandleTypeDef * hospi, OSPI_HyperbusCfgTypeDef * cfg, uint32_t Timeout)

Function description

Configure the Hyperbus parameters.

Parameters

- **hospi:** : OSPI handle
- **cfg:** : Structure containing the Hyperbus configuration
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

HAL_OSPI_HyperbusCmd

Function name

HAL_StatusTypeDef HAL_OSPI_HyperbusCmd (OSPI_HandleTypeDef * hospi, OSPI_HyperbusCmdTypeDef * cmd, uint32_t Timeout)

Function description

Set the Hyperbus command configuration.

Parameters

- **hospi:** : OSPI handle
- **cmd:** : Structure containing the Hyperbus command
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

HAL_OSPI_Transmit

Function name

HAL_StatusTypeDef HAL_OSPI_Transmit (OSPI_HandleTypeDef * hospi, uint8_t * pData, uint32_t Timeout)

Function description

Transmit an amount of data in blocking mode.

Parameters

- **hospi:** : OSPI handle
- **pData:** : pointer to data buffer
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Write Mode

HAL_OSPI_Receive

Function name

HAL_StatusTypeDef HAL_OSPI_Receive (OSPI_HandleTypeDef * hospi, uint8_t * pData, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **hospi:** : OSPI handle
- **pData:** : pointer to data buffer
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read Mode

HAL_OSPI_Transmit_IT

Function name

HAL_StatusTypeDef HAL_OSPI_Transmit_IT (OSPI_HandleTypeDef * hospi, uint8_t * pData)

Function description

Send an amount of data in non-blocking mode with interrupt.

Parameters

- **hospi:** : OSPI handle
- **pData:** : pointer to data buffer

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Write Mode

HAL_OSPI_Receive_IT

Function name

HAL_StatusTypeDef HAL_OSPI_Receive_IT (OSPI_HandleTypeDef * hospi, uint8_t * pData)

Function description

Receive an amount of data in non-blocking mode with interrupt.

Parameters

- **hospi:** : OSPI handle
- **pData:** : pointer to data buffer

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read Mode

HAL_OSPI_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_OSPI_Transmit_DMA (OSPI_HandleTypeDef * hospi, uint8_t * pData)

Function description

Send an amount of data in non-blocking mode with DMA.

Parameters

- **hospi:** : OSPI handle
- **pData:** : pointer to data buffer

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Write Mode
- If DMA peripheral access is configured as halfword, the number of data and the fifo threshold should be aligned on halfword
- If DMA peripheral access is configured as word, the number of data and the fifo threshold should be aligned on word

HAL_OSPI_Receive_DMA

Function name

HAL_StatusTypeDef HAL_OSPI_Receive_DMA (OSPI_HandleTypeDef * hospi, uint8_t * pData)

Function description

Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hospi:** : OSPI handle
- **pData:** : pointer to data buffer.

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read Mode
- If DMA peripheral access is configured as halfword, the number of data and the fifo threshold should be aligned on halfword
- If DMA peripheral access is configured as word, the number of data and the fifo threshold should be aligned on word

HAL_OSPI_AutoPolling

Function name

HAL_StatusTypeDef HAL_OSPI_AutoPolling (OSPI_HandleTypeDef * hospi, OSPI_AutoPollingTypeDef * cfg, uint32_t Timeout)

Function description

Configure the OSPI Automatic Polling Mode in blocking mode.

Parameters

- **hospi:** : OSPI handle
- **cfg:** : structure that contains the polling configuration information.
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

Notes

- This function is used only in Automatic Polling Mode

HAL_OSPI_AutoPolling_IT

Function name

HAL_StatusTypeDef HAL_OSPI_AutoPolling_IT (OSPI_HandleTypeDef * hospi, OSPI_AutoPollingTypeDef * cfg)

Function description

Configure the OSPI Automatic Polling Mode in non-blocking mode.

Parameters

- **hospi:** : OSPI handle
- **cfg:** : structure that contains the polling configuration information.

Return values

- **HAL:** status

Notes

- This function is used only in Automatic Polling Mode

HAL_OSPI_MemoryMapped

Function name

HAL_StatusTypeDef HAL_OSPI_MemoryMapped (OSPI_HandleTypeDef * hospi, OSPI_MemoryMappedTypeDef * cfg)

Function description

Configure the Memory Mapped mode.

Parameters

- **hospi:** : OSPI handle
- **cfg:** : structure that contains the memory mapped configuration information.

Return values

- **HAL:** status

Notes

- This function is used only in Memory mapped Mode

HAL_OSPI_ErrorCallback

Function name

void HAL_OSPI_ErrorCallback (OSPI_HandleTypeDef * hospi)

Function description

Transfer Error callback.

Parameters

- **hospi:** : OSPI handle

Return values

- **None:**

HAL_OSPI_AbortCpltCallback

Function name

void HAL_OSPI_AbortCpltCallback (OSPI_HandleTypeDef * hospi)

Function description

Abort completed callback.

Parameters

- **hospi:** : OSPI handle

Return values

- **None:**

HAL_OSPI_FifoThresholdCallback

Function name

void HAL_OSPI_FifoThresholdCallback (OSPI_HandleTypeDef * hospi)

Function description

FIFO Threshold callback.

Parameters

- **hospi:** : OSPI handle

Return values

- **None:**

HAL_OSPI_CmdCpltCallback

Function name

void HAL_OSPI_CmdCpltCallback (OSPI_HandleTypeDef * hospi)

Function description

Command completed callback.

Parameters

- **hospi:** : OSPI handle

Return values

- **None:**

HAL_OSPI_RxCpltCallback

Function name

void HAL_OSPI_RxCpltCallback (OSPI_HandleTypeDef * hospi)

Function description

Rx Transfer completed callback.

Parameters

- **hospi:** : OSPI handle

Return values

- **None:**

HAL_OSPI_TxCpltCallback

Function name

void HAL_OSPI_TxCpltCallback (OSPI_HandleTypeDef * hospi)

Function description

Tx Transfer completed callback.

Parameters

- **hospi:** : OSPI handle

Return values

- **None:**

HAL_OSPI_RxHalfCpltCallback

Function name

void HAL_OSPI_RxHalfCpltCallback (OSPI_HandleTypeDef * hospi)

Function description

Rx Half Transfer completed callback.

Parameters

- **hospi:** : OSPI handle

Return values

- **None:**

HAL_OSPI_TxHalfCpltCallback

Function name

void HAL_OSPI_TxHalfCpltCallback (OSPI_HandleTypeDef * hospi)

Function description

Tx Half Transfer completed callback.

Parameters

- **hospi:** : OSPI handle

Return values

- **None:**

HAL_OSPI_StatusMatchCallback

Function name

void HAL_OSPI_StatusMatchCallback (OSPI_HandleTypeDef * hospi)

Function description

Status Match callback.

Parameters

- **hospi:** : OSPI handle

Return values

- **None:**

HAL_OSPI_TimeOutCallback

Function name

void HAL_OSPI_TimeOutCallback (OSPI_HandleTypeDef * hospi)

Function description

Timeout callback.

Parameters

- **hospi:** : OSPI handle

Return values

- **None:**

HAL_OSPI_RegisterCallback

Function name

HAL_StatusTypeDef HAL_OSPI_RegisterCallback (OSPI_HandleTypeDef * hospi, HAL_OSPI_CallbackIDTypeDef CallbackID, pOSPI_CallbackTypeDef pCallback)

Function description

Register a User OSPI Callback To be used instead of the weak (surcharged) predefined callback.

Parameters

- **hospi:** : OSPI handle
- **CallbackID:** : ID of the callback to be registered This parameter can be one of the following values:
  - HAL_OSPI_ERROR_CB_ID OSPI Error Callback ID
  - HAL_OSPI_ABORT_CB_ID OSPI Abort Callback ID
  - HAL_OSPI_FIFO_THRESHOLD_CB_ID OSPI FIFO Threshold Callback ID
  - HAL_OSPI_CMD_CPLT_CB_ID OSPI Command Complete Callback ID
  - HAL_OSPI_RX_CPLT_CB_ID OSPI Rx Complete Callback ID
  - HAL_OSPI_TX_CPLT_CB_ID OSPI Tx Complete Callback ID
  - HAL_OSPI_RX_HALF_CPLT_CB_ID OSPI Rx Half Complete Callback ID
  - HAL_OSPI_TX_HALF_CPLT_CB_ID OSPI Tx Half Complete Callback ID
  - HAL_OSPI_STATUS_MATCH_CB_ID OSPI Status Match Callback ID
  - HAL_OSPI_TIMEOUT_CB_ID OSPI Timeout Callback ID
  - HAL_OSPI_MSP_INIT_CB_ID OSPI MspInit callback ID
  - HAL_OSPI_MSP_DEINIT_CB_ID OSPI MspDeInit callback ID
- **pCallback:** : pointer to the Callback function

Return values

- **status:**

HAL_OSPI_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_OSPI_UnRegisterCallback (OSPI_HandleTypeDef * hospi, HAL_OSPI_CallbackIDTypeDef CallbackID)

Function description

Unregister a User OSPI Callback OSPI Callback is redirected to the weak (surcharged) predefined callback.

Parameters

- **hospi:** : OSPI handle
- **CallbackID:** : ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_OSPI_ERROR_CB_ID OSPI Error Callback ID
  - HAL_OSPI_ABORT_CB_ID OSPI Abort Callback ID
  - HAL_OSPI_FIFO_THRESHOLD_CB_ID OSPI FIFO Threshold Callback ID
  - HAL_OSPI_CMD_CPLT_CB_ID OSPI Command Complete Callback ID
  - HAL_OSPI_RX_CPLT_CB_ID OSPI Rx Complete Callback ID
  - HAL_OSPI_TX_CPLT_CB_ID OSPI Tx Complete Callback ID
  - HAL_OSPI_RX_HALF_CPLT_CB_ID OSPI Rx Half Complete Callback ID
  - HAL_OSPI_TX_HALF_CPLT_CB_ID OSPI Tx Half Complete Callback ID
  - HAL_OSPI_STATUS_MATCH_CB_ID OSPI Status Match Callback ID
  - HAL_OSPI_TIMEOUT_CB_ID OSPI Timeout Callback ID
  - HAL_OSPI_MSP_INIT_CB_ID OSPI MspInit callback ID
  - HAL_OSPI_MSP_DEINIT_CB_ID OSPI MspDeInit callback ID

Return values

- **status:**

HAL_OSPI_Abort

Function name

HAL_StatusTypeDef HAL_OSPI_Abort (OSPI_HandleTypeDef * hospi)

Function description

Abort the current transmission.

Parameters

- **hospi:** : OSPI handle

Return values

- **HAL:** status

HAL_OSPI_Abort_IT

Function name

HAL_StatusTypeDef HAL_OSPI_Abort_IT (OSPI_HandleTypeDef * hospi)

Function description

Abort the current transmission (non-blocking function)

Parameters

- **hospi:** : OSPI handle

Return values

- **HAL:** status

HAL_OSPI_SetFifoThreshold

Function name

HAL_StatusTypeDef HAL_OSPI_SetFifoThreshold (OSPI_HandleTypeDef * hospi, uint32_t Threshold)

Function description

Set OSPI Fifo threshold.

Parameters

- **hospi:** : OSPI handle.
- **Threshold:** : Threshold of the Fifo.

Return values

- **HAL:** status

HAL_OSPI_GetFifoThreshold

Function name

uint32_t HAL_OSPI_GetFifoThreshold (OSPI_HandleTypeDef * hospi)

Function description

Get OSPI Fifo threshold.

Parameters

- **hospi:** : OSPI handle.

Return values

- **Fifo:** threshold

HAL_OSPI_SetTimeout

Function name

HAL_StatusTypeDef HAL_OSPI_SetTimeout (OSPI_HandleTypeDef * hospi, uint32_t Timeout)

Function description

Set OSPI timeout.

Parameters

- **hospi:** : OSPI handle.
- **Timeout:** : Timeout for the memory access.

Return values

- **None:**

HAL_OSPI_GetError

Function name

uint32_t HAL_OSPI_GetError (OSPI_HandleTypeDef * hospi)

Function description

Return the OSPI error code.

Parameters

- **hospi:** : OSPI handle

Return values

- **OSPI:** Error Code

HAL_OSPI_GetState

Function name

uint32_t HAL_OSPI_GetState (OSPI_HandleTypeDef * hospi)

Function description

Return the OSPI handle state.

Parameters

- **hospi:** : OSPI handle

Return values

- **HAL:** state

HAL_OSPIM_Config

Function name

HAL_StatusTypeDef HAL_OSPIM_Config (OSPI_HandleTypeDef * hospi, OSPIM_CfgTypeDef * cfg, uint32_t Timeout)

Function description

Configure the OctoSPI IO manager.

Parameters

- **hospi:** : OSPI handle
- **cfg:** : Configuration of the IO Manager for the instance
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

59.3 OSPI Firmware driver defines

The following section lists the various define and macros of the module.

59.3.1 OSPI

OSPI

OSPI Address DTR Mode

HAL_OSPI_ADDRESS_DTR_DISABLE

DTR mode disabled for address phase

HAL_OSPI_ADDRESS_DTR_ENABLE

DTR mode enabled for address phase

OSPI Address Mode

HAL_OSPI_ADDRESS_NONE

No address

HAL_OSPI_ADDRESS_1_LINE

Address on a single line

HAL_OSPI_ADDRESS_2_LINES

Address on two lines

HAL_OSPI_ADDRESS_4_LINES

Address on four lines

HAL_OSPI_ADDRESS_8_LINES

Address on eight lines

OSPI Address Size

HAL_OSPI_ADDRESS_8_BITS

8-bit address

HAL_OSPI_ADDRESS_16_BITS

16-bit address

HAL_OSPI_ADDRESS_24_BITS

24-bit address

HAL_OSPI_ADDRESS_32_BITS

32-bit address

OSPI Hyperbus Address Space

HAL_OSPI_MEMORY_ADDRESS_SPACE

HyperBus memory mode

HAL_OSPI_REGISTER_ADDRESS_SPACE

HyperBus register mode

OSPI Alternate Bytes DTR Mode

HAL_OSPI_ALTERNATE_BYTES_DTR_DISABLE

DTR mode disabled for alternate bytes phase

HAL_OSPI_ALTERNATE_BYTES_DTR_ENABLE

DTR mode enabled for alternate bytes phase

OSPI Alternate Bytes Mode

HAL_OSPI_ALTERNATE_BYTES_NONE

No alternate bytes

HAL_OSPI_ALTERNATE_BYTES_1_LINE

Alternate bytes on a single line

HAL_OSPI_ALTERNATE_BYTES_2_LINES

Alternate bytes on two lines

HAL_OSPI_ALTERNATE_BYTES_4_LINES

Alternate bytes on four lines

HAL_OSPI_ALTERNATE_BYTES_8_LINES

Alternate bytes on eight lines

OSPI Alternate Bytes Size

HAL_OSPI_ALTERNATE_BYTES_8_BITS

8-bit alternate bytes

HAL_OSPI_ALTERNATE_BYTES_16_BITS

16-bit alternate bytes

HAL_OSPI_ALTERNATE_BYTES_24_BITS

24-bit alternate bytes

HAL_OSPI_ALTERNATE_BYTES_32_BITS

32-bit alternate bytes

OSPI Automatic Stop

HAL_OSPI_AUTOMATIC_STOP_DISABLE

AutoPolling stops only with abort or OSPI disabling

HAL_OSPI_AUTOMATIC_STOP_ENABLE

AutoPolling stops as soon as there is a match

OSPI Clock Mode

HAL_OSPI_CLOCK_MODE_0

CLK must stay low while nCS is high

HAL_OSPI_CLOCK_MODE_3

CLK must stay high while nCS is high

OSPI Data DTR Mode

HAL_OSPI_DATA_DTR_DISABLE

DTR mode disabled for data phase

HAL_OSPI_DATA_DTR_ENABLE

DTR mode enabled for data phase

OSPI Data Mode

HAL_OSPI_DATA_NONE

No data

HAL_OSPI_DATA_1_LINE

Data on a single line

HAL_OSPI_DATA_2_LINES

Data on two lines

HAL_OSPI_DATA_4_LINES

Data on four lines

HAL_OSPI_DATA_8_LINES

Data on eight lines

OSPI Delay Block Bypaas

HAL_OSPI_DELAY_BLOCK_USED

Sampling clock is delayed by the delay block

HAL_OSPI_DELAY_BLOCK_BYPASSED

Delay block is bypassed

OSPI Delay Hold Quarter Cycle

HAL_OSPI_DHQC_DISABLE

No Delay

HAL_OSPI_DHQC_ENABLE

Delay Hold 1/4 cycle

OSPI DQS Mode

HAL_OSPI_DQS_DISABLE

DQS disabled

HAL_OSPI_DQS_ENABLE

DQS enabled

OSPI Dual-Quad

HAL_OSPI_DUALQUAD_DISABLE

Dual-Quad mode disabled

HAL_OSPI_DUALQUAD_ENABLE

Dual-Quad mode enabled

OSPI Error Code

HAL_OSPI_ERROR_NONE

No error

HAL_OSPI_ERROR_TIMEOUT

Timeout error

HAL_OSPI_ERROR_TRANSFER

Transfer error

HAL_OSPI_ERROR_DMA

DMA transfer error

HAL_OSPI_ERROR_INVALID_PARAM

Invalid parameters error

HAL_OSPI_ERROR_INVALID_SEQUENCE

Sequence of the state machine is incorrect

HAL_OSPI_ERROR_INVALID_CALLBACK

Invalid callback error

OSPI Exported Macros

__HAL_OSPI_RESET_HANDLE_STATE

Description:

- Reset OSPI handle state.

Parameters:

- __HANDLE__: specifies the OSPI Handle.

Return value:

- None

__HAL_OSPI_ENABLE

Description:

- Enable the OSPI peripheral.

Parameters:

- __HANDLE__: specifies the OSPI Handle.

Return value:

- None

__HAL_OSPI_DISABLE

Description:

- Disable the OSPI peripheral.

Parameters:

- __HANDLE__: specifies the OSPI Handle.

Return value:

- None

__HAL_OSPI_ENABLE_IT

Description:

- Enable the specified OSPI interrupt.

Parameters:

- __HANDLE__: specifies the OSPI Handle.
- __INTERRUPT__: specifies the OSPI interrupt source to enable. This parameter can be one of the following values:
  - HAL_OSPI_IT_TO: OSPI Timeout interrupt
  - HAL_OSPI_IT_SM: OSPI Status match interrupt
  - HAL_OSPI_IT_FT: OSPI FIFO threshold interrupt
  - HAL_OSPI_IT_TC: OSPI Transfer complete interrupt
  - HAL_OSPI_IT_TE: OSPI Transfer error interrupt

Return value:

- None

__HAL_OSPI_DISABLE_IT

Description:

- Disable the specified OSPI interrupt.

Parameters:

- __HANDLE__: specifies the OSPI Handle.
- __INTERRUPT__: specifies the OSPI interrupt source to disable. This parameter can be one of the following values:
  - HAL_OSPI_IT_TO: OSPI Timeout interrupt
  - HAL_OSPI_IT_SM: OSPI Status match interrupt
  - HAL_OSPI_IT_FT: OSPI FIFO threshold interrupt
  - HAL_OSPI_IT_TC: OSPI Transfer complete interrupt
  - HAL_OSPI_IT_TE: OSPI Transfer error interrupt

Return value:

- None

__HAL_OSPI_GET_IT_SOURCE

Description:

- Check whether the specified OSPI interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the OSPI Handle.
- __INTERRUPT__: specifies the OSPI interrupt source to check. This parameter can be one of the following values:
  - HAL_OSPI_IT_TO: OSPI Timeout interrupt
  - HAL_OSPI_IT_SM: OSPI Status match interrupt
  - HAL_OSPI_IT_FT: OSPI FIFO threshold interrupt
  - HAL_OSPI_IT_TC: OSPI Transfer complete interrupt
  - HAL_OSPI_IT_TE: OSPI Transfer error interrupt

Return value:

- The: new state of __INTERRUPT__ (TRUE or FALSE).

__HAL_OSPI_GET_FLAG

Description:

- Check whether the selected OSPI flag is set or not.

Parameters:

- __HANDLE__: specifies the OSPI Handle.
- __FLAG__: specifies the OSPI flag to check. This parameter can be one of the following values:
  - HAL_OSPI_FLAG_BUSY: OSPI Busy flag
  - HAL_OSPI_FLAG_TO: OSPI Timeout flag
  - HAL_OSPI_FLAG_SM: OSPI Status match flag
  - HAL_OSPI_FLAG_FT: OSPI FIFO threshold flag
  - HAL_OSPI_FLAG_TC: OSPI Transfer complete flag
  - HAL_OSPI_FLAG_TE: OSPI Transfer error flag

Return value:

- None

__HAL_OSPI_CLEAR_FLAG

Description:

- Clears the specified OSPI's flag status.

Parameters:

- __HANDLE__: specifies the OSPI Handle.
- __FLAG__: specifies the OSPI clear register flag that needs to be set This parameter can be one of the following values:
  - HAL_OSPI_FLAG_TO: OSPI Timeout flag
  - HAL_OSPI_FLAG_SM: OSPI Status match flag
  - HAL_OSPI_FLAG_TC: OSPI Transfer complete flag
  - HAL_OSPI_FLAG_TE: OSPI Transfer error flag

Return value:

- None

OSPI Flags

HAL_OSPI_FLAG_BUSY

Busy flag: operation is ongoing

HAL_OSPI_FLAG_TO

Timeout flag: timeout occurs in memory-mapped mode

HAL_OSPI_FLAG_SM

Status match flag: received data matches in autopolling mode

HAL_OSPI_FLAG_FT

Fifo threshold flag: Fifo threshold reached or data left after read from memory is complete

HAL_OSPI_FLAG_TC

Transfer complete flag: programmed number of data have been transferred or the transfer has been aborted

HAL_OSPI_FLAG_TE

Transfer error flag: invalid address is being accessed

OSPI Flash Id

HAL_OSPI_FLASH_ID_1

FLASH 1 selected

HAL_OSPI_FLASH_ID_2

FLASH 2 selected

OSPI Free Running Clock

HAL_OSPI_FREERUNCLK_DISABLE

CLK is not free running

HAL_OSPI_FREERUNCLK_ENABLE

CLK is free running (always provided)

OSPI Instruction DTR Mode

HAL_OSPI_INSTRUCTION_DTR_DISABLE

DTR mode disabled for instruction phase

HAL_OSPI_INSTRUCTION_DTR_ENABLE

DTR mode enabled for instruction phase

OSPI Instruction Mode

HAL_OSPI_INSTRUCTION_NONE

No instruction

HAL_OSPI_INSTRUCTION_1_LINE

Instruction on a single line

HAL_OSPI_INSTRUCTION_2_LINES

Instruction on two lines

HAL_OSPI_INSTRUCTION_4_LINES

Instruction on four lines

HAL_OSPI_INSTRUCTION_8_LINES

Instruction on eight lines

OSPI Instruction Size

HAL_OSPI_INSTRUCTION_8_BITS

8-bit instruction

HAL_OSPI_INSTRUCTION_16_BITS

16-bit instruction

HAL_OSPI_INSTRUCTION_24_BITS

24-bit instruction

HAL_OSPI_INSTRUCTION_32_BITS

32-bit instruction

OSPI Interrupts

HAL_OSPI_IT_TO

Interrupt on the timeout flag

HAL_OSPI_IT_SM

Interrupt on the status match flag

HAL_OSPI_IT_FT

Interrupt on the fifo threshold flag

HAL_OSPI_IT_TC

Interrupt on the transfer complete flag

HAL_OSPI_IT_TE

Interrupt on the transfer error flag

OSPI Hyperbus Latency Mode

HAL_OSPI_VARIABLE_LATENCY

Variable initial latency

HAL_OSPI_FIXED_LATENCY

Fixed latency

OSPI Match Mode

HAL_OSPI_MATCH_MODE_AND

AND match mode between unmasked bits

HAL_OSPI_MATCH_MODE_OR

OR match mode between unmasked bits

OSPI Memory Type

HAL_OSPI_MEMTYPE_MICRON

Micron mode

HAL_OSPI_MEMTYPE_MACRONIX

Macronix mode

HAL_OSPI_MEMTYPE_APMEMORY

AP Memory mode

HAL_OSPI_MEMTYPE_MACRONIX_RAM

Macronix RAM mode

HAL_OSPI_MEMTYPE_HYPERBUS

Hyperbus mode

OSPI Operation Type

HAL_OSPI_OPTYPE_COMMON_CFG

Common configuration (indirect or auto-polling mode)

HAL_OSPI_OPTYPE_READ_CFG

Read configuration (memory-mapped mode)

HAL_OSPI_OPTYPE_WRITE_CFG

Write configuration (memory-mapped mode)

HAL_OSPI_OPTYPE_WRAP_CFG

Wrap configuration (memory-mapped mode)

OSPI Sample Shifting

HAL_OSPI_SAMPLE_SHIFTING_NONE

No shift

HAL_OSPI_SAMPLE_SHIFTING_HALFCYCLE

1/2 cycle shift

OSPI SIOO Mode

HAL_OSPI_SIOO_INST_EVERY_CMD

Send instruction on every transaction

HAL_OSPI_SIOO_INST_ONLY_FIRST_CMD

Send instruction only for the first command

OSPI State

HAL_OSPI_STATE_RESET

Initial state

HAL_OSPI_STATE_HYPERBUS_INIT

Initialization done in hyperbus mode but timing configuration not done

HAL_OSPI_STATE_READY

Driver ready to be used

HAL_OSPI_STATE_CMD_CFG

Command (regular or hyperbus) configured, ready for an action

HAL_OSPI_STATE_READ_CMD_CFG

Read command configuration done, not the write command configuration

HAL_OSPI_STATE_WRITE_CMD_CFG

Write command configuration done, not the read command configuration

HAL_OSPI_STATE_BUSY_CMD

Command without data on-going

HAL_OSPI_STATE_BUSY_TX

Indirect Tx on-going

HAL_OSPI_STATE_BUSY_RX

Indirect Rx on-going

HAL_OSPI_STATE_BUSY_AUTO_POLLING

Auto-polling on-going

HAL_OSPI_STATE_BUSY_MEM_MAPPED

Memory-mapped on-going

HAL_OSPI_STATE_ABORT

Abort on-going

HAL_OSPI_STATE_ERROR

Blocking error, driver should be re-initialized

OSPI Timeout Activation

HAL_OSPI_TIMEOUT_COUNTER_DISABLE

Timeout counter disabled, nCS remains active

HAL_OSPI_TIMEOUT_COUNTER_ENABLE

Timeout counter enabled, nCS released when timeout expires

OSPI Timeout definition

HAL_OSPI_TIMEOUT_DEFAULT_VALUE

OSPI Wrap-Size

HAL_OSPI_WRAP_NOT_SUPPORTED

wrapped reads are not supported by the memory

HAL_OSPI_WRAP_16_BYTES

external memory supports wrap size of 16 bytes

HAL_OSPI_WRAP_32_BYTES

external memory supports wrap size of 32 bytes

HAL_OSPI_WRAP_64_BYTES

external memory supports wrap size of 64 bytes

HAL_OSPI_WRAP_128_BYTES

external memory supports wrap size of 128 bytes

OSPI Hyperbus Write Zero Latency Activation

HAL_OSPI_LATENCY_ON_WRITE

Latency on write accesses

HAL_OSPI_NO_LATENCY_ON_WRITE

No latency on write accesses

60 HAL OTFDEC Generic Driver

60.1 OTFDEC Firmware driver registers structures

60.1.1 OTFDEC_RegionConfigTypeDef

OTFDEC_RegionConfigTypeDef is defined in the stm32h7xx_hal_otfdec.h

Data Fields

- ***uint32_t Nonce***
- ***uint32_t StartAddress***
- ***uint32_t EndAddress***
- ***uint16_t Version***

Field Documentation

- ***uint32_t OTFDEC_RegionConfigTypeDef::Nonce[2]***
  OTFDEC region nonce
- ***uint32_t OTFDEC_RegionConfigTypeDef::StartAddress***
  OTFDEC region start address
- ***uint32_t OTFDEC_RegionConfigTypeDef::EndAddress***
  OTFDEC region end address
- ***uint16_t OTFDEC_RegionConfigTypeDef::Version***
  OTFDEC region firmware version

60.1.2 __OTFDEC_HandleTypeDef

__OTFDEC_HandleTypeDef is defined in the stm32h7xx_hal_otfdec.h

Data Fields

- ***OTFDEC_TypeDef * Instance***
- ***HAL_OTFDEC_StateTypeDef State***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t ErrorCode***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***OTFDEC_TypeDef* __OTFDEC_HandleTypeDef::Instance***
  OTFDEC registers base address
- ***HAL_OTFDEC_StateTypeDef __OTFDEC_HandleTypeDef::State***
  OTFDEC state
- ***HAL_LockTypeDef __OTFDEC_HandleTypeDef::Lock***
  OTFDEC locking object
- ***__IO uint32_t __OTFDEC_HandleTypeDef::ErrorCode***
  OTFDEC error code
- ***void(* __OTFDEC_HandleTypeDef::ErrorCallback)(struct __OTFDEC_HandleTypeDef *hotfdec)***
  OTFDEC error callback
- ***void(* __OTFDEC_HandleTypeDef::MspInitCallback)(struct __OTFDEC_HandleTypeDef *hotfdec)***
  OTFDEC Msp Init callback
- ***void(* __OTFDEC_HandleTypeDef::MspDeInitCallback)(struct __OTFDEC_HandleTypeDef *hotfdec)***
  OTFDEC Msp DeInit callback

60.2 OTFDEC Firmware driver API description

The following section lists the various functions of the OTFDEC library.

60.2.1 How to use this driver

The OTFDEC HAL driver can be used as follows:

1. Declare an OTFDEC_HandleTypeDef handle structure (eg. OTFDEC_HandleTypeDef hotfdec).
2. Initialize the OTFDEC low level resources by implementing the HAL_OTFDEC_MspInit() API:
   - Enable the OTFDEC interface clock.
   - NVIC configuration if interrupts are used
     - Configure the OTFDEC interrupt priority.
     - Enable the NVIC OTFDEC IRQ handle.
3. Initialize the OTFDEC peripheral by calling the HAL_OTFDEC_Init() API.
4. For each region,
   - Configure the region deciphering mode by calling the HAL_OTFDEC_RegionSetMode() API.
   - Write the region Key by calling the HAL_OTFDEC_RegionSetKey() API. If desired, read the key CRC by calling HAL_OTFDEC_RegionGetKeyCRC() API and compare the result with the theoretically expected CRC.
   - Initialize the OTFDEC region config structure with the Nonce, protected region start and end addresses and firmware version, and wrap-up the region configuration by calling HAL_OTFDEC_RegionConfig() API.
5. At this point, the OTFDEC region configuration is done and the deciphering is enabled. The region can be deciphered on the fly after having made sure the OctoSPI is configured in memory-mapped mode.

Note: Warning: the OTFDEC deciphering is based on a different endianness compared to the AES-CTR as implemented in the AES peripheral. E.g., if the OTFEC resorts to the Key (B0, B1, B2, B3) where Bi are 32-bit longwords and B0 is the Least Significant Word, the AES has to be configured with the Key (B3, B2, B1, B0) to report the same result (with the same swapping applied to the Initialization Vector).

Callback registration

The compilation flag USE_HAL_OTFDEC_REGISTER_CALLBACKS, when set to 1, allows the user to configure dynamically the driver callbacks. Use Functions HAL_OTFDEC_RegisterCallback() to register an interrupt callback.

Function HAL_OTFDEC_RegisterCallback() allows to register following callbacks:

- ErrorCallback : OTFDEC error callback
- MspInitCallback : OTFDEC Msp Init callback
- MspDeInitCallback : OTFDEC Msp DeInit callback This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_OTFDEC_UnRegisterCallback to reset a callback to the default weak function.

HAL_OTFDEC_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- ErrorCallback : OTFDEC error callback
- MspInitCallback : OTFDEC Msp Init callback
- MspDeInitCallback : OTFDEC Msp DeInit callback

By default, after the HAL_OTFDEC_Init() and when the state is HAL_OTFDEC_STATE_RESET all callbacks are set to the corresponding weak functions: example HAL_OTFDEC_ErrorCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_OTFDEC_Init()HAL_OTFDEC_DeInit() only when these callbacks are null (not registered beforehand).

If MspInit or MspDeInit are not null, the HAL_OTFDEC_Init()/HAL_OTFDEC_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in HAL_OTFDEC_STATE_READY state only. Exception done MspInit/MspDeInit functions that can be registered/unregistered in HAL_OTFDEC_STATE_READY or HAL_OTFDEC_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit.

Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_OTFDEC_RegisterCallback() before calling HAL_OTFDEC_DeInit() or HAL_OTFDEC_Init() function.

When the compilation flag USE_HAL_OTFDEC_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

60.2.2 Initialization and de-initialization functions

This section contains the following APIs:

- *HAL_OTFDEC_Init()*
- *HAL_OTFDEC_DeInit()*
- *HAL_OTFDEC_MspInit()*
- *HAL_OTFDEC_MspDeInit()*
- *HAL_OTFDEC_RegisterCallback()*
- *HAL_OTFDEC_UnRegisterCallback()*

60.2.3 OTFDEC IRQ handler management

This section provides OTFDEC IRQ handler function.

This section contains the following APIs:

- *HAL_OTFDEC_IRQHandler()*
- *HAL_OTFDEC_ErrorCallback()*

60.2.4 Peripheral Control functions

This subsection permits to configure the OTFDEC peripheral

This section contains the following APIs:

- *HAL_OTFDEC_RegionKeyLock()*
- *HAL_OTFDEC_RegionSetKey()*
- *HAL_OTFDEC_RegionSetMode()*
- *HAL_OTFDEC_RegionConfig()*
- *HAL_OTFDEC_KeyCRCComputation()*
- *HAL_OTFDEC_RegionEnable()*
- *HAL_OTFDEC_RegionDisable()*

60.2.5 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral.

This section contains the following APIs:

- *HAL_OTFDEC_GetState()*
- *HAL_OTFDEC_RegionGetKeyCRC()*
- *HAL_OTFDEC_RegionGetConfig()*

60.2.6 Detailed description of functions

HAL_OTFDEC_Init

Function name

HAL_StatusTypeDef HAL_OTFDEC_Init (OTFDEC_HandleTypeDef * hotfdec)

Function description

Initialize the OTFDEC peripheral and create the associated handle.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module

Return values

- **HAL:** status

HAL_OTFDEC_DeInit

Function name

HAL_StatusTypeDef HAL_OTFDEC_DeInit (OTFDEC_HandleTypeDef * hotfdec)

Function description

DeInitialize the OTFDEC peripheral.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module

Return values

- **HAL:** status

HAL_OTFDEC_MspInit

Function name

void HAL_OTFDEC_MspInit (OTFDEC_HandleTypeDef * hotfdec)

Function description

Initialize the OTFDEC MSP.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module

Return values

- **None:**

HAL_OTFDEC_MspDeInit

Function name

void HAL_OTFDEC_MspDeInit (OTFDEC_HandleTypeDef * hotfdec)

Function description

DeInitialize OTFDEC MSP.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module

Return values

- **None:**

HAL_OTFDEC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_OTFDEC_RegisterCallback (OTFDEC_HandleTypeDef * hotfdec, HAL_OTFDEC_CallbackIDTypeDef CallbackID, pOTFDEC_CallbackTypeDef pCallback)

Function description

Register a User OTFDEC Callback To be used instead of the weak predefined callback.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_OTFDEC_ERROR_CB_ID OTFDEC error callback ID
  - HAL_OTFDEC_MSPINIT_CB_ID MspInit callback ID
  - HAL_OTFDEC_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_OTFDEC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_OTFDEC_UnRegisterCallback (OTFDEC_HandleTypeDef * hotfdec, HAL_OTFDEC_CallbackIDTypeDef CallbackID)

Function description

Unregister a OTFDEC Callback OTFDEC callback is redirected to the weak predefined callback.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_OTFDEC_ERROR_CB_ID OTFDEC error callback ID
  - HAL_OTFDEC_MSPINIT_CB_ID MspInit callback ID
  - HAL_OTFDEC_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_OTFDEC_IRQHandler

Function name

void HAL_OTFDEC_IRQHandler (OTFDEC_HandleTypeDef * hotfdec)

Function description

Handle OTFDEC interrupt request.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module

Return values

- **None:**

HAL_OTFDEC_ErrorCallback

Function name

void HAL_OTFDEC_ErrorCallback (OTFDEC_HandleTypeDef * hotfdec)

Function description

OTFDEC error callback.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module

Return values

- **None:**

HAL_OTFDEC_RegionKeyLock

Function name

HAL_StatusTypeDef HAL_OTFDEC_RegionKeyLock (OTFDEC_HandleTypeDef * hotfdec, uint32_t RegionIndex)

Function description

Lock region keys.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **RegionIndex:** index of region the keys of which are locked

Return values

- **HAL:** state

Notes

- Writes to this region KEYRx registers are ignored until next OTFDEC reset.

HAL_OTFDEC_RegionSetKey

Function name

HAL_StatusTypeDef HAL_OTFDEC_RegionSetKey (OTFDEC_HandleTypeDef * hotfdec, uint32_t RegionIndex, uint32_t * pKey)

Function description

Set region keys.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **RegionIndex:** index of region the keys of which are set
- **pKey:** pointer at set of keys

Return values

- **HAL:** state

Notes

- The API reads the key CRC computed by the peripheral and compares it with that theoretically expected. An error is reported if they are different.

HAL_OTFDEC_RegionSetMode

Function name

HAL_StatusTypeDef HAL_OTFDEC_RegionSetMode (OTFDEC_HandleTypeDef * hotfdec, uint32_t RegionIndex, uint32_t mode)

Function description

Set region mode.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **RegionIndex:** index of region the mode of which is set
- **mode:** This parameter can be only:
  - OTFDEC_REG_MODE_INSTRUCTION_ACCESSES_ONLY Only instruction accesses are decrypted
  - OTFDEC_REG_MODE_DATA_ACCESSES_ONLY Only data accesses are decrypted
  - OTFDEC_REG_MODE_INSTRUCTION_OR_DATA_ACCESSES All read accesses are decrypted (instruction or data)
  - OTFDEC_REG_MODE_INSTRUCTION_ACCESSES_ONLY_WITH_CIPHER Only instruction accesses are decrypted with proprietary cipher activated

Return values

- **HAL:** state

HAL_OTFDEC_RegionConfig

Function name

HAL_StatusTypeDef HAL_OTFDEC_RegionConfig (OTFDEC_HandleTypeDef * hotfdec, uint32_t RegionIndex, OTFDEC_RegionConfigTypeDef * Config, uint32_t lock)

Function description

Set region configuration.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **RegionIndex:** index of region that is configured
- **Config:** pointer on structure containing the region configuration parameters
- **lock:** configuration lock enable or disable parameter This parameter can be one of the following values:
  - OTFDEC_REG_CONFIGR_LOCK_DISABLE OTFDEC region configuration is not locked
  - OTFDEC_REG_CONFIGR_LOCK_ENABLE OTFDEC region configuration is locked

Return values

- **HAL:** state

Notes

- Region deciphering is enabled at the end of this function

HAL_OTFDEC_KeyCRCComputation

Function name

uint32_t HAL_OTFDEC_KeyCRCComputation (uint32_t * pKey)

Function description

Compute Key CRC.

Parameters

- **pKey:** pointer at set of keys

Return values

- **CRC:** value

HAL_OTFDEC_RegionEnable

Function name

HAL_StatusTypeDef HAL_OTFDEC_RegionEnable (OTFDEC_HandleTypeDef * hotfdec, uint32_t RegionIndex)

Function description

Enable region deciphering.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **RegionIndex:** index of region the deciphering is enabled

Return values

- **HAL:** state

Notes

- An error is reported when the configuration is locked.

HAL_OTFDEC_RegionDisable

Function name

HAL_StatusTypeDef HAL_OTFDEC_RegionDisable (OTFDEC_HandleTypeDef * hotfdec, uint32_t RegionIndex)

Function description

Disable region deciphering.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **RegionIndex:** index of region the deciphering is disabled

Return values

- **HAL:** state

Notes

- An error is reported when the configuration is locked.

HAL_OTFDEC_GetState

Function name

HAL_OTFDEC_StateTypeDef HAL_OTFDEC_GetState (OTFDEC_HandleTypeDef * hotfdec)

Function description

Return the OTFDEC state.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module

Return values

- **HAL:** state

HAL_OTFDEC_RegionGetKeyCRC

Function name

uint32_t HAL_OTFDEC_RegionGetKeyCRC (OTFDEC_HandleTypeDef * hotfdec, uint32_t RegionIndex)

Function description

Return region keys CRC.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **RegionIndex:** index of region the keys CRC of which is read

Return values

- **Key:** CRC

HAL_OTFDEC_RegionGetConfig

Function name

HAL_StatusTypeDef HAL_OTFDEC_RegionGetConfig (OTFDEC_HandleTypeDef * hotfdec, uint32_t RegionIndex, OTFDEC_RegionConfigTypeDef * Config)

Function description

Return region configuration parameters.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **RegionIndex:** index of region the configuration of which is read
- **Config:** pointer on structure that will be filled up with the region configuration parameters

Return values

- **HAL:** state

60.3 OTFDEC Firmware driver defines

The following section lists the various define and macros of the module.

60.3.1 OTFDEC

OTFDEC

OTFDEC Error Definition

HAL_OTFDEC_ERROR_NONE

No error

HAL_OTFDEC_SECURITY_ERROR

Security error

HAL_OTFDEC_EXECUTE_ERROR

Execute-only Execute-Never error

HAL_OTFDEC_KEY_ERROR

Key error

HAL_OTFDEC_ERROR_INVALID_CALLBACK

Invalid Callback error

OTFDEC Exported Macros

__HAL_OTFDEC_RESET_HANDLE_STATE

Description:

- Reset OTFDEC handle state.

Parameters:

- __HANDLE__: pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module

Return value:

- None

__HAL_OTFDEC_ENABLE_IT

Description:

- Enable OTFDEC peripheral interrupts combination.

Parameters:

- __HANDLE__: pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- __INTERRUPT__: mask on enabled interrupts This parameter can be one of the following values:
  - OTFDEC_SEC_ERROR_INT OTFDEC security error interrupt
  - OTFDEC_EXE_ERROR_INT OTFDEC execution error interrupt
  - OTFDEC_KEY_ERROR_INT OTFDEC key error interrupt
  - OTFDEC_SEC_EXE_ERROR_INT OTFDEC security and execution errors interrupts
  - OTFDEC_SEC_KEY_ERROR_INT OTFDEC security and key errors interrupts
  - OTFDEC_EXE_KEY_ERROR_INT OTFDEC execution and key errors interrupts
  - OTFDEC_ALL_INT OTFDEC all interrupts

Return value:

- None

__HAL_OTFDEC_DISABLE_IT

Description:

- Disable OTFDEC peripheral interrupts combination.

Parameters:

- __HANDLE__: pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- __INTERRUPT__: mask on disabled interrupts This parameter can be one of the following values:
  - OTFDEC_SEC_ERROR_INT OTFDEC security error interrupt
  - OTFDEC_EXE_ERROR_INT OTFDEC execution error interrupt
  - OTFDEC_KEY_ERROR_INT OTFDEC key error interrupt
  - OTFDEC_SEC_EXE_ERROR_INT OTFDEC security and execution errors interrupts
  - OTFDEC_SEC_KEY_ERROR_INT OTFDEC security and key errors interrupts
  - OTFDEC_EXE_KEY_ERROR_INT OTFDEC execution and key errors interrupts
  - OTFDEC_ALL_INT OTFDEC all interrupts

Return value:

- None

__HAL_OTFDEC_GET_FLAG

Description:

- Check whether the specified combination of OTFDEC interrupt flags is set or not.

Parameters:

- __HANDLE__: pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- __FLAG__: mask on combination of interrupts flags This parameter can be one of the following values:
  - OTFDEC_SEC_ERROR_INT OTFDEC security error interrupt flag
  - OTFDEC_EXE_ERROR_INT OTFDEC execution error interrupt flag
  - OTFDEC_KEY_ERROR_INT OTFDEC key error interrupt flag
  - OTFDEC_SEC_EXE_ERROR_INT OTFDEC security and execution errors interrupts flags
  - OTFDEC_SEC_KEY_ERROR_INT OTFDEC security and key errors interrupts flags
  - OTFDEC_EXE_KEY_ERROR_INT OTFDEC execution and key errors interrupts flag
  - OTFDEC_ALL_INT OTFDEC all interrupts flags

Return value:

- The: state of __FLAG__ (TRUE or FALSE).

__HAL_OTFDEC_CLEAR_FLAG

Description:

- Clear the specified combination of OTFDEC interrupt flags.

Parameters:

- __HANDLE__: pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- __FLAG__: mask on combination of interrupts flags This parameter can be one of the following values:
  - OTFDEC_SEC_ERROR_INT OTFDEC security error interrupt flag
  - OTFDEC_EXE_ERROR_INT OTFDEC execution error interrupt flag
  - OTFDEC_KEY_ERROR_INT OTFDEC key error interrupt flag
  - OTFDEC_SEC_EXE_ERROR_INT OTFDEC security and execution errors interrupts flags
  - OTFDEC_SEC_KEY_ERROR_INT OTFDEC security and key errors interrupts flags
  - OTFDEC_EXE_KEY_ERROR_INT OTFDEC execution and key errors interrupts flag
  - OTFDEC_ALL_INT OTFDEC all interrupts flags

Return value:

- None

OTFDEC Interrupts

OTFDEC_SEC_ERROR_INT

OTFDEC security error interrupt

OTFDEC_EXE_ERROR_INT

OTFDEC execution error interrupt

OTFDEC_KEY_ERROR_INT

OTFDEC key error interrupt

OTFDEC_SEC_EXE_ERROR_INT

OTFDEC security and execution errors interrupts

OTFDEC_SEC_KEY_ERROR_INT

OTFDEC security and key errors interrupts

OTFDEC_EXE_KEY_ERROR_INT

OTFDEC execution and key errors interrupts

OTFDEC_ALL_INT

OTFDEC all interrupts

OTFDEC Regions Index

OTFDEC_REGION1

OTFDEC region 1

OTFDEC_REGION2

OTFDEC region 2

OTFDEC_REGION3

OTFDEC region 3

OTFDEC_REGION4

OTFDEC region 4

OTFDEC Region Configuration Lock

OTFDEC_REG_CONFIGR_LOCK_DISABLE

OTFDEC region configuration lock disable

OTFDEC_REG_CONFIGR_LOCK_ENABLE

OTFDEC region configuration lock enable

OTFDEC Region Enable

OTFDEC_REG_CONFIGR_REG_DISABLE

OTFDEC region encryption or on-the-fly decryption disable

OTFDEC_REG_CONFIGR_REG_ENABLE

OTFDEC region encryption or on-the-fly decryption enable

OTFDEC Region Operating Mode

OTFDEC_REG_MODE_INSTRUCTION_ACCESSES_ONLY

Only instruction accesses are decrypted

OTFDEC_REG_MODE_DATA_ACCESSES_ONLY

Only data accesses are decrypted

OTFDEC_REG_MODE_INSTRUCTION_OR_DATA_ACCESSES

All read accesses are decrypted

OTFDEC_REG_MODE_INSTRUCTION_ACCESSES_ONLY_WITH_CIPHER

Only instruction accesses are decrypted with proprietary cipher activated

61 HAL PCD Generic Driver

61.1 PCD Firmware driver registers structures

61.1.1 __PCD_HandleTypeDef

__PCD_HandleTypeDef is defined in the stm32h7xx_hal_pcd.h

Data Fields

- ***PCD_TypeDef * Instance***
- ***PCD_InitTypeDef Init***
- ***__IO uint8_t USB_Address***
- ***PCD_EPTypeDef IN_ep***
- ***PCD_EPTypeDef OUT_ep***
- ***HAL_LockTypeDef Lock***
- ***__IO PCD_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***uint32_t Setup***
- ***PCD_LPM_StateTypeDef LPM_State***
- ***uint32_t BESL***
- ***uint32_t FrameNumber***
- ***uint32_t lpm_active***
- ***uint32_t battery_charging_active***
- ***void * pData***
- ***void(* SOFCallback***
- ***void(* SetupStageCallback***
- ***void(* ResetCallback***
- ***void(* SuspendCallback***
- ***void(* ResumeCallback***
- ***void(* ConnectCallback***
- ***void(* DisconnectCallback***
- ***void(* DataOutStageCallback***
- ***void(* DataInStageCallback***
- ***void(* ISOOUTIncompleteCallback***
- ***void(* ISOINIncompleteCallback***
- ***void(* BCDCallback***
- ***void(* LPMCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***PCD_TypeDef* __PCD_HandleTypeDef::Instance***
  Register base address
- ***PCD_InitTypeDef __PCD_HandleTypeDef::Init***
  PCD required parameters
- ***__IO uint8_t __PCD_HandleTypeDef::USB_Address***
  USB Address
- ***PCD_EPTypeDef __PCD_HandleTypeDef::IN_ep[16]***
  IN endpoint parameters

- ***PCD_EPTypeDef __PCD_HandleTypeDef::OUT_ep[16]***
  OUT endpoint parameters
- ***HAL_LockTypeDef __PCD_HandleTypeDef::Lock***
  PCD peripheral status
- ***__IO PCD_StateTypeDef __PCD_HandleTypeDef::State***
  PCD communication state
- ***__IO uint32_t __PCD_HandleTypeDef::ErrorCode***
  PCD Error code
- ***uint32_t __PCD_HandleTypeDef::Setup[12]***
  Setup packet buffer
- ***PCD_LPM_StateTypeDef __PCD_HandleTypeDef::LPM_State***
  LPM State
- ***uint32_t __PCD_HandleTypeDef::BESL***
- ***uint32_t __PCD_HandleTypeDef::FrameNumber***
  Store Current Frame number
- ***uint32_t __PCD_HandleTypeDef::lpm_active***
  Enable or disable the Link Power Management . This parameter can be set to ENABLE or DISABLE
- ***uint32_t __PCD_HandleTypeDef::battery_charging_active***
  Enable or disable Battery charging. This parameter can be set to ENABLE or DISABLE
- ***void* __PCD_HandleTypeDef::pData***
  Pointer to upper stack Handler
- ***void(* __PCD_HandleTypeDef::SOFCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD SOF callback
- ***void(* __PCD_HandleTypeDef::SetupStageCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Setup Stage callback
- ***void(* __PCD_HandleTypeDef::ResetCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Reset callback
- ***void(* __PCD_HandleTypeDef::SuspendCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Suspend callback
- ***void(* __PCD_HandleTypeDef::ResumeCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Resume callback
- ***void(* __PCD_HandleTypeDef::ConnectCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Connect callback
- ***void(* __PCD_HandleTypeDef::DisconnectCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Disconnect callback
- ***void(* __PCD_HandleTypeDef::DataOutStageCallback)(struct __PCD_HandleTypeDef *hpcd, uint8_t epnum)***
  USB OTG PCD Data OUT Stage callback
- ***void(* __PCD_HandleTypeDef::DataInStageCallback)(struct __PCD_HandleTypeDef *hpcd, uint8_t epnum)***
  USB OTG PCD Data IN Stage callback
- ***void(* __PCD_HandleTypeDef::ISOOUTIncompleteCallback)(struct __PCD_HandleTypeDef *hpcd, uint8_t epnum)***
  USB OTG PCD ISO OUT Incomplete callback
- ***void(* __PCD_HandleTypeDef::ISOINIncompleteCallback)(struct __PCD_HandleTypeDef *hpcd, uint8_t epnum)***
  USB OTG PCD ISO IN Incomplete callback
- ***void(* __PCD_HandleTypeDef::BCDCallback)(struct __PCD_HandleTypeDef *hpcd, PCD_BCD_MsgTypeDef msg)***
  USB OTG PCD BCD callback

- ***void(* __PCD_HandleTypeDef::LPMCallback)(struct __PCD_HandleTypeDef *hpcd, PCD_LPM_MsgTypeDef msg)***
  USB OTG PCD LPM callback
- ***void(* __PCD_HandleTypeDef::MspInitCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Msp Init callback
- ***void(* __PCD_HandleTypeDef::MspDeInitCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Msp DeInit callback

61.2 PCD Firmware driver API description

The following section lists the various functions of the PCD library.

61.2.1 How to use this driver

The PCD HAL driver can be used as follows:

1. Declare a PCD_HandleTypeDef handle structure, for example: PCD_HandleTypeDef hpcd;
2. Fill parameters of Init structure in HCD handle
3. Call HAL_PCD_Init() API to initialize the PCD peripheral (Core, Device core, ...)
4. Initialize the PCD low level resources through the HAL_PCD_MspInit() API:
   a. Enable the PCD/USB Low Level interface clock using
      - __HAL_RCC_USB_OTG_FS_CLK_ENABLE();
      - __HAL_RCC_USB_OTG_HS_CLK_ENABLE(); (For High Speed Mode)
   b. Initialize the related GPIO clocks
   c. Configure PCD pin-out
   d. Configure PCD NVIC interrupt
5. Associate the Upper USB device stack to the HAL PCD Driver:
   a. hpcd.pData = pdev;
6. Enable PCD transmission and reception:
   a. HAL_PCD_Start();

61.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

This section contains the following APIs:

- ***HAL_PCD_Init()***
- ***HAL_PCD_DeInit()***
- ***HAL_PCD_MspInit()***
- ***HAL_PCD_MspDeInit()***
- ***HAL_PCD_RegisterCallback()***
- ***HAL_PCD_UnRegisterCallback()***
- ***HAL_PCD_RegisterDataOutStageCallback()***
- ***HAL_PCD_UnRegisterDataOutStageCallback()***
- ***HAL_PCD_RegisterDataInStageCallback()***
- ***HAL_PCD_UnRegisterDataInStageCallback()***
- ***HAL_PCD_RegisterIsoOutIncpltCallback()***
- ***HAL_PCD_UnRegisterIsoOutIncpltCallback()***
- ***HAL_PCD_RegisterIsoInIncpltCallback()***
- ***HAL_PCD_UnRegisterIsoInIncpltCallback()***
- ***HAL_PCD_RegisterBcdCallback()***
- ***HAL_PCD_UnRegisterBcdCallback()***
- ***HAL_PCD_RegisterLpmCallback()***
- ***HAL_PCD_UnRegisterLpmCallback()***

61.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the PCD data transfers.
This section contains the following APIs:

- *HAL_PCD_Start()*
- *HAL_PCD_Stop()*
- *HAL_PCD_IRQHandler()*
- *HAL_PCD_DataOutStageCallback()*
- *HAL_PCD_DataInStageCallback()*
- *HAL_PCD_SetupStageCallback()*
- *HAL_PCD_SOFCallback()*
- *HAL_PCD_ResetCallback()*
- *HAL_PCD_SuspendCallback()*
- *HAL_PCD_ResumeCallback()*
- *HAL_PCD_ISOOUTIncompleteCallback()*
- *HAL_PCD_ISOINIncompleteCallback()*
- *HAL_PCD_ConnectCallback()*
- *HAL_PCD_DisconnectCallback()*

61.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the PCD data transfers.
This section contains the following APIs:

- *HAL_PCD_DevConnect()*
- *HAL_PCD_DevDisconnect()*
- *HAL_PCD_SetAddress()*
- *HAL_PCD_EP_Open()*
- *HAL_PCD_EP_Close()*
- *HAL_PCD_EP_Receive()*
- *HAL_PCD_EP_GetRxCount()*
- *HAL_PCD_EP_Transmit()*
- *HAL_PCD_EP_SetStall()*
- *HAL_PCD_EP_ClrStall()*
- *HAL_PCD_EP_Abort()*
- *HAL_PCD_EP_Flush()*
- *HAL_PCD_ActivateRemoteWakeup()*
- *HAL_PCD_DeActivateRemoteWakeup()*
- *HAL_PCD_SetTestMode()*

61.2.5 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.
This section contains the following APIs:

- *HAL_PCD_GetState()*
- *HAL_PCD_SetTestMode()*

61.2.6 Detailed description of functions

HAL_PCD_Init

Function name

HAL_StatusTypeDef HAL_PCD_Init (PCD_HandleTypeDef * hpcd)

Function description

Initializes the PCD according to the specified parameters in the PCD_InitTypeDef and initialize the associated handle.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_DeInit

Function name

HAL_StatusTypeDef HAL_PCD_DeInit (PCD_HandleTypeDef * hpcd)

Function description

DeInitializes the PCD peripheral.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_MspInit

Function name

void HAL_PCD_MspInit (PCD_HandleTypeDef * hpcd)

Function description

Initializes the PCD MSP.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_MspDeInit

Function name

void HAL_PCD_MspDeInit (PCD_HandleTypeDef * hpcd)

Function description

DeInitializes PCD MSP.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_RegisterCallback

Function name

HAL_StatusTypeDef HAL_PCD_RegisterCallback (PCD_HandleTypeDef * hpcd, HAL_PCD_CallbackIDTypeDef CallbackID, pPCD_CallbackTypeDef pCallback)

Function description

Register a User USB PCD Callback To be used instead of the weak predefined callback.

Parameters

- **hpcd:** USB PCD handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_PCD_SOF_CB_ID USB PCD SOF callback ID
  - HAL_PCD_SETUPSTAGE_CB_ID USB PCD Setup callback ID
  - HAL_PCD_RESET_CB_ID USB PCD Reset callback ID
  - HAL_PCD_SUSPEND_CB_ID USB PCD Suspend callback ID
  - HAL_PCD_RESUME_CB_ID USB PCD Resume callback ID
  - HAL_PCD_CONNECT_CB_ID USB PCD Connect callback ID
  - HAL_PCD_DISCONNECT_CB_ID USB PCD Disconnect callback ID
  - HAL_PCD_MSPINIT_CB_ID MspDeInit callback ID
  - HAL_PCD_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_PCD_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_PCD_UnRegisterCallback (PCD_HandleTypeDef * hpcd, HAL_PCD_CallbackIDTypeDef CallbackID)

Function description

Unregister an USB PCD Callback USB PCD callback is redirected to the weak predefined callback.

Parameters

- **hpcd:** USB PCD handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_PCD_SOF_CB_ID USB PCD SOF callback ID
  - HAL_PCD_SETUPSTAGE_CB_ID USB PCD Setup callback ID
  - HAL_PCD_RESET_CB_ID USB PCD Reset callback ID
  - HAL_PCD_SUSPEND_CB_ID USB PCD Suspend callback ID
  - HAL_PCD_RESUME_CB_ID USB PCD Resume callback ID
  - HAL_PCD_CONNECT_CB_ID USB PCD Connect callback ID
  - HAL_PCD_DISCONNECT_CB_ID USB PCD Disconnect callback ID
  - HAL_PCD_MSPINIT_CB_ID MspDeInit callback ID
  - HAL_PCD_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_PCD_RegisterDataOutStageCallback

Function name

HAL_StatusTypeDef HAL_PCD_RegisterDataOutStageCallback (PCD_HandleTypeDef * hpcd, pPCD_DataOutStageCallbackTypeDef pCallback)

Function description

Register USB PCD Data OUT Stage Callback To be used instead of the weak HAL_PCD_DataOutStageCallback() predefined callback.

Parameters

- **hpcd:** PCD handle
- **pCallback:** pointer to the USB PCD Data OUT Stage Callback function

Return values

- **HAL:** status

HAL_PCD_UnRegisterDataOutStageCallback

Function name

HAL_StatusTypeDef HAL_PCD_UnRegisterDataOutStageCallback (PCD_HandleTypeDef * hpcd)

Function description

Unregister the USB PCD Data OUT Stage Callback USB PCD Data OUT Stage Callback is redirected to the weak HAL_PCD_DataOutStageCallback() predefined callback.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_RegisterDataInStageCallback

Function name

HAL_StatusTypeDef HAL_PCD_RegisterDataInStageCallback (PCD_HandleTypeDef * hpcd, pPCD_DataInStageCallbackTypeDef pCallback)

Function description

Register USB PCD Data IN Stage Callback To be used instead of the weak HAL_PCD_DataInStageCallback() predefined callback.

Parameters

- **hpcd:** PCD handle
- **pCallback:** pointer to the USB PCD Data IN Stage Callback function

Return values

- **HAL:** status

HAL_PCD_UnRegisterDataInStageCallback

Function name

HAL_StatusTypeDef HAL_PCD_UnRegisterDataInStageCallback (PCD_HandleTypeDef * hpcd)

Function description

Unregister the USB PCD Data IN Stage Callback USB PCD Data OUT Stage Callback is redirected to the weak HAL_PCD_DataInStageCallback() predefined callback.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_RegisterIsoOutIncpltCallback

Function name

HAL_StatusTypeDef HAL_PCD_RegisterIsoOutIncpltCallback (PCD_HandleTypeDef * hpcd, pPCD_IsoOutIncpltCallbackTypeDef pCallback)

Function description

Register USB PCD Iso OUT incomplete Callback To be used instead of the weak HAL_PCD_ISOOUTIncompleteCallback() predefined callback.

Parameters

- **hpcd:** PCD handle
- **pCallback:** pointer to the USB PCD Iso OUT incomplete Callback function

Return values

- **HAL:** status

HAL_PCD_UnRegisterIsoOutIncpltCallback

Function name

HAL_StatusTypeDef HAL_PCD_UnRegisterIsoOutIncpltCallback (PCD_HandleTypeDef * hpcd)

Function description

Unregister the USB PCD Iso OUT incomplete Callback USB PCD Iso OUT incomplete Callback is redirected to the weak HAL_PCD_ISOOUTIncompleteCallback() predefined callback.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_RegisterIsoInIncpltCallback

Function name

HAL_StatusTypeDef HAL_PCD_RegisterIsoInIncpltCallback (PCD_HandleTypeDef * hpcd, pPCD_IsoInIncpltCallbackTypeDef pCallback)

Function description

Register USB PCD Iso IN incomplete Callback To be used instead of the weak HAL_PCD_ISOINIncompleteCallback() predefined callback.

Parameters

- **hpcd:** PCD handle
- **pCallback:** pointer to the USB PCD Iso IN incomplete Callback function

Return values

- **HAL:** status

HAL_PCD_UnRegisterIsoInIncpltCallback

Function name

HAL_StatusTypeDef HAL_PCD_UnRegisterIsoInIncpltCallback (PCD_HandleTypeDef * hpcd)

Function description

Unregister the USB PCD Iso IN incomplete Callback USB PCD Iso IN incomplete Callback is redirected to the weak HAL_PCD_ISOINIncompleteCallback() predefined callback.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_RegisterBcdCallback

Function name

HAL_StatusTypeDef HAL_PCD_RegisterBcdCallback (PCD_HandleTypeDef * hpcd, pPCD_BcdCallbackTypeDef pCallback)

Function description

Register USB PCD BCD Callback To be used instead of the weak HAL_PCDEx_BCD_Callback() predefined callback.

Parameters

- **hpcd:** PCD handle
- **pCallback:** pointer to the USB PCD BCD Callback function

Return values

- **HAL:** status

HAL_PCD_UnRegisterBcdCallback

Function name

HAL_StatusTypeDef HAL_PCD_UnRegisterBcdCallback (PCD_HandleTypeDef * hpcd)

Function description

Unregister the USB PCD BCD Callback USB BCD Callback is redirected to the weak HAL_PCDEx_BCD_Callback() predefined callback.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_RegisterLpmCallback

Function name

HAL_StatusTypeDef HAL_PCD_RegisterLpmCallback (PCD_HandleTypeDef * hpcd, pPCD_LpmCallbackTypeDef pCallback)

Function description

Register USB PCD LPM Callback To be used instead of the weak HAL_PCDEx_LPM_Callback() predefined callback.

Parameters

- **hpcd:** PCD handle
- **pCallback:** pointer to the USB PCD LPM Callback function

Return values

- **HAL:** status

HAL_PCD_UnRegisterLpmCallback

Function name

HAL_StatusTypeDef HAL_PCD_UnRegisterLpmCallback (PCD_HandleTypeDef * hpcd)

Function description

Unregister the USB PCD LPM Callback USB LPM Callback is redirected to the weak HAL_PCDEx_LPM_Callback() predefined callback.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_Start

Function name

HAL_StatusTypeDef HAL_PCD_Start (PCD_HandleTypeDef * hpcd)

Function description

Start the USB device.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_Stop

Function name

HAL_StatusTypeDef HAL_PCD_Stop (PCD_HandleTypeDef * hpcd)

Function description

Stop the USB device.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_IRQHandler

Function name

void HAL_PCD_IRQHandler (PCD_HandleTypeDef * hpcd)

Function description

Handles PCD interrupt request.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_SOFCallback

Function name

void HAL_PCD_SOFCallback (PCD_HandleTypeDef * hpcd)

Function description

USB Start Of Frame callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_SetupStageCallback

Function name

void HAL_PCD_SetupStageCallback (PCD_HandleTypeDef * hpcd)

Function description

Setup stage callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_ResetCallback

Function name

void HAL_PCD_ResetCallback (PCD_HandleTypeDef * hpcd)

Function description

USB Reset callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_SuspendCallback

Function name

void HAL_PCD_SuspendCallback (PCD_HandleTypeDef * hpcd)

Function description

Suspend event callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_ResumeCallback

Function name

void HAL_PCD_ResumeCallback (PCD_HandleTypeDef * hpcd)

Function description

Resume event callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_ConnectCallback

Function name

void HAL_PCD_ConnectCallback (PCD_HandleTypeDef * hpcd)

Function description

Connection event callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_DisconnectCallback

Function name

void HAL_PCD_DisconnectCallback (PCD_HandleTypeDef * hpcd)

Function description

Disconnection event callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_DataOutStageCallback

Function name

void HAL_PCD_DataOutStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)

Function description

Data OUT stage callback.

Parameters

- **hpcd:** PCD handle
- **epnum:** endpoint number

Return values

- **None:**

HAL_PCD_DataInStageCallback

Function name

void HAL_PCD_DataInStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)

Function description

Data IN stage callback.

Parameters

- **hpcd:** PCD handle
- **epnum:** endpoint number

Return values

- **None:**

HAL_PCD_ISOOUTIncompleteCallback

Function name

void HAL_PCD_ISOOUTIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)

Function description

Incomplete ISO OUT callback.

Parameters

- **hpcd:** PCD handle
- **epnum:** endpoint number

Return values

- **None:**

HAL_PCD_ISOINIncompleteCallback

Function name

void HAL_PCD_ISOINIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)

Function description

Incomplete ISO IN callback.

Parameters

- **hpcd:** PCD handle
- **epnum:** endpoint number

Return values

- **None:**

HAL_PCD_DevConnect

Function name

HAL_StatusTypeDef HAL_PCD_DevConnect (PCD_HandleTypeDef * hpcd)

Function description

Connect the USB device.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_DevDisconnect

Function name

HAL_StatusTypeDef HAL_PCD_DevDisconnect (PCD_HandleTypeDef * hpcd)

Function description

Disconnect the USB device.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_SetAddress

Function name

HAL_StatusTypeDef HAL_PCD_SetAddress (PCD_HandleTypeDef * hpcd, uint8_t address)

Function description

Set the USB Device address.

Parameters

- **hpcd:** PCD handle
- **address:** new device address

Return values

- **HAL:** status

HAL_PCD_EP_Open

Function name

HAL_StatusTypeDef HAL_PCD_EP_Open (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint16_t ep_mps, uint8_t ep_type)

Function description

Open and configure an endpoint.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address
- **ep_mps:** endpoint max packet size
- **ep_type:** endpoint type

Return values

- **HAL:** status

HAL_PCD_EP_Close

Function name

HAL_StatusTypeDef HAL_PCD_EP_Close (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)

Function description

Deactivate an endpoint.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address

Return values

- **HAL:** status

HAL_PCD_EP_Receive

Function name

HAL_StatusTypeDef HAL_PCD_EP_Receive (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf, uint32_t len)

Function description

Receive an amount of data.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address
- **pBuf:** pointer to the reception buffer
- **len:** amount of data to be received

Return values

- **HAL:** status

HAL_PCD_EP_Transmit

Function name

HAL_StatusTypeDef HAL_PCD_EP_Transmit (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf, uint32_t len)

Function description

Send an amount of data.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address
- **pBuf:** pointer to the transmission buffer
- **len:** amount of data to be sent

Return values

- **HAL:** status

HAL_PCD_EP_SetStall

Function name

HAL_StatusTypeDef HAL_PCD_EP_SetStall (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)

Function description

Set a STALL condition over an endpoint.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address

Return values

- **HAL:** status

HAL_PCD_EP_ClrStall

Function name

HAL_StatusTypeDef HAL_PCD_EP_ClrStall (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)

Function description

Clear a STALL condition over in an endpoint.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address

Return values

- **HAL:** status

HAL_PCD_EP_Flush

Function name

HAL_StatusTypeDef HAL_PCD_EP_Flush (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)

Function description

Flush an endpoint.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address

Return values

- **HAL:** status

HAL_PCD_EP_Abort

Function name

HAL_StatusTypeDef HAL_PCD_EP_Abort (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)

Function description

Abort an USB EP transaction.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address

Return values

- **HAL:** status

HAL_PCD_ActivateRemoteWakeup

Function name

HAL_StatusTypeDef HAL_PCD_ActivateRemoteWakeup (PCD_HandleTypeDef * hpcd)

Function description

Activate remote wakeup signalling.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_DeActivateRemoteWakeup

Function name

HAL_StatusTypeDef HAL_PCD_DeActivateRemoteWakeup (PCD_HandleTypeDef * hpcd)

Function description

De-activate remote wakeup signalling.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_SetTestMode

Function name

HAL_StatusTypeDef HAL_PCD_SetTestMode (PCD_HandleTypeDef * hpcd, uint8_t testmode)

Function description

Set the USB Device high speed test mode.

Parameters

- **hpcd:** PCD handle
- **testmode:** USB Device high speed test mode

Return values

- **HAL:** status

HAL_PCD_EP_GetRxCount

Function name

uint32_t HAL_PCD_EP_GetRxCount (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)

Function description

Get Received Data Size.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address

Return values

- **Data:** Size

HAL_PCD_GetState

Function name

PCD_StateTypeDef HAL_PCD_GetState (PCD_HandleTypeDef * hpcd)

Function description

Return the PCD handle state.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** state

61.3 PCD Firmware driver defines

The following section lists the various define and macros of the module.

61.3.1 PCD

PCD

PCD Error Code definition

HAL_PCD_ERROR_INVALID_CALLBACK

Invalid Callback error

PCD Exported Macros

__HAL_PCD_ENABLE

__HAL_PCD_DISABLE

__HAL_PCD_GET_FLAG

__HAL_PCD_CLEAR_FLAG

__HAL_PCD_IS_INVALID_INTERRUPT

__HAL_PCD_UNGATE_PHYCLOCK

__HAL_PCD_GATE_PHYCLOCK

__HAL_PCD_IS_PHY_SUSPENDED

__HAL_USB_OTG_HS_WAKEUP_EXTI_ENABLE_IT

__HAL_USB_OTG_HS_WAKEUP_EXTI_DISABLE_IT

__HAL_USB_OTG_FS_WAKEUP_EXTI_ENABLE_IT

__HAL_USB_OTG_FS_WAKEUP_EXTI_DISABLE_IT

PCD PHY Module

PCD_PHY_ULPI

PCD_PHY_EMBEDDED

PCD_PHY_UTMI

PCD Speed

PCD_SPEED_HIGH

PCD_SPEED_HIGH_IN_FULL

PCD_SPEED_FULL

62 HAL PCD Extension Driver

62.1 PCDEx Firmware driver API description

The following section lists the various functions of the PCDEx library.

62.1.1 Extended features functions

This section provides functions allowing to:

- Update FIFO configuration

This section contains the following APIs:

- *HAL_PCDEx_SetTxFiFo()*
- *HAL_PCDEx_SetRxFiFo()*
- *HAL_PCDEx_ActivateLPM()*
- *HAL_PCDEx_DeActivateLPM()*
- *HAL_PCDEx_BCD_VBUSDetect()*
- *HAL_PCDEx_ActivateBCD()*
- *HAL_PCDEx_DeActivateBCD()*
- *HAL_PCDEx_LPM_Callback()*
- *HAL_PCDEx_BCD_Callback()*

62.1.2 Detailed description of functions

HAL_PCDEx_SetTxFiFo

Function name

HAL_StatusTypeDef HAL_PCDEx_SetTxFiFo (PCD_HandleTypeDef * hpcd, uint8_t fifo, uint16_t size)

Function description

Set Tx FIFO.

Parameters

- **hpcd:** PCD handle
- **fifo:** The number of Tx fifo
- **size:** Fifo size

Return values

- **HAL:** status

HAL_PCDEx_SetRxFiFo

Function name

HAL_StatusTypeDef HAL_PCDEx_SetRxFiFo (PCD_HandleTypeDef * hpcd, uint16_t size)

Function description

Set Rx FIFO.

Parameters

- **hpcd:** PCD handle
- **size:** Size of Rx fifo

Return values

- **HAL:** status

HAL_PCDEx_ActivateLPM

Function name

HAL_StatusTypeDef HAL_PCDEx_ActivateLPM (PCD_HandleTypeDef * hpcd)

Function description

Activate LPM feature.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCDEx_DeActivateLPM

Function name

HAL_StatusTypeDef HAL_PCDEx_DeActivateLPM (PCD_HandleTypeDef * hpcd)

Function description

Deactivate LPM feature.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCDEx_ActivateBCD

Function name

HAL_StatusTypeDef HAL_PCDEx_ActivateBCD (PCD_HandleTypeDef * hpcd)

Function description

Activate BatteryCharging feature.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCDEx_DeActivateBCD

Function name

HAL_StatusTypeDef HAL_PCDEx_DeActivateBCD (PCD_HandleTypeDef * hpcd)

Function description

Deactivate BatteryCharging feature.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCDEx_BCD_VBUSDetect

Function name

void HAL_PCDEx_BCD_VBUSDetect (PCD_HandleTypeDef * hpcd)

Function description

Handle BatteryCharging Process.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCDEx_LPM_Callback

Function name

void HAL_PCDEx_LPM_Callback (PCD_HandleTypeDef * hpcd, PCD_LPM_MsgTypeDef msg)

Function description

Send LPM message to user layer callback.

Parameters

- **hpcd:** PCD handle
- **msg:** LPM message

Return values

- **HAL:** status

HAL_PCDEx_BCD_Callback

Function name

void HAL_PCDEx_BCD_Callback (PCD_HandleTypeDef * hpcd, PCD_BCD_MsgTypeDef msg)

Function description

Send BatteryCharging message to user layer callback.

Parameters

- **hpcd:** PCD handle
- **msg:** LPM message

Return values

- **HAL:** status

63 HAL PSSI Generic Driver

63.1 PSSI Firmware driver registers structures

63.1.1 PSSI_InitTypeDef

PSSI_InitTypeDef is defined in the stm32h7xx_hal_pssi.h

Data Fields

- ***uint32_t DataWidth***
- ***uint32_t BusWidth***
- ***uint32_t ControlSignal***
- ***uint32_t ClockPolarity***
- ***uint32_t DataEnablePolarity***
- ***uint32_t ReadyPolarity***

Field Documentation

- ***uint32_t PSSI_InitTypeDef::DataWidth***
- ***uint32_t PSSI_InitTypeDef::BusWidth***
- ***uint32_t PSSI_InitTypeDef::ControlSignal***
- ***uint32_t PSSI_InitTypeDef::ClockPolarity***
- ***uint32_t PSSI_InitTypeDef::DataEnablePolarity***
- ***uint32_t PSSI_InitTypeDef::ReadyPolarity***

63.1.2 __PSSI_HandleTypeDef

__PSSI_HandleTypeDef is defined in the stm32h7xx_hal_pssi.h

Data Fields

- ***PSSI_TypeDef * Instance***
- ***PSSI_InitTypeDef Init***
- ***uint32_t * pBuffPtr***
- ***uint32_t XferCount***
- ***uint32_t XferSize***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***void(* TxCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_PSSI_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***PSSI_TypeDef* __PSSI_HandleTypeDef::Instance***
  PSSI register base address.
- ***PSSI_InitTypeDef __PSSI_HandleTypeDef::Init***
  PSSI Initialization Structure.

- ***uint32_t* __PSSI_HandleTypeDef::pBuffPtr***
  PSSI Data buffer.
- ***uint32_t __PSSI_HandleTypeDef::XferCount***
  PSSI transfer count
- ***uint32_t __PSSI_HandleTypeDef::XferSize***
  PSSI transfer size
- ***DMA_HandleTypeDef* __PSSI_HandleTypeDef::hdmatx***
  PSSI Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __PSSI_HandleTypeDef::hdmarx***
  PSSI Rx DMA Handle parameters
- ***void(* __PSSI_HandleTypeDef::TxCpltCallback)(struct __PSSI_HandleTypeDef *hpssi)***
  PSSI transfer complete callback.
- ***void(* __PSSI_HandleTypeDef::RxCpltCallback)(struct __PSSI_HandleTypeDef *hpssi)***
  PSSI transfer complete callback.
- ***void(* __PSSI_HandleTypeDef::ErrorCallback)(struct __PSSI_HandleTypeDef *hpssi)***
  PSSI transfer complete callback.
- ***void(* __PSSI_HandleTypeDef::AbortCpltCallback)(struct __PSSI_HandleTypeDef *hpssi)***
  PSSI transfer error callback.
- ***void(* __PSSI_HandleTypeDef::MspInitCallback)(struct __PSSI_HandleTypeDef *hpssi)***
  PSSI Msp Init callback.
- ***void(* __PSSI_HandleTypeDef::MspDeInitCallback)(struct __PSSI_HandleTypeDef *hpssi)***
  PSSI Msp DeInit callback.
- ***HAL_LockTypeDef __PSSI_HandleTypeDef::Lock***
  PSSI lock.
- ***__IO HAL_PSSI_StateTypeDef __PSSI_HandleTypeDef::State***
  PSSI transfer state.
- ***__IO uint32_t __PSSI_HandleTypeDef::ErrorCode***
  PSSI error code.

63.2 PSSI Firmware driver API description

The following section lists the various functions of the PSSI library.

63.2.1 How to use this driver

The PSSI HAL driver can be used as follows:

1. Declare a PSSI_HandleTypeDef handle structure, for example: PSSI_HandleTypeDef hpssi;

2. Initialize the PSSI low level resources by implementing the @ref HAL_PSSI_MspInit() API:
   a. Enable the PSSIx interface clock
   b. PSSI pins configuration
      - Enable the clock for the PSSI GPIOs
      - Configure PSSI pins as alternate function open-drain
   c. NVIC configuration if you need to use interrupt process
      - Configure the PSSIx interrupt priority
      - Enable the NVIC PSSI IRQ Channel
   d. DMA Configuration if you need to use DMA process
      - Declare DMA_HandleTypeDef handles structure for the transmit and receive
      - Enable the DMAx interface clock
      - Configure the DMA handle parameters
      - Configure the DMA Tx and Rx
      - Associate the initialized DMA handle to the hpssi DMA Tx and Rx handle
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx and Rx
3. Configure the Communication Bus Width, Control Signals, Input Polarity and Output Polarity in the hpssi Init structure.
4. Initialize the PSSI registers by calling the @ref HAL_PSSI_Init(), configure also the low level Hardware (GPIO, CLOCK, NVIC...etc) by calling the customized @ref HAL_PSSI_MspInit(&hpssi) API.
5. For PSSI IO operations, two operation modes are available within this driver :

Polling mode IO operation

- Transmit an amount of data by byte in blocking mode using @ref HAL_PSSI_Transmit()
- Receive an amount of data by byte in blocking mode using @ref HAL_PSSI_Receive()

DMA mode IO operation

- Transmit an amount of data in non-blocking mode (DMA) using @ref HAL_PSSI_Transmit_DMA()
- At transmission end of transfer, @ref HAL_PSSI_TxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_PSSI_TxCpltCallback()
- Receive an amount of data in non-blocking mode (DMA) using @ref HAL_PSSI_Receive_DMA()
- At reception end of transfer, @ref HAL_PSSI_RxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_PSSI_RxCpltCallback()
- In case of transfer Error, @ref HAL_PSSI_ErrorCallback() function is executed and user can add his own code by customization of function pointer @ref HAL_PSSI_ErrorCallback()
- Abort a PSSI process communication with Interrupt using @ref HAL_PSSI_Abort_IT()
- End of abort process, @ref HAL_PSSI_AbortCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_PSSI_AbortCpltCallback()

PSSI HAL driver macros list

Below the list of most used macros in PSSI HAL driver.

- @ref HAL_PSSI_ENABLE : Enable the PSSI peripheral
- @ref HAL_PSSI_DISABLE : Disable the PSSI peripheral
- @ref HAL_PSSI_GET_FLAG : Check whether the specified PSSI flag is set or not
- @ref HAL_PSSI_CLEAR_FLAG : Clear the specified PSSI pending flag
- @ref HAL_PSSI_ENABLE_IT : Enable the specified PSSI interrupt
- @ref HAL_PSSI_DISABLE_IT : Disable the specified PSSI interrupt

Callback registration

Note: You can refer to the PSSI HAL driver header file for more useful macros

63.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and deinitialize the PSSIx peripheral:

- User must implement HAL_PSSI_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_PSSI_Init() to configure the selected device with the selected configuration:
  - Data Width
  - Control Signals
  - Input Clock polarity
  - Output Clock polarity
- Call the function HAL_PSSI_DeInit() to restore the default configuration of the selected PSSIx peripheral.

This section contains the following APIs:

- ***HAL_PSSI_Init()***
- ***HAL_PSSI_DeInit()***
- ***HAL_PSSI_MspInit()***
- ***HAL_PSSI_MspDeInit()***
- ***HAL_PSSI_RegisterCallback()***
- ***HAL_PSSI_UnRegisterCallback()***

63.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the PSSI data transfers.

1. There are two modes of transfer:
   - Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
   - No-Blocking mode : The communication is performed using DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated the DMA IRQ .
2. Blocking mode functions are :
   - HAL_PSSI_Transmit()
   - HAL_PSSI_Receive()
3. No-Blocking mode functions with DMA are :
   - HAL_PSSI_Transmit_DMA()
   - HAL_PSSI_Receive_DMA()
4. A set of Transfer Complete Callbacks are provided in non Blocking mode:
   - HAL_PSSI_TxCpltCallback()
   - HAL_PSSI_RxCpltCallback()
   - HAL_PSSI_ErrorCallback()
   - HAL_PSSI_AbortCpltCallback()

This section contains the following APIs:

- ***HAL_PSSI_Transmit()***
- ***HAL_PSSI_Receive()***
- ***HAL_PSSI_Transmit_DMA()***
- ***HAL_PSSI_Receive_DMA()***
- ***HAL_PSSI_Abort_DMA()***

63.2.4 Peripheral State, Mode and Error functions

This subsection permit to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- ***HAL_PSSI_GetState()***
- ***HAL_PSSI_GetError()***

63.2.5 Detailed description of functions

HAL_PSSI_Init

Function name

HAL_StatusTypeDef HAL_PSSI_Init (PSSI_HandleTypeDef * hpssi)

Function description

Initializes the PSSI according to the specified parameters in the PSSI_InitTypeDef and initialize the associated handle.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **HAL:** status

HAL_PSSI_DeInit

Function name

HAL_StatusTypeDef HAL_PSSI_DeInit (PSSI_HandleTypeDef * hpssi)

Function description

DeInitialize the PSSI peripheral.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **HAL:** status

HAL_PSSI_MspInit

Function name

void HAL_PSSI_MspInit (PSSI_HandleTypeDef * hpssi)

Function description

Initialize the PSSI MSP.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **None:**

HAL_PSSI_MspDeInit

Function name

void HAL_PSSI_MspDeInit (PSSI_HandleTypeDef * hpssi)

Function description

DeInitialize the PSSI MSP.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **None:**

HAL_PSSI_RegisterCallback

Function name

HAL_StatusTypeDef HAL_PSSI_RegisterCallback (PSSI_HandleTypeDef * hpssi, HAL_PSSI_CallbackIDTypeDef CallbackID, pPSSI_CallbackTypeDef pCallback)

Function description

Register a User PSSI Callback To be used instead of the weak predefined callback.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_PSSI_TX_COMPLETE_CB_ID Tx Transfer completed callback ID
  - HAL_PSSI_RX_COMPLETE_CB_ID Rx Transfer completed callback ID
  - HAL_PSSI_ERROR_CB_ID Error callback ID
  - HAL_PSSI_ABORT_CB_ID Abort callback ID
  - HAL_PSSI_MSPINIT_CB_ID MspInit callback ID
  - HAL_PSSI_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

Notes

- The HAL_PSSI_RegisterCallback() may be called before HAL_PSSI_Init() in HAL_PSSI_STATE_RESET to register callbacks for HAL_PSSI_MSPINIT_CB_ID and HAL_PSSI_MSPDEINIT_CB_ID.

HAL_PSSI_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_PSSI_UnRegisterCallback (PSSI_HandleTypeDef * hpssi, HAL_PSSI_CallbackIDTypeDef CallbackID)

Function description

Unregister an PSSI Callback PSSI callback is redirected to the weak predefined callback.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_PSSI_TX_COMPLETE_CB_ID Tx Transfer completed callback ID
  - HAL_PSSI_RX_COMPLETE_CB_ID Rx Transfer completed callback ID
  - HAL_PSSI_ERROR_CB_ID Error callback ID
  - HAL_PSSI_ABORT_CB_ID Abort callback ID
  - HAL_PSSI_MSPINIT_CB_ID MspInit callback ID
  - HAL_PSSI_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

Notes

- The HAL_PSSI_UnRegisterCallback() may be called before HAL_PSSI_Init() in HAL_PSSI_STATE_RESET to un-register callbacks for HAL_PSSI_MSPINIT_CB_ID and HAL_PSSI_MSPDEINIT_CB_ID.

HAL_PSSI_Transmit

Function name

HAL_StatusTypeDef HAL_PSSI_Transmit (PSSI_HandleTypeDef * hpssi, uint8_t * pData, uint32_t Size, uint32_t Timeout)

Function description

Transmits in master mode an amount of data in blocking mode.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent (in bytes)
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PSSI_Receive

Function name

HAL_StatusTypeDef HAL_PSSI_Receive (PSSI_HandleTypeDef * hpssi, uint8_t * pData, uint32_t Size, uint32_t Timeout)

Function description

Receives an amount of data in blocking mode.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received (in bytes)
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PSSI_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_PSSI_Transmit_DMA (PSSI_HandleTypeDef * hpssi, uint32_t * pData, uint32_t Size)

Function description

Transmit an amount of data in non-blocking mode with DMA.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent (in bytes)

Return values

- **HAL:** status

HAL_PSSI_Receive_DMA

Function name

HAL_StatusTypeDef HAL_PSSI_Receive_DMA (PSSI_HandleTypeDef * hpssi, uint32_t * pData, uint32_t Size)

Function description

Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received (in bytes)

Return values

- **HAL:** status

HAL_PSSI_Abort_DMA

Function name

HAL_StatusTypeDef HAL_PSSI_Abort_DMA (PSSI_HandleTypeDef * hpssi)

Function description

Abort a DMA process communication with Interrupt.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **HAL:** status

HAL_PSSI_GetState

Function name

HAL_PSSI_StateTypeDef HAL_PSSI_GetState (PSSI_HandleTypeDef * hpssi)

Function description

Return the PSSI handle state.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **HAL:** state

HAL_PSSI_GetError

Function name

uint32_t HAL_PSSI_GetError (PSSI_HandleTypeDef * hpssi)

Function description

Return the PSSI error code.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **PSSI:** Error Code

HAL_PSSI_IRQHandler

Function name

void HAL_PSSI_IRQHandler (PSSI_HandleTypeDef * hpssi)

Function description

This function handles PSSI event interrupt request.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **None:**

HAL_PSSI_TxCpltCallback

Function name

void HAL_PSSI_TxCpltCallback (PSSI_HandleTypeDef * hpssi)

Function description

Tx Transfer complete callback.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **None:**

HAL_PSSI_RxCpltCallback

Function name

void HAL_PSSI_RxCpltCallback (PSSI_HandleTypeDef * hpssi)

Function description

Rx Transfer complete callback.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **None:**

HAL_PSSI_ErrorCallback

Function name

void HAL_PSSI_ErrorCallback (PSSI_HandleTypeDef * hpssi)

Function description

PSSI error callback.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **None:**

HAL_PSSI_AbortCpltCallback

Function name

void HAL_PSSI_AbortCpltCallback (PSSI_HandleTypeDef * hpssi)

Function description

PSSI abort callback.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **None:**

63.3 PSSI Firmware driver defines

The following section lists the various define and macros of the module.

63.3.1 PSSI

PSSI

PSSI Bus Width

HAL_PSSI_8LINES

8 data lines

HAL_PSSI_16LINES

16 data lines

PSSI Data Width

HAL_PSSI_8BITS

8 Bits

HAL_PSSI_16BITS

16 Bits

HAL_PSSI_32BITS

32 Bits

PSSI definitions

PSSI_MAX_NBYTE_SIZE

PSSI_TIMEOUT_TRANSMIT

Timeout Value

PSSI_CR_OUTEN_INPUT

Input Mode

PSSI_CR_OUTEN_OUTPUT

Output Mode

PSSI_CR_DMA_ENABLE

DMA Mode Enable

PSSI_CR_DMA_DISABLE

DMA Mode Disable

PSSI_CR_16BITS

16 Lines Mode

PSSI_CR_8BITS

8 Lines Mode

PSSI_FLAG_RTT1B

1 Byte Fifo Flag

PSSI_FLAG_RTT4B

4 Bytes Fifo Flag

PSSI Error Code

HAL_PSSI_ERROR_NONE

No error

HAL_PSSI_ERROR_NOT_SUPPORTED

Not supported operation

HAL_PSSI_ERROR_UNDER_RUN

FIFO Under-run error

HAL_PSSI_ERROR_OVER_RUN

FIFO Over-run error

HAL_PSSI_ERROR_DMA

Dma error

HAL_PSSI_ERROR_TIMEOUT

Timeout error

HAL_PSSI_ERROR_INVALID_CALLBACK

Invalid callback error

PSSI Exported Macros

HAL_PSSI_RESET_HANDLE_STATE

Description:

- Reset PSSI handle state.

Parameters:

- __HANDLE__: specifies the PSSI handle.

Return value:

- None

HAL_PSSI_ENABLE

Description:

- Enable the PSSI.

Parameters:

- __HANDLE__: PSSI handle

Return value:

- None.

HAL_PSSI_DISABLE

Description:

- Disable the PSSI.

Parameters:

- __HANDLE__: PSSI handle

Return value:

- None.

HAL_PSSI_GET_STATUS

Description:

- Get the PSSI pending flags.

Parameters:

- __HANDLE__: PSSI handle
- __FLAG__: flag to check. This parameter can be any combination of the following values:
  - PSSI_FLAG_RTT1B: FIFO is ready to transfer one byte
  - PSSI_FLAG_RTT4B: FIFO is ready to transfer four bytes

Return value:

- The: state of FLAG.

HAL_PSSI_GET_FLAG

Description:

- Get the PSSI pending flags.

Parameters:

- __HANDLE__: PSSI handle
- __FLAG__: flag to check. This parameter can be any combination of the following values:
  - PSSI_FLAG_OVR_RIS: Data Buffer overrun/underrun error flag

Return value:

- The: state of FLAG.

HAL_PSSI_CLEAR_FLAG

Description:

- Clear the PSSI pending flags.

Parameters:

- __HANDLE__: PSSI handle
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - PSSI_FLAG_OVR_RIS: Data Buffer overrun/underrun error flag

Return value:

- None

HAL_PSSI_ENABLE_IT

Description:

- Enable the specified PSSI interrupts.

Parameters:

- __HANDLE__: PSSI handle
- __INTERRUPT__: specifies the PSSI interrupt sources to be enabled. This parameter can be any combination of the following values:
  - PSSI_FLAG_OVR_RIS: Configuration error mask

Return value:

- None

HAL_PSSI_DISABLE_IT

Description:

- Disable the specified PSSI interrupts.

Parameters:

- __HANDLE__: PSSI handle
- __INTERRUPT__: specifies the PSSI interrupt sources to be disabled. This parameter can be any combination of the following values:
  - PSSI_IT_OVR_IE: Configuration error mask

Return value:

- None

HAL_PSSI_GET_IT_SOURCE

Description:

- Check whether the specified PSSI interrupt source is enabled or not.

Parameters:

- __HANDLE__: PSSI handle
- __INTERRUPT__: specifies the PSSI interrupt source to check. This parameter can be one of the following values:
  - PSSI_IT_OVR_IE: Data Buffer overrun/underrun error interrupt mask

Return value:

- The: state of INTERRUPT source.

IS_PSSI_CONTROL_SIGNAL

Description:

- Check whether the PSSI Control signal is valid.

Parameters:

- __CONTROL__: Control signals configuration

Return value:

- Valid: or not.

IS_PSSI_BUSWIDTH

Description:

- Check whether the PSSI Bus Width is valid.

Parameters:

- __BUSWIDTH__: PSSI Bush width

Return value:

- Valid: or not.

IS_PSSI_CLOCK_POLARITY

Description:

- Check whether the PSSI Clock Polarity is valid.

Parameters:

- __CLOCKPOL__: PSSI Clock Polarity

Return value:

- Valid: or not.

IS_PSSI_DE_POLARITY

Description:

- Check whether the PSSI Data Enable Polarity is valid.

Parameters:

- __DEPOL__: PSSI DE Polarity

Return value:

- Valid: or not.

IS_PSSI_RDY_POLARITY

Description:

- Check whether the PSSI Ready Polarity is valid.

Parameters:

- __RDYPOL__: PSSI RDY Polarity

Return value:

- Valid: or not.

PSSI Interrupts

PSSI_FLAG_OVR_RIS

Overrun, Underrun errors flag

PSSI_FLAG_MASK

Overrun, Underrun errors Mask

PSSI_FLAG_OVR_MIS

Overrun, Underrun masked errors flag

PSSI mode

HAL_PSSI_UNIDIRECTIONAL

Uni-directional mode

HAL_PSSI_BIDIRECTIONAL

Bi-directional mode

64 HAL PWR Generic Driver

64.1 PWR Firmware driver registers structures

64.1.1 PWR_PVDTypeDef

PWR_PVDTypeDef is defined in the stm32h7xx_hal_pwr.h

Data Fields

- ***uint32_t PVDLevel***
- ***uint32_t Mode***

Field Documentation

- ***uint32_t PWR_PVDTypeDef::PVDLevel***

  PVDLevel: Specifies the PVD detection level. This parameter can be a value of ***PWR_PVD_detection_level***.
- ***uint32_t PWR_PVDTypeDef::Mode***

  Mode: Specifies the EXTI operating mode for the PVD event. This parameter can be a value of ***PWR_PVD_Mode***.

64.2 PWR Firmware driver API description

The following section lists the various functions of the PWR library.

64.2.1 PWR peripheral overview

1. The Power control (PWR) provides an overview of the supply architecture for the different power domains and of the supply configuration controller. In the H7 family, the number of power domains is different between device lines. This difference is due to characteristics of each device.
2. Domain architecture overview for the different H7 lines: (+) Dual core lines are STM32H745, STM32H747, STM32H755 and STM32H757. These devices have 3 power domains (D1, D2 and D3). The domain D1 contains a CPU (Cortex-M7), a Flash memory and some peripherals. The D2 domain contains peripherals and a CPU (Cortex-M4). The D3 domain contains the system control, I/O logic and low-power peripherals. (+) STM32H72x, STM32H73x, STM32H742, STM32H743, STM32H750 and STM32H753 devices have 3 power domains (D1, D2 and D3). The domain D1 contains a CPU (Cortex-M7), a Flash memory and some peripherals. The D2 domain contains peripherals. The D3 domains contains the system control, I/O logic and low-power peripherals. (+) STM32H7Axxx and STM32H7Bxxx devices have 2 power domains (CD and SRD). The core domain (CD) contains a CPU (Cortex-M7), a Flash memory and peripherals. The SmartRun domain contains the system control, I/O logic and low-power peripherals.
3. Every entity have low power mode as described below :
4. The CPU low power modes are : (+) CPU CRUN. (+) CPU CSLEEP. (+) CPU CSTOP.
5. The domain low power modes are : (+) DRUN. (+) DSTOP. (+) DSTANDBY.
6. The SYSTEM low power modes are : (+) RUN* : The Run* mode is entered after a POR reset and a wakeup from Standby. In Run* mode, the performance is limited and the system supply configuration shall be programmed. The system enters Run mode only when the ACTVOSRDY bit in PWR control status register 1 (PWR_CSR1) is set to 1. (+) RUN. (+) STOP. (+) STANDBY.

(#) The Power control (PWR) provides an overview of the supply architecture for the different power domains and of the supply configuration controller. In the H7 family, the number of power domains is different between device lines. This difference is due to characteristics of each device. (#) Domain architecture overview for the different H7 lines:

- Dual core lines are STM32H745, STM32H747, STM32H755 and STM32H757. These devices have 3 power domains (D1, D2 and D3). The domain D1 contains a CPU (Cortex-M7), a Flash memory and some peripherals. The D2 domain contains peripherals and a CPU (Cortex-M4). The D3 domain contains the system control, I/O logic and low-power peripherals.

- STM32H72x, STM32H73x, STM32H742, STM32H743, STM32H750 and STM32H753 devices have 3 power domains (D1, D2 and D3). The domain D1 contains a CPU (Cortex-M7), a Flash memory and some peripherals. The D2 domain contains peripherals. The D3 domains contains the system control, I/O logic and low-power peripherals.
- STM32H7Axxx and STM32H7Bxxx devices have 2 power domains (CD and SRD). The core domain (CD) contains a CPU (Cortex-M7), a Flash memory and peripherals. The SmartRun domain contains the system control, I/O logic and low-power peripherals. (#) Every entity have low power mode as described below : (#) The CPU low power modes are :
- CPU CRUN.
- CPU CSLEEP.
- CPU CSTOP. (#) The domain low power modes are :
- DRUN.
- DSTOP.
- DSTANDBY. (#) The SYSTEM low power modes are :
- RUN* : The Run* mode is entered after a POR reset and a wakeup from Standby. In Run* mode, the performance is limited and the system supply configuration shall be programmed. The system enters Run mode only when the ACTVOSRDY bit in PWR control status register 1 (PWR_CSR1) is set to 1.
- RUN.
- STOP.
- STANDBY.

64.2.2 How to use this driver

1. Power management peripheral is active by default at startup level in STM32h7xx lines.
2. Call HAL_PWR_EnableBkUpAccess() and HAL_PWR_DisableBkUpAccess() functions to enable/disable access to the backup domain (RTC registers, RTC backup data registers and backup SRAM).
3. Call HAL_PWR_ConfigPVD() after setting parameters to be configured (event mode and voltage threshold) in order to set up the Power Voltage Detector, then use HAL_PWR_EnablePVD() and HAL_PWR_DisablePVD() functions to start and stop the PVD detection. (+) PVD level could be one of the following values :
   – 1V95
   – 2V1
   – 2V25
   – 2V4
   – 2V55
   – 2V7
   – 2V85
   – External voltage level
4. Call HAL_PWR_EnableWakeUpPin() and HAL_PWR_DisableWakeUpPin() functions with the right parameter to configure the wake up pin polarity (Low or High) and to enable and disable it.
5. Call HAL_PWR_EnterSLEEPMode() function to enter the current Core in SLEEP mode. Wake-up from SLEEP mode could be following to an event or an interrupt according to low power mode intrinsic request called (__WFI() or __WFE()). Please ensure to clear all CPU pending events by calling HAL_PWREx_ClearPendingEvent() function when trying to enter the Cortex-Mx in SLEEP mode with __WFE() entry.
6. Call HAL_PWR_EnterSTOPMode() function to enter the whole system to Stop 0 mode for single core devices. For dual core devices, this API will enter the domain (containing Cortex-Mx that executing this function) in DSTOP mode. According to the used parameter, user could select the regulator to be kept actif in low power mode and wake-up event type. Please ensure to clear all CPU pending events by calling HAL_PWREx_ClearPendingEvent() function when trying to enter the Cortex-Mx in CSTOP mode with __WFE() entry.
7. Call HAL_PWR_EnterSTANDBYMode() function to enter the whole system in STANDBY mode for single core devices. For dual core devices, this API will enter the domain (containing Cortex-Mx that executing this function) in DSTANDBY mode.

8. Call HAL_PWR_EnableSleepOnExit() and HAL_PWR_DisableSleepOnExit() APIs to enable and disable the Cortex-Mx re-entring in SLEEP mode after an interruption handling is over.
9. Call HAL_PWR_EnableSEVOnPend() and HAL_PWR_DisableSEVOnPend() functions to configure the Cortex-Mx to wake-up after any pending event / interrupt even if it's disabled or has insufficient priority to cause exception entry.
10. Call HAL_PWR_PVD_IRQHandler() function to handle the PWR PVD interrupt request.

(#) Power management peripheral is active by default at startup level in STM32h7xx lines. (#) Call HAL_PWR_EnableBkUpAccess() and HAL_PWR_DisableBkUpAccess() functions to enable/disable access to the backup domain (RTC registers, RTC backup data registers and backup SRAM). (#) Call HAL_PWR_ConfigPVD() after setting parameters to be configured (event mode and voltage threshold) in order to set up the Power Voltage Detector, then use HAL_PWR_EnablePVD() and HAL_PWR_DisablePVD() functions to start and stop the PVD detection.

- PVD level could be one of the following values :
  - 1V95
  - 2V1
  - 2V25
  - 2V4
  - 2V55
  - 2V7
  - 2V85
  - External voltage level (#) Call HAL_PWR_EnableWakeUpPin() and HAL_PWR_DisableWakeUpPin() functions with the right parameter to configure the wake up pin polarity (Low or High) and to enable and disable it. (#) Call HAL_PWR_EnterSLEEPMode() function to enter the current Core in SLEEP mode. Wake-up from SLEEP mode could be following to an event or an interrupt according to low power mode intrinsic request called (__WFI() or __WFE()). Please ensure to clear all CPU pending events by calling HAL_PWREx_ClearPendingEvent() function when trying to enter the Cortex-Mx in SLEEP mode with __WFE() entry. (#) Call HAL_PWR_EnterSTOPMode() function to enter the whole system to Stop 0 mode for single core devices. For dual core devices, this API will enter the domain (containing Cortex-Mx that executing this function) in DSTOP mode. According to the used parameter, user could select the regulator to be kept actif in low power mode and wake-up event type. Please ensure to clear all CPU pending events by calling HAL_PWREx_ClearPendingEvent() function when trying to enter the Cortex-Mx in CSTOP mode with __WFE() entry. (#) Call HAL_PWR_EnterSTANDBYMode() function to enter the whole system in STANDBY mode for single core devices. For dual core devices, this API will enter the domain (containing Cortex-Mx that executing this function) in DSTANDBY mode. (#) Call HAL_PWR_EnableSleepOnExit() and HAL_PWR_DisableSleepOnExit() APIs to enable and disable the Cortex-Mx re-entring in SLEEP mode after an interruption handling is over. (#) Call HAL_PWR_EnableSEVOnPend() and HAL_PWR_DisableSEVOnPend() functions to configure the Cortex-Mx to wake-up after any pending event / interrupt even if it's disabled or has insufficient priority to cause exception entry. (#) Call HAL_PWR_PVD_IRQHandler() function to handle the PWR PVD interrupt request.

PWR HAL driver macros list

Below the list of most used macros in PWR HAL driver.

- __HAL_PWR_VOLTAGESCALING_CONFIG() : Configure the main internal regulator output voltage.
- __HAL_PWR_GET_FLAG() : Get the PWR pending flags.
- __HAL_PWR_CLEAR_FLAG() : Clear the PWR pending flags.

64.2.3 Initialization and De-Initialization Functions

This section provides functions allowing to deinitialize power peripheral.

After system reset, the backup domain (RTC registers, RTC backup data registers and backup SRAM) is protected against possible unwanted write accesses. The HAL_PWR_EnableBkUpAccess() function enables the access to the backup domain. The HAL_PWR_DisableBkUpAccess() function disables the access to the backup domain.

This section contains the following APIs:

- ***HAL_PWR_DeInit()***

- *HAL_PWR_EnableBkUpAccess()*
- *HAL_PWR_DisableBkUpAccess()*

64.2.4 Peripheral Control Functions

This section provides functions allowing to control power peripheral.

PVD configuration

- The PVD is used to monitor the VDD power supply by comparing it to a threshold selected by the PVD Level (PLS[7:0] bits in the PWR_CR1 register).
- A PVDO flag is available to indicate if VDD is higher or lower than the PVD threshold. This event is internally connected to the EXTI line 16 to generate an interrupt if enabled. It is configurable through __HAL_PWR_PVD_EXTI_ENABLE_IT() macro.
- The PVD is stopped in STANDBY mode.

Wake-up pin configuration

- Wake-up pin is used to wake up the system from STANDBY mode. The pin pull is configurable through the WKUPEPR register to be in No-pull, Pull-up and Pull-down. The pin polarity is configurable through the WKUPEPR register to be active on rising or falling edges.
- There are up to six Wake-up pin in the STM32H7 devices family.

Low Power modes configuration

The device present 3 principles low-power modes features:

- SLEEP mode : Cortex-Mx is stopped and all PWR domains are remaining active (Powered and Clocked).
- STOP mode : Cortex-Mx is stopped, clocks are stopped and the regulator is running. The Main regulator or the LP regulator could be selected.
- STANDBY mode : All PWR domains enter DSTANDBY mode and the VCORE supply regulator is powered off.

SLEEP mode

- Entry: The SLEEP mode is entered by using the HAL_PWR_EnterSLEEPMode(Regulator, SLEEPEntry) function.
  - PWR_SLEEPENTRY_WFI: enter SLEEP mode with WFI instruction.
  - PWR_SLEEPENTRY_WFE: enter SLEEP mode with WFE instruction.

Note: The Regulator parameter is not used for the STM32H7 family and is kept as parameter just to maintain compatibility with the lower power families (STM32L).

- Exit: Any peripheral interrupt acknowledged by the nested vectored interrupt controller (NVIC) can wake up the device from SLEEP mode.

STOP mode

In system STOP mode, all clocks in the 1.2V domain are stopped, the PLL, the HSI, and the HSE RC oscillators are disabled. Internal SRAM and register contents are preserved. The voltage regulator can be configured either in normal or low-power mode. To minimize the consumption in STOP mode, FLASH can be powered off before entering the STOP mode using the HAL_PWREx_EnableFlashPowerDown() function. It can be switched on again by software after exiting the STOP mode using the HAL_PWREx_DisableFlashPowerDown() function.

- Entry: The STOP mode is entered using the HAL_PWR_EnterSTOPMode(Regulator, STOPEntry) function with:
  - Regulator:
    - PWR_MAINREGULATOR_ON: Main regulator ON.
    - PWR_LOWPOWERREGULATOR_ON: Low Power regulator ON.
  - STOPEntry:
    - PWR_STOPENTRY_WFI: enter STOP mode with WFI instruction.
    - PWR_STOPENTRY_WFE: enter STOP mode with WFE instruction.

- Exit: Any EXTI Line (Internal or External) configured in Interrupt/Event mode.

STANDBY mode

- The system STANDBY mode allows to achieve the lowest power consumption. It is based on the Cortex-Mx deep SLEEP mode, with the voltage regulator disabled. The system is consequently powered off. The PLL, the HSI oscillator and the HSE oscillator are also switched off. SRAM and register contents are lost except for the RTC registers, RTC backup registers, backup SRAM and standby circuitry.

The voltage regulator is OFF. (++) Entry: (+++) The STANDBY mode is entered using the HAL_PWR_EnterSTANDBYMode() function. (++) Exit: (+++) WKUP pin rising or falling edge, RTC alarm (Alarm A and Alarm B), RTC wakeup, tamper event, time stamp event, external reset in NRST pin, IWDG reset.

Auto-wakeup (AWU) from low-power mode

- The MCU can be woken up from low-power mode by an RTC Alarm event, an RTC Wakeup event, a tamper event or a time-stamp event, without depending on an external interrupt (Auto-wakeup mode).
- RTC auto-wakeup (AWU) from the STOP and STANDBY modes
  - To wake up from the STOP mode with an RTC alarm event, it is necessary to configure the RTC to generate the RTC alarm using the HAL_RTC_SetAlarm_IT() function.
  - To wake up from the STOP mode with an RTC Tamper or time stamp event, it is necessary to configure the RTC to detect the tamper or time stamp event using the HAL_RTCEx_SetTimeStamp_IT() or HAL_RTCEx_SetTamper_IT() functions.
  - To wake up from the STOP mode with an RTC WakeUp event, it is necessary to configure the RTC to generate the RTC WakeUp event using the HAL_RTCEx_SetWakeUpTimer_IT() function.

This section contains the following APIs:

- ***HAL_PWR_ConfigPVD()***
- ***HAL_PWR_EnablePVD()***
- ***HAL_PWR_DisablePVD()***
- ***HAL_PWR_EnableWakeUpPin()***
- ***HAL_PWR_DisableWakeUpPin()***
- ***HAL_PWR_EnterSLEEPMode()***
- ***HAL_PWR_EnterSTOPMode()***
- ***HAL_PWR_EnterSTANDBYMode()***
- ***HAL_PWR_EnableSleepOnExit()***
- ***HAL_PWR_DisableSleepOnExit()***
- ***HAL_PWR_EnableSEVOnPend()***
- ***HAL_PWR_DisableSEVOnPend()***
- ***HAL_PWR_PVD_IRQHandler()***
- ***HAL_PWR_PVDCallback()***

64.2.5 Interrupt Handling Functions

This section provides functions allowing to handle the PVD pending interrupts.

This section contains the following APIs:

- ***HAL_PWR_PVD_IRQHandler()***
- ***HAL_PWR_PVDCallback()***

64.2.6 Detailed description of functions

HAL_PWR_DeInit

Function name

void HAL_PWR_DeInit (void)

Function description

Deinitialize the HAL PWR peripheral registers to their default reset values.

Return values

- **None.:**

Notes

- This functionality is not available in this product. The prototype is kept just to maintain compatibility with other products.

HAL_PWR_EnableBkUpAccess

Function name

void HAL_PWR_EnableBkUpAccess (void)

Function description

Enable access to the backup domain (RTC registers, RTC backup data registers and backup SRAM).

Return values

- **None.:**

Notes

- If the HSE divided by 2, 3, ..31 is used as the RTC clock, the Backup Domain Access should be kept enabled.

HAL_PWR_DisableBkUpAccess

Function name

void HAL_PWR_DisableBkUpAccess (void)

Function description

Disable access to the backup domain (RTC registers, RTC backup data registers and backup SRAM).

Return values

- **None.:**

Notes

- If the HSE divided by 2, 3, ..31 is used as the RTC clock, the Backup Domain Access should be kept enabled.

HAL_PWR_ConfigPVD

Function name

void HAL_PWR_ConfigPVD (PWR_PVDTypeDef * sConfigPVD)

Function description

Configure the event mode and the voltage threshold detected by the Programmable Voltage Detector(PVD).

Parameters

- **sConfigPVD:** : Pointer to an PWR_PVDTypeDef structure that contains the configuration information for the PVD.

Return values

- **None.:**

Notes

- Refer to the electrical characteristics of your device datasheet for more details about the voltage threshold corresponding to each detection level.
- For dual core devices, please ensure to configure the EXTI lines for the different Cortex-Mx through PWR_Exported_Macro provided by this driver. All combination are allowed: wake up only Cortex-M7, wake up only Cortex-M4 or wake up Cortex-M7 and Cortex-M4.

HAL_PWR_EnablePVD

Function name

void HAL_PWR_EnablePVD (void)

Function description

Enable the Programmable Voltage Detector (PVD).

Return values

- **None.:**

HAL_PWR_DisablePVD

Function name

void HAL_PWR_DisablePVD (void)

Function description

Disable the Programmable Voltage Detector (PVD).

Return values

- **None.:**

HAL_PWR_EnableWakeUpPin

Function name

void HAL_PWR_EnableWakeUpPin (uint32_t WakeUpPinPolarity)

Function description

Enable the WakeUp PINx functionality.

Parameters

- **WakeUpPinPolarity:** : Specifies which Wake-Up pin to enable. This parameter can be one of the following legacy values, which sets the default (rising edge):
  - PWR_WAKEUP_PIN1, PWR_WAKEUP_PIN2, PWR_WAKEUP_PIN3, PWR_WAKEUP_PIN4, PWR_WAKEUP_PIN5, PWR_WAKEUP_PIN6. or one of the following values where the user can explicitly states the enabled pin and the chosen polarity:
  - PWR_WAKEUP_PIN1_HIGH, PWR_WAKEUP_PIN1_LOW, PWR_WAKEUP_PIN2_HIGH, PWR_WAKEUP_PIN2_LOW, PWR_WAKEUP_PIN3_HIGH, PWR_WAKEUP_PIN3_LOW, PWR_WAKEUP_PIN4_HIGH, PWR_WAKEUP_PIN4_LOW, PWR_WAKEUP_PIN5_HIGH, PWR_WAKEUP_PIN5_LOW, PWR_WAKEUP_PIN6_HIGH, PWR_WAKEUP_PIN6_LOW.

Return values

- **None.:**

Notes

- PWR_WAKEUP_PINx and PWR_WAKEUP_PINx_HIGH are equivalent.
- The PWR_WAKEUP_PIN3_HIGH, PWR_WAKEUP_PIN3_LOW, PWR_WAKEUP_PIN5_HIGH and PWR_WAKEUP_PIN5_LOW are available only for devices that includes GPIOI port.

HAL_PWR_DisableWakeUpPin

Function name

void HAL_PWR_DisableWakeUpPin (uint32_t WakeUpPinx)

Function description

Disable the WakeUp PINx functionality.

Parameters

- **WakeUpPinx:** : Specifies the Power Wake-Up pin to disable. This parameter can be one of the following values:
  - PWR_WAKEUP_PIN1, PWR_WAKEUP_PIN2, PWR_WAKEUP_PIN3, PWR_WAKEUP_PIN4, PWR_WAKEUP_PIN5, PWR_WAKEUP_PIN6, PWR_WAKEUP_PIN1_HIGH, PWR_WAKEUP_PIN1_LOW, PWR_WAKEUP_PIN2_HIGH, PWR_WAKEUP_PIN2_LOW, PWR_WAKEUP_PIN3_HIGH, PWR_WAKEUP_PIN3_LOW, PWR_WAKEUP_PIN4_HIGH, PWR_WAKEUP_PIN4_LOW, PWR_WAKEUP_PIN5_HIGH, PWR_WAKEUP_PIN5_LOW, PWR_WAKEUP_PIN6_HIGH, PWR_WAKEUP_PIN6_LOW.

Return values

- **None.:**

Notes

- The PWR_WAKEUP_PIN3_HIGH, PWR_WAKEUP_PIN3_LOW, PWR_WAKEUP_PIN5_HIGH and PWR_WAKEUP_PIN5_LOW are available only for devices that includes GPIOI port.

HAL_PWR_EnterSTOPMode

Function name

void HAL_PWR_EnterSTOPMode (uint32_t Regulator, uint8_t STOPEntry)

Function description

Enter STOP mode.

Parameters

- **Regulator:** : Specifies the regulator state in STOP mode. This parameter can be one of the following values:
  - PWR_MAINREGULATOR_ON : STOP mode with regulator ON.
  - PWR_LOWPOWERREGULATOR_ON : STOP mode with low power regulator ON.
- **STOPEntry:** : Specifies if STOP mode in entered with WFI or WFE intrinsic instruction. This parameter can be one of the following values:
  - PWR_STOPENTRY_WFI : Enter STOP mode with WFI instruction.
  - PWR_STOPENTRY_WFE : Enter STOP mode with WFE instruction.

Return values

- **None.:**

Notes

- For single core devices, this API will enter the system in STOP mode with all domains in DSTOP, if RUN_D3/RUN_SRD bit in CPUCR register is cleared. For dual core devices, this API will enter the domain (containing Cortex-Mx that executing this function) in DSTOP mode. If all Cortex-Mx domains are in DSTOP and RUN_D3 bit in CPUCR register is cleared, all the system will enter in STOP mode.
- In System STOP mode, all I/O pins keep the same state as in Run mode.
- When exiting System STOP mode by issuing an interrupt or a wakeup event, the HSI RC oscillator is selected as default system wakeup clock.
- In System STOP mode, when the voltage regulator operates in low power mode, an additional startup delay is incurred when the system is waking up. By keeping the internal regulator ON during STOP mode, the consumption is higher although the startup time is reduced.

HAL_PWR_EnterSLEEPMode

Function name

void HAL_PWR_EnterSLEEPMode (uint32_t Regulator, uint8_t SLEEPEntry)

Function description

Enter the current core in SLEEP mode (CSLEEP).

Parameters

- **Regulator:** : Specifies the regulator state in SLEEP mode. This parameter can be one of the following values:
  - PWR_MAINREGULATOR_ON : SLEEP mode with regulator ON.
  - PWR_LOWPOWERREGULATOR_ON : SLEEP mode with low power regulator ON.
- **SLEEPEntry:** : Specifies if SLEEP mode is entered with WFI or WFE intrinsic instruction. This parameter can be one of the following values:
  - PWR_SLEEPENTRY_WFI : enter SLEEP mode with WFI instruction.
  - PWR_SLEEPENTRY_WFE : enter SLEEP mode with WFE instruction.

Return values

- **None.:**

Notes

- This parameter is not used for the STM32H7 family and is kept as parameter just to maintain compatibility with the lower power families.
- Ensure to clear pending events before calling this API through HAL_PWREx_ClearPendingEvent() when the SLEEP entry is WFE.

HAL_PWR_EnterSTANDBYMode

Function name

void HAL_PWR_EnterSTANDBYMode (void)

Function description

Enter STANDBY mode.

Return values

- **None.:**

Notes

- For single core devices, this API will enter the system in STANDBY mode with all domains in DSTANDBY, if RUN_D3/RUN_SRD bit in CPUCR register is cleared. For dual core devices, this API will enter the domain (containing Cortex-Mx that executing this function) in DSTANDBY mode. If all Cortex-Mx domains are in DSTANDBY and RUN_D3 bit in CPUCR register is cleared, all the system will enter in STANDBY mode.
- The system enters Standby mode only when all domains are in DSTANDBY.
- When the System exit STANDBY mode by issuing an interrupt or a wakeup event, the HSI RC oscillator is selected as system clock.
- It is recommended to disable all regulators before entring STANDBY mode for power consumption saving purpose.

HAL_PWR_PVD_IRQHandler

Function name

void HAL_PWR_PVD_IRQHandler (void)

Function description

This function handles the PWR PVD interrupt request.

Return values

- **None.:**

Notes

- This API should be called under the PVD_AVD_IRQHandler().

HAL_PWR_PVDCallback

Function name

void HAL_PWR_PVDCallback (void)

Function description

PWR PVD interrupt callback.

Return values

- **None.:**

HAL_PWR_EnableSleepOnExit

Function name

void HAL_PWR_EnableSleepOnExit (void)

Function description

Indicate Sleep-On-Exit feature when returning from Handler mode to Thread mode.

Return values

- **None.:**

Notes

- Set SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters SLEEP mode when an interruption handling is over. Setting this bit is useful when the processor is expected to run only on interruptions handling.

HAL_PWR_DisableSleepOnExit

Function name

void HAL_PWR_DisableSleepOnExit (void)

Function description

Disable Sleep-On-Exit feature when returning from Handler mode to Thread mode.

Return values

- **None:**

Notes

- Clears SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters SLEEP mode when an interruption handling is over.

HAL_PWR_EnableSEVOnPend

Function name

void HAL_PWR_EnableSEVOnPend (void)

Function description

Enable CORTEX SEVONPEND feature.

Return values

- **None.:**

Notes

- Sets SEVONPEND bit of SCR register. When this bit is set, any pending event / interrupt even if it's disabled or has insufficient priority to cause exception entry wakes up the Cortex-Mx.

HAL_PWR_DisableSEVOnPend

Function name

void HAL_PWR_DisableSEVOnPend (void)

Function description

Disable CORTEX SEVONPEND feature.

Return values

- **None.:**

Notes

- Resets SEVONPEND bit of SCR register. When this bit is reset, only enabled pending causes exception entry wakes up the Cortex-Mx.

64.3 PWR Firmware driver defines

The following section lists the various define and macros of the module.

64.3.1 PWR

PWR

PWR Enable WUP Mask

PWR_EWUP_MASK

PWR Exported Macro

__HAL_PWR_VOLTAGESCALING_CONFIG

Description:

- Configure the main internal regulator output voltage.

Parameters:

- __REGULATOR__: : Specifies the regulator output voltage to achieve a trade-off between performance and power consumption when the device does not operate at the maximum frequency (refer to the datasheet for more details). This parameter can be one of the following values:
  - PWR_REGULATOR_VOLTAGE_SCALE0 : Regulator voltage output Scale 0 mode.
  - PWR_REGULATOR_VOLTAGE_SCALE1 : Regulator voltage output Scale 1 mode.
  - PWR_REGULATOR_VOLTAGE_SCALE2 : Regulator voltage output Scale 2 mode.
  - PWR_REGULATOR_VOLTAGE_SCALE3 : Regulator voltage output Scale 3 mode.

Return value:

- None.

Notes:

- For STM32H74x and STM32H75x lines, configuring Voltage Scale 0 is only possible when Vcore is supplied from LDO (Low DropOut). The SYSCFG Clock must be enabled through __HAL_RCC_SYSCFG_CLK_ENABLE() macro before configuring Voltage Scale 0 using __HAL_PWR_VOLTAGESCALING_CONFIG(). Transition to Voltage Scale 0 is only possible when the system is already in Voltage Scale 1. Transition from Voltage Scale 0 is only possible to Voltage Scale 1 then once in Voltage Scale 1 it is possible to switch to another voltage scale. After each regulator voltage setting, wait on VOSRDY flag to be set using macro __HAL_PWR_GET_FLAG(). To enter low power mode , and if current regulator voltage is Voltage Scale 0 then first switch to Voltage Scale 1 before entering low power mode.

__HAL_PWR_GET_FLAG

Description:

- Check PWR flags are set or not.

Parameters:

- __FLAG__: : Specifies the flag to check. This parameter can be one of the following values:
  - PWR_FLAG_PVDO : PVD Output. This flag is valid only if PVD is enabled by the HAL_PWR_EnablePVD() function. The PVD is stopped by STANDBY mode. For this reason, this bit is equal to 0 after STANDBY or reset until the PVDE bit is set.
  - PWR_FLAG_AVDO : AVD Output. This flag is valid only if AVD is enabled by the HAL_PWREx_EnableAVD() function. The AVD is stopped by STANDBY mode. For this reason, this bit is equal to 0 after STANDBY or reset until the AVDE bit is set.
  - PWR_FLAG_ACTVOSRDY : This flag indicates that the Regulator voltage scaling output selection is ready.
  - PWR_FLAG_BRR : Backup regulator ready flag. This bit is not reset when the device wakes up from STANDBY mode or by a system reset or power-on reset.
  - PWR_FLAG_VOSRDY : This flag indicates that the Regulator voltage scaling output selection is ready. mode or by a system reset or power-on reset.
  - PWR_FLAG_USB33RDY : This flag indicates that the USB supply from regulator is ready.
  - PWR_FLAG_TEMPH : This flag indicates that the temperature equal or above high threshold level.
  - PWR_FLAG_TEMPL : This flag indicates that the temperature equal or below low threshold level.
  - PWR_FLAG_VBATH : This flag indicates that VBAT level equal or above high threshold level.
  - PWR_FLAG_VBATL : This flag indicates that VBAT level equal or below low threshold level.
  - PWR_FLAG_STOP : This flag indicates that the system entered in STOP mode.
  - PWR_FLAG_SB : This flag indicates that the system entered in STANDBY mode.
  - PWR_FLAG_SB_D1 : This flag indicates that the D1 domain entered in STANDBY mode.
  - PWR_FLAG_SB_D2 : This flag indicates that the D2 domain entered in STANDBY mode.
  - PWR_FLAG2_STOP : This flag indicates that the system entered in STOP mode.
  - PWR_FLAG2_SB : This flag indicates that the system entered in STANDBY mode.
  - PWR_FLAG2_SB_D1 : This flag indicates that the D1 domain entered in STANDBY mode.
  - PWR_FLAG2_SB_D2 : This flag indicates that the D2 domain entered in STANDBY mode.
  - PWR_FLAG_CPU_HOLD : This flag indicates that the CPU1 wakes up with hold.
  - PWR_FLAG_CPU2_HOLD : This flag indicates that the CPU2 wakes up with hold.
  - PWR_FLAG_SMPSEXTRDY : This flag indicates that the SMPS External supply is sready.
  - PWR_FLAG_SCUEN : This flag indicates that the supply configuration update is enabled.
  - PWR_FLAG_MMCVDO : This flag indicates that the VDDMMC is above or equal to 1.2 V.

Return value:

- The: (__FLAG__) state (TRUE or FALSE).

Notes:

- The PWR_FLAG_PVDO, PWR_FLAG_AVDO, PWR_FLAG_ACTVOSRDY, PWR_FLAG_BRR, PWR_FLAG_VOSRDY, PWR_FLAG_USB33RDY, PWR_FLAG_TEMPH, PWR_FLAG_TEMPL, PWR_FLAG_VBATH, PWR_FLAG_VBATL, PWR_FLAG_STOP and PWR_FLAG_SB flags are used for all H7 family lines. The PWR_FLAG2_STOP, PWR_FLAG2_SB, PWR_FLAG2_SB_D1, PWR_FLAG2_SB_D2, PWR_FLAG_CPU_HOLD and PWR_FLAG_CPU2_HOLD flags are used only for H7 dual core lines. The PWR_FLAG_SB_D1 and PWR_FLAG_SB_D2 flags are used for all H7 family except STM32H7Axxx and STM32H7Bxxx lines. The PWR_FLAG_MMCVDO flag is used only for STM32H7Axxx and STM32H7Bxxx lines. The PWR_FLAG_SCUEN flag is used for devices that support only LDO regulator. The PWR_FLAG_SMPSEXTRDY flag is used for devices that support LDO and SMPS regulators.

__HAL_PWR_GET_WAKEUPFLAG

Description:

- Check PWR wake up flags are set or not.

Parameters:

- __FLAG__: specifies the wake up flag to check. This parameter can be one of the following values:
  - PWR_FLAG_WKUP1 : This parameter clear Wake up line 1 flag.
  - PWR_FLAG_WKUP2 : This parameter clear Wake up line 2 flag.
  - PWR_FLAG_WKUP3 : This parameter clear Wake up line 3 flag.
  - PWR_FLAG_WKUP4 : This parameter clear Wake up line 4 flag.
  - PWR_FLAG_WKUP5 : This parameter clear Wake up line 5 flag.
  - PWR_FLAG_WKUP6 : This parameter clear Wake up line 6 flag.

Return value:

- The: (__FLAG__) state (TRUE or FALSE).

Notes:

- The PWR_FLAG_WKUP3 and PWR_FLAG_WKUP5 are available only for devices that support GPIOI port.

__HAL_PWR_CLEAR_FLAG

Description:

- Clear CPU PWR flags.

Parameters:

- __FLAG__: : Specifies the flag to clear.

Return value:

- None.

Notes:

- This parameter is not used for the STM32H7 family and is kept as parameter just to maintain compatibility with other families. This macro clear all CPU flags STOPF, SBF, SBF_D1, and SBF_D2. This parameter can be one of the following values : PWR_CPU_FLAGS : Clear HOLD2F, STOPF, SBF, SBF_D1, and SBF_D2 CPU flags.

__HAL_PWR_CLEAR_WAKEUPFLAG

Description:

- Clear PWR wake up flags.

Parameters:

- __FLAG__: : Specifies the wake up flag to be cleared. This parameter can be one of the following values :
  - PWR_FLAG_WKUP1 : This parameter clear Wake up line 1 flag.
  - PWR_FLAG_WKUP2 : This parameter clear Wake up line 2 flag.
  - PWR_FLAG_WKUP3 : This parameter clear Wake up line 3 flag.
  - PWR_FLAG_WKUP4 : This parameter clear Wake up line 4 flag.
  - PWR_FLAG_WKUP5 : This parameter clear Wake up line 5 flag.
  - PWR_FLAG_WKUP6 : This parameter clear Wake up line 6 flag.

Return value:

- None.

Notes:

- The PWR_FLAG_WKUP3 and PWR_FLAG_WKUP5 are available only for devices that support GPIOI port.

__HAL_PWR_PVD_EXTI_ENABLE_IT

Description:

- Enable the PVD EXTI Line 16.

Return value:

- None.

__HAL_PWR_PVD_EXTID2_ENABLE_IT

Description:

- Enable the PVD EXTI D2 Line 16.

Return value:

- None.

__HAL_PWR_PVD_EXTI_DISABLE_IT

Description:

- Disable the PVD EXTI Line 16.

Return value:

- None.

__HAL_PWR_PVD_EXTID2_DISABLE_IT

Description:

- Disable the PVD EXTI D2 Line 16.

Return value:

- None.

__HAL_PWR_PVD_EXTI_ENABLE_EVENT

Description:

- Enable event on PVD EXTI Line 16.

Return value:

- None.

__HAL_PWR_PVD_EXTID2_ENABLE_EVENT

Description:

- Enable event on PVD EXTI D2 Line.

Return value:

- None.

__HAL_PWR_PVD_EXTI_DISABLE_EVENT

Description:

- Disable event on PVD EXTI Line 16.

Return value:

- None.

__HAL_PWR_PVD_EXTID2_DISABLE_EVENT

Description:

- Disable event on PVD EXTI D2 Line.

Return value:

- None.

__HAL_PWR_PVD_EXTI_ENABLE_RISING_EDGE

Description:

- Enable the PVD Rising Interrupt Trigger.

Return value:

- None.

__HAL_PWR_PVD_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the PVD Rising Interrupt Trigger.

Return value:

- None.

__HAL_PWR_PVD_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the PVD Falling Interrupt Trigger.

Return value:

- None.

__HAL_PWR_PVD_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable the PVD Falling Interrupt Trigger.

Return value:

- None.

__HAL_PWR_PVD_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable the PVD Rising & Falling Interrupt Trigger.

Return value:

- None.

__HAL_PWR_PVD_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable the PVD Rising & Falling Interrupt Trigger.

Return value:

- None.

__HAL_PWR_PVD_EXTI_GET_FLAG

Description:

- Check whether the specified PVD EXTI interrupt flag is set or not.

Return value:

- EXTI: PVD Line Status.

__HAL_PWR_PVD_EXTID2_GET_FLAG

Description:

- Checks whether the specified PVD EXTI interrupt flag is set or not.

Return value:

- EXTI: D2 PVD Line Status.

__HAL_PWR_PVD_EXTI_CLEAR_FLAG

Description:

- Clear the PVD EXTI flag.

Return value:

- None.

__HAL_PWR_PVD_EXTID2_CLEAR_FLAG

Description:

- Clear the PVD EXTI D2 flag.

Return value:

- None.

__HAL_PWR_PVD_EXTI_GENERATE_SWIT

Description:

- Generates a Software interrupt on PVD EXTI line.

Return value:

- None.

PWR Flag

PWR_FLAG_STOP

PWR_FLAG_SB_D1

PWR_FLAG_SB_D2

PWR_FLAG_SB

PWR_FLAG_CPU_HOLD

PWR_FLAG_CPU2_HOLD

PWR_FLAG2_STOP

PWR_FLAG2_SB_D1

PWR_FLAG2_SB_D2

PWR_FLAG2_SB

PWR_FLAG_PVDO

PWR_FLAG_AVDO

PWR_FLAG_ACTVOSRDY

PWR_FLAG_ACTVOS

PWR_FLAG_BRR

PWR_FLAG_VOSRDY

PWR_FLAG_SMPSEXTRDY

PWR_FLAG_USB33RDY

PWR_FLAG_TEMPH

PWR_FLAG_TEMPL

PWR_FLAG_VBATH

PWR_FLAG_VBATL

PWR_FLAG_WKUP1

PWR_FLAG_WKUP2

PWR_FLAG_WKUP3

PWR_FLAG_WKUP4

PWR_FLAG_WKUP5

PWR_FLAG_WKUP6

PWR Private macros to check input parameters

IS_PWR_PVD_LEVEL

IS_PWR_PVD_MODE

IS_PWR_REGULATOR

IS_PWR_SLEEP_ENTRY

IS_PWR_STOP_ENTRY

IS_PWR_REGULATOR_VOLTAGE

PWR PVD detection level

PWR_PVDLEVEL_0

Programmable voltage detector level 0 selection : 1V95

PWR_PVDLEVEL_1

Programmable voltage detector level 1 selection : 2V1

PWR_PVDLEVEL_2

Programmable voltage detector level 2 selection : 2V25

PWR_PVDLEVEL_3

Programmable voltage detector level 3 selection : 2V4

PWR_PVDLEVEL_4

Programmable voltage detector level 4 selection : 2V55

PWR_PVDLEVEL_5

Programmable voltage detector level 5 selection : 2V7

PWR_PVDLEVEL_6

Programmable voltage detector level 6 selection : 2V85

PWR_PVDLEVEL_7

External input analog voltage (Compare internally to VREF)

PWR PVD EXTI Line

PWR_EXTI_LINE_PVD

External interrupt line 16 Connected to the PVD EXTI Line

PWR PVD Mode

PWR_PVD_MODE_NORMAL

Basic mode is used

PWR_PVD_MODE_IT_RISING

Interrupt Mode with Rising edge trigger detection

PWR_PVD_MODE_IT_FALLING

Interrupt Mode with Falling edge trigger detection

PWR_PVD_MODE_IT_RISING_FALLING

Interrupt Mode with Rising/Falling edge trigger detection

PWR_PVD_MODE_EVENT_RISING

Event Mode with Rising edge trigger detection

PWR_PVD_MODE_EVENT_FALLING

Event Mode with Falling edge trigger detection

PWR_PVD_MODE_EVENT_RISING_FALLING

Event Mode with Rising/Falling edge trigger detection

PWR PVD Mode Mask

PVD_RISING_EDGE

PVD_FALLING_EDGE

PVD_RISING_FALLING_EDGE

PWR Regulator state in SLEEP/STOP mode

PWR_MAINREGULATOR_ON

PWR_LOWPOWERREGULATOR_ON

PWR Regulator Voltage Scale

PWR_REGULATOR_VOLTAGE_SCALE0

PWR_REGULATOR_VOLTAGE_SCALE1

PWR_REGULATOR_VOLTAGE_SCALE2

PWR_REGULATOR_VOLTAGE_SCALE3

PWR SLEEP mode entry

PWR_SLEEPENTRY_WFI

PWR_SLEEPENTRY_WFE

PWR STOP mode entry

PWR_STOPENTRY_WFI

PWR_STOPENTRY_WFE

65 HAL PWR Extension Driver

65.1 PWREx Firmware driver registers structures

65.1.1 PWREx_AVDTypeDef

PWREx_AVDTypeDef is defined in the stm32h7xx_hal_pwr_ex.h

Data Fields

- ***uint32_t AVDLevel***
- ***uint32_t Mode***

Field Documentation

- ***uint32_t PWREx_AVDTypeDef::AVDLevel***

  AVDLevel : Specifies the AVD detection level. This parameter can be a value of ***PWREx_AVD_detection_level***
- ***uint32_t PWREx_AVDTypeDef::Mode***

  Mode : Specifies the EXTI operating mode for the AVD event. This parameter can be a value of ***PWREx_AVD_Mode***.

65.1.2 PWREx_WakeupPinTypeDef

PWREx_WakeupPinTypeDef is defined in the stm32h7xx_hal_pwr_ex.h

Data Fields

- ***uint32_t WakeUpPin***
- ***uint32_t PinPolarity***
- ***uint32_t PinPull***

Field Documentation

- ***uint32_t PWREx_WakeupPinTypeDef::WakeUpPin***

  WakeUpPin: Specifies the Wake-Up pin to be enabled. This parameter can be a value of ***PWREx_WakeUp_Pins***
- ***uint32_t PWREx_WakeupPinTypeDef::PinPolarity***

  PinPolarity: Specifies the Wake-Up pin polarity. This parameter can be a value of ***PWREx_PIN_Polarity***
- ***uint32_t PWREx_WakeupPinTypeDef::PinPull***

  PinPull: Specifies the Wake-Up pin pull. This parameter can be a value of ***PWREx_PIN_Pull***

65.2 PWREx Firmware driver API description

The following section lists the various functions of the PWREx library.

65.2.1 How to use this driver

1. Call HAL_PWREx_ConfigSupply() function to configure the regulator supply with the following different setups according to hardware (support SMPS): (+) PWR_DIRECT_SMPS_SUPPLY (+) PWR_SMPS_1V8_SUPPLIES_LDO (+) PWR_SMPS_2V5_SUPPLIES_LDO (+) PWR_SMPS_1V8_SUPPLIES_EXT_AND_LDO (+) PWR_SMPS_2V5_SUPPLIES_EXT_AND_LDO (+) PWR_SMPS_1V8_SUPPLIES_EXT (+) PWR_SMPS_2V5_SUPPLIES_EXT (+) PWR_LDO_SUPPLY (+) PWR_EXTERNAL_SOURCE_SUPPLY
2. Call HAL_PWREx_GetSupplyConfig() function to get the current supply setup.
3. Call HAL_PWREx_ControlVoltageScaling() function to configure the main internal regulator output voltage. The voltage scaling could be one of the following scales : (+) PWR_REGULATOR_VOLTAGE_SCALE0 (+) PWR_REGULATOR_VOLTAGE_SCALE1 (+) PWR_REGULATOR_VOLTAGE_SCALE2 (+) PWR_REGULATOR_VOLTAGE_SCALE3

4. Call HAL_PWREx_GetVoltageRange() function to get the current output voltage applied to the main regulator.
5. Call HAL_PWREx_ControlStopModeVoltageScaling() function to configure the main internal regulator output voltage in STOP mode. The voltage scaling in STOP mode could be one of the following scales : (+) PWR_REGULATOR_SVOS_SCALE3 (+) PWR_REGULATOR_SVOS_SCALE4 (+) PWR_REGULATOR_SVOS_SCALE5
6. Call HAL_PWREx_GetStopModeVoltageRange() function to get the current output voltage applied to the main regulator in STOP mode.
7. Call HAL_PWREx_EnterSTOP2Mode() function to enter the system in STOP mode with core domain in D2STOP mode. This API is used only for STM32H7Axxx and STM32H7Bxxx devices. Please ensure to clear all CPU pending events by calling HAL_PWREx_ClearPendingEvent() function when trying to enter the Cortex-Mx in DEEP-SLEEP mode with __WFE() entry.
8. Call HAL_PWREx_EnterSTOPMode() function to enter the selected domain in DSTOP mode. Call this API with all available power domains to enter the system in STOP mode. Please ensure to clear all CPU pending events by calling HAL_PWREx_ClearPendingEvent() function when trying to enter the Cortex-Mx in DEEP-SLEEP mode with __WFE() entry.
9. Call HAL_PWREx_ClearPendingEvent() function always before entring the Cortex-Mx in any low power mode (SLEEP/DEEP-SLEEP) using WFE entry.
10. Call HAL_PWREx_EnterSTANDBYMode() function to enter the selected domain in DSTANDBY mode. Call this API with all available power domains to enter the system in STANDBY mode.
11. Call HAL_PWREx_ConfigD3Domain() function to setup the D3/SRD domain state (RUN/STOP) when the system enter to low power mode.
12. Call HAL_PWREx_ClearDomainFlags() function to clear the CPU flags for the selected power domain. This API is used only for dual core devices.
13. Call HAL_PWREx_HoldCore() and HAL_PWREx_ReleaseCore() functions to hold and release the selected CPU and and their domain peripherals when exiting STOP mode. These APIs are used only for dual core devices.
14. Call HAL_PWREx_EnableFlashPowerDown() and HAL_PWREx_DisableFlashPowerDown() functions to enable and disable the Flash Power Down in STOP mode.
15. Call HAL_PWREx_EnableMemoryShutOff() and HAL_PWREx_DisableMemoryShutOff() functions to enable and disable the memory block shut-off in DStop or DStop2. These APIs are used only for STM32H7Axxx and STM32H7Bxxx lines.
16. Call HAL_PWREx_EnableWakeUpPin() and HAL_PWREx_DisableWakeUpPin() functions to enable and disable the Wake-up pin functionality for the selected pin.
17. Call HAL_PWREx_GetWakeupFlag() and HAL_PWREx_ClearWakeupFlag() functions to manage wake-up flag for the selected pin.
18. Call HAL_PWREx_WAKEUP_PIN_IRQHandler() function to handle all wake-up pins interrupts.
19. Call HAL_PWREx_EnableBkUpReg() and HAL_PWREx_DisableBkUpReg() functions to enable and disable the backup domain regulator.
20. Call HAL_PWREx_EnableUSBReg(), HAL_PWREx_DisableUSBReg(), HAL_PWREx_EnableUSBVoltageDetector() and HAL_PWREx_DisableUSBVoltageDetector() functions to manage USB power regulation functionalities.
21. Call HAL_PWREx_EnableBatteryCharging() and HAL_PWREx_DisableBatteryCharging() functions to enable and disable the battery charging feature with the selected resistor.
22. Call HAL_PWREx_EnableAnalogBooster() and HAL_PWREx_DisableAnalogBooster() functions to enable and disable the AVD boost feature when the VDD supply voltage is below 2V7.
23. Call HAL_PWREx_EnableMonitoring() and HAL_PWREx_DisableMonitoring() functions to enable and disable the VBAT and Temperature monitoring. When VBAT and Temperature monitoring feature is enables, use HAL_PWREx_GetTemperatureLevel() and HAL_PWREx_GetVBATLevel() to get respectively the Temperature level and VBAT level.
24. Call HAL_PWREx_GetMMCVoltage() and HAL_PWREx_DisableMonitoring() function to get VDDMMC voltage level. This API is used only for STM32H7Axxx and STM32H7Bxxx lines

25. Call HAL_PWREx_ConfigAVD() after setting parameter to be configured (event mode and voltage threshold) in order to set up the Analog Voltage Detector then use HAL_PWREx_EnableAVD() and HAL_PWREx_DisableAVD() functions to start and stop the AVD detection. (+) AVD level could be one of the following values :
    – 1V7
    – 2V1
    – 2V5
    – 2V8
26. Call HAL_PWREx_PVD_AVD_IRQHandler() function to handle the PWR PVD and AVD interrupt request.

(#) Call HAL_PWREx_ConfigSupply() function to configure the regulator supply with the following different setups according to hardware (support SMPS):

- PWR_DIRECT_SMPS_SUPPLY
- PWR_SMPS_1V8_SUPPLIES_LDO
- PWR_SMPS_2V5_SUPPLIES_LDO
- PWR_SMPS_1V8_SUPPLIES_EXT_AND_LDO
- PWR_SMPS_2V5_SUPPLIES_EXT_AND_LDO
- PWR_SMPS_1V8_SUPPLIES_EXT
- PWR_SMPS_2V5_SUPPLIES_EXT
- PWR_LDO_SUPPLY
- PWR_EXTERNAL_SOURCE_SUPPLY (#) Call HAL_PWREx_GetSupplyConfig() function to get the current supply setup. (#) Call HAL_PWREx_ControlVoltageScaling() function to configure the main internal regulator output voltage. The voltage scaling could be one of the following scales :
- PWR_REGULATOR_VOLTAGE_SCALE0
- PWR_REGULATOR_VOLTAGE_SCALE1
- PWR_REGULATOR_VOLTAGE_SCALE2
- PWR_REGULATOR_VOLTAGE_SCALE3 (#) Call HAL_PWREx_GetVoltageRange() function to get the current output voltage applied to the main regulator. (#) Call HAL_PWREx_ControlStopModeVoltageScaling() function to configure the main internal regulator output voltage in STOP mode. The voltage scaling in STOP mode could be one of the following scales :
- PWR_REGULATOR_SVOS_SCALE3
- PWR_REGULATOR_SVOS_SCALE4

- PWR_REGULATOR_SVOS_SCALE5 (#) Call HAL_PWREx_GetStopModeVoltageRange() function to get the current output voltage applied to the main regulator in STOP mode. (#) Call HAL_PWREx_EnterSTOP2Mode() function to enter the system in STOP mode with core domain in D2STOP mode. This API is used only for STM32H7Axxx and STM32H7Bxxx devices. Please ensure to clear all CPU pending events by calling HAL_PWREx_ClearPendingEvent() function when trying to enter the Cortex-Mx in DEEP-SLEEP mode with __WFE() entry. (#) Call HAL_PWREx_EnterSTOPMode() function to enter the selected domain in DSTOP mode. Call this API with all available power domains to enter the system in STOP mode. Please ensure to clear all CPU pending events by calling HAL_PWREx_ClearPendingEvent() function when trying to enter the Cortex-Mx in DEEP-SLEEP mode with __WFE() entry. (#) Call HAL_PWREx_ClearPendingEvent() function always before entring the Cortex-Mx in any low power mode (SLEEP/DEEP-SLEEP) using WFE entry. (#) Call HAL_PWREx_EnterSTANDBYMode() function to enter the selected domain in DSTANDBY mode. Call this API with all available power domains to enter the system in STANDBY mode. (#) Call HAL_PWREx_ConfigD3Domain() function to setup the D3/SRD domain state (RUN/STOP) when the system enter to low power mode. (#) Call HAL_PWREx_ClearDomainFlags() function to clear the CPU flags for the selected power domain. This API is used only for dual core devices. (#) Call HAL_PWREx_HoldCore() and HAL_PWREx_ReleaseCore() functions to hold and release the selected CPU and and their domain peripherals when exiting STOP mode. These APIs are used only for dual core devices. (#) Call HAL_PWREx_EnableFlashPowerDown() and HAL_PWREx_DisableFlashPowerDown() functions to enable and disable the Flash Power Down in STOP mode. (#) Call HAL_PWREx_EnableMemoryShutOff() and HAL_PWREx_DisableMemoryShutOff() functions to enable and disable the memory block shut-off in DStop or DStop2. These APIs are used only for STM32H7Axxx and STM32H7Bxxx lines. (#) Call HAL_PWREx_EnableWakeUpPin() and HAL_PWREx_DisableWakeUpPin() functions to enable and disable the Wake-up pin functionality for the selected pin. (#) Call HAL_PWREx_GetWakeupFlag() and HAL_PWREx_ClearWakeupFlag() functions to manage wake-up flag for the selected pin. (#) Call HAL_PWREx_WAKEUP_PIN_IRQHandler() function to handle all wake-up pins interrupts. (#) Call HAL_PWREx_EnableBkUpReg() and HAL_PWREx_DisableBkUpReg() functions to enable and disable the backup domain regulator. (#) Call HAL_PWREx_EnableUSBReg(), HAL_PWREx_DisableUSBReg(), HAL_PWREx_EnableUSBVoltageDetector() and HAL_PWREx_DisableUSBVoltageDetector() functions to manage USB power regulation functionalities. (#) Call HAL_PWREx_EnableBatteryCharging() and HAL_PWREx_DisableBatteryCharging() functions to enable and disable the battery charging feature with the selected resistor. (#) Call HAL_PWREx_EnableAnalogBooster() and HAL_PWREx_DisableAnalogBooster() functions to enable and disable the AVD boost feature when the VDD supply voltage is below 2V7. (#) Call HAL_PWREx_EnableMonitoring() and HAL_PWREx_DisableMonitoring() functions to enable and disable the VBAT and Temperature monitoring. When VBAT and Temperature monitoring feature is enables, use HAL_PWREx_GetTemperatureLevel() and HAL_PWREx_GetVBATLevel() to get respectively the Temperature level and VBAT level. (#) Call HAL_PWREx_GetMMCVoltage() and HAL_PWREx_DisableMonitoring() function to get VDDMMC voltage level. This API is used only for STM32H7Axxx and STM32H7Bxxx lines (#) Call HAL_PWREx_ConfigAVD() after setting parameter to be configured (event mode and voltage threshold) in order to set up the Analog Voltage Detector then use HAL_PWREx_EnableAVD() and HAL_PWREx_DisableAVD() functions to start and stop the AVD detection.
- AVD level could be one of the following values :
  - 1V7
  - 2V1
  - 2V5
  - 2V8 (#) Call HAL_PWREx_PVD_AVD_IRQHandler() function to handle the PWR PVD and AVD interrupt request.

65.2.2 Power supply control functions

1. When the system is powered on, the POR monitors VDD supply. Once VDD is above the POR threshold level, the voltage regulator is enabled in the default supply configuration: (+) The Voltage converter output level is set at 1V0 in accordance with the VOS3 level configured in PWR (D3/SRD) domain control register (PWR_D3CR/PWR_SRDCR). (+) The system is kept in reset mode as long as VCORE is not ok. (+) Once VCORE is ok, the system is taken out of reset and the HSI oscillator is enabled. (+) Once the oscillator is stable, the system is initialized: Flash memory and option bytes are loaded and the CPU starts in Run* mode. (+) The software shall then initialize the system including supply configuration programming using the HAL_PWREx_ConfigSupply(). (+) Once the supply configuration has been configured, the HAL_PWREx_ConfigSupply() function checks the ACTVOSRDY bit in PWR control status register 1 (PWR_CSR1) to guarantee a valid voltage levels:
   – As long as ACTVOSRDY indicates that voltage levels are invalid, the system is in limited Run* mode, write accesses to the RAMs are not permitted and VOS shall not be changed.
   – Once ACTVOSRDY indicates that voltage levels are valid, the system is in normal Run mode, write accesses to RAMs are allowed and VOS can be changed.

(#) When the system is powered on, the POR monitors VDD supply. Once VDD is above the POR threshold level, the voltage regulator is enabled in the default supply configuration:

- The Voltage converter output level is set at 1V0 in accordance with the VOS3 level configured in PWR (D3/SRD) domain control register (PWR_D3CR/PWR_SRDCR).
- The system is kept in reset mode as long as VCORE is not ok.
- Once VCORE is ok, the system is taken out of reset and the HSI oscillator is enabled.
- Once the oscillator is stable, the system is initialized: Flash memory and option bytes are loaded and the CPU starts in Run* mode.
- The software shall then initialize the system including supply configuration programming using the HAL_PWREx_ConfigSupply().
- Once the supply configuration has been configured, the HAL_PWREx_ConfigSupply() function checks the ACTVOSRDY bit in PWR control status register 1 (PWR_CSR1) to guarantee a valid voltage levels:
  – As long as ACTVOSRDY indicates that voltage levels are invalid, the system is in limited Run* mode, write accesses to the RAMs are not permitted and VOS shall not be changed.
  – Once ACTVOSRDY indicates that voltage levels are valid, the system is in normal Run mode, write accesses to RAMs are allowed and VOS can be changed.

This section contains the following APIs:

- ***HAL_PWREx_ConfigSupply()***
- ***HAL_PWREx_GetSupplyConfig()***
- ***HAL_PWREx_ControlVoltageScaling()***
- ***HAL_PWREx_GetVoltageRange()***
- ***HAL_PWREx_ControlStopModeVoltageScaling()***
- ***HAL_PWREx_GetStopModeVoltageRange()***

65.2.3 Low power control functions

Domains Low Power modes configuration

This section provides the extended low power mode control APIs. The system presents 3 principles domains (D1, D2 and D3) that can be operated in low-power modes (DSTOP or DSTANDBY mode):

- DSTOP mode to enters a domain to STOP mode:
  - D1 domain and/or D2 domain enters DSTOP mode only when the CPU subsystem is in CSTOP mode and has allocated peripheral in the domain. In DSTOP mode the domain bus matrix clock is stopped.
  - The system enters STOP mode using one of the following scenarios:
    - D1 domain enters DSTANDBY mode (powered off) and D2, D3 domains enter DSTOP mode.
    - D2 domain enters DSTANDBY mode (powered off) and D1, D3 domains enter DSTOP mode.
    - D3 domain enters DSTANDBY mode (powered off) and D1, D2 domains enter DSTOP mode.
    - D1 and D2 domains enter DSTANDBY mode (powered off) and D3 domain enters DSTOP mode.
    - D1 and D3 domains enter DSTANDBY mode (powered off) and D2 domain enters DSTOP mode.
    - D2 and D3 domains enter DSTANDBY mode (powered off) and D1 domain enters DSTOP mode.
    - D1, D2 and D3 domains enter DSTOP mode.
  - When the system enters STOP mode, the clocks are stopped and the regulator is running in main or low power mode.
  - D3 domain can be kept in Run mode regardless of the CPU status when enter STOP mode by using HAL_PWREx_ConfigD3Domain(D3State) function.
- DSTANDBY mode to enters a domain to STANDBY mode:
  - The DSTANDBY mode is entered when the PDDS_Dn bit in PWR CPU control register (PWR_CPUCR) for the Dn domain selects Standby mode.
  - The system enters STANDBY mode only when D1, D2 and D3 domains enter DSTANDBY mode. Consequently the VCORE supply regulator is powered off.

DSTOP mode

In DStop mode the domain bus matrix clock is stopped. The Flash memory can enter low-power Stop mode when it is enabled through FLPS in PWR_CR1 register. This allows a trade-off between domain DStop restart time and low power consumption.

In DStop mode domain peripherals using the LSI or LSE clock and peripherals having a kernel clock request are still able to operate.

Before entering DSTOP mode it is recommended to call SCB_CleanDCache function in order to clean the D-Cache and guarantee the data integrity for the SRAM memories.

- Entry: The DSTOP mode is entered using the HAL_PWREx_EnterSTOPMode(Regulator, STOPEntry, Domain) function with:
  - Regulator:
    - PWR_MAINREGULATOR_ON : Main regulator ON.
    - PWR_LOWPOWERREGULATOR_ON : Low Power regulator ON.
  - STOPEntry:
    - PWR_STOPENTRY_WFI : enter STOP mode with WFI instruction
    - PWR_STOPENTRY_WFE : enter STOP mode with WFE instruction
  - Domain:
    - PWR_D1_DOMAIN : Enters D1/CD domain to DSTOP mode.
    - PWR_D2_DOMAIN : Enters D2 domain to DSTOP mode.
    - PWR_D3_DOMAIN : Enters D3/SRD domain to DSTOP mode.
- Exit: Any EXTI Line (Internal or External) configured in Interrupt/Event mode.

DSTANDBY mode

In DStandby mode:

- The domain bus matrix clock is stopped.
- The domain is powered down and the domain RAM and register contents are lost.

Before entering DSTANDBY mode it is recommended to call SCB_CleanDCache function in order to clean the D-Cache and guarantee the data integrity for the SRAM memories.

- Entry: The DSTANDBY mode is entered using the HAL_PWREx_EnterSTANDBYMode (Domain) function with:
  - Domain:
    - PWR_D1_DOMAIN : Enters D1/CD domain to DSTANDBY mode.
    - PWR_D2_DOMAIN : Enters D2 domain to DSTANDBY mode.
    - PWR_D3_DOMAIN : Enters D3/SRD domain to DSTANDBY mode.
- Exit: WKUP pin rising or falling edge, RTC alarm (Alarm A and Alarm B), RTC wakeup, tamper event, time stamp event, external reset in NRST pin, IWDG reset.

Keep D3/SRD in RUN mode

D3/SRD domain can be kept in Run mode regardless of the CPU status when entering STOP mode by using HAL_PWREx_ConfigD3Domain(D3State) function with :

- D3State:
  - PWR_D3_DOMAIN_STOP : D3/SDR domain follows the CPU sub-system mode.
  - PWR_D3_DOMAIN_RUN : D3/SRD domain remains in Run mode regardless of CPU subsystem mode.

FLASH Power Down configuration

By setting the FLPS bit in the PWR_CR1 register using the HAL_PWREx_EnableFlashPowerDown() function, the Flash memory also enters power down mode when the device enters STOP mode. When the Flash memory is in power down mode, an additional startup delay is incurred when waking up from STOP mode.

Wakeup Pins configuration

Wakeup pins allow the system to exit from Standby mode. The configuration of wakeup pins is done with the HAL_PWREx_EnableWakeUpPin(sPinParams) function with:

- sPinParams: structure to enable and configure a wakeup pin:
  - WakeUpPin: Wakeup pin to be enabled.
  - PinPolarity: Wakeup pin polarity (rising or falling edge).
  - PinPull: Wakeup pin pull (no pull, pull-up or pull-down).

The wakeup pins are internally connected to the EXTI lines [55-60] to generate an interrupt if enabled. The EXTI lines configuration is done by the HAL_EXTI_Dx_EventInputConfig() functions defined in the stm32h7xxhal.c file.

When a wakeup pin event is received the HAL_PWREx_WAKEUP_PIN_IRQHandler is called and the appropriate flag is set in the PWR_WKUPFR register. Then in the HAL_PWREx_WAKEUP_PIN_IRQHandler function the wakeup pin flag will be cleared and the appropriate user callback will be called. The user can add his own code by customization of function pointer HAL_PWREx_WKUPx_Callback.

This section contains the following APIs:

- ***HAL_PWREx_EnterSTOPMode()***
- ***HAL_PWREx_ClearPendingEvent()***
- ***HAL_PWREx_EnterSTANDBYMode()***
- ***HAL_PWREx_ConfigD3Domain()***
- ***HAL_PWREx_ClearDomainFlags()***
- ***HAL_PWREx_HoldCore()***
- ***HAL_PWREx_ReleaseCore()***
- ***HAL_PWREx_EnableFlashPowerDown()***
- ***HAL_PWREx_DisableFlashPowerDown()***
- ***HAL_PWREx_EnableWakeUpPin()***
- ***HAL_PWREx_DisableWakeUpPin()***
- ***HAL_PWREx_GetWakeupFlag()***
- ***HAL_PWREx_ClearWakeupFlag()***
- ***HAL_PWREx_WAKEUP_PIN_IRQHandler()***
- ***HAL_PWREx_WKUP1_Callback()***

- *HAL_PWREx_WKUP2_Callback()*
- *HAL_PWREx_WKUP3_Callback()*
- *HAL_PWREx_WKUP4_Callback()*
- *HAL_PWREx_WKUP5_Callback()*
- *HAL_PWREx_WKUP6_Callback()*

65.2.4 Peripherals control functions

Main and Backup Regulators configuration

- The backup domain includes 4 Kbytes of backup SRAM accessible only from the CPU, and addressed in 32-bit, 16-bit or 8-bit mode. Its content is retained even in Standby or VBAT mode when the low power backup regulator is enabled. It can be considered as an internal EEPROM when VBAT is always present. You can use the HAL_PWREx_EnableBkUpReg() function to enable the low power backup regulator.
- When the backup domain is supplied by VDD (analog switch connected to VDD) the backup SRAM is powered from VDD which replaces the VBAT power supply to save battery life.
- The backup SRAM is not mass erased by a tamper event. It is read protected to prevent confidential data, such as cryptographic private key, from being accessed. The backup SRAM can be erased only through the Flash interface when a protection level change from level 1 to level 0 is requested.

Note: Refer to the description of Read protection (RDP) in the Flash programming manual.

- The main internal regulator can be configured to have a tradeoff between performance and power consumption when the device does not operate at the maximum frequency. This is done through HAL_PWREx_ControlVoltageScaling(VOS) function which configure the VOS bit in PWR_D3CR register.
- The main internal regulator can be configured to operate in Low Power mode when the system enters STOP mode to further reduce power consumption. This is done through HAL_PWREx_ControlStopModeVoltageScaling(SVOS) function which configure the SVOS bit in PWR_CR1 register. The selected SVOS4 and SVOS5 levels add an additional startup delay when exiting from system Stop mode.

Note: Refer to the product datasheets for more details.

USB Regulator configuration

- The USB transceivers are supplied from a dedicated VDD33USB supply that can be provided either by the integrated USB regulator, or by an external USB supply.
- The USB regulator is enabled by HAL_PWREx_EnableUSBReg() function, the VDD33USB is then provided from the USB regulator.
- When the USB regulator is enabled, the VDD33USB supply level detector shall be enabled through HAL_PWREx_EnableUSBVoltageDetector() function.
- The USB regulator is disabled through HAL_PWREx_DisableUSBReg() function and VDD33USB can be provided from an external supply. In this case VDD33USB and VDD50USB shall be connected together.

VBAT battery charging

- When VDD is present, the external battery connected to VBAT can be charged through an internal resistance. VBAT charging can be performed either through a 5 KOhm resistor or through a 1.5 KOhm resistor.
- VBAT charging is enabled by HAL_PWREx_EnableBatteryCharging (ResistorValue) function with:
  - ResistorValue:
    - PWR_BATTERY_CHARGING_RESISTOR_5: 5 KOhm resistor.
    - PWR_BATTERY_CHARGING_RESISTOR_1_5: 1.5 KOhm resistor.
- VBAT charging is disabled by HAL_PWREx_DisableBatteryCharging() function.

This section contains the following APIs:

- *HAL_PWREx_EnableBkUpReg()*
- *HAL_PWREx_DisableBkUpReg()*
- *HAL_PWREx_EnableUSBReg()*

- *HAL_PWREx_DisableUSBReg()*
- *HAL_PWREx_EnableUSBVoltageDetector()*
- *HAL_PWREx_DisableUSBVoltageDetector()*
- *HAL_PWREx_EnableBatteryCharging()*
- *HAL_PWREx_DisableBatteryCharging()*

65.2.5 Power Monitoring functions

VBAT and Temperature supervision

- The VBAT battery voltage supply can be monitored by comparing it with two threshold levels: VBAThigh and VBATlow. VBATH flag and VBATL flags in the PWR control register 2 (PWR_CR2), indicate if VBAT is higher or lower than the threshold.
- The temperature can be monitored by comparing it with two threshold levels, TEMPhigh and TEMPlow. TEMPH and TEMPL flags, in the PWR control register 2 (PWR_CR2), indicate whether the device temperature is higher or lower than the threshold.
- The VBAT and the temperature monitoring is enabled by HAL_PWREx_EnableMonitoring() function and disabled by HAL_PWREx_DisableMonitoring() function.
- The HAL_PWREx_GetVBATLevel() function returns the VBAT level which can be : PWR_VBAT_BELOW_LOW_THRESHOLD or PWR_VBAT_ABOVE_HIGH_THRESHOLD or PWR_VBAT_BETWEEN_HIGH_LOW_THRESHOLD.
- The HAL_PWREx_GetTemperatureLevel() function returns the Temperature level which can be : PWR_TEMP_BELOW_LOW_THRESHOLD or PWR_TEMP_ABOVE_HIGH_THRESHOLD or PWR_TEMP_BETWEEN_HIGH_LOW_THRESHOLD.

AVD configuration

- The AVD is used to monitor the VDDA power supply by comparing it to a threshold selected by the AVD Level (ALS[3:0] bits in the PWR_CR1 register).
- A AVDO flag is available to indicate if VDDA is higher or lower than the AVD threshold. This event is internally connected to the EXTI line 16 to generate an interrupt if enabled. It is configurable through __HAL_PWR_AVD_EXTI_ENABLE_IT() macro.
- The AVD is stopped in System Standby mode.

This section contains the following APIs:

- *HAL_PWREx_EnableMonitoring()*
- *HAL_PWREx_DisableMonitoring()*
- *HAL_PWREx_GetTemperatureLevel()*
- *HAL_PWREx_GetVBATLevel()*
- *HAL_PWREx_ConfigAVD()*
- *HAL_PWREx_EnableAVD()*
- *HAL_PWREx_DisableAVD()*
- *HAL_PWREx_PVD_AVD_IRQHandler()*
- *HAL_PWREx_AVDCallback()*

65.2.6 Detailed description of functions

HAL_PWREx_ConfigSupply

Function name

HAL_StatusTypeDef HAL_PWREx_ConfigSupply (uint32_t SupplySource)

Function description

Configure the system Power Supply.

Parameters

- **SupplySource:** : Specifies the Power Supply source to set after a system startup. This parameter can be one of the following values :
  - PWR_DIRECT_SMPS_SUPPLY : The SMPS supplies the Vcore Power Domains. The LDO is Bypassed.
  - PWR_SMPS_1V8_SUPPLIES_LDO : The SMPS 1.8V output supplies the LDO. The Vcore Power Domains are supplied from the LDO.
  - PWR_SMPS_2V5_SUPPLIES_LDO : The SMPS 2.5V output supplies the LDO. The Vcore Power Domains are supplied from the LDO.
  - PWR_SMPS_1V8_SUPPLIES_EXT_AND_LDO : The SMPS 1.8V output supplies external circuits and the LDO. The Vcore Power Domains are supplied from the LDO.
  - PWR_SMPS_2V5_SUPPLIES_EXT_AND_LDO : The SMPS 2.5V output supplies external circuits and the LDO. The Vcore Power Domains are supplied from the LDO.
  - PWR_SMPS_1V8_SUPPLIES_EXT : The SMPS 1.8V output supplies external circuits. The LDO is Bypassed. The Vcore Power Domains are supplied from external source.
  - PWR_SMPS_2V5_SUPPLIES_EXT : The SMPS 2.5V output supplies external circuits. The LDO is Bypassed. The Vcore Power Domains are supplied from external source.
  - PWR_LDO_SUPPLY : The LDO regulator supplies the Vcore Power Domains. The SMPS regulator is Bypassed.
  - PWR_EXTERNAL_SOURCE_SUPPLY : The SMPS and the LDO are Bypassed. The Vcore Power Domains are supplied from external source.

Return values

- **HAL:** status.

Notes

- The PWR_LDO_SUPPLY and PWR_EXTERNAL_SOURCE_SUPPLY are used by all H7 lines. The PWR_DIRECT_SMPS_SUPPLY, PWR_SMPS_1V8_SUPPLIES_LDO, PWR_SMPS_2V5_SUPPLIES_LDO, PWR_SMPS_1V8_SUPPLIES_EXT_AND_LDO, PWR_SMPS_2V5_SUPPLIES_EXT_AND_LDO, PWR_SMPS_1V8_SUPPLIES_EXT and PWR_SMPS_2V5_SUPPLIES_EXT are used only for lines that supports SMPS regulator.

HAL_PWREx_GetSupplyConfig

Function name

uint32_t HAL_PWREx_GetSupplyConfig (void)

Function description

Get the power supply configuration.

Return values

- **The:** supply configuration.

HAL_PWREx_ControlVoltageScaling

Function name

HAL_StatusTypeDef HAL_PWREx_ControlVoltageScaling (uint32_t VoltageScaling)

Function description

Configure the main internal regulator output voltage.

Parameters

- **VoltageScaling:** : Specifies the regulator output voltage to achieve a tradeoff between performance and power consumption. This parameter can be one of the following values :
  - PWR_REGULATOR_VOLTAGE_SCALE0 : Regulator voltage output Scale 0 mode.
  - PWR_REGULATOR_VOLTAGE_SCALE1 : Regulator voltage output range 1 mode.
  - PWR_REGULATOR_VOLTAGE_SCALE2 : Regulator voltage output range 2 mode.
  - PWR_REGULATOR_VOLTAGE_SCALE3 : Regulator voltage output range 3 mode.

Return values

- **HAL:** Status

Notes

- For STM32H74x and STM32H75x lines, configuring Voltage Scale 0 is only possible when Vcore is supplied from LDO (Low DropOut). The SYSCFG Clock must be enabled through __HAL_RCC_SYSCFG_CLK_ENABLE() macro before configuring Voltage Scale 0. To enter low power mode , and if current regulator voltage is Voltage Scale 0 then first switch to Voltage Scale 1 before entering low power mode.

HAL_PWREx_GetVoltageRange

Function name

uint32_t HAL_PWREx_GetVoltageRange (void)

Function description

Get the main internal regulator output voltage.

Return values

- **The:** current applied VOS selection.

HAL_PWREx_ControlStopModeVoltageScaling

Function name

HAL_StatusTypeDef HAL_PWREx_ControlStopModeVoltageScaling (uint32_t VoltageScaling)

Function description

Configure the main internal regulator output voltage in STOP mode.

Parameters

- **VoltageScaling:** : Specifies the regulator output voltage when the system enters Stop mode to achieve a tradeoff between performance and power consumption. This parameter can be one of the following values:
  - PWR_REGULATOR_SVOS_SCALE3 : Regulator voltage output range 3 mode.
  - PWR_REGULATOR_SVOS_SCALE4 : Regulator voltage output range 4 mode.
  - PWR_REGULATOR_SVOS_SCALE5 : Regulator voltage output range 5 mode.

Return values

- **HAL:** Status.

Notes

- The Stop mode voltage scaling for SVOS4 and SVOS5 sets the voltage regulator in Low-power (LP) mode to further reduce power consumption. When preselecting SVOS3, the use of the voltage regulator low-power mode (LP) can be selected by LPDS register bit.
- The selected SVOS4 and SVOS5 levels add an additional startup delay when exiting from system Stop mode.

HAL_PWREx_GetStopModeVoltageRange

Function name

uint32_t HAL_PWREx_GetStopModeVoltageRange (void)

Function description

Get the main internal regulator output voltage in STOP mode.

Return values

- **The:** actual applied VOS selection.

HAL_PWREx_EnterSTOPMode

Function name

void HAL_PWREx_EnterSTOPMode (uint32_t Regulator, uint8_t STOPEntry, uint32_t Domain)

Function description

Enter a Domain to DSTOP mode.

Parameters

- **Regulator:** : Specifies the regulator state in STOP mode. This parameter can be one of the following values:
  - PWR_MAINREGULATOR_ON : STOP mode with regulator ON.
  - PWR_LOWPOWERREGULATOR_ON : STOP mode with low power regulator ON.
- **STOPEntry:** : Specifies if STOP mode in entered with WFI or WFE intrinsic instruction. This parameter can be one of the following values:
  - PWR_STOPENTRY_WFI : Enter STOP mode with WFI instruction.
  - PWR_STOPENTRY_WFE : Enter STOP mode with WFE instruction.
- **Domain:** : Specifies the Domain to enter in DSTOP mode. This parameter can be one of the following values:
  - PWR_D1_DOMAIN : Enter D1/CD Domain to DSTOP mode.
  - PWR_D2_DOMAIN : Enter D2 Domain to DSTOP mode.
  - PWR_D3_DOMAIN : Enter D3/SRD Domain to DSTOP mode.

Return values

- **None.:**

Notes

- This API gives flexibility to manage independently each domain STOP mode. For dual core lines, this API should be executed with the corresponding Cortex-Mx to enter domain to DSTOP mode. When it is executed by all available Cortex-Mx, the system enter to STOP mode. For single core lines, calling this API with domain parameter set to PWR_D1_DOMAIN (D1/CD), the whole system will enter in STOP mode independently of PWR_CPUCR_PDDS_Dx bits values if RUN_D3 bit in the CPUCR_RUN_D3 is cleared.
- In DStop mode the domain bus matrix clock is stopped.
- The system D3/SRD domain enter Stop mode only when the CPU subsystem is in CStop mode, the EXTI wakeup sources are inactive and at least one PDDS_Dn bit in PWR CPU control register (PWR_CPUCR) for any domain request Stop.
- Before entering DSTOP mode it is recommended to call SCB_CleanDCache function in order to clean the D-Cache and guarantee the data integrity for the SRAM memories.
- In System Stop mode, the domain peripherals that use the LSI or LSE clock, and the peripherals that have a kernel clock request to select HSI or CSI as source, are still able to operate.

HAL_PWREx_EnterSTANDBYMode

Function name

void HAL_PWREx_EnterSTANDBYMode (uint32_t Domain)

Function description

Enter a Domain to DSTANDBY mode.

Parameters

- **Domain:** : Specifies the Domain to enter to STANDBY mode. This parameter can be one of the following values:
  - PWR_D1_DOMAIN: Enter D1/CD Domain to DSTANDBY mode.
  - PWR_D2_DOMAIN: Enter D2 Domain to DSTANDBY mode.
  - PWR_D3_DOMAIN: Enter D3/SRD Domain to DSTANDBY mode.

Return values

- **None:**

Notes

- This API gives flexibility to manage independently each domain STANDBY mode. For dual core lines, this API should be executed with the corresponding Cortex-Mx to enter domain to DSTANDBY mode. When it is executed by all available Cortex-Mx, the system enter STANDBY mode. For single core lines, calling this API with D1/SRD the selected domain will enter the whole system in STOP if PWR_CPUCR_PDDS_D3 = 0 and enter the whole system in STANDBY if PWR_CPUCR_PDDS_D3 = 1.
- The DStandby mode is entered when all PDDS_Dn bits in PWR_CPUCR for the Dn domain select Standby mode. When the system enters Standby mode, the voltage regulator is disabled.
- When D2 or D3 domain is in DStandby mode and the CPU sets the domain PDDS_Dn bit to select Stop mode, the domain remains in DStandby mode. The domain will only exit DStandby when the CPU allocates a peripheral in the domain.
- The system D3/SRD domain enters Standby mode only when the D1 and D2 domain are in DStandby.
- Before entering DSTANDBY mode it is recommended to call SCB_CleanDCache function in order to clean the D-Cache and guarantee the data integrity for the SRAM memories.

HAL_PWREx_ConfigD3Domain

Function name

void HAL_PWREx_ConfigD3Domain (uint32_t D3State)

Function description

Configure the D3/SRD Domain state when the System in low power mode.

Parameters

- **D3State:** : Specifies the D3/SRD state. This parameter can be one of the following values :
  - PWR_D3_DOMAIN_STOP : D3/SRD domain will follow the most deep CPU sub-system low power mode.
  - PWR_D3_DOMAIN_RUN : D3/SRD domain will stay in RUN mode regardless of the CPU sub-system low power mode.

Return values

- **None:**

HAL_PWREx_ClearPendingEvent

Function name

void HAL_PWREx_ClearPendingEvent (void)

Function description

Clear pending event.

Return values

- **None.:**

Notes

- This API clears the pending event in order to enter a given CPU to CSLEEP or CSTOP. It should be called just before APIs performing enter low power mode using Wait For Event request.
- Cortex-M7 must be in CRUN mode when calling this API by Cortex-M4.

HAL_PWREx_ClearDomainFlags

Function name

void HAL_PWREx_ClearDomainFlags (uint32_t DomainFlags)

Function description

Clear HOLD2F, HOLD1F, STOPF, SBF, SBF_D1, and SBF_D2 flags for a given domain.

Parameters

- **DomainFlags:** : Specifies the Domain flags to be cleared. This parameter can be one of the following values:
  - PWR_D1_DOMAIN_FLAGS : Clear D1 Domain flags.
  - PWR_D2_DOMAIN_FLAGS : Clear D2 Domain flags.
  - PWR_ALL_DOMAIN_FLAGS : Clear D1 and D2 Domain flags.

Return values

- **None.:**

HAL_PWREx_HoldCore

Function name

HAL_StatusTypeDef HAL_PWREx_HoldCore (uint32_t CPU)

Function description

Hold the CPU and their domain peripherals when exiting STOP mode.

Parameters

- **CPU:** : Specifies the core to be held. This parameter can be one of the following values:
  - PWR_CORE_CPU1: Hold CPU1 and set CPU2 as master.
  - PWR_CORE_CPU2: Hold CPU2 and set CPU1 as master.

Return values

- **HAL:** status

HAL_PWREx_ReleaseCore

Function name

void HAL_PWREx_ReleaseCore (uint32_t CPU)

Function description

Release the CPU and their domain peripherals after a wake-up from STOP mode.

Parameters

- **CPU:** Specifies the core to be released. This parameter can be one of the following values:
  - PWR_CORE_CPU1: Release the CPU1 and their domain peripherals from holding.
  - PWR_CORE_CPU2: Release the CPU2 and their domain peripherals from holding.

Return values

- **None:**

HAL_PWREx_EnableFlashPowerDown

Function name

void HAL_PWREx_EnableFlashPowerDown (void)

Function description

Enable the Flash Power Down in Stop mode.

Return values

- **None.:**

Notes

- When Flash Power Down is enabled the Flash memory enters low-power mode when D1/SRD domain is in DStop mode. This feature allows to obtain the best trade-off between low-power consumption and restart time when exiting from DStop mode.

HAL_PWREx_DisableFlashPowerDown

Function name

void HAL_PWREx_DisableFlashPowerDown (void)

Function description

Disable the Flash Power Down in Stop mode.

Return values

- **None.:**

Notes

- When Flash Power Down is disabled the Flash memory is kept on normal mode when D1/SRD domain is in DStop mode. This feature allows to obtain the best trade-off between low-power consumption and restart time when exiting from DStop mode.

HAL_PWREx_EnableWakeUpPin

Function name

void HAL_PWREx_EnableWakeUpPin (PWREx_WakeupPinTypeDef * sPinParams)

Function description

Enable the Wake-up PINx functionality.

Parameters

- **sPinParams:** : Pointer to a PWREx_WakeupPinTypeDef structure that contains the configuration information for the wake-up Pin.

Return values

- **None.:**

Notes

- For dual core devices, please ensure to configure the EXTI lines for the different Cortex-Mx. All combination are allowed: wake up only Cortex-M7, wake up only Cortex-M4 and wake up Cortex-M7 and Cortex-M4.

HAL_PWREx_DisableWakeUpPin

Function name

void HAL_PWREx_DisableWakeUpPin (uint32_t WakeUpPin)

Function description

Disable the Wake-up PINx functionality.

Parameters

- **WakeUpPin:** : Specifies the Wake-Up pin to be disabled. This parameter can be one of the following values:
  - PWR_WAKEUP_PIN1 : Disable PA0 wake-up PIN.
  - PWR_WAKEUP_PIN2 : Disable PA2 wake-up PIN.
  - PWR_WAKEUP_PIN3 : Disable PI8 wake-up PIN.
  - PWR_WAKEUP_PIN4 : Disable PC13 wake-up PIN.
  - PWR_WAKEUP_PIN5 : Disable PI11 wake-up PIN.
  - PWR_WAKEUP_PIN6 : Disable PC1 wake-up PIN.

Return values

- **None:**

Notes

- The PWR_WAKEUP_PIN3 and PWR_WAKEUP_PIN5 are available only for devices that support GPIOI port.

HAL_PWREx_GetWakeupFlag

Function name

uint32_t HAL_PWREx_GetWakeupFlag (uint32_t WakeUpFlag)

Function description

Get the Wake-Up Pin pending flags.

Parameters

- **WakeUpFlag:** : Specifies the Wake-Up PIN flag to be checked. This parameter can be one of the following values:
  - PWR_WAKEUP_FLAG1 : Get wakeup event received from PA0.
  - PWR_WAKEUP_FLAG2 : Get wakeup event received from PA2.
  - PWR_WAKEUP_FLAG3 : Get wakeup event received from PI8.
  - PWR_WAKEUP_FLAG4 : Get wakeup event received from PC13.
  - PWR_WAKEUP_FLAG5 : Get wakeup event received from PI11.
  - PWR_WAKEUP_FLAG6 : Get wakeup event received from PC1.
  - PWR_WAKEUP_FLAG_ALL : Get Wakeup event received from all wake up pins.

Return values

- **The:** Wake-Up pin flag.

Notes

- The PWR_WAKEUP_FLAG3 and PWR_WAKEUP_FLAG5 are available only for devices that support GPIOI port.

HAL_PWREx_ClearWakeupFlag

Function name

HAL_StatusTypeDef HAL_PWREx_ClearWakeupFlag (uint32_t WakeUpFlag)

Function description

Clear the Wake-Up pin pending flag.

Parameters

- **WakeUpFlag:** Specifies the Wake-Up PIN flag to clear. This parameter can be one of the following values:
  - PWR_WAKEUP_FLAG1 : Clear the wakeup event received from PA0.
  - PWR_WAKEUP_FLAG2 : Clear the wakeup event received from PA2.
  - PWR_WAKEUP_FLAG3 : Clear the wakeup event received from PI8.
  - PWR_WAKEUP_FLAG4 : Clear the wakeup event received from PC13.
  - PWR_WAKEUP_FLAG5 : Clear the wakeup event received from PI11.
  - PWR_WAKEUP_FLAG6 : Clear the wakeup event received from PC1.
  - PWR_WAKEUP_FLAG_ALL : Clear the wakeup events received from all wake up pins.

Return values

- **HAL:** status.

Notes

- The PWR_WAKEUP_FLAG3 and PWR_WAKEUP_FLAG5 are available only for devices that support GPIOI port.

HAL_PWREx_WAKEUP_PIN_IRQHandler

Function name

void HAL_PWREx_WAKEUP_PIN_IRQHandler (void)

Function description

This function handles the PWR WAKEUP PIN interrupt request.

Return values

- **None.:**

Notes

- This API should be called under the WAKEUP_PIN_IRQHandler().

HAL_PWREx_WKUP1_Callback

Function name

void HAL_PWREx_WKUP1_Callback (void)

Function description

PWR WKUP1 interrupt callback.

Return values

- **None.:**

HAL_PWREx_WKUP2_Callback

Function name

void HAL_PWREx_WKUP2_Callback (void)

Function description

PWR WKUP2 interrupt callback.

Return values

- **None.:**

HAL_PWREx_WKUP3_Callback

Function name

void HAL_PWREx_WKUP3_Callback (void)

Function description

PWR WKUP3 interrupt callback.

Return values

- **None.:**

HAL_PWREx_WKUP4_Callback

Function name

void HAL_PWREx_WKUP4_Callback (void)

Function description

PWR WKUP4 interrupt callback.

Return values

- **None.:**

HAL_PWREx_WKUP5_Callback

Function name

void HAL_PWREx_WKUP5_Callback (void)

Function description

PWR WKUP5 interrupt callback.

Return values

- **None.:**

HAL_PWREx_WKUP6_Callback

Function name

void HAL_PWREx_WKUP6_Callback (void)

Function description

PWR WKUP6 interrupt callback.

Return values

- **None.:**

HAL_PWREx_EnableBkUpReg

Function name

HAL_StatusTypeDef HAL_PWREx_EnableBkUpReg (void)

Function description

Enable the Backup Regulator.

Return values

- **HAL:** status.

HAL_PWREx_DisableBkUpReg

Function name

HAL_StatusTypeDef HAL_PWREx_DisableBkUpReg (void)

Function description

Disable the Backup Regulator.

Return values

- **HAL:** status.

HAL_PWREx_EnableUSBReg

Function name

HAL_StatusTypeDef HAL_PWREx_EnableUSBReg (void)

Function description

Enable the USB Regulator.

Return values

- **HAL:** status.

HAL_PWREx_DisableUSBReg

Function name

HAL_StatusTypeDef HAL_PWREx_DisableUSBReg (void)

Function description

Disable the USB Regulator.

Return values

- **HAL:** status.

HAL_PWREx_EnableUSBVoltageDetector

Function name

void HAL_PWREx_EnableUSBVoltageDetector (void)

Function description

Enable the USB voltage level detector.

Return values

- **None.:**

HAL_PWREx_DisableUSBVoltageDetector

Function name

void HAL_PWREx_DisableUSBVoltageDetector (void)

Function description

Disable the USB voltage level detector.

Return values

- **None.:**

HAL_PWREx_EnableBatteryCharging

Function name

void HAL_PWREx_EnableBatteryCharging (uint32_t ResistorValue)

Function description

Enable the Battery charging.

Parameters

- **ResistorValue:** : Specifies the charging resistor. This parameter can be one of the following values :
  - PWR_BATTERY_CHARGING_RESISTOR_5 : 5 KOhm resistor.
  - PWR_BATTERY_CHARGING_RESISTOR_1_5 : 1.5 KOhm resistor.

Return values

- **None.:**

Notes

- When VDD is present, charge the external battery through an internal resistor.

HAL_PWREx_DisableBatteryCharging

Function name

void HAL_PWREx_DisableBatteryCharging (void)

Function description

Disable the Battery charging.

Return values

- **None.:**

HAL_PWREx_EnableMonitoring

Function name

void HAL_PWREx_EnableMonitoring (void)

Function description

Enable the VBAT and temperature monitoring.

Return values

- **HAL:** status.

HAL_PWREx_DisableMonitoring

Function name

void HAL_PWREx_DisableMonitoring (void)

Function description

Disable the VBAT and temperature monitoring.

Return values

- **HAL:** status.

HAL_PWREx_GetTemperatureLevel

Function name

uint32_t HAL_PWREx_GetTemperatureLevel (void)

Function description

Indicate whether the junction temperature is between, above or below the thresholds.

Return values

- **Temperature:** level.

HAL_PWREx_GetVBATLevel

Function name

uint32_t HAL_PWREx_GetVBATLevel (void)

Function description

Indicate whether the Battery voltage level is between, above or below the thresholds.

Return values

- **VBAT:** level.

HAL_PWREx_ConfigAVD

Function name

void HAL_PWREx_ConfigAVD (PWREx_AVDTypeDef * sConfigAVD)

Function description

Configure the event mode and the voltage threshold detected by the Analog Voltage Detector (AVD).

Parameters

- **sConfigAVD:** : Pointer to an PWREx_AVDTypeDef structure that contains the configuration information for the AVD.

Return values

- **None.:**

Notes

- Refer to the electrical characteristics of your device datasheet for more details about the voltage threshold corresponding to each detection level.
- For dual core devices, please ensure to configure the EXTI lines for the different Cortex-Mx through PWR_Exported_Macro provided by this driver. All combination are allowed: wake up only Cortex-M7, wake up only Cortex-M4 and wake up Cortex-M7 and Cortex-M4.

HAL_PWREx_EnableAVD

Function name

void HAL_PWREx_EnableAVD (void)

Function description

Enable the Analog Voltage Detector (AVD).

Return values

- **None.:**

HAL_PWREx_DisableAVD

Function name

void HAL_PWREx_DisableAVD (void)

Function description

Disable the Analog Voltage Detector(AVD).

Return values

- **None.:**

HAL_PWREx_PVD_AVD_IRQHandler

Function name

void HAL_PWREx_PVD_AVD_IRQHandler (void)

Function description

This function handles the PWR PVD/AVD interrupt request.

Return values

- **None:**

Notes

- This API should be called under the PVD_AVD_IRQHandler().

HAL_PWREx_AVDCallback

Function name

void HAL_PWREx_AVDCallback (void)

Function description

PWR AVD interrupt callback.

Return values

- **None.:**

65.3 PWREx Firmware driver defines

The following section lists the various define and macros of the module.

65.3.1 PWREx

PWREx

PWREx AVD detection level

PWR_AVDLEVEL_0

Analog voltage detector level 0 selection : 1V7

PWR_AVDLEVEL_1

Analog voltage detector level 1 selection : 2V1

PWR_AVDLEVEL_2

Analog voltage detector level 2 selection : 2V5

PWR_AVDLEVEL_3

Analog voltage detector level 3 selection : 2V8

PWREx AVD EXTI Line 16

PWR_EXTI_LINE_AVD

External interrupt line 16 Connected to the AVD EXTI Line

PWREx AVD Mode

PWR_AVD_MODE_NORMAL

Basic mode is used

PWR_AVD_MODE_IT_RISING

External Interrupt Mode with Rising edge trigger detection

PWR_AVD_MODE_IT_FALLING

External Interrupt Mode with Falling edge trigger detection

PWR_AVD_MODE_IT_RISING_FALLING

External Interrupt Mode with Rising/Falling edge trigger detection

PWR_AVD_MODE_EVENT_RISING

Event Mode with Rising edge trigger detection

PWR_AVD_MODE_EVENT_FALLING

Event Mode with Falling edge trigger detection

PWR_AVD_MODE_EVENT_RISING_FALLING

Event Mode with Rising/Falling edge trigger detection

PWR Extended AVD Mode Mask

AVD_MODE_IT

AVD_MODE_EVT

AVD_RISING_EDGE

AVD_FALLING_EDGE

AVD_RISING_FALLING_EDGE

PWREx Core definition

PWR_CORE_CPU1

PWR_CORE_CPU2

PWREx D3 Domain State

PWR_D3_DOMAIN_STOP

PWR_D3_DOMAIN_RUN

PWREx Domains definition

PWR_D1_DOMAIN

PWR_D2_DOMAIN

PWR_D3_DOMAIN

PWREx Domain Flags definition

PWR_D1_DOMAIN_FLAGS

PWR_D2_DOMAIN_FLAGS

PWR_ALL_DOMAIN_FLAGS

PWREx Exported Macro

__HAL_PWR_AVD_EXTI_ENABLE_IT

Description:

- Enable the AVD EXTI Line 16.

Return value:

- None.

__HAL_PWR_AVD_EXTID2_ENABLE_IT

Description:

- Enable the AVD EXTI D2 Line 16.

Return value:

- None.

__HAL_PWR_AVD_EXTI_DISABLE_IT

Description:

- Disable the AVD EXTI Line 16.

Return value:

- None.

__HAL_PWR_AVD_EXTID2_DISABLE_IT

Description:

- Disable the AVD EXTI D2 Line 16.

Return value:

- None.

__HAL_PWR_AVD_EXTI_ENABLE_EVENT

Description:

- Enable event on AVD EXTI Line 16.

Return value:

- None.

__HAL_PWR_AVD_EXTID2_ENABLE_EVENT

Description:

- Enable event on AVD EXTI D2 Line 16.

Return value:

- None.

__HAL_PWR_AVD_EXTI_DISABLE_EVENT

Description:

- Disable event on AVD EXTI Line 16.

Return value:

- None.

__HAL_PWR_AVD_EXTID2_DISABLE_EVENT

Description:

- Disable event on AVD EXTI D2 Line 16.

Return value:

- None.

__HAL_PWR_AVD_EXTI_ENABLE_RISING_EDGE

Description:

- Enable the AVD Extended Interrupt Rising Trigger.

Return value:

- None.

__HAL_PWR_AVD_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the AVD Extended Interrupt Rising Trigger.

Return value:

- None.

__HAL_PWR_AVD_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the AVD Extended Interrupt Falling Trigger.

Return value:

- None.

__HAL_PWR_AVD_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable the AVD Extended Interrupt Falling Trigger.

Return value:

- None.

__HAL_PWR_AVD_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable the AVD Extended Interrupt Rising and Falling Trigger.

Return value:

- None.

__HAL_PWR_AVD_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable the AVD Extended Interrupt Rising & Falling Trigger.

Return value:

- None.

__HAL_PWR_AVD_EXTI_GET_FLAG

Description:

- Check whether the specified AVD EXTI interrupt flag is set or not.

Return value:

- EXTI: AVD Line Status.

__HAL_PWR_AVD_EXTID2_GET_FLAG

Description:

- Check whether the specified AVD EXTI D2 interrupt flag is set or not.

Return value:

- EXTI: D2 AVD Line Status.

__HAL_PWR_AVD_EXTI_CLEAR_FLAG

Description:

- Clear the AVD EXTI flag.

Return value:

- None.

__HAL_PWR_AVD_EXTID2_CLEAR_FLAG

Description:

- Clear the AVD EXTI D2 flag.

Return value:

- None.

__HAL_PWR_AVD_EXTI_GENERATE_SWIT

Description:

- Generates a Software interrupt on AVD EXTI line.

Return value:

- None.

PWREx Private macros to check input parameters

IS_PWR_SUPPLY

IS_PWR_STOP_MODE_REGULATOR_VOLTAGE

IS_PWR_DOMAIN

IS_D3_STATE

IS_PWR_WAKEUP_PIN

IS_PWR_WAKEUP_PIN_POLARITY

IS_PWR_WAKEUP_PIN_PULL

IS_PWR_WAKEUP_FLAG

IS_PWR_AVD_LEVEL

IS_PWR_AVD_MODE

IS_PWR_BATTERY_RESISTOR_SELECT

IS_PWR_D1_CPU

IS_PWR_CORE

IS_PWR_D2_CPU

IS_PWR_DOMAIN_FLAG

PWREx Pin Polarity configuration

PWR_PIN_POLARITY_HIGH

PWR_PIN_POLARITY_LOW

PWREx Pin Pull configuration

PWR_PIN_NO_PULL

PWR_PIN_PULL_UP

PWR_PIN_PULL_DOWN

PWREx Regulator Voltage Scale

PWR_REGULATOR_SVOS_SCALE5

PWR_REGULATOR_SVOS_SCALE4

PWR_REGULATOR_SVOS_SCALE3

PWR Extended Flag Setting Time Out Value

PWR_FLAG_SETTING_DELAY

PWREx Supply configuration

PWR_LDO_SUPPLY

Core domains are supplied from the LDO

PWR_DIRECT_SMPS_SUPPLY

Core domains are supplied from the SMPS only

PWR_SMPS_1V8_SUPPLIES_LDO

The SMPS 1.8V output supplies the LDO which supplies the Core domains

PWR_SMPS_2V5_SUPPLIES_LDO

The SMPS 2.5V output supplies the LDO which supplies the Core domains

PWR_SMPS_1V8_SUPPLIES_EXT_AND_LDO

The SMPS 1.8V output supplies an external circuits and the LDO. The Core domains are supplied from the LDO

PWR_SMPS_2V5_SUPPLIES_EXT_AND_LDO

The SMPS 2.5V output supplies an external circuits and the LDO. The Core domains are supplied from the LDO

PWR_SMPS_1V8_SUPPLIES_EXT

The SMPS 1.8V output supplies an external source which supplies the Core domains

PWR_SMPS_2V5_SUPPLIES_EXT

The SMPS 2.5V output supplies an external source which supplies the Core domains

PWR_EXTERNAL_SOURCE_SUPPLY

The SMPS disabled and the LDO Bypass. The Core domains are supplied from an external source

PWR_SUPPLY_CONFIG_MASK

PWREx Temperature Thresholds

PWR_TEMP_BETWEEN_HIGH_LOW_THRESHOLD

PWR_TEMP_BELOW_LOW_THRESHOLD

PWR_TEMP_ABOVE_HIGH_THRESHOLD

PWR battery charging resistor selection

PWR_BATTERY_CHARGING_RESISTOR_5

VBAT charging through a 5 kOhms resistor

PWR_BATTERY_CHARGING_RESISTOR_1_5

VBAT charging through a 1.5 kOhms resistor

PWREx VBAT Thresholds

PWR_VBAT_BETWEEN_HIGH_LOW_THRESHOLD

PWR_VBAT_BELOW_LOW_THRESHOLD

PWR_VBAT_ABOVE_HIGH_THRESHOLD

PWREx Wake-Up Pins

PWR_WAKEUP_PIN6

PWR_WAKEUP_PIN5

PWR_WAKEUP_PIN4

PWR_WAKEUP_PIN3

PWR_WAKEUP_PIN2

PWR_WAKEUP_PIN1

PWR_WAKEUP_PIN6_HIGH

PWR_WAKEUP_PIN5_HIGH

PWR_WAKEUP_PIN4_HIGH

PWR_WAKEUP_PIN3_HIGH

PWR_WAKEUP_PIN2_HIGH

PWR_WAKEUP_PIN1_HIGH

PWR_WAKEUP_PIN6_LOW

PWR_WAKEUP_PIN5_LOW

PWR_WAKEUP_PIN4_LOW

PWR_WAKEUP_PIN3_LOW

PWR_WAKEUP_PIN2_LOW

PWR_WAKEUP_PIN1_LOW

PWREx Wakeup Pins Flags.

PWR_WAKEUP_FLAG1

Wakeup flag on PA0

PWR_WAKEUP_FLAG2

Wakeup flag on PA2

PWR_WAKEUP_FLAG3

Wakeup flag on PI8

PWR_WAKEUP_FLAG4

Wakeup flag on PC13

PWR_WAKEUP_FLAG5

Wakeup flag on PI11

PWR_WAKEUP_FLAG6

Wakeup flag on PC1

PWR_WAKEUP_FLAG_ALL

PWREx Wake-Up Pins masks and offsets

PWR_EXTI_WAKEUP_PINS_MASK

PWR_WAKEUP_PINS_PULL_SHIFT_OFFSET

66 HAL QSPI Generic Driver

66.1 QSPI Firmware driver registers structures

66.1.1 QSPI_InitTypeDef

QSPI_InitTypeDef is defined in the stm32h7xx_hal_qspi.h

Data Fields

- ***uint32_t ClockPrescaler***
- ***uint32_t FifoThreshold***
- ***uint32_t SampleShifting***
- ***uint32_t FlashSize***
- ***uint32_t ChipSelectHighTime***
- ***uint32_t ClockMode***
- ***uint32_t FlashID***
- ***uint32_t DualFlash***

Field Documentation

- ***uint32_t QSPI_InitTypeDef::ClockPrescaler***
- ***uint32_t QSPI_InitTypeDef::FifoThreshold***
- ***uint32_t QSPI_InitTypeDef::SampleShifting***
- ***uint32_t QSPI_InitTypeDef::FlashSize***
- ***uint32_t QSPI_InitTypeDef::ChipSelectHighTime***
- ***uint32_t QSPI_InitTypeDef::ClockMode***
- ***uint32_t QSPI_InitTypeDef::FlashID***
- ***uint32_t QSPI_InitTypeDef::DualFlash***

66.1.2 __QSPI_HandleTypeDef

__QSPI_HandleTypeDef is defined in the stm32h7xx_hal_qspi.h

Data Fields

- ***QUADSPI_TypeDef * Instance***
- ***QSPI_InitTypeDef Init***
- ***uint8_t * pTxBuffPtr***
- ***__IO uint32_t TxXferSize***
- ***__IO uint32_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***__IO uint32_t RxXferSize***
- ***__IO uint32_t RxXferCount***
- ***MDMA_HandleTypeDef * hmdma***
- ***__IO HAL_LockTypeDef Lock***
- ***__IO HAL_QSPI_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***uint32_t Timeout***
- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* FifoThresholdCallback***
- ***void(* CmdCpltCallback***

- *void(* RxCpltCallback*
- *void(* TxCpltCallback*
- *void(* StatusMatchCallback*
- *void(* TimeOutCallback*
- *void(* MspInitCallback*
- *void(* MspDeInitCallback*

Field Documentation

- *QUADSPI_TypeDef* __QSPI_HandleTypeDef::Instance*
- *QSPI_InitTypeDef __QSPI_HandleTypeDef::Init*
- *uint8_t* __QSPI_HandleTypeDef::pTxBuffPtr*
- *__IO uint32_t __QSPI_HandleTypeDef::TxXferSize*
- *__IO uint32_t __QSPI_HandleTypeDef::TxXferCount*
- *uint8_t* __QSPI_HandleTypeDef::pRxBuffPtr*
- *__IO uint32_t __QSPI_HandleTypeDef::RxXferSize*
- *__IO uint32_t __QSPI_HandleTypeDef::RxXferCount*
- *MDMA_HandleTypeDef* __QSPI_HandleTypeDef::hmdma*
- *__IO HAL_LockTypeDef __QSPI_HandleTypeDef::Lock*
- *__IO HAL_QSPI_StateTypeDef __QSPI_HandleTypeDef::State*
- *__IO uint32_t __QSPI_HandleTypeDef::ErrorCode*
- *uint32_t __QSPI_HandleTypeDef::Timeout*
- *void(* __QSPI_HandleTypeDef::ErrorCallback)(struct __QSPI_HandleTypeDef *hqspi)*
- *void(* __QSPI_HandleTypeDef::AbortCpltCallback)(struct __QSPI_HandleTypeDef *hqspi)*
- *void(* __QSPI_HandleTypeDef::FifoThresholdCallback)(struct __QSPI_HandleTypeDef *hqspi)*
- *void(* __QSPI_HandleTypeDef::CmdCpltCallback)(struct __QSPI_HandleTypeDef *hqspi)*
- *void(* __QSPI_HandleTypeDef::RxCpltCallback)(struct __QSPI_HandleTypeDef *hqspi)*
- *void(* __QSPI_HandleTypeDef::TxCpltCallback)(struct __QSPI_HandleTypeDef *hqspi)*
- *void(* __QSPI_HandleTypeDef::StatusMatchCallback)(struct __QSPI_HandleTypeDef *hqspi)*
- *void(* __QSPI_HandleTypeDef::TimeOutCallback)(struct __QSPI_HandleTypeDef *hqspi)*
- *void(* __QSPI_HandleTypeDef::MspInitCallback)(struct __QSPI_HandleTypeDef *hqspi)*
- *void(* __QSPI_HandleTypeDef::MspDeInitCallback)(struct __QSPI_HandleTypeDef *hqspi)*

66.1.3 QSPI_CommandTypeDef

QSPI_CommandTypeDef is defined in the stm32h7xx_hal_qspi.h

Data Fields

- *uint32_t Instruction*
- *uint32_t Address*
- *uint32_t AlternateBytes*
- *uint32_t AddressSize*
- *uint32_t AlternateBytesSize*
- *uint32_t DummyCycles*
- *uint32_t InstructionMode*
- *uint32_t AddressMode*
- *uint32_t AlternateByteMode*
- *uint32_t DataMode*
- *uint32_t NbData*

- *uint32_t DdrMode*
- *uint32_t DdrHoldHalfCycle*
- *uint32_t SIOOMode*

Field Documentation

- *uint32_t QSPI_CommandTypeDef::Instruction*
- *uint32_t QSPI_CommandTypeDef::Address*
- *uint32_t QSPI_CommandTypeDef::AlternateBytes*
- *uint32_t QSPI_CommandTypeDef::AddressSize*
- *uint32_t QSPI_CommandTypeDef::AlternateBytesSize*
- *uint32_t QSPI_CommandTypeDef::DummyCycles*
- *uint32_t QSPI_CommandTypeDef::InstructionMode*
- *uint32_t QSPI_CommandTypeDef::AddressMode*
- *uint32_t QSPI_CommandTypeDef::AlternateByteMode*
- *uint32_t QSPI_CommandTypeDef::DataMode*
- *uint32_t QSPI_CommandTypeDef::NbData*
- *uint32_t QSPI_CommandTypeDef::DdrMode*
- *uint32_t QSPI_CommandTypeDef::DdrHoldHalfCycle*
- *uint32_t QSPI_CommandTypeDef::SIOOMode*

66.1.4 QSPI_AutoPollingTypeDef

QSPI_AutoPollingTypeDef is defined in the stm32h7xx_hal_qspi.h

Data Fields

- *uint32_t Match*
- *uint32_t Mask*
- *uint32_t Interval*
- *uint32_t StatusBytesSize*
- *uint32_t MatchMode*
- *uint32_t AutomaticStop*

Field Documentation

- *uint32_t QSPI_AutoPollingTypeDef::Match*
- *uint32_t QSPI_AutoPollingTypeDef::Mask*
- *uint32_t QSPI_AutoPollingTypeDef::Interval*
- *uint32_t QSPI_AutoPollingTypeDef::StatusBytesSize*
- *uint32_t QSPI_AutoPollingTypeDef::MatchMode*
- *uint32_t QSPI_AutoPollingTypeDef::AutomaticStop*

66.1.5 QSPI_MemoryMappedTypeDef

QSPI_MemoryMappedTypeDef is defined in the stm32h7xx_hal_qspi.h

Data Fields

- *uint32_t TimeOutPeriod*
- *uint32_t TimeOutActivation*

Field Documentation

- *uint32_t QSPI_MemoryMappedTypeDef::TimeOutPeriod*
- *uint32_t QSPI_MemoryMappedTypeDef::TimeOutActivation*

66.2 QSPI Firmware driver API description

The following section lists the various functions of the QSPI library.

66.2.1 How to use this driver

Initialization

1. As prerequisite, fill in the HAL_QSPI_MspInit() :
   - Enable QuadSPI clock interface with __HAL_RCC_QSPI_CLK_ENABLE().
   - Reset QuadSPI Peripheral with __HAL_RCC_QSPI_FORCE_RESET() and __HAL_RCC_QSPI_RELEASE_RESET().
   - Enable the clocks for the QuadSPI GPIOS with __HAL_RCC_GPIOx_CLK_ENABLE().
   - Configure these QuadSPI pins in alternate mode using HAL_GPIO_Init().
   - If interrupt mode is used, enable and configure QuadSPI global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().
   - If DMA mode is used, enable the clocks for the QuadSPI MDMA with __HAL_RCC_MDMA_CLK_ENABLE(), configure MDMA with HAL_MDMA_Init(), link it with QuadSPI handle using __HAL_LINKDMA(), enable and configure MDMA global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().
2. Configure the flash size, the clock prescaler, the fifo threshold, the clock mode, the sample shifting and the CS high time using the HAL_QSPI_Init() function.

Indirect functional mode

1. Configure the command sequence using the HAL_QSPI_Command() or HAL_QSPI_Command_IT() functions :
   - Instruction phase : the mode used and if present the instruction opcode.
   - Address phase : the mode used and if present the size and the address value.
   - Alternate-bytes phase : the mode used and if present the size and the alternate bytes values.
   - Dummy-cycles phase : the number of dummy cycles (mode used is same as data phase).
   - Data phase : the mode used and if present the number of bytes.
   - Double Data Rate (DDR) mode : the activation (or not) of this mode and the delay if activated.
   - Sending Instruction Only Once (SIOO) mode : the activation (or not) of this mode.
2. If no data is required for the command, it is sent directly to the memory :
   - In polling mode, the output of the function is done when the transfer is complete.
   - In interrupt mode, HAL_QSPI_CmdCpltCallback() will be called when the transfer is complete.
3. For the indirect write mode, use HAL_QSPI_Transmit(), HAL_QSPI_Transmit_DMA() or HAL_QSPI_Transmit_IT() after the command configuration :
   - In polling mode, the output of the function is done when the transfer is complete.
   - In interrupt mode, HAL_QSPI_FifoThresholdCallback() will be called when the fifo threshold is reached and HAL_QSPI_TxCpltCallback() will be called when the transfer is complete.
   - In DMA mode,HAL_QSPI_TxCpltCallback() will be called when the transfer is complete.
4. For the indirect read mode, use HAL_QSPI_Receive(), HAL_QSPI_Receive_DMA() or HAL_QSPI_Receive_IT() after the command configuration :
   - In polling mode, the output of the function is done when the transfer is complete.
   - In interrupt mode, HAL_QSPI_FifoThresholdCallback() will be called when the fifo threshold is reached and HAL_QSPI_RxCpltCallback() will be called when the transfer is complete.
   - In DMA mode,HAL_QSPI_RxCpltCallback() will be called when the transfer is complete.

Auto-polling functional mode

1. Configure the command sequence and the auto-polling functional mode using the HAL_QSPI_AutoPolling() or HAL_QSPI_AutoPolling_IT() functions :
   - Instruction phase : the mode used and if present the instruction opcode.
   - Address phase : the mode used and if present the size and the address value.
   - Alternate-bytes phase : the mode used and if present the size and the alternate bytes values.
   - Dummy-cycles phase : the number of dummy cycles (mode used is same as data phase).
   - Data phase : the mode used.
   - Double Data Rate (DDR) mode : the activation (or not) of this mode and the delay if activated.
   - Sending Instruction Only Once (SIOO) mode : the activation (or not) of this mode.
   - The size of the status bytes, the match value, the mask used, the match mode (OR/AND), the polling interval and the automatic stop activation.
2. After the configuration :
   - In polling mode, the output of the function is done when the status match is reached. The automatic stop is activated to avoid an infinite loop.
   - In interrupt mode, HAL_QSPI_StatusMatchCallback() will be called each time the status match is reached.

MDMA functional mode

1. Configure the SourceInc and DestinationInc of MDMA parameters in the HAL_QSPI_MspInit() function :
   - MDMA settings for write operation : (+) The DestinationInc should be MDMA_DEST_INC_DISABLE (+) The SourceInc must be a value of MDMA_Source_increment_mode (Except the MDMA_SRC_INC_DOUBLEWORD). (+) The SourceDataSize must be a value of MDMA Source data size (Except the MDMA_SRC_DATASIZE_DOUBLEWORD) aligned with MDMA_Source_increment_mode . (+) The DestDataSize must be a value of MDMA Destination data size (Except the MDMA_DEST_DATASIZE_DOUBLEWORD)
   - MDMA settings for read operation : (+) The SourceInc should be MDMA_SRC_INC_DISABLE (+) The DestinationInc must be a value of MDMA_Destination_increment_mode (Except the MDMA_DEST_INC_DOUBLEWORD). (+) The SourceDataSize must be a value of MDMA Source data size (Except the MDMA_SRC_DATASIZE_DOUBLEWORD) . (+) The DestDataSize must be a value of MDMA Destination data size (Except the MDMA_DEST_DATASIZE_DOUBLEWORD) aligned with MDMA_Destination_increment_mode.
   - The buffer Transfer Length (BufferTransferLength) = number of bytes in the FIFO (FifoThreshold) of the Quadspi.
2. In case of wrong MDMA setting
   - For write operation : (+) If the DestinationInc is different to MDMA_DEST_INC_DISABLE , it will be disabled by the HAL_QSPI_Transmit_DMA().
   - For read operation : (+) If the SourceInc is not set to MDMA_SRC_INC_DISABLE , it will be disabled by the HAL_QSPI_Receive_DMA().

(#) Configure the SourceInc and DestinationInc of MDMA parameters in the HAL_QSPI_MspInit() function : (++) MDMA settings for write operation :

- The DestinationInc should be MDMA_DEST_INC_DISABLE
- The SourceInc must be a value of MDMA_Source_increment_mode (Except the MDMA_SRC_INC_DOUBLEWORD).
- The SourceDataSize must be a value of MDMA Source data size (Except the MDMA_SRC_DATASIZE_DOUBLEWORD) aligned with MDMA_Source_increment_mode .
- The DestDataSize must be a value of MDMA Destination data size (Except the MDMA_DEST_DATASIZE_DOUBLEWORD)
  - MDMA settings for read operation :
- The SourceInc should be MDMA_SRC_INC_DISABLE
- The DestinationInc must be a value of MDMA_Destination_increment_mode (Except the MDMA_DEST_INC_DOUBLEWORD).

- The SourceDataSize must be a value of MDMA Source data size (Except the MDMA_SRC_DATASIZE_DOUBLEWORD) .
- The DestDataSize must be a value of MDMA Destination data size (Except the MDMA_DEST_DATASIZE_DOUBLEWORD) aligned with MDMA_Destination_increment_mode.
  - The buffer Transfer Length (BufferTransferLength) = number of bytes in the FIFO (FifoThreshold) of the Quadspi. (#)In case of wrong MDMA setting
  - For write operation :
- If the DestinationInc is different to MDMA_DEST_INC_DISABLE , it will be disabled by the HAL_QSPI_Transmit_DMA().
  - For read operation :
- If the SourceInc is not set to MDMA_SRC_INC_DISABLE , it will be disabled by the HAL_QSPI_Receive_DMA().

Memory-mapped functional mode

1. Configure the command sequence and the memory-mapped functional mode using the HAL_QSPI_MemoryMapped() functions :
   - Instruction phase : the mode used and if present the instruction opcode.
   - Address phase : the mode used and the size.
   - Alternate-bytes phase : the mode used and if present the size and the alternate bytes values.
   - Dummy-cycles phase : the number of dummy cycles (mode used is same as data phase).
   - Data phase : the mode used.
   - Double Data Rate (DDR) mode : the activation (or not) of this mode and the delay if activated.
   - Sending Instruction Only Once (SIOO) mode : the activation (or not) of this mode.
   - The timeout activation and the timeout period.
2. After the configuration, the QuadSPI will be used as soon as an access on the AHB is done on the address range. HAL_QSPI_TimeOutCallback() will be called when the timeout expires.

Errors management and abort functionality

1. HAL_QSPI_GetError() function gives the error raised during the last operation.
2. HAL_QSPI_Abort() and HAL_QSPI_Abort_IT() functions aborts any on-going operation and flushes the fifo :
   - In polling mode, the output of the function is done when the transfer complete bit is set and the busy bit cleared.
   - In interrupt mode, HAL_QSPI_AbortCpltCallback() will be called when the transfer complete bit is set.

Control functions

1. HAL_QSPI_GetState() function gives the current state of the HAL QuadSPI driver.
2. HAL_QSPI_SetTimeout() function configures the timeout value used in the driver.
3. HAL_QSPI_SetFifoThreshold() function configures the threshold on the Fifo of the QSPI IP.
4. HAL_QSPI_GetFifoThreshold() function gives the current of the Fifo's threshold
5. HAL_QSPI_SetFlashID() function configures the index of the flash memory to be accessed.

Callback registration

The compilation define USE_HAL_QSPI_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_QSPI_RegisterCallback() to register a user callback, it allows to register following callbacks:

- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- FifoThresholdCallback : callback when the fifo threshold is reached.
- CmdCpltCallback : callback when a command without data is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- TxCpltCallback : callback when a transmission transfer is completed.
- StatusMatchCallback : callback when a status match occurs.

- TimeOutCallback : callback when the timeout perioed expires.
- MspInitCallback : QSPI MspInit.
- MspDeInitCallback : QSPI MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. Use function HAL_QSPI_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. It allows to reset following callbacks:
- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- FifoThresholdCallback : callback when the fifo threshold is reached.
- CmdCpltCallback : callback when a command without data is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- TxCpltCallback : callback when a transmission transfer is completed.
- StatusMatchCallback : callback when a status match occurs.
- TimeOutCallback : callback when the timeout perioed expires.
- MspInitCallback : QSPI MspInit.
- MspDeInitCallback : QSPI MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the HAL_QSPI_Init and if the state is HAL_QSPI_STATE_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the HAL_QSPI_Init and HAL_QSPI_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_QSPI_Init and HAL_QSPI_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/ unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_QSPI_RegisterCallback before calling HAL_QSPI_DeInit or HAL_QSPI_Init function. When The compilation define USE_HAL_QSPI_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

Workarounds linked to Silicon Limitation

1. Workarounds Implemented inside HAL Driver
   – Extra data written in the FIFO at the end of a read transfer

66.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to :

- Initialize the QuadSPI.
- De-initialize the QuadSPI.

This section contains the following APIs:

- ***HAL_QSPI_Init()***
- ***HAL_QSPI_DeInit()***
- ***HAL_QSPI_MspInit()***
- ***HAL_QSPI_MspDeInit()***

66.2.3 IO operation functions

This subsection provides a set of functions allowing to :

- Handle the interrupts.
- Handle the command sequence.
- Transmit data in blocking, interrupt or DMA mode.
- Receive data in blocking, interrupt or DMA mode.
- Manage the auto-polling functional mode.
- Manage the memory-mapped functional mode.

This section contains the following APIs:

- *HAL_QSPI_IRQHandler()*
- *HAL_QSPI_Command()*
- *HAL_QSPI_Command_IT()*
- *HAL_QSPI_Transmit()*
- *HAL_QSPI_Receive()*
- *HAL_QSPI_Transmit_IT()*
- *HAL_QSPI_Receive_IT()*
- *HAL_QSPI_Transmit_DMA()*
- *HAL_QSPI_Receive_DMA()*
- *HAL_QSPI_AutoPolling()*
- *HAL_QSPI_AutoPolling_IT()*
- *HAL_QSPI_MemoryMapped()*
- *HAL_QSPI_ErrorCallback()*
- *HAL_QSPI_AbortCpltCallback()*
- *HAL_QSPI_CmdCpltCallback()*
- *HAL_QSPI_RxCpltCallback()*
- *HAL_QSPI_TxCpltCallback()*
- *HAL_QSPI_FifoThresholdCallback()*
- *HAL_QSPI_StatusMatchCallback()*
- *HAL_QSPI_TimeOutCallback()*
- *HAL_QSPI_RegisterCallback()*
- *HAL_QSPI_UnRegisterCallback()*

66.2.4 Peripheral Control and State functions

This subsection provides a set of functions allowing to :

- Check in run-time the state of the driver.
- Check the error code set during last operation.
- Abort any operation.

This section contains the following APIs:

- *HAL_QSPI_GetState()*
- *HAL_QSPI_GetError()*
- *HAL_QSPI_Abort()*
- *HAL_QSPI_Abort_IT()*
- *HAL_QSPI_SetTimeout()*
- *HAL_QSPI_SetFifoThreshold()*
- *HAL_QSPI_GetFifoThreshold()*
- *HAL_QSPI_SetFlashID()*

66.2.5 Detailed description of functions

HAL_QSPI_Init

Function name

HAL_StatusTypeDef HAL_QSPI_Init (QSPI_HandleTypeDef * hqspi)

Function description

Initialize the QSPI mode according to the specified parameters in the QSPI_InitTypeDef and initialize the associated handle.

Parameters

- **hqspi:** QSPI handle

Return values

- **HAL:** status

HAL_QSPI_DeInit

Function name

HAL_StatusTypeDef HAL_QSPI_DeInit (QSPI_HandleTypeDef * hqspi)

Function description

De-Initialize the QSPI peripheral.

Parameters

- **hqspi:** QSPI handle

Return values

- **HAL:** status

HAL_QSPI_MspInit

Function name

void HAL_QSPI_MspInit (QSPI_HandleTypeDef * hqspi)

Function description

Initialize the QSPI MSP.

Parameters

- **hqspi:** QSPI handle

Return values

- **None:**

HAL_QSPI_MspDeInit

Function name

void HAL_QSPI_MspDeInit (QSPI_HandleTypeDef * hqspi)

Function description

DeInitialize the QSPI MSP.

Parameters

- **hqspi:** QSPI handle

Return values

- **None:**

HAL_QSPI_IRQHandler

Function name

void HAL_QSPI_IRQHandler (QSPI_HandleTypeDef * hqspi)

Function description

Handle QSPI interrupt request.

Parameters

- **hqspi:** QSPI handle

Return values

- **None:**

HAL_QSPI_Command

Function name

HAL_StatusTypeDef HAL_QSPI_Command (QSPI_HandleTypeDef * hqspi, QSPI_CommandTypeDef * cmd, uint32_t Timeout)

Function description

Set the command configuration.

Parameters

- **hqspi:** QSPI handle
- **cmd:** : structure that contains the command configuration information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read or Write Modes

HAL_QSPI_Transmit

Function name

HAL_StatusTypeDef HAL_QSPI_Transmit (QSPI_HandleTypeDef * hqspi, uint8_t * pData, uint32_t Timeout)

Function description

Transmit an amount of data in blocking mode.

Parameters

- **hqspi:** QSPI handle
- **pData:** pointer to data buffer
- **Timeout:** Timeout duration

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Write Mode

HAL_QSPI_Receive

Function name

HAL_StatusTypeDef HAL_QSPI_Receive (QSPI_HandleTypeDef * hqspi, uint8_t * pData, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **hqspi:** QSPI handle
- **pData:** pointer to data buffer
- **Timeout:** Timeout duration

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read Mode

HAL_QSPI_Command_IT

Function name

HAL_StatusTypeDef HAL_QSPI_Command_IT (QSPI_HandleTypeDef * hqspi, QSPI_CommandTypeDef * cmd)

Function description

Set the command configuration in interrupt mode.

Parameters

- **hqspi:** QSPI handle
- **cmd:** structure that contains the command configuration information

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read or Write Modes

HAL_QSPI_Transmit_IT

Function name

HAL_StatusTypeDef HAL_QSPI_Transmit_IT (QSPI_HandleTypeDef * hqspi, uint8_t * pData)

Function description

Send an amount of data in non-blocking mode with interrupt.

Parameters

- **hqspi:** QSPI handle
- **pData:** pointer to data buffer

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Write Mode

HAL_QSPI_Receive_IT

Function name

HAL_StatusTypeDef HAL_QSPI_Receive_IT (QSPI_HandleTypeDef * hqspi, uint8_t * pData)

Function description

Receive an amount of data in non-blocking mode with interrupt.

Parameters

- **hqspi:** QSPI handle
- **pData:** pointer to data buffer

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read Mode

HAL_QSPI_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_QSPI_Transmit_DMA (QSPI_HandleTypeDef * hqspi, uint8_t * pData)

Function description

Send an amount of data in non-blocking mode with DMA.

Parameters

- **hqspi:** QSPI handle
- **pData:** pointer to data buffer

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Write Mode

HAL_QSPI_Receive_DMA

Function name

HAL_StatusTypeDef HAL_QSPI_Receive_DMA (QSPI_HandleTypeDef * hqspi, uint8_t * pData)

Function description

Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hqspi:** QSPI handle
- **pData:** pointer to data buffer.

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read Mode

HAL_QSPI_AutoPolling

Function name

HAL_StatusTypeDef HAL_QSPI_AutoPolling (QSPI_HandleTypeDef * hqspi, QSPI_CommandTypeDef * cmd, QSPI_AutoPollingTypeDef * cfg, uint32_t Timeout)

Function description

Configure the QSPI Automatic Polling Mode in blocking mode.

Parameters

- **hqspi:** QSPI handle
- **cmd:** structure that contains the command configuration information.
- **cfg:** structure that contains the polling configuration information.
- **Timeout:** Timeout duration

Return values

- **HAL:** status

Notes

- This function is used only in Automatic Polling Mode

HAL_QSPI_AutoPolling_IT

Function name

HAL_StatusTypeDef HAL_QSPI_AutoPolling_IT (QSPI_HandleTypeDef * hqspi, QSPI_CommandTypeDef * cmd, QSPI_AutoPollingTypeDef * cfg)

Function description

Configure the QSPI Automatic Polling Mode in non-blocking mode.

Parameters

- **hqspi:** QSPI handle
- **cmd:** structure that contains the command configuration information.
- **cfg:** structure that contains the polling configuration information.

Return values

- **HAL:** status

Notes

- This function is used only in Automatic Polling Mode

HAL_QSPI_MemoryMapped

Function name

HAL_StatusTypeDef HAL_QSPI_MemoryMapped (QSPI_HandleTypeDef * hqspi, QSPI_CommandTypeDef * cmd, QSPI_MemoryMappedTypeDef * cfg)

Function description

Configure the Memory Mapped mode.

Parameters

- **hqspi:** QSPI handle
- **cmd:** structure that contains the command configuration information.
- **cfg:** structure that contains the memory mapped configuration information.

Return values

- **HAL:** status

Notes

- This function is used only in Memory mapped Mode

HAL_QSPI_ErrorCallback

Function name

void HAL_QSPI_ErrorCallback (QSPI_HandleTypeDef * hqspi)

Function description

Transfer Error callback.

Parameters

- **hqspi:** QSPI handle

Return values

- **None:**

HAL_QSPI_AbortCpltCallback

Function name

void HAL_QSPI_AbortCpltCallback (QSPI_HandleTypeDef * hqspi)

Function description

Abort completed callback.

Parameters

- **hqspi:** QSPI handle

Return values

- **None:**

HAL_QSPI_FifoThresholdCallback

Function name

void HAL_QSPI_FifoThresholdCallback (QSPI_HandleTypeDef * hqspi)

Function description

FIFO Threshold callback.

Parameters

- **hqspi:** QSPI handle

Return values

- **None:**

HAL_QSPI_CmdCpltCallback

Function name

void HAL_QSPI_CmdCpltCallback (QSPI_HandleTypeDef * hqspi)

Function description

Command completed callback.

Parameters

- **hqspi:** QSPI handle

Return values

- **None:**

HAL_QSPI_RxCpltCallback

Function name

void HAL_QSPI_RxCpltCallback (QSPI_HandleTypeDef * hqspi)

Function description

Rx Transfer completed callback.

Parameters

- **hqspi:** QSPI handle

Return values

- **None:**

HAL_QSPI_TxCpltCallback

Function name

void HAL_QSPI_TxCpltCallback (QSPI_HandleTypeDef * hqspi)

Function description

Tx Transfer completed callback.

Parameters

- **hqspi:** QSPI handle

Return values

- **None:**

HAL_QSPI_StatusMatchCallback

Function name

void HAL_QSPI_StatusMatchCallback (QSPI_HandleTypeDef * hqspi)

Function description

Status Match callback.

Parameters

- **hqspi:** QSPI handle

Return values

- **None:**

HAL_QSPI_TimeOutCallback

Function name

void HAL_QSPI_TimeOutCallback (QSPI_HandleTypeDef * hqspi)

Function description

Timeout callback.

Parameters

- **hqspi:** QSPI handle

Return values

- **None:**

HAL_QSPI_RegisterCallback

Function name

HAL_StatusTypeDef HAL_QSPI_RegisterCallback (QSPI_HandleTypeDef * hqspi, HAL_QSPI_CallbackIDTypeDef CallbackId, pQSPI_CallbackTypeDef pCallback)

Function description

Register a User QSPI Callback To be used instead of the weak (surcharged) predefined callback.

Parameters

- **hqspi:** QSPI handle
- **CallbackId:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_QSPI_ERROR_CB_ID QSPI Error Callback ID
  - HAL_QSPI_ABORT_CB_ID QSPI Abort Callback ID
  - HAL_QSPI_FIFO_THRESHOLD_CB_ID QSPI FIFO Threshold Callback ID
  - HAL_QSPI_CMD_CPLT_CB_ID QSPI Command Complete Callback ID
  - HAL_QSPI_RX_CPLT_CB_ID QSPI Rx Complete Callback ID
  - HAL_QSPI_TX_CPLT_CB_ID QSPI Tx Complete Callback ID
  - HAL_QSPI_STATUS_MATCH_CB_ID QSPI Status Match Callback ID
  - HAL_QSPI_TIMEOUT_CB_ID QSPI Timeout Callback ID
  - HAL_QSPI_MSP_INIT_CB_ID QSPI MspInit callback ID
  - HAL_QSPI_MSP_DEINIT_CB_ID QSPI MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **status:**

HAL_QSPI_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_QSPI_UnRegisterCallback (QSPI_HandleTypeDef * hqspi, HAL_QSPI_CallbackIDTypeDef CallbackId)

Function description

Unregister a User QSPI Callback QSPI Callback is redirected to the weak (surcharged) predefined callback.

Parameters

- **hqspi:** QSPI handle
- **CallbackId:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_QSPI_ERROR_CB_ID QSPI Error Callback ID
  - HAL_QSPI_ABORT_CB_ID QSPI Abort Callback ID
  - HAL_QSPI_FIFO_THRESHOLD_CB_ID QSPI FIFO Threshold Callback ID
  - HAL_QSPI_CMD_CPLT_CB_ID QSPI Command Complete Callback ID
  - HAL_QSPI_RX_CPLT_CB_ID QSPI Rx Complete Callback ID
  - HAL_QSPI_TX_CPLT_CB_ID QSPI Tx Complete Callback ID
  - HAL_QSPI_STATUS_MATCH_CB_ID QSPI Status Match Callback ID
  - HAL_QSPI_TIMEOUT_CB_ID QSPI Timeout Callback ID
  - HAL_QSPI_MSP_INIT_CB_ID QSPI MspInit callback ID
  - HAL_QSPI_MSP_DEINIT_CB_ID QSPI MspDeInit callback ID

Return values

- **status:**

HAL_QSPI_GetState

Function name

HAL_QSPI_StateTypeDef HAL_QSPI_GetState (QSPI_HandleTypeDef * hqspi)

Function description

Return the QSPI handle state.

Parameters

- **hqspi:** QSPI handle

Return values

- **HAL:** state

HAL_QSPI_GetError

Function name

uint32_t HAL_QSPI_GetError (QSPI_HandleTypeDef * hqspi)

Function description

Return the QSPI error code.

Parameters

- **hqspi:** QSPI handle

Return values

- **QSPI:** Error Code

HAL_QSPI_Abort

Function name

HAL_StatusTypeDef HAL_QSPI_Abort (QSPI_HandleTypeDef * hqspi)

Function description

Abort the current transmission.

Parameters

- **hqspi:** QSPI handle

Return values

- **HAL:** status

HAL_QSPI_Abort_IT

Function name

HAL_StatusTypeDef HAL_QSPI_Abort_IT (QSPI_HandleTypeDef * hqspi)

Function description

Abort the current transmission (non-blocking function)

Parameters

- **hqspi:** QSPI handle

Return values

- **HAL:** status

HAL_QSPI_SetTimeout

Function name

void HAL_QSPI_SetTimeout (QSPI_HandleTypeDef * hqspi, uint32_t Timeout)

Function description

Set QSPI timeout.

Parameters

- **hqspi:** QSPI handle.
- **Timeout:** Timeout for the QSPI memory access.

Return values

- **None:**

HAL_QSPI_SetFifoThreshold

Function name

HAL_StatusTypeDef HAL_QSPI_SetFifoThreshold (QSPI_HandleTypeDef * hqspi, uint32_t Threshold)

Function description

Set QSPI Fifo threshold.

Parameters

- **hqspi:** QSPI handle.
- **Threshold:** Threshold of the Fifo (value between 1 and 16).

Return values

- **HAL:** status

HAL_QSPI_GetFifoThreshold

Function name

uint32_t HAL_QSPI_GetFifoThreshold (QSPI_HandleTypeDef * hqspi)

Function description

Get QSPI Fifo threshold.

Parameters

- **hqspi:** QSPI handle.

Return values

- **Fifo:** threshold (value between 1 and 16)

HAL_QSPI_SetFlashID

Function name

HAL_StatusTypeDef HAL_QSPI_SetFlashID (QSPI_HandleTypeDef * hqspi, uint32_t FlashID)

Function description

Set FlashID.

Parameters

- **hqspi:** QSPI handle.
- **FlashID:** Index of the flash memory to be accessed. This parameter can be a value of QSPI Flash Select.

Return values

- **HAL:** status

Notes

- The FlashID is ignored when dual flash mode is enabled.

66.3 QSPI Firmware driver defines

The following section lists the various define and macros of the module.

66.3.1 QSPI

QSPI

QSPI Address Mode

QSPI_ADDRESS_NONE

No address

QSPI_ADDRESS_1_LINE

Address on a single line

QSPI_ADDRESS_2_LINES

Address on two lines

QSPI_ADDRESS_4_LINES

Address on four lines

QSPI Address Size

QSPI_ADDRESS_8_BITS

8-bit address

QSPI_ADDRESS_16_BITS

16-bit address

QSPI_ADDRESS_24_BITS

24-bit address

QSPI_ADDRESS_32_BITS

32-bit address

QSPI Alternate Bytes Mode

QSPI_ALTERNATE_BYTES_NONE

No alternate bytes

QSPI_ALTERNATE_BYTES_1_LINE

Alternate bytes on a single line

QSPI_ALTERNATE_BYTES_2_LINES

Alternate bytes on two lines

QSPI_ALTERNATE_BYTES_4_LINES

Alternate bytes on four lines

QSPI Alternate Bytes Size

QSPI_ALTERNATE_BYTES_8_BITS

8-bit alternate bytes

QSPI_ALTERNATE_BYTES_16_BITS

16-bit alternate bytes

QSPI_ALTERNATE_BYTES_24_BITS

24-bit alternate bytes

QSPI_ALTERNATE_BYTES_32_BITS

32-bit alternate bytes

QSPI Automatic Stop

QSPI_AUTOMATIC_STOP_DISABLE

AutoPolling stops only with abort or QSPI disabling

QSPI_AUTOMATIC_STOP_ENABLE

AutoPolling stops as soon as there is a match

QSPI ChipSelect High Time

QSPI_CS_HIGH_TIME_1_CYCLE

nCS stay high for at least 1 clock cycle between commands

QSPI_CS_HIGH_TIME_2_CYCLE

nCS stay high for at least 2 clock cycles between commands

QSPI_CS_HIGH_TIME_3_CYCLE

nCS stay high for at least 3 clock cycles between commands

QSPI_CS_HIGH_TIME_4_CYCLE

nCS stay high for at least 4 clock cycles between commands

QSPI_CS_HIGH_TIME_5_CYCLE

nCS stay high for at least 5 clock cycles between commands

QSPI_CS_HIGH_TIME_6_CYCLE

nCS stay high for at least 6 clock cycles between commands

QSPI_CS_HIGH_TIME_7_CYCLE

nCS stay high for at least 7 clock cycles between commands

QSPI_CS_HIGH_TIME_8_CYCLE

nCS stay high for at least 8 clock cycles between commands

QSPI Clock Mode

QSPI_CLOCK_MODE_0

Clk stays low while nCS is released

QSPI_CLOCK_MODE_3

Clk goes high while nCS is released

QSPI Data Mode

QSPI_DATA_NONE

No data

QSPI_DATA_1_LINE

Data on a single line

QSPI_DATA_2_LINES

Data on two lines

QSPI_DATA_4_LINES

Data on four lines

QSPI DDR Data Output Delay

QSPI_DDR_HHC_ANALOG_DELAY

Delay the data output using analog delay in DDR mode

QSPI_DDR_HHC_HALF_CLK_DELAY

Delay the data output by one half of system clock in DDR mode

QSPI DDR Mode

QSPI_DDR_MODE_DISABLE

Double data rate mode disabled

QSPI_DDR_MODE_ENABLE

Double data rate mode enabled

QSPI Dual Flash Mode

QSPI_DUALFLASH_ENABLE

Dual-flash mode enabled

QSPI_DUALFLASH_DISABLE

Dual-flash mode disabled

QSPI Error Code

HAL_QSPI_ERROR_NONE

No error

HAL_QSPI_ERROR_TIMEOUT

Timeout error

HAL_QSPI_ERROR_TRANSFER

Transfer error

HAL_QSPI_ERROR_DMA

DMA transfer error

HAL_QSPI_ERROR_INVALID_PARAM

Invalid parameters error

HAL_QSPI_ERROR_INVALID_CALLBACK

Invalid callback error

QSPI Exported Macros

__HAL_QSPI_RESET_HANDLE_STATE

Description:

- Reset QSPI handle state.

Parameters:

- __HANDLE__: QSPI handle.

Return value:

- None

__HAL_QSPI_ENABLE

Description:

- Enable the QSPI peripheral.

Parameters:

- __HANDLE__: specifies the QSPI Handle.

Return value:

- None

__HAL_QSPI_DISABLE

Description:

- Disable the QSPI peripheral.

Parameters:

- __HANDLE__: specifies the QSPI Handle.

Return value:

- None

__HAL_QSPI_ENABLE_IT

Description:

- Enable the specified QSPI interrupt.

Parameters:

- __HANDLE__: specifies the QSPI Handle.
- __INTERRUPT__: specifies the QSPI interrupt source to enable. This parameter can be one of the following values:
  - QSPI_IT_TO: QSPI Timeout interrupt
  - QSPI_IT_SM: QSPI Status match interrupt
  - QSPI_IT_FT: QSPI FIFO threshold interrupt
  - QSPI_IT_TC: QSPI Transfer complete interrupt
  - QSPI_IT_TE: QSPI Transfer error interrupt

Return value:

- None

__HAL_QSPI_DISABLE_IT

Description:

- Disable the specified QSPI interrupt.

Parameters:

- __HANDLE__: specifies the QSPI Handle.
- __INTERRUPT__: specifies the QSPI interrupt source to disable. This parameter can be one of the following values:
  - QSPI_IT_TO: QSPI Timeout interrupt
  - QSPI_IT_SM: QSPI Status match interrupt
  - QSPI_IT_FT: QSPI FIFO threshold interrupt
  - QSPI_IT_TC: QSPI Transfer complete interrupt
  - QSPI_IT_TE: QSPI Transfer error interrupt

Return value:

- None

__HAL_QSPI_GET_IT_SOURCE

Description:

- Check whether the specified QSPI interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the QSPI Handle.
- __INTERRUPT__: specifies the QSPI interrupt source to check. This parameter can be one of the following values:
  - QSPI_IT_TO: QSPI Timeout interrupt
  - QSPI_IT_SM: QSPI Status match interrupt
  - QSPI_IT_FT: QSPI FIFO threshold interrupt
  - QSPI_IT_TC: QSPI Transfer complete interrupt
  - QSPI_IT_TE: QSPI Transfer error interrupt

Return value:

- The: new state of __INTERRUPT__ (TRUE or FALSE).

__HAL_QSPI_GET_FLAG

Description:

- Check whether the selected QSPI flag is set or not.

Parameters:

- __HANDLE__: specifies the QSPI Handle.
- __FLAG__: specifies the QSPI flag to check. This parameter can be one of the following values:
  - QSPI_FLAG_BUSY: QSPI Busy flag
  - QSPI_FLAG_TO: QSPI Timeout flag
  - QSPI_FLAG_SM: QSPI Status match flag
  - QSPI_FLAG_FT: QSPI FIFO threshold flag
  - QSPI_FLAG_TC: QSPI Transfer complete flag
  - QSPI_FLAG_TE: QSPI Transfer error flag

Return value:

- None

__HAL_QSPI_CLEAR_FLAG

Description:

- Clears the specified QSPI's flag status.

Parameters:

- __HANDLE__: specifies the QSPI Handle.
- __FLAG__: specifies the QSPI clear register flag that needs to be set This parameter can be one of the following values:
  - QSPI_FLAG_TO: QSPI Timeout flag
  - QSPI_FLAG_SM: QSPI Status match flag
  - QSPI_FLAG_TC: QSPI Transfer complete flag
  - QSPI_FLAG_TE: QSPI Transfer error flag

Return value:

- None

QSPI Flags

QSPI_FLAG_BUSY

Busy flag: operation is ongoing

QSPI_FLAG_TO

Timeout flag: timeout occurs in memory-mapped mode

QSPI_FLAG_SM

Status match flag: received data matches in autopolling mode

QSPI_FLAG_FT

Fifo threshold flag: Fifo threshold reached or data left after read from memory is complete

QSPI_FLAG_TC

Transfer complete flag: programmed number of data have been transferred or the transfer has been aborted

QSPI_FLAG_TE

Transfer error flag: invalid address is being accessed

QSPI Flash Select

QSPI_FLASH_ID_1

FLASH 1 selected

QSPI_FLASH_ID_2

FLASH 2 selected

QSPI Instruction Mode

QSPI_INSTRUCTION_NONE

No instruction

QSPI_INSTRUCTION_1_LINE

Instruction on a single line

QSPI_INSTRUCTION_2_LINES

Instruction on two lines

QSPI_INSTRUCTION_4_LINES

Instruction on four lines

QSPI Interrupts

QSPI_IT_TO

Interrupt on the timeout flag

QSPI_IT_SM

Interrupt on the status match flag

QSPI_IT_FT

Interrupt on the fifo threshold flag

QSPI_IT_TC

Interrupt on the transfer complete flag

QSPI_IT_TE

Interrupt on the transfer error flag

QSPI Match Mode

QSPI_MATCH_MODE_AND

AND match mode between unmasked bits

QSPI_MATCH_MODE_OR

OR match mode between unmasked bits

QSPI Sample Shifting

QSPI_SAMPLE_SHIFTING_NONE

No clock cycle shift to sample data

QSPI_SAMPLE_SHIFTING_HALFCYCLE

1/2 clock cycle shift to sample data

QSPI Send Instruction Mode

QSPI_SIOO_INST_EVERY_CMD

Send instruction on every transaction

QSPI_SIOO_INST_ONLY_FIRST_CMD

Send instruction only for the first command

QSPI Timeout Activation

QSPI_TIMEOUT_COUNTER_DISABLE

Timeout counter disabled, nCS remains active

QSPI_TIMEOUT_COUNTER_ENABLE

Timeout counter enabled, nCS released when timeout expires

QSPI Timeout definition

HAL_QSPI_TIMEOUT_DEFAULT_VALUE

67 HAL RAMECC Generic Driver

67.1 RAMECC Firmware driver registers structures

67.1.1 __RAMECC_HandleTypeDef

__RAMECC_HandleTypeDef is defined in the stm32h7xx_hal_ramecc.h

Data Fields

- ***RAMECC_MonitorTypeDef * Instance***
- ***__IO HAL_RAMECC_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* DetectErrorCallback***

Field Documentation

- ***RAMECC_MonitorTypeDef* __RAMECC_HandleTypeDef::Instance***
  Register base address
- ***__IO HAL_RAMECC_StateTypeDef __RAMECC_HandleTypeDef::State***
  RAMECC state
- ***__IO uint32_t __RAMECC_HandleTypeDef::ErrorCode***
  RAMECC Error Code
- ***void(* __RAMECC_HandleTypeDef::DetectErrorCallback)(struct __RAMECC_HandleTypeDef *hramecc)***
  RAMECC error detect callback

67.2 RAMECC Firmware driver API description

The following section lists the various functions of the RAMECC library.

67.2.1 How to use this driver

1. Enable and latch error information through HAL_RAMECC_Init().
2. For a given Monitor, enable and disable interrupt through HAL_RAMECC_EnableNotification(). To enable a notification for a given RAMECC instance, use global interrupts. To enable a notification for only RAMECC monitor, use monitor interrupts. All possible notifications are defined in the driver header file under RAMECC_Interrupt group.

Silent mode

- Use HAL_RAMECC_StartMonitor() to start RAMECC latch failing information without enabling any notification.

Interrupt mode

- Use HAL_RAMECC_EnableNotification() to enable interrupts for a given error.
- Configure the RAMECC interrupt priority using HAL_NVIC_SetPriority().
- Enable the RAMECC IRQ handler using HAL_NVIC_EnableIRQ().
- Start RAMECC latch failing information using HAL_RAMECC_StartMonitor().

Failing information

1. Use HAL_RAMECC_GetFailingAddress() function to return the RAMECC failing address.
2. Use HAL_RAMECC_GetFailingDataLow() function to return the RAMECC failing data low.
3. Use HAL_RAMECC_GetFailingDataHigh() function to return the RAMECC failing data high.

4. Use HAL_RAMECC_GetHammingErrorCode() function to return the RAMECC Hamming bits injected.
5. Use HAL_RAMECC_IsECCSingleErrorDetected() function to check if a single error was detected and corrected.
6. Use HAL_RAMECC_IsECCDoubleErrorDetected() function to check if a double error was dedetected.

RAMECC HAL driver macros list

Below the list of used macros in RAMECC HAL driver.

- __HAL_RAMECC_ENABLE_IT : Enable the specified ECCRAM Monitor interrupts.
- __HAL_RAMECC_DISABLE_IT : Disable the specified ECCRAM Monitor interrupts.
- __HAL_RAMECC_GET_FLAG : Return the current RAMECC Monitor selected flag.
- __HAL_RAMECC_CLEAR_FLAG : Clear the current RAMECC Monitor selected flag.

67.2.2 Initialization and de-initialization functions

This section provides functions allowing to initialize the RAMECC Monitor.

The HAL_RAMECC_Init() function follows the RAMECC configuration procedures as described in reference manual. The HAL_RAMECC_DeInit() function allows to deinitialize the RAMECC monitor.

This section contains the following APIs:

- ***HAL_RAMECC_Init()***
- ***HAL_RAMECC_DeInit()***

67.2.3 Monitoring operation functions

This section provides functions allowing to:

- Configure latching error information.
- Configure RAMECC Global/Monitor interrupts.
- Register and Unregister RAMECC callbacks
- Handle RAMECC interrupt request

This section contains the following APIs:

- ***HAL_RAMECC_StartMonitor()***
- ***HAL_RAMECC_StopMonitor()***
- ***HAL_RAMECC_EnableNotification()***
- ***HAL_RAMECC_DisableNotification()***
- ***HAL_RAMECC_RegisterCallback()***
- ***HAL_RAMECC_UnRegisterCallback()***
- ***HAL_RAMECC_IRQHandler()***

67.2.4 Error information functions

This section provides functions allowing to:

- Get failing address.
- Get failing data low.
- Get failing data high.
- Get hamming bits injected.
- Check single error flag.
- Check double error flag.

This section contains the following APIs:

- ***HAL_RAMECC_GetFailingAddress()***
- ***HAL_RAMECC_GetFailingDataLow()***
- ***HAL_RAMECC_GetFailingDataHigh()***
- ***HAL_RAMECC_GetHammingErrorCode()***
- ***HAL_RAMECC_IsECCSingleErrorDetected()***

- *HAL_RAMECC_IsECCDoubleErrorDetected()*

67.2.5 State and Error Functions

This section provides functions allowing to check and get the RAMECC state and the error code .

The HAL_RAMECC_GetState() function allows to get the RAMECC peripheral state. The HAL_RAMECC_GetError() function allows to Get the RAMECC peripheral error code.

This section contains the following APIs:

- *HAL_RAMECC_GetState()*
- *HAL_RAMECC_GetError()*

67.2.6 Detailed description of functions

HAL_RAMECC_Init

Function name

HAL_StatusTypeDef HAL_RAMECC_Init (RAMECC_HandleTypeDef * hramecc)

Function description

Initialize the RAMECC by clearing flags and disabling interrupts.

Parameters

- **hramecc:** Pointer to a RAMECC_HandleTypeDef structure that contains the configuration information for the specified RAMECC Monitor.

Return values

- **HAL:** status.

HAL_RAMECC_DeInit

Function name

HAL_StatusTypeDef HAL_RAMECC_DeInit (RAMECC_HandleTypeDef * hramecc)

Function description

DeInitializes the RAMECC peripheral.

Parameters

- **hramecc:** Pointer to a RAMECC_HandleTypeDef structure that contains the configuration information for the specified RAMECC Monitor.

Return values

- **HAL:** status.

HAL_RAMECC_StartMonitor

Function name

HAL_StatusTypeDef HAL_RAMECC_StartMonitor (RAMECC_HandleTypeDef * hramecc)

Function description

Starts the RAMECC latching error information.

Parameters

- **hramecc:** Pointer to a RAMECC_HandleTypeDef structure that contains the configuration information for the specified RAMECC Monitor.

Return values

- **HAL:** status.

HAL_RAMECC_StopMonitor

Function name

HAL_StatusTypeDef HAL_RAMECC_StopMonitor (RAMECC_HandleTypeDef * hramecc)

Function description

Stop the RAMECC latching error information.

Parameters

- **hramecc:** Pointer to a RAMECC_HandleTypeDef structure that contains the configuration information for the specified RAMECC Monitor.

Return values

- **HAL:** status.

HAL_RAMECC_EnableNotification

Function name

HAL_StatusTypeDef HAL_RAMECC_EnableNotification (RAMECC_HandleTypeDef * hramecc, uint32_t Notifications)

Function description

Enable the RAMECC error interrupts.

Parameters

- **hramecc:** Pointer to a RAMECC_HandleTypeDef structure that contains the configuration information for the specified RAMECC Monitor.
- **Notifications:** Select the notification.

Return values

- **HAL:** status.

HAL_RAMECC_DisableNotification

Function name

HAL_StatusTypeDef HAL_RAMECC_DisableNotification (RAMECC_HandleTypeDef * hramecc, uint32_t Notifications)

Function description

Disable the RAMECC error interrupts.

Parameters

- **hramecc:** Pointer to a RAMECC_HandleTypeDef structure that contains the configuration information for the specified RAMECC Monitor.
- **Notifications:** Select the notification.

Return values

- **HAL:** status.

HAL_RAMECC_IRQHandler

Function name

void HAL_RAMECC_IRQHandler (RAMECC_HandleTypeDef * hramecc)

Function description

Handles RAMECC interrupt request.

Parameters

- **hramecc:** Pointer to a RAMECC_HandleTypeDef structure that contains the configuration information for the specified RAMECC Monitor.

Return values

- **None.:**

HAL_RAMECC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_RAMECC_RegisterCallback (RAMECC_HandleTypeDef * hramecc, void(*) (RAMECC_HandleTypeDef *_hramecc) pCallback)

Function description

Register callbacks.

Parameters

- **hramecc:** Pointer to a RAMECC_HandleTypeDef structure that contains the configuration information for the specified RAMECC Monitor.
- **pCallback:** pointer to private callbacsk function which has pointer to a RAMECC_HandleTypeDef structure as parameter.

Return values

- **HAL:** status.

HAL_RAMECC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_RAMECC_UnRegisterCallback (RAMECC_HandleTypeDef * hramecc)

Function description

UnRegister callbacks.

Parameters

- **hramecc:** Pointer to a RAMECC_HandleTypeDef structure that contains the configuration information for the specified RAMECC Monitor.

Return values

- **HAL:** status.

HAL_RAMECC_GetFailingAddress

Function name

uint32_t HAL_RAMECC_GetFailingAddress (RAMECC_HandleTypeDef * hramecc)

Function description

Return the RAMECC failing address.

Parameters

- **hramecc:** Pointer to a RAMECC_HandleTypeDef structure that contains the configuration information for the specified RAMECC Monitor.

Return values

- **Failing:** address offset.

HAL_RAMECC_GetFailingDataLow

Function name

uint32_t HAL_RAMECC_GetFailingDataLow (RAMECC_HandleTypeDef * hramecc)

Function description

Return the RAMECC data low.

Parameters

- **hramecc:** Pointer to a RAMECC_HandleTypeDef structure that contains the configuration information for the specified RAMECC Monitor.

Return values

- **Failing:** data low.

HAL_RAMECC_GetFailingDataHigh

Function name

uint32_t HAL_RAMECC_GetFailingDataHigh (RAMECC_HandleTypeDef * hramecc)

Function description

Return the RAMECC data high.

Parameters

- **hramecc:** Pointer to a RAMECC_HandleTypeDef structure that contains the configuration information for the specified RAMECC Monitor.

Return values

- **Failing:** data high.

HAL_RAMECC_GetHammingErrorCode

Function name

uint32_t HAL_RAMECC_GetHammingErrorCode (RAMECC_HandleTypeDef * hramecc)

Function description

Return the RAMECC Hamming bits injected.

Parameters

- **hramecc:** Pointer to a RAMECC_HandleTypeDef structure that contains the configuration information for the specified RAMECC Monitor.

Return values

- **Hamming:** bits injected.

HAL_RAMECC_IsECCSingleErrorDetected

Function name

uint32_t HAL_RAMECC_IsECCSingleErrorDetected (RAMECC_HandleTypeDef * hramecc)

Function description

Check if an ECC single error was occurred.

Parameters

- **hramecc:** Pointer to a RAMECC_HandleTypeDef structure that contains the configuration information for the specified RAMECC Monitor.

Return values

- **State:** of bit (1 or 0).

HAL_RAMECC_IsECCDoubleErrorDetected

Function name

uint32_t HAL_RAMECC_IsECCDoubleErrorDetected (RAMECC_HandleTypeDef * hramecc)

Function description

Check if an ECC double error was occurred.

Parameters

- **hramecc:** Pointer to a RAMECC_HandleTypeDef structure that contains the configuration information for the specified RAMECC Monitor.

Return values

- **State:** of bit (1 or 0).

HAL_RAMECC_GetState

Function name

HAL_RAMECC_StateTypeDef HAL_RAMECC_GetState (RAMECC_HandleTypeDef * hramecc)

Function description

Get the RAMECC peripheral state.

Parameters

- **hramecc:** : Pointer to a RAMECC_HandleTypeDef structure that contains the configuration information for the specified RAMECC instance.

Return values

- **RAMECC:** state.

HAL_RAMECC_GetError

Function name

uint32_t HAL_RAMECC_GetError (RAMECC_HandleTypeDef * hramecc)

Function description

Get the RAMECC peripheral error code.

Parameters

- **hramecc:** : Pointer to a RAMECC_HandleTypeDef structure that contains the configuration information for the specified RAMECC instance.

Return values

- **RAMECC:** error code.

67.3 RAMECC Firmware driver defines

The following section lists the various define and macros of the module.

67.3.1 RAMECC

RAMECC

RAMECC Error Codes

HAL_RAMECC_ERROR_NONE

RAMECC No Error

HAL_RAMECC_ERROR_TIMEOUT

RAMECC Timeout Error

HAL_RAMECC_ERROR_BUSY

RAMECC Busy Error

HAL_RAMECC_ERROR_INVALID_CALLBACK

Invalid Callback error

RAMECC Exported Macros

__HAL_RAMECC_ENABLE_GLOBAL_IT

__HAL_RAMECC_ENABLE_MONITOR_IT

__HAL_RAMECC_ENABLE_IT

Description:

- Enable the specified RAMECC interrupts.

Parameters:

- __HANDLE__: : RAMECC handle.
- __INTERRUPT__: specifies the RAMECC interrupt sources to be enabled or disabled. This parameter can be one of the following values:
  - RAMECC_IT_GLOBAL_E : Global interrupt enable mask.
  - RAMECC_IT_GLOBAL_SEE : Global ECC single error interrupt enable.
  - RAMECC_IT_GLOBAL_DEE : Global ECC double error interrupt enable.
  - RAMECC_IT_GLOBAL_DEBWE : Global ECC double error on byte write (BW) interrupt enable.
  - RAMECC_IT_GLOBAL_ALL : All Global ECC interrupts enable mask.
  - RAMECC_IT_MONITOR_SEE : Monitor ECC single error interrupt enable.
  - RAMECC_IT_MONITOR_DEE : Monitor ECC double error interrupt enable.
  - RAMECC_IT_MONITOR_DEBWE : Monitor ECC double error on byte write (BW) interrupt enable.
  - RAMECC_IT_MONITOR_ALL : All Monitor ECC interrupts enable mask.

Return value:

- None

__HAL_RAMECC_DISABLE_GLOBAL_IT

__HAL_RAMECC_DISABLE_MONITOR_IT

__HAL_RAMECC_DISABLE_IT

Description:

- Disable the specified RAMECC interrupts.

Parameters:

- __HANDLE__: : RAMECC handle.
- __INTERRUPT__: specifies the RAMECC interrupt sources to be enabled or disabled. This parameter can be one of the following values:
  - RAMECC_IT_GLOBAL_E : Global interrupt enable mask.
  - RAMECC_IT_GLOBAL_SEE : Global ECC single error interrupt enable.
  - RAMECC_IT_GLOBAL_DEE : Global ECC double error interrupt enable.
  - RAMECC_IT_GLOBAL_DEBWE : Global ECC double error on byte write (BW) interrupt enable.
  - RAMECC_IT_GLOBAL_ALL : All Global ECC interrupts enable mask.
  - RAMECC_IT_MONITOR_SEE : Monitor ECC single error interrupt enable.
  - RAMECC_IT_MONITOR_DEE : Monitor ECC double error interrupt enable.
  - RAMECC_IT_MONITOR_DEBWE : Monitor ECC double error on byte write (BW) interrupt enable.
  - RAMECC_IT_MONITOR_ALL : All Monitor ECC interrupts enable mask.

Return value:

- None

__HAL_RAMECC_GET_GLOBAL_IT_SOURCE

__HAL_RAMECC_GET_MONITOR_IT_SOURCE

__HAL_RAMECC_GET_IT_SOURCE

Description:

- Check whether the specified RAMECC interrupt source is enabled or not.

Parameters:

- __HANDLE__: : Specifies the RAMECC Handle.
- __INTERRUPT__: : Specifies the RAMECC interrupt source to check. This parameter can be one of the following values:
  - RAMECC_IT_GLOBAL_E : Global interrupt enable mask.
  - RAMECC_IT_GLOBAL_SEE : Global ECC single error interrupt enable.
  - RAMECC_IT_GLOBAL_DEE : Global ECC double error interrupt enable.
  - RAMECC_IT_GLOBAL_DEBWE : Global ECC double error on byte write (BW) interrupt enable.
  - RAMECC_IT_GLOBAL_ALL : All Global ECC interrupts enable mask.
  - RAMECC_IT_MONITOR_SEE : Monitor ECC single error interrupt enable.
  - RAMECC_IT_MONITOR_DEE : Monitor ECC double error interrupt enable.
  - RAMECC_IT_MONITOR_DEBWE : Monitor ECC double error on byte write (BW) interrupt enable.
  - RAMECC_IT_MONITOR_ALL : All Monitor ECC interrupts enable mask.

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_RAMECC_GET_FLAG

Description:

- Get the RAMECC pending flags.

Parameters:

- __HANDLE__: : RAMECC handle.
- __FLAG__: : specifies the flag to clear. This parameter can be any combination of the following values:
  - RAMECC_FLAG_SEDCF : RAMECC instance ECC single error detected and corrected flag.
  - RAMECC_FLAG_DEDF : RAMECC instance ECC double error detected flag.
  - RAMECC_FLAG_DEBWDF : RAMECC instance ECC double error on byte write (BW) detected flag.
  - RAMECC_FLAGS_ALL : RAMECC instance all flag.

Return value:

- The: state of __FLAG__ (SET or RESET).

__HAL_RAMECC_CLEAR_FLAG

Description:

- Clear the RAMECC pending flags.

Parameters:

- __HANDLE__: : RAMECC handle.
- __FLAG__: : specifies the flag to clear. This parameter can be any combination of the following values:
  - RAMECC_FLAG_SEDCF : RAMECC instance ECC single error detected and corrected flag.
  - RAMECC_FLAG_DEDF : RAMECC instance ECC double error detected flag.
  - RAMECC_FLAG_DEBWDF : RAMECC instance ECC double error on byte write (BW) detected flag.
  - RAMECC_FLAGS_ALL : RAMECC instance all flag.

Return value:

- None.

__HAL_RAMECC_RESET_HANDLE_STATE

Description:

- Reset the RAMECC handle state.

Parameters:

- __HANDLE__: : Specifies the RAMECC Handle.

Return value:

- None.

RAMECC Monitor flags

RAMECC_FLAG_SINGLEERR_R

RAMECC_FLAG_DOUBLEERR_R

RAMECC_FLAG_DOUBLEERR_W

RAMECC_FLAGS_ALL

RAMECC interrupts

RAMECC_IT_GLOBAL_ID

RAMECC_IT_MONITOR_ID

RAMECC_IT_GLOBAL_ENABLE

RAMECC_IT_GLOBAL_SINGLEERR_R

RAMECC_IT_GLOBAL_DOUBLEERR_R

RAMECC_IT_GLOBAL_DOUBLEERR_W

RAMECC_IT_GLOBAL_ALL

RAMECC_IT_MONITOR_SINGLEERR_R

RAMECC_IT_MONITOR_DOUBLEERR_R

RAMECC_IT_MONITOR_DOUBLEERR_W

RAMECC_IT_MONITOR_ALL

68 HAL RCC Generic Driver

68.1 RCC Firmware driver registers structures

68.1.1 RCC_PLLInitTypeDef

RCC_PLLInitTypeDef is defined in the stm32h7xx_hal_rcc.h

Data Fields

- ***uint32_t PLLState***
- ***uint32_t PLLSource***
- ***uint32_t PLLM***
- ***uint32_t PLLN***
- ***uint32_t PLLP***
- ***uint32_t PLLQ***
- ***uint32_t PLLR***
- ***uint32_t PLLRGE***
- ***uint32_t PLLVCOSEL***
- ***uint32_t PLLFRACN***

Field Documentation

- ***uint32_t RCC_PLLInitTypeDef::PLLState***

  The new state of the PLL. This parameter can be a value of ***RCC_PLL_Config***
- ***uint32_t RCC_PLLInitTypeDef::PLLSource***

  RCC_PLLSource: PLL entry clock source. This parameter must be a value of ***RCC_PLL_Clock_Source***
- ***uint32_t RCC_PLLInitTypeDef::PLLM***

  PLLM: Division factor for PLL VCO input clock. This parameter must be a number between Min_Data = 1 and Max_Data = 63
- ***uint32_t RCC_PLLInitTypeDef::PLLN***

  PLLN: Multiplication factor for PLL VCO output clock. This parameter must be a number between Min_Data = 4 and Max_Data = 512 or between Min_Data = 8 and Max_Data = 420(*) (*) : For stm32h7a3xx and stm32h7b3xx family lines.
- ***uint32_t RCC_PLLInitTypeDef::PLLP***

  PLLP: Division factor for system clock. This parameter must be a number between Min_Data = 2 and Max_Data = 128 odd division factors are not allowed
- ***uint32_t RCC_PLLInitTypeDef::PLLQ***

  PLLQ: Division factor for peripheral clocks. This parameter must be a number between Min_Data = 1 and Max_Data = 128
- ***uint32_t RCC_PLLInitTypeDef::PLLR***

  PLLR: Division factor for peripheral clocks. This parameter must be a number between Min_Data = 1 and Max_Data = 128
- ***uint32_t RCC_PLLInitTypeDef::PLLRGE***

  PLLRGE: PLL1 clock Input range This parameter must be a value of ***RCC_PLL1_VCI_Range***
- ***uint32_t RCC_PLLInitTypeDef::PLLVCOSEL***

  PLLVCOSEL: PLL1 clock Output range This parameter must be a value of ***RCC_PLL1_VCO_Range***
- ***uint32_t RCC_PLLInitTypeDef::PLLFRACN***

  PLLFRACN: Specifies Fractional Part Of The Multiplication Factor for PLL1 VCO It should be a value between 0 and 8191

68.1.2 RCC_OscInitTypeDef

RCC_OscInitTypeDef is defined in the stm32h7xx_hal_rcc.h

Data Fields

- ***uint32_t OscillatorType***
- ***uint32_t HSEState***
- ***uint32_t LSEState***
- ***uint32_t HSIState***
- ***uint32_t HSICalibrationValue***
- ***uint32_t LSIState***
- ***uint32_t HSI48State***
- ***uint32_t CSIState***
- ***uint32_t CSICalibrationValue***
- ***RCC_PLLInitTypeDef PLL***

Field Documentation

- ***uint32_t RCC_OscInitTypeDef::OscillatorType***
  The oscillators to be configured. This parameter can be a value of ***RCC_Oscillator_Type***
- ***uint32_t RCC_OscInitTypeDef::HSEState***
  The new state of the HSE. This parameter can be a value of ***RCC_HSE_Config***
- ***uint32_t RCC_OscInitTypeDef::LSEState***
  The new state of the LSE. This parameter can be a value of ***RCC_LSE_Config***
- ***uint32_t RCC_OscInitTypeDef::HSIState***
  The new state of the HSI. This parameter can be a value of ***RCC_HSI_Config***
- ***uint32_t RCC_OscInitTypeDef::HSICalibrationValue***
  The calibration trimming value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x3F for STM32H7 rev.Y This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7F for STM32H7 rev.B and above
- ***uint32_t RCC_OscInitTypeDef::LSIState***
  The new state of the LSI. This parameter can be a value of ***RCC_LSI_Config***
- ***uint32_t RCC_OscInitTypeDef::HSI48State***
  The new state of the HSI48. This parameter can be a value of ***RCC_HSI48_Config***
- ***uint32_t RCC_OscInitTypeDef::CSIState***
  The new state of the CSI. This parameter can be a value of ***RCC_CSI_Config***
- ***uint32_t RCC_OscInitTypeDef::CSICalibrationValue***
  The calibration trimming value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x1F for STM32H7 rev.Y This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x3F for STM32H7 rev.B and above
- ***RCC_PLLInitTypeDef RCC_OscInitTypeDef::PLL***
  PLL structure parameters

68.1.3 RCC_ClkInitTypeDef

RCC_ClkInitTypeDef is defined in the stm32h7xx_hal_rcc.h

Data Fields

- ***uint32_t ClockType***
- ***uint32_t SYSCLKSource***
- ***uint32_t SYSCLKDivider***
- ***uint32_t AHBCLKDivider***
- ***uint32_t APB3CLKDivider***
- ***uint32_t APB1CLKDivider***
- ***uint32_t APB2CLKDivider***
- ***uint32_t APB4CLKDivider***

Field Documentation

- ***uint32_t RCC_ClkInitTypeDef::ClockType***
  The clock to be configured. This parameter can be a value of ***RCC_System_Clock_Type***
- ***uint32_t RCC_ClkInitTypeDef::SYSCLKSource***
  The clock source (SYSCLKS) used as system clock. This parameter can be a value of ***RCC_System_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::SYSCLKDivider***
  The system clock divider. This parameter can be a value of ***RCC_SYS_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::AHBCLKDivider***
  The AHB clock (HCLK) divider. This clock is derived from the system clock (SYSCLK). This parameter can be a value of ***RCC_HCLK_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::APB3CLKDivider***
  The APB3 clock (D1PCLK1) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_APB3_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::APB1CLKDivider***
  The APB1 clock (PCLK1) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_APB1_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::APB2CLKDivider***
  The APB2 clock (PCLK2) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_APB2_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::APB4CLKDivider***
  The APB4 clock (D3PCLK1) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_APB4_Clock_Source***

68.2 RCC Firmware driver API description

The following section lists the various functions of the RCC library.

68.2.1 RCC specific features

After reset the device is running from Internal High Speed oscillator (HSI 64MHz) with Flash 0 wait state,and all peripherals are off except internal SRAM, Flash, JTAG and PWR

- There is no pre-scaler on High speed (AHB) and Low speed (APB) buses; all peripherals mapped on these buses are running at HSI speed.
- The clock for all peripherals is switched off, except the SRAM and FLASH.
- All GPIOs are in analogue mode , except the JTAG pins which are assigned to be used for debug purpose.

Once the device started from reset, the user application has to:

- Configure the clock source to be used to drive the System clock (if the application needs higher frequency/ performance)
- Configure the System clock frequency and Flash settings
- Configure the AHB and APB buses pre-scalers
- Enable the clock for the peripheral(s) to be used
- Configure the clock kernel source(s) for peripherals which clocks are not derived from the System clock through :RCC_D1CCIPR,RCC_D2CCIP1R,RCC_D2CCIP2R and RCC_D3CCIPR registers

68.2.2 RCC Limitations

A delay between an RCC peripheral clock enable and the effective peripheral enabling should be taken into account in order to manage the peripheral read/write from/to registers.

- This delay depends on the peripheral mapping.
- If peripheral is mapped on AHB: the delay is 2 AHB clock cycle after the clock enable bit is set on the hardware register
- If peripheral is mapped on APB: the delay is 2 APB clock cycle after the clock enable bit is set on the hardware register

Implemented Workaround:

- For AHB & APB peripherals, a dummy read to the peripheral register has been inserted in each __HAL_RCC_PPP_CLK_ENABLE() macro.

68.2.3 Initialization and de-initialization functions

This section provides functions allowing to configure the internal/external oscillators (HSE, HSI, LSE,CSI, LSI,HSI48, PLL, CSS and MCO) and the System buses clocks (SYSCLK, AHB3, AHB1 AHB2,AHB4,APB3, APB1L, APB1H, APB2, and APB4).

Internal/external clock and PLL configuration

1. HSI (high-speed internal), 64 MHz factory-trimmed RC used directly or through the PLL as System clock source.
2. CSI is a low-power RC oscillator which can be used directly as system clock, peripheral clock, or PLL input.But even with frequency calibration, is less accurate than an external crystal oscillator or ceramic resonator.
3. LSI (low-speed internal), 32 KHz low consumption RC used as IWDG and/or RTC clock source.
4. HSE (high-speed external), 4 to 48 MHz crystal oscillator used directly or through the PLL as System clock source. Can be used also as RTC clock source.
5. LSE (low-speed external), 32 KHz oscillator used as RTC clock source.
6. PLL , The RCC features three independent PLLs (clocked by HSI , HSE or CSI), featuring three different output clocks and able to work either in integer or Fractional mode.
   – A main PLL, PLL1, which is generally used to provide clocks to the CPU and to some peripherals.
   – Two dedicated PLLs, PLL2 and PLL3, which are used to generate the kernel clock for peripherals.
7. CSS (Clock security system), once enabled and if a HSE clock failure occurs (HSE used directly or through PLL as System clock source), the System clock is automatically switched to HSI and an interrupt is generated if enabled. The interrupt is linked to the Cortex-M NMI (Non-Mask-able Interrupt) exception vector.
8. MCO1 (micro controller clock output), used to output HSI, LSE, HSE, PLL1(PLL1_Q) or HSI48 clock (through a configurable pre-scaler) on PA8 pin.
9. MCO2 (micro controller clock output), used to output HSE, PLL2(PLL2_P), SYSCLK, LSI, CSI, or PLL1(PLL1_P) clock (through a configurable pre-scaler) on PC9 pin.

System, AHB and APB buses clocks configuration

1. Several clock sources can be used to drive the System clock (SYSCLK): CSI,HSI, HSE and PLL. The AHB clock (HCLK) is derived from System core clock through configurable pre-scaler and used to clock the CPU, memory and peripherals mapped on AHB and APB bus of the 3 Domains (D1, D2, D3)* through configurable pre-scalers and used to clock the peripherals mapped on these buses. You can use "HAL_RCC_GetSysClockFreq()" function to retrieve system clock frequency.

Note: *All the peripheral clocks are derived from the System clock (SYSCLK) except those with dual clock domain where kernel source clock could be selected through RCC_D1CCIPR,RCC_D2CCIP1R,RCC_D2CCIP2R and RCC_D3CCIPR registers. (*) : 2 Domains (CD and SRD) for stm32h7a3xx and stm32h7b3xx family lines.*

This section contains the following APIs:

- ***HAL_RCC_DeInit()***
- ***HAL_RCC_OscConfig()***
- ***HAL_RCC_ClockConfig()***

68.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the RCC Clocks frequencies.

This section contains the following APIs:

- ***HAL_RCC_MCOConfig()***
- ***HAL_RCC_EnableCSS()***
- ***HAL_RCC_DisableCSS()***
- ***HAL_RCC_GetSysClockFreq()***
- ***HAL_RCC_GetHCLKFreq()***
- ***HAL_RCC_GetPCLK1Freq()***

- *HAL_RCC_GetPCLK2Freq()*
- *HAL_RCC_GetOscConfig()*
- *HAL_RCC_GetClockConfig()*
- *HAL_RCC_NMI_IRQHandler()*
- *HAL_RCC_CSSCallback()*

68.2.5 Detailed description of functions

HAL_RCC_DeInit

Function name

HAL_StatusTypeDef HAL_RCC_DeInit (void)

Function description

Resets the RCC clock configuration to the default reset state.

Return values

- **HAL:** status

Notes

- The default reset state of the clock configuration is given below: HSI ON and used as system clock sourceHSE, PLL1, PLL2 and PLL3 OFFAHB, APB Bus pre-scaler set to 1.CSS, MCO1 and MCO2 OFFAll interrupts disabled
- This function doesn't modify the configuration of the Peripheral clocksLSI, LSE and RTC clocks

HAL_RCC_OscConfig

Function name

HAL_StatusTypeDef HAL_RCC_OscConfig (RCC_OscInitTypeDef * RCC_OscInitStruct)

Function description

Initializes the RCC Oscillators according to the specified parameters in the RCC_OscInitTypeDef.

Parameters

- **RCC_OscInitStruct:** pointer to an RCC_OscInitTypeDef structure that contains the configuration information for the RCC Oscillators.

Return values

- **HAL:** status

Notes

- The PLL is not disabled when used as system clock.
- Transitions LSE Bypass to LSE On and LSE On to LSE Bypass are not supported by this function. User should request a transition to LSE Off first and then LSE On or LSE Bypass.
- Transition HSE Bypass to HSE On and HSE On to HSE Bypass are not supported by this function. User should request a transition to HSE Off first and then HSE On or HSE Bypass.

HAL_RCC_ClockConfig

Function name

HAL_StatusTypeDef HAL_RCC_ClockConfig (RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t FLatency)

Function description

Initializes the CPU, AHB and APB buses clocks according to the specified parameters in the RCC_ClkInitStruct.

Parameters

- **RCC_ClkInitStruct:** pointer to an RCC_OscInitTypeDef structure that contains the configuration information for the RCC peripheral.
- **FLatency:** FLASH Latency, this parameter depend on device selected

Return values

- **None:**

Notes

- The SystemCoreClock CMSIS variable is used to store System Core Clock Frequency and updated by HAL_InitTick() function called within this function
- The HSI is used (enabled by hardware) as system clock source after start-up from Reset, wake-up from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled).
- A switch from one clock source to another occurs only if the target clock source is ready (clock stable after start-up delay or PLL locked). If a clock source which is not yet ready is selected, the switch will occur when the clock source will be ready. You can use HAL_RCC_GetClockConfig() function to know which clock is currently used as system clock source.
- Depending on the device voltage range, the software has to set correctly D1CPRE[3:0] bits to ensure that Domain1 core clock not exceed the maximum allowed frequency (for more details refer to section above "Initialization/de-initialization functions")

HAL_RCC_MCOConfig

Function name

void HAL_RCC_MCOConfig (uint32_t RCC_MCOx, uint32_t RCC_MCOSource, uint32_t RCC_MCODiv)

Function description

Selects the clock source to output on MCO1 pin(PA8) or on MCO2 pin(PC9).

Parameters

- **RCC_MCOx:** specifies the output direction for the clock source. This parameter can be one of the following values:
  - RCC_MCO1: Clock source to output on MCO1 pin(PA8).
  - RCC_MCO2: Clock source to output on MCO2 pin(PC9).
- **RCC_MCOSource:** specifies the clock source to output. This parameter can be one of the following values:
  - RCC_MCO1SOURCE_HSI: HSI clock selected as MCO1 source
  - RCC_MCO1SOURCE_LSE: LSE clock selected as MCO1 source
  - RCC_MCO1SOURCE_HSE: HSE clock selected as MCO1 source
  - RCC_MCO1SOURCE_PLL1QCLK: PLL1Q clock selected as MCO1 source
  - RCC_MCO1SOURCE_HSI48: HSI48 (48MHZ) selected as MCO1 source
  - RCC_MCO2SOURCE_SYSCLK: System clock (SYSCLK) selected as MCO2 source
  - RCC_MCO2SOURCE_PLL2PCLK: PLL2P clock selected as MCO2 source
  - RCC_MCO2SOURCE_HSE: HSE clock selected as MCO2 source
  - RCC_MCO2SOURCE_PLLCLK: PLL1P clock selected as MCO2 source
  - RCC_MCO2SOURCE_CSICLK: CSI clock selected as MCO2 source
  - RCC_MCO2SOURCE_LSICLK: LSI clock selected as MCO2 source
- **RCC_MCODiv:** specifies the MCOx pre-scaler. This parameter can be one of the following values:
  - RCC_MCODIV_1 up to RCC_MCODIV_15 : divider applied to MCOx clock

Return values

- **None:**

Notes

- PA8/PC9 should be configured in alternate function mode.

HAL_RCC_EnableCSS

Function name

void HAL_RCC_EnableCSS (void)

Function description

Enables the Clock Security System.

Return values

- **None:**

Notes

- If a failure is detected on the HSE oscillator clock, this oscillator is automatically disabled and an interrupt is generated to inform the software about the failure (Clock Security System Interrupt, CSSI), allowing the MCU to perform rescue operations. The CSSI is linked to the Cortex-M NMI (Non-Mask-able Interrupt) exception vector.

HAL_RCC_DisableCSS

Function name

void HAL_RCC_DisableCSS (void)

Function description

Disables the Clock Security System.

Return values

- **None:**

HAL_RCC_GetSysClockFreq

Function name

uint32_t HAL_RCC_GetSysClockFreq (void)

Function description

Returns the SYSCLK frequency.

Return values

- **SYSCLK:** frequency

Notes

- The system frequency computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:
- If SYSCLK source is CSI, function returns values based on CSI_VALUE(*)
- If SYSCLK source is HSI, function returns values based on HSI_VALUE(**)
- If SYSCLK source is HSE, function returns values based on HSE_VALUE(***)
- If SYSCLK source is PLL, function returns values based on CSI_VALUE(*), HSI_VALUE(**) or HSE_VALUE(***) multiplied/divided by the PLL factors.
- (*) CSI_VALUE is a constant defined in stm32h7xx_hal_conf.h file (default value 4 MHz) but the real value may vary depending on the variations in voltage and temperature.
- (**) HSI_VALUE is a constant defined in stm32h7xx_hal_conf.h file (default value 64 MHz) but the real value may vary depending on the variations in voltage and temperature.
- (***) HSE_VALUE is a constant defined in stm32h7xx_hal_conf.h file (default value 25 MHz), user has to ensure that HSE_VALUE is same as the real frequency of the crystal used. Otherwise, this function may have wrong result.
- The result of this function could be not correct when using fractional value for HSE crystal.
- This function can be used by the user application to compute the baud rate for the communication peripherals or configure other parameters.
- Each time SYSCLK changes, this function must be called to update the right SYSCLK value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCC_GetHCLKFreq

Function name

uint32_t HAL_RCC_GetHCLKFreq (void)

Function description

Returns the HCLK frequency.

Return values

- **HCLK:** frequency

Notes

- Each time HCLK changes, this function must be called to update the right HCLK value. Otherwise, any configuration based on this function will be incorrect.
- The SystemD2Clock CMSIS variable is used to store System domain2 Clock Frequency and updated within this function

HAL_RCC_GetPCLK1Freq

Function name

uint32_t HAL_RCC_GetPCLK1Freq (void)

Function description

Returns the PCLK1 frequency.

Return values

- **PCLK1:** frequency

Notes

- Each time PCLK1 changes, this function must be called to update the right PCLK1 value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCC_GetPCLK2Freq

Function name

uint32_t HAL_RCC_GetPCLK2Freq (void)

Function description

Returns the D2 PCLK2 frequency.

Return values

- **PCLK1:** frequency

Notes

- Each time PCLK2 changes, this function must be called to update the right PCLK2 value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCC_GetOscConfig

Function name

void HAL_RCC_GetOscConfig (RCC_OscInitTypeDef * RCC_OscInitStruct)

Function description

Configures the RCC_OscInitStruct according to the internal RCC configuration registers.

Parameters

- **RCC_OscInitStruct:** pointer to an RCC_OscInitTypeDef structure that will be configured.

Return values

- **None:**

HAL_RCC_GetClockConfig

Function name

void HAL_RCC_GetClockConfig (RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t * pFLatency)

Function description

Configures the RCC_ClkInitStruct according to the internal RCC configuration registers.

Parameters

- **RCC_ClkInitStruct:** pointer to an RCC_ClkInitTypeDef structure that will be configured.
- **pFLatency:** Pointer on the Flash Latency.

Return values

- **None:**

HAL_RCC_NMI_IRQHandler

Function name

void HAL_RCC_NMI_IRQHandler (void)

Function description

This function handles the RCC CSS interrupt request.

Return values

- **None:**

Notes

- This API should be called under the NMI_Handler().

HAL_RCC_CSSCallback

Function name

void HAL_RCC_CSSCallback (void)

Function description

RCC Clock Security System interrupt callback.

Return values

- **none:**

68.3 RCC Firmware driver defines

The following section lists the various define and macros of the module.

68.3.1 RCC

RCC

RCC APB1 Clock Source

RCC_APB1_DIV1

RCC_APB1_DIV2

RCC_APB1_DIV4

RCC_APB1_DIV8

RCC_APB1_DIV16

RCC APB2 Clock Source

RCC_APB2_DIV1

RCC_APB2_DIV2

RCC_APB2_DIV4

RCC_APB2_DIV8

RCC_APB2_DIV16

RCC APB3 Clock Source

RCC_APB3_DIV1

RCC_APB3_DIV2

RCC_APB3_DIV4

RCC_APB3_DIV8

RCC_APB3_DIV16

RCC APB4 Clock Source

RCC_APB4_DIV1

RCC_APB4_DIV2

RCC_APB4_DIV4

RCC_APB4_DIV8

RCC_APB4_DIV16

RCC CSI Config

RCC_CSI_OFF

RCC_CSI_ON

RCC_CSICALIBRATION_DEFAULT

RCC Exported Constants

HAL_RCC_REV_Y_HSITRIM_Pos

HAL_RCC_REV_Y_HSITRIM_Msk

HAL_RCC_REV_Y_CSITRIM_Pos

HAL_RCC_REV_Y_CSITRIM_Msk

RCC Exported Macros

__HAL_RCC_MDMA_CLK_ENABLE

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_DMA2D_CLK_ENABLE

__HAL_RCC_JPGDECEN_CLK_ENABLE

__HAL_RCC_FMC_CLK_ENABLE

__HAL_RCC_QSPI_CLK_ENABLE

__HAL_RCC_SDMMC1_CLK_ENABLE

__HAL_RCC_MDMA_CLK_DISABLE

__HAL_RCC_DMA2D_CLK_DISABLE

__HAL_RCC_JPGDECEN_CLK_DISABLE

__HAL_RCC_FMC_CLK_DISABLE

__HAL_RCC_QSPI_CLK_DISABLE

__HAL_RCC_SDMMC1_CLK_DISABLE

__HAL_RCC_MDMA_IS_CLK_ENABLED

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_DMA2D_IS_CLK_ENABLED

__HAL_RCC_JPGDECEN_IS_CLK_ENABLED

__HAL_RCC_FMC_IS_CLK_ENABLED

__HAL_RCC_QSPI_IS_CLK_ENABLED

__HAL_RCC_SDMMC1_IS_CLK_ENABLED

__HAL_RCC_MDMA_IS_CLK_DISABLED

__HAL_RCC_DMA2D_IS_CLK_DISABLED

__HAL_RCC_JPGDECEN_IS_CLK_DISABLED

__HAL_RCC_FMC_IS_CLK_DISABLED

__HAL_RCC_QSPI_IS_CLK_DISABLED

__HAL_RCC_SDMMC1_IS_CLK_DISABLED

__HAL_RCC_DMA1_CLK_ENABLE

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_DMA2_CLK_ENABLE

__HAL_RCC_ADC12_CLK_ENABLE

__HAL_RCC_ART_CLK_ENABLE

__HAL_RCC_ETH1MAC_CLK_ENABLE

__HAL_RCC_ETH1TX_CLK_ENABLE

__HAL_RCC_ETH1RX_CLK_ENABLE

__HAL_RCC_USB1_OTG_HS_CLK_ENABLE

__HAL_RCC_USB1_OTG_HS_ULPI_CLK_ENABLE

__HAL_RCC_USB2_OTG_FS_CLK_ENABLE

__HAL_RCC_USB2_OTG_FS_ULPI_CLK_ENABLE

__HAL_RCC_DMA1_CLK_DISABLE

__HAL_RCC_DMA2_CLK_DISABLE

__HAL_RCC_ADC12_CLK_DISABLE

__HAL_RCC_ART_CLK_DISABLE

__HAL_RCC_ETH1MAC_CLK_DISABLE

__HAL_RCC_ETH1TX_CLK_DISABLE

__HAL_RCC_ETH1RX_CLK_DISABLE

__HAL_RCC_USB1_OTG_HS_CLK_DISABLE

__HAL_RCC_USB1_OTG_HS_ULPI_CLK_DISABLE

__HAL_RCC_USB2_OTG_FS_CLK_DISABLE

__HAL_RCC_USB2_OTG_FS_ULPI_CLK_DISABLE

__HAL_RCC_DMA1_IS_CLK_ENABLED

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_DMA2_IS_CLK_ENABLED

__HAL_RCC_ADC12_IS_CLK_ENABLED

__HAL_RCC_ART_IS_CLK_ENABLED

__HAL_RCC_ETH1MAC_IS_CLK_ENABLED

__HAL_RCC_ETH1TX_IS_CLK_ENABLED

__HAL_RCC_ETH1RX_IS_CLK_ENABLED

__HAL_RCC_USB1_OTG_HS_IS_CLK_ENABLED

__HAL_RCC_USB1_OTG_HS_ULPI_IS_CLK_ENABLED

__HAL_RCC_USB2_OTG_FS_IS_CLK_ENABLED

__HAL_RCC_USB2_OTG_FS_ULPI_IS_CLK_ENABLED

__HAL_RCC_DMA1_IS_CLK_DISABLED

__HAL_RCC_DMA2_IS_CLK_DISABLED

__HAL_RCC_ADC12_IS_CLK_DISABLED

__HAL_RCC_ART_IS_CLK_DISABLED

__HAL_RCC_ETH1MAC_IS_CLK_DISABLED

__HAL_RCC_ETH1TX_IS_CLK_DISABLED

__HAL_RCC_ETH1RX_IS_CLK_DISABLED

__HAL_RCC_USB1_OTG_HS_IS_CLK_DISABLED

__HAL_RCC_USB1_OTG_HS_ULPI_IS_CLK_DISABLED

__HAL_RCC_USB2_OTG_FS_IS_CLK_DISABLED

__HAL_RCC_USB2_OTG_FS_ULPI_IS_CLK_DISABLED

__HAL_RCC_DCMI_CLK_ENABLE

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_CRYP_CLK_ENABLE

__HAL_RCC_HASH_CLK_ENABLE

__HAL_RCC_RNG_CLK_ENABLE

__HAL_RCC_SDMMC2_CLK_ENABLE

__HAL_RCC_D2SRAM1_CLK_ENABLE

__HAL_RCC_D2SRAM2_CLK_ENABLE

__HAL_RCC_D2SRAM3_CLK_ENABLE

__HAL_RCC_DCMI_CLK_DISABLE

__HAL_RCC_CRYP_CLK_DISABLE

__HAL_RCC_HASH_CLK_DISABLE

__HAL_RCC_RNG_CLK_DISABLE

__HAL_RCC_SDMMC2_CLK_DISABLE

__HAL_RCC_D2SRAM1_CLK_DISABLE

__HAL_RCC_D2SRAM2_CLK_DISABLE

__HAL_RCC_D2SRAM3_CLK_DISABLE

__HAL_RCC_DCMI_IS_CLK_ENABLED

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_CRYP_IS_CLK_ENABLED

__HAL_RCC_HASH_IS_CLK_ENABLED

__HAL_RCC_RNG_IS_CLK_ENABLED

__HAL_RCC_SDMMC2_IS_CLK_ENABLED

__HAL_RCC_D2SRAM1_IS_CLK_ENABLED

__HAL_RCC_D2SRAM2_IS_CLK_ENABLED

__HAL_RCC_D2SRAM3_IS_CLK_ENABLED

__HAL_RCC_DCMI_IS_CLK_DISABLED

__HAL_RCC_CRYP_IS_CLK_DISABLED

__HAL_RCC_HASH_IS_CLK_DISABLED

__HAL_RCC_RNG_IS_CLK_DISABLED

__HAL_RCC_SDMMC2_IS_CLK_DISABLED

__HAL_RCC_D2SRAM1_IS_CLK_DISABLED

__HAL_RCC_D2SRAM2_IS_CLK_DISABLED

__HAL_RCC_D2SRAM3_IS_CLK_DISABLED

__HAL_RCC_GPIOA_CLK_ENABLE

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_GPIOB_CLK_ENABLE

__HAL_RCC_GPIOC_CLK_ENABLE

__HAL_RCC_GPIOD_CLK_ENABLE

__HAL_RCC_GPIOE_CLK_ENABLE

__HAL_RCC_GPIOF_CLK_ENABLE

__HAL_RCC_GPIOG_CLK_ENABLE

__HAL_RCC_GPIOH_CLK_ENABLE

__HAL_RCC_GPIOI_CLK_ENABLE

__HAL_RCC_GPIOJ_CLK_ENABLE

__HAL_RCC_GPIOK_CLK_ENABLE

__HAL_RCC_CRC_CLK_ENABLE

__HAL_RCC_BDMA_CLK_ENABLE

__HAL_RCC_ADC3_CLK_ENABLE

__HAL_RCC_HSEM_CLK_ENABLE

__HAL_RCC_BKPRAM_CLK_ENABLE

__HAL_RCC_GPIOA_CLK_DISABLE

__HAL_RCC_GPIOB_CLK_DISABLE

__HAL_RCC_GPIOC_CLK_DISABLE

__HAL_RCC_GPIOD_CLK_DISABLE

__HAL_RCC_GPIOE_CLK_DISABLE

__HAL_RCC_GPIOF_CLK_DISABLE

__HAL_RCC_GPIOG_CLK_DISABLE

__HAL_RCC_GPIOH_CLK_DISABLE

__HAL_RCC_GPIOI_CLK_DISABLE

__HAL_RCC_GPIOJ_CLK_DISABLE

__HAL_RCC_GPIOK_CLK_DISABLE

__HAL_RCC_CRC_CLK_DISABLE

__HAL_RCC_BDMA_CLK_DISABLE

__HAL_RCC_ADC3_CLK_DISABLE

__HAL_RCC_HSEM_CLK_DISABLE

__HAL_RCC_BKPRAM_CLK_DISABLE

__HAL_RCC_GPIOA_IS_CLK_ENABLED

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_GPIOB_IS_CLK_ENABLED

__HAL_RCC_GPIOC_IS_CLK_ENABLED

__HAL_RCC_GPIOD_IS_CLK_ENABLED

__HAL_RCC_GPIOE_IS_CLK_ENABLED

__HAL_RCC_GPIOF_IS_CLK_ENABLED

__HAL_RCC_GPIOG_IS_CLK_ENABLED

__HAL_RCC_GPIOH_IS_CLK_ENABLED

__HAL_RCC_GPIOI_IS_CLK_ENABLED

__HAL_RCC_GPIOJ_IS_CLK_ENABLED

__HAL_RCC_GPIOK_IS_CLK_ENABLED

__HAL_RCC_CRC_IS_CLK_ENABLED

__HAL_RCC_BDMA_IS_CLK_ENABLED

__HAL_RCC_ADC3_IS_CLK_ENABLED

__HAL_RCC_HSEM_IS_CLK_ENABLED

__HAL_RCC_BKPRAM_IS_CLK_ENABLED

__HAL_RCC_GPIOA_IS_CLK_DISABLED

__HAL_RCC_GPIOB_IS_CLK_DISABLED

__HAL_RCC_GPIOC_IS_CLK_DISABLED

__HAL_RCC_GPIOD_IS_CLK_DISABLED

__HAL_RCC_GPIOE_IS_CLK_DISABLED

__HAL_RCC_GPIOF_IS_CLK_DISABLED

__HAL_RCC_GPIOG_IS_CLK_DISABLED

__HAL_RCC_GPIOH_IS_CLK_DISABLED

__HAL_RCC_GPIOI_IS_CLK_DISABLED

__HAL_RCC_GPIOJ_IS_CLK_DISABLED

__HAL_RCC_GPIOK_IS_CLK_DISABLED

__HAL_RCC_CRC_IS_CLK_DISABLED

__HAL_RCC_BDMA_IS_CLK_DISABLED

__HAL_RCC_ADC3_IS_CLK_DISABLED

__HAL_RCC_HSEM_IS_CLK_DISABLED

__HAL_RCC_BKPRAM_IS_CLK_DISABLED

__HAL_RCC_LTDC_CLK_ENABLE

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_WWDG1_CLK_ENABLE

__HAL_RCC_LTDC_CLK_DISABLE

__HAL_RCC_WWDG1_CLK_DISABLE

__HAL_RCC_LTDC_IS_CLK_ENABLED

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_WWDG1_IS_CLK_ENABLED

__HAL_RCC_LTDC_IS_CLK_DISABLED

__HAL_RCC_WWDG1_IS_CLK_DISABLED

__HAL_RCC_TIM2_CLK_ENABLE

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_TIM3_CLK_ENABLE

__HAL_RCC_TIM4_CLK_ENABLE

__HAL_RCC_TIM5_CLK_ENABLE

__HAL_RCC_TIM6_CLK_ENABLE

__HAL_RCC_TIM7_CLK_ENABLE

__HAL_RCC_TIM12_CLK_ENABLE

__HAL_RCC_TIM13_CLK_ENABLE

__HAL_RCC_TIM14_CLK_ENABLE

__HAL_RCC_LPTIM1_CLK_ENABLE

__HAL_RCC_WWDG2_CLK_ENABLE

__HAL_RCC_SPI2_CLK_ENABLE

__HAL_RCC_SPI3_CLK_ENABLE

__HAL_RCC_SPDIFRX_CLK_ENABLE

__HAL_RCC_USART2_CLK_ENABLE

__HAL_RCC_USART3_CLK_ENABLE

__HAL_RCC_UART4_CLK_ENABLE

__HAL_RCC_UART5_CLK_ENABLE

__HAL_RCC_I2C1_CLK_ENABLE

__HAL_RCC_I2C2_CLK_ENABLE

__HAL_RCC_I2C3_CLK_ENABLE

__HAL_RCC_CEC_CLK_ENABLE

__HAL_RCC_DAC12_CLK_ENABLE

__HAL_RCC_UART7_CLK_ENABLE

__HAL_RCC_UART8_CLK_ENABLE

__HAL_RCC_CRS_CLK_ENABLE

__HAL_RCC_SWPMI1_CLK_ENABLE

__HAL_RCC_OPAMP_CLK_ENABLE

__HAL_RCC_MDIOS_CLK_ENABLE

__HAL_RCC_FDCAN_CLK_ENABLE

__HAL_RCC_TIM2_CLK_DISABLE

__HAL_RCC_TIM3_CLK_DISABLE

__HAL_RCC_TIM4_CLK_DISABLE

__HAL_RCC_TIM5_CLK_DISABLE

__HAL_RCC_TIM6_CLK_DISABLE

__HAL_RCC_TIM7_CLK_DISABLE

__HAL_RCC_TIM12_CLK_DISABLE

__HAL_RCC_TIM13_CLK_DISABLE

__HAL_RCC_TIM14_CLK_DISABLE

__HAL_RCC_LPTIM1_CLK_DISABLE

__HAL_RCC_WWDG2_CLK_DISABLE

__HAL_RCC_SPI2_CLK_DISABLE

__HAL_RCC_SPI3_CLK_DISABLE

__HAL_RCC_SPDIFRX_CLK_DISABLE

__HAL_RCC_USART2_CLK_DISABLE

__HAL_RCC_USART3_CLK_DISABLE

__HAL_RCC_UART4_CLK_DISABLE

__HAL_RCC_UART5_CLK_DISABLE

__HAL_RCC_I2C1_CLK_DISABLE

__HAL_RCC_I2C2_CLK_DISABLE

__HAL_RCC_I2C3_CLK_DISABLE

__HAL_RCC_CEC_CLK_DISABLE

__HAL_RCC_DAC12_CLK_DISABLE

__HAL_RCC_UART7_CLK_DISABLE

__HAL_RCC_UART8_CLK_DISABLE

__HAL_RCC_CRS_CLK_DISABLE

__HAL_RCC_SWPMI1_CLK_DISABLE

__HAL_RCC_OPAMP_CLK_DISABLE

__HAL_RCC_MDIOS_CLK_DISABLE

__HAL_RCC_FDCAN_CLK_DISABLE

__HAL_RCC_TIM2_IS_CLK_ENABLED

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_TIM3_IS_CLK_ENABLED

__HAL_RCC_TIM4_IS_CLK_ENABLED

__HAL_RCC_TIM5_IS_CLK_ENABLED

__HAL_RCC_TIM6_IS_CLK_ENABLED

__HAL_RCC_TIM7_IS_CLK_ENABLED

__HAL_RCC_TIM12_IS_CLK_ENABLED

__HAL_RCC_TIM13_IS_CLK_ENABLED

__HAL_RCC_TIM14_IS_CLK_ENABLED

__HAL_RCC_LPTIM1_IS_CLK_ENABLED

__HAL_RCC_WWDG2_IS_CLK_ENABLED

__HAL_RCC_SPI2_IS_CLK_ENABLED

__HAL_RCC_SPI3_IS_CLK_ENABLED

__HAL_RCC_SPDIFRX_IS_CLK_ENABLED

__HAL_RCC_USART2_IS_CLK_ENABLED

__HAL_RCC_USART3_IS_CLK_ENABLED

__HAL_RCC_UART4_IS_CLK_ENABLED

__HAL_RCC_UART5_IS_CLK_ENABLED

__HAL_RCC_I2C1_IS_CLK_ENABLED

__HAL_RCC_I2C2_IS_CLK_ENABLED

__HAL_RCC_I2C3_IS_CLK_ENABLED

__HAL_RCC_CEC_IS_CLK_ENABLED

__HAL_RCC_DAC12_IS_CLK_ENABLED

__HAL_RCC_UART7_IS_CLK_ENABLED

__HAL_RCC_UART8_IS_CLK_ENABLED

__HAL_RCC_CRS_IS_CLK_ENABLED

__HAL_RCC_SWPMI1_IS_CLK_ENABLED

__HAL_RCC_OPAMP_IS_CLK_ENABLED

__HAL_RCC_MDIOS_IS_CLK_ENABLED

__HAL_RCC_FDCAN_IS_CLK_ENABLED

__HAL_RCC_TIM2_IS_CLK_DISABLED

__HAL_RCC_TIM3_IS_CLK_DISABLED

__HAL_RCC_TIM4_IS_CLK_DISABLED

__HAL_RCC_TIM5_IS_CLK_DISABLED

__HAL_RCC_TIM6_IS_CLK_DISABLED

__HAL_RCC_TIM7_IS_CLK_DISABLED

__HAL_RCC_TIM12_IS_CLK_DISABLED

__HAL_RCC_TIM13_IS_CLK_DISABLED

__HAL_RCC_TIM14_IS_CLK_DISABLED

__HAL_RCC_LPTIM1_IS_CLK_DISABLED

__HAL_RCC_WWDG2_IS_CLK_DISABLED

__HAL_RCC_SPI2_IS_CLK_DISABLED

__HAL_RCC_SPI3_IS_CLK_DISABLED

__HAL_RCC_SPDIFRX_IS_CLK_DISABLED

__HAL_RCC_USART2_IS_CLK_DISABLED

__HAL_RCC_USART3_IS_CLK_DISABLED

__HAL_RCC_UART4_IS_CLK_DISABLED

__HAL_RCC_UART5_IS_CLK_DISABLED

__HAL_RCC_I2C1_IS_CLK_DISABLED

__HAL_RCC_I2C2_IS_CLK_DISABLED

__HAL_RCC_I2C3_IS_CLK_DISABLED

__HAL_RCC_CEC_IS_CLK_DISABLED

__HAL_RCC_DAC12_IS_CLK_DISABLED

__HAL_RCC_UART7_IS_CLK_DISABLED

__HAL_RCC_UART8_IS_CLK_DISABLED

__HAL_RCC_CRS_IS_CLK_DISABLED

__HAL_RCC_SWPMI1_IS_CLK_DISABLED

__HAL_RCC_OPAMP_IS_CLK_DISABLED

__HAL_RCC_MDIOS_IS_CLK_DISABLED

__HAL_RCC_FDCAN_IS_CLK_DISABLED

__HAL_RCC_TIM1_CLK_ENABLE

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_TIM8_CLK_ENABLE

__HAL_RCC_USART1_CLK_ENABLE

__HAL_RCC_USART6_CLK_ENABLE

__HAL_RCC_SPI1_CLK_ENABLE

__HAL_RCC_SPI4_CLK_ENABLE

__HAL_RCC_TIM15_CLK_ENABLE

__HAL_RCC_TIM16_CLK_ENABLE

__HAL_RCC_TIM17_CLK_ENABLE

__HAL_RCC_SPI5_CLK_ENABLE

__HAL_RCC_SAI1_CLK_ENABLE

__HAL_RCC_SAI2_CLK_ENABLE

__HAL_RCC_SAI3_CLK_ENABLE

__HAL_RCC_DFSDM1_CLK_ENABLE

__HAL_RCC_HRTIM1_CLK_ENABLE

__HAL_RCC_TIM1_CLK_DISABLE

__HAL_RCC_TIM8_CLK_DISABLE

__HAL_RCC_USART1_CLK_DISABLE

__HAL_RCC_USART6_CLK_DISABLE

__HAL_RCC_SPI1_CLK_DISABLE

__HAL_RCC_SPI4_CLK_DISABLE

__HAL_RCC_TIM15_CLK_DISABLE

__HAL_RCC_TIM16_CLK_DISABLE

__HAL_RCC_TIM17_CLK_DISABLE

__HAL_RCC_SPI5_CLK_DISABLE

__HAL_RCC_SAI1_CLK_DISABLE

__HAL_RCC_SAI2_CLK_DISABLE

__HAL_RCC_SAI3_CLK_DISABLE

__HAL_RCC_DFSDM1_CLK_DISABLE

__HAL_RCC_HRTIM1_CLK_DISABLE

__HAL_RCC_TIM1_IS_CLK_ENABLED

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_TIM8_IS_CLK_ENABLED

__HAL_RCC_USART1_IS_CLK_ENABLED

__HAL_RCC_USART6_IS_CLK_ENABLED

__HAL_RCC_SPI1_IS_CLK_ENABLED

__HAL_RCC_SPI4_IS_CLK_ENABLED

__HAL_RCC_TIM15_IS_CLK_ENABLED

__HAL_RCC_TIM16_IS_CLK_ENABLED

__HAL_RCC_TIM17_IS_CLK_ENABLED

__HAL_RCC_SPI5_IS_CLK_ENABLED

__HAL_RCC_SAI1_IS_CLK_ENABLED

__HAL_RCC_SAI2_IS_CLK_ENABLED

__HAL_RCC_SAI3_IS_CLK_ENABLED

__HAL_RCC_DFSDM1_IS_CLK_ENABLED

__HAL_RCC_HRTIM1_IS_CLK_ENABLED

__HAL_RCC_TIM1_IS_CLK_DISABLED

__HAL_RCC_TIM8_IS_CLK_DISABLED

__HAL_RCC_USART1_IS_CLK_DISABLED

__HAL_RCC_USART6_IS_CLK_DISABLED

__HAL_RCC_SPI1_IS_CLK_DISABLED

__HAL_RCC_SPI4_IS_CLK_DISABLED

__HAL_RCC_TIM15_IS_CLK_DISABLED

__HAL_RCC_TIM16_IS_CLK_DISABLED

__HAL_RCC_TIM17_IS_CLK_DISABLED

__HAL_RCC_SPI5_IS_CLK_DISABLED

__HAL_RCC_SAI1_IS_CLK_DISABLED

__HAL_RCC_SAI2_IS_CLK_DISABLED

__HAL_RCC_SAI3_IS_CLK_DISABLED

__HAL_RCC_DFSDM1_IS_CLK_DISABLED

__HAL_RCC_HRTIM1_IS_CLK_DISABLED

__HAL_RCC_SYSCFG_CLK_ENABLE

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_LPUART1_CLK_ENABLE

__HAL_RCC_SPI6_CLK_ENABLE

__HAL_RCC_I2C4_CLK_ENABLE

__HAL_RCC_LPTIM2_CLK_ENABLE

__HAL_RCC_LPTIM3_CLK_ENABLE

__HAL_RCC_LPTIM4_CLK_ENABLE

__HAL_RCC_LPTIM5_CLK_ENABLE

__HAL_RCC_COMP12_CLK_ENABLE

__HAL_RCC_VREF_CLK_ENABLE

__HAL_RCC_SAI4_CLK_ENABLE

__HAL_RCC_RTC_CLK_ENABLE

__HAL_RCC_SYSCFG_CLK_DISABLE

__HAL_RCC_LPUART1_CLK_DISABLE

__HAL_RCC_SPI6_CLK_DISABLE

__HAL_RCC_I2C4_CLK_DISABLE

__HAL_RCC_LPTIM2_CLK_DISABLE

__HAL_RCC_LPTIM3_CLK_DISABLE

__HAL_RCC_LPTIM4_CLK_DISABLE

__HAL_RCC_LPTIM5_CLK_DISABLE

__HAL_RCC_COMP12_CLK_DISABLE

__HAL_RCC_VREF_CLK_DISABLE

__HAL_RCC_RTC_CLK_DISABLE

__HAL_RCC_SAI4_CLK_DISABLE

__HAL_RCC_SYSCFG_IS_CLK_ENABLED

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_LPUART1_IS_CLK_ENABLED

__HAL_RCC_SPI6_IS_CLK_ENABLED

__HAL_RCC_I2C4_IS_CLK_ENABLED

__HAL_RCC_LPTIM2_IS_CLK_ENABLED

__HAL_RCC_LPTIM3_IS_CLK_ENABLED

__HAL_RCC_LPTIM4_IS_CLK_ENABLED

__HAL_RCC_LPTIM5_IS_CLK_ENABLED

__HAL_RCC_COMP12_IS_CLK_ENABLED

__HAL_RCC_VREF_IS_CLK_ENABLED

__HAL_RCC_RTC_IS_CLK_ENABLED

__HAL_RCC_SAI4_IS_CLK_ENABLED

__HAL_RCC_SYSCFG_IS_CLK_DISABLED

__HAL_RCC_LPUART1_IS_CLK_DISABLED

__HAL_RCC_SPI6_IS_CLK_DISABLED

__HAL_RCC_I2C4_IS_CLK_DISABLED

__HAL_RCC_LPTIM2_IS_CLK_DISABLED

__HAL_RCC_LPTIM3_IS_CLK_DISABLED

__HAL_RCC_LPTIM4_IS_CLK_DISABLED

__HAL_RCC_LPTIM5_IS_CLK_DISABLED

__HAL_RCC_COMP12_IS_CLK_DISABLED

__HAL_RCC_VREF_IS_CLK_DISABLED

__HAL_RCC_RTC_IS_CLK_DISABLED

__HAL_RCC_SAI4_IS_CLK_DISABLED

__HAL_RCC_C1_MDMA_CLK_ENABLE

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_C1_DMA2D_CLK_ENABLE

__HAL_RCC_C1_JPGDECEN_CLK_ENABLE

__HAL_RCC_C1_FMC_CLK_ENABLE

__HAL_RCC_C1_QSPI_CLK_ENABLE

__HAL_RCC_C1_SDMMC1_CLK_ENABLE

__HAL_RCC_C1_MDMA_CLK_DISABLE

__HAL_RCC_C1_DMA2D_CLK_DISABLE

__HAL_RCC_C1_JPGDECEN_CLK_DISABLE

__HAL_RCC_C1_FMC_CLK_DISABLE

__HAL_RCC_C1_QSPI_CLK_DISABLE

__HAL_RCC_C1_SDMMC1_CLK_DISABLE

__HAL_RCC_C1_DMA1_CLK_ENABLE

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_C1_DMA2_CLK_ENABLE

__HAL_RCC_C1_ADC12_CLK_ENABLE

__HAL_RCC_C1_ART_CLK_ENABLE

__HAL_RCC_C1_ETH1MAC_CLK_ENABLE

__HAL_RCC_C1_ETH1TX_CLK_ENABLE

__HAL_RCC_C1_ETH1RX_CLK_ENABLE

__HAL_RCC_C1_USB1_OTG_HS_CLK_ENABLE

__HAL_RCC_C1_USB1_OTG_HS_ULPI_CLK_ENABLE

__HAL_RCC_C1_USB2_OTG_FS_CLK_ENABLE

__HAL_RCC_C1_USB2_OTG_FS_ULPI_CLK_ENABLE

__HAL_RCC_C1_DMA1_CLK_DISABLE

__HAL_RCC_C1_DMA2_CLK_DISABLE

__HAL_RCC_C1_ADC12_CLK_DISABLE

__HAL_RCC_C1_ART_CLK_DISABLE

__HAL_RCC_C1_ETH1MAC_CLK_DISABLE

__HAL_RCC_C1_ETH1TX_CLK_DISABLE

__HAL_RCC_C1_ETH1RX_CLK_DISABLE

__HAL_RCC_C1_USB1_OTG_HS_CLK_DISABLE

__HAL_RCC_C1_USB1_OTG_HS_ULPI_CLK_DISABLE

__HAL_RCC_C1_USB2_OTG_FS_CLK_DISABLE

__HAL_RCC_C1_USB2_OTG_FS_ULPI_CLK_DISABLE

__HAL_RCC_C1_DCMI_CLK_ENABLE

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_C1_CRYP_CLK_ENABLE

__HAL_RCC_C1_HASH_CLK_ENABLE

__HAL_RCC_C1_RNG_CLK_ENABLE

__HAL_RCC_C1_SDMMC2_CLK_ENABLE

__HAL_RCC_C1_D2SRAM1_CLK_ENABLE

__HAL_RCC_C1_D2SRAM2_CLK_ENABLE

__HAL_RCC_C1_D2SRAM3_CLK_ENABLE

__HAL_RCC_C1_DCMI_CLK_DISABLE

__HAL_RCC_C1_CRYP_CLK_DISABLE

__HAL_RCC_C1_HASH_CLK_DISABLE

__HAL_RCC_C1_RNG_CLK_DISABLE

__HAL_RCC_C1_SDMMC2_CLK_DISABLE

__HAL_RCC_C1_D2SRAM1_CLK_DISABLE

__HAL_RCC_C1_D2SRAM2_CLK_DISABLE

__HAL_RCC_C1_D2SRAM3_CLK_DISABLE

__HAL_RCC_C1_GPIOA_CLK_ENABLE

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_C1_GPIOB_CLK_ENABLE

__HAL_RCC_C1_GPIOC_CLK_ENABLE

__HAL_RCC_C1_GPIOD_CLK_ENABLE

__HAL_RCC_C1_GPIOE_CLK_ENABLE

__HAL_RCC_C1_GPIOF_CLK_ENABLE

__HAL_RCC_C1_GPIOG_CLK_ENABLE

__HAL_RCC_C1_GPIOH_CLK_ENABLE

__HAL_RCC_C1_GPIOI_CLK_ENABLE

__HAL_RCC_C1_GPIOJ_CLK_ENABLE

__HAL_RCC_C1_GPIOK_CLK_ENABLE

__HAL_RCC_C1_CRC_CLK_ENABLE

__HAL_RCC_C1_BDMA_CLK_ENABLE

__HAL_RCC_C1_ADC3_CLK_ENABLE

__HAL_RCC_C1_HSEM_CLK_ENABLE

__HAL_RCC_C1_BKPRAM_CLK_ENABLE

__HAL_RCC_C1_GPIOA_CLK_DISABLE

__HAL_RCC_C1_GPIOB_CLK_DISABLE

__HAL_RCC_C1_GPIOC_CLK_DISABLE

__HAL_RCC_C1_GPIOD_CLK_DISABLE

__HAL_RCC_C1_GPIOE_CLK_DISABLE

__HAL_RCC_C1_GPIOF_CLK_DISABLE

__HAL_RCC_C1_GPIOG_CLK_DISABLE

__HAL_RCC_C1_GPIOH_CLK_DISABLE

__HAL_RCC_C1_GPIOI_CLK_DISABLE

__HAL_RCC_C1_GPIOJ_CLK_DISABLE

__HAL_RCC_C1_GPIOK_CLK_DISABLE

__HAL_RCC_C1_CRC_CLK_DISABLE

__HAL_RCC_C1_BDMA_CLK_DISABLE

__HAL_RCC_C1_ADC3_CLK_DISABLE

__HAL_RCC_C1_HSEM_CLK_DISABLE

__HAL_RCC_C1_BKPRAM_CLK_DISABLE

__HAL_RCC_C1_LTDC_CLK_ENABLE

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_C1_DSI_CLK_ENABLE

__HAL_RCC_C1_WWDG1_CLK_ENABLE

__HAL_RCC_C1_LTDC_CLK_DISABLE

__HAL_RCC_C1_DSI_CLK_DISABLE

__HAL_RCC_C1_WWDG1_CLK_DISABLE

__HAL_RCC_C1_TIM2_CLK_ENABLE

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_C1_TIM3_CLK_ENABLE

__HAL_RCC_C1_TIM4_CLK_ENABLE

__HAL_RCC_C1_TIM5_CLK_ENABLE

__HAL_RCC_C1_TIM6_CLK_ENABLE

__HAL_RCC_C1_TIM7_CLK_ENABLE

__HAL_RCC_C1_TIM12_CLK_ENABLE

__HAL_RCC_C1_TIM13_CLK_ENABLE

__HAL_RCC_C1_TIM14_CLK_ENABLE

__HAL_RCC_C1_LPTIM1_CLK_ENABLE

__HAL_RCC_C1_WWDG2_CLK_ENABLE

__HAL_RCC_C1_SPI2_CLK_ENABLE

__HAL_RCC_C1_SPI3_CLK_ENABLE

__HAL_RCC_C1_SPDIFRX_CLK_ENABLE

__HAL_RCC_C1_USART2_CLK_ENABLE

__HAL_RCC_C1_USART3_CLK_ENABLE

__HAL_RCC_C1_UART4_CLK_ENABLE

__HAL_RCC_C1_UART5_CLK_ENABLE

__HAL_RCC_C1_I2C1_CLK_ENABLE

__HAL_RCC_C1_I2C2_CLK_ENABLE

__HAL_RCC_C1_I2C3_CLK_ENABLE

__HAL_RCC_C1_CEC_CLK_ENABLE

__HAL_RCC_C1_DAC12_CLK_ENABLE

__HAL_RCC_C1_UART7_CLK_ENABLE

__HAL_RCC_C1_UART8_CLK_ENABLE

__HAL_RCC_C1_CRS_CLK_ENABLE

__HAL_RCC_C1_SWPMI_CLK_ENABLE

__HAL_RCC_C1_OPAMP_CLK_ENABLE

__HAL_RCC_C1_MDIOS_CLK_ENABLE

__HAL_RCC_C1_FDCAN_CLK_ENABLE

__HAL_RCC_C1_TIM2_CLK_DISABLE

__HAL_RCC_C1_TIM3_CLK_DISABLE

__HAL_RCC_C1_TIM4_CLK_DISABLE

__HAL_RCC_C1_TIM5_CLK_DISABLE

__HAL_RCC_C1_TIM6_CLK_DISABLE

__HAL_RCC_C1_TIM7_CLK_DISABLE

__HAL_RCC_C1_TIM12_CLK_DISABLE

__HAL_RCC_C1_TIM13_CLK_DISABLE

__HAL_RCC_C1_TIM14_CLK_DISABLE

__HAL_RCC_C1_LPTIM1_CLK_DISABLE

__HAL_RCC_C1_WWDG2_CLK_DISABLE

__HAL_RCC_C1_SPI2_CLK_DISABLE

__HAL_RCC_C1_SPI3_CLK_DISABLE

__HAL_RCC_C1_SPDIFRX_CLK_DISABLE

__HAL_RCC_C1_USART2_CLK_DISABLE

__HAL_RCC_C1_USART3_CLK_DISABLE

__HAL_RCC_C1_UART4_CLK_DISABLE

__HAL_RCC_C1_UART5_CLK_DISABLE

__HAL_RCC_C1_I2C1_CLK_DISABLE

__HAL_RCC_C1_I2C2_CLK_DISABLE

__HAL_RCC_C1_I2C3_CLK_DISABLE

__HAL_RCC_C1_CEC_CLK_DISABLE

__HAL_RCC_C1_DAC12_CLK_DISABLE

__HAL_RCC_C1_UART7_CLK_DISABLE

__HAL_RCC_C1_UART8_CLK_DISABLE

__HAL_RCC_C1_CRS_CLK_DISABLE

__HAL_RCC_C1_SWPMI_CLK_DISABLE

__HAL_RCC_C1_OPAMP_CLK_DISABLE

__HAL_RCC_C1_MDIOS_CLK_DISABLE

__HAL_RCC_C1_FDCAN_CLK_DISABLE

__HAL_RCC_C1_TIM1_CLK_ENABLE

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_C1_TIM8_CLK_ENABLE

__HAL_RCC_C1_USART1_CLK_ENABLE

__HAL_RCC_C1_USART6_CLK_ENABLE

__HAL_RCC_C1_SPI1_CLK_ENABLE

__HAL_RCC_C1_SPI4_CLK_ENABLE

__HAL_RCC_C1_TIM15_CLK_ENABLE

__HAL_RCC_C1_TIM16_CLK_ENABLE

__HAL_RCC_C1_TIM17_CLK_ENABLE

__HAL_RCC_C1_SPI5_CLK_ENABLE

__HAL_RCC_C1_SAI1_CLK_ENABLE

__HAL_RCC_C1_SAI2_CLK_ENABLE

__HAL_RCC_C1_SAI3_CLK_ENABLE

__HAL_RCC_C1_DFSDM1_CLK_ENABLE

__HAL_RCC_C1_HRTIM1_CLK_ENABLE

__HAL_RCC_C1_TIM1_CLK_DISABLE

__HAL_RCC_C1_TIM8_CLK_DISABLE

__HAL_RCC_C1_USART1_CLK_DISABLE

__HAL_RCC_C1_USART6_CLK_DISABLE

__HAL_RCC_C1_SPI1_CLK_DISABLE

__HAL_RCC_C1_SPI4_CLK_DISABLE

__HAL_RCC_C1_TIM15_CLK_DISABLE

__HAL_RCC_C1_TIM16_CLK_DISABLE

__HAL_RCC_C1_TIM17_CLK_DISABLE

__HAL_RCC_C1_SPI5_CLK_DISABLE

__HAL_RCC_C1_SAI1_CLK_DISABLE

__HAL_RCC_C1_SAI2_CLK_DISABLE

__HAL_RCC_C1_SAI3_CLK_DISABLE

__HAL_RCC_C1_DFSDM1_CLK_DISABLE

__HAL_RCC_C1_HRTIM1_CLK_DISABLE

__HAL_RCC_C1_SYSCFG_CLK_ENABLE

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_C1_LPUART1_CLK_ENABLE

__HAL_RCC_C1_SPI6_CLK_ENABLE

__HAL_RCC_C1_I2C4_CLK_ENABLE

__HAL_RCC_C1_LPTIM2_CLK_ENABLE

__HAL_RCC_C1_LPTIM3_CLK_ENABLE

__HAL_RCC_C1_LPTIM4_CLK_ENABLE

__HAL_RCC_C1_LPTIM5_CLK_ENABLE

__HAL_RCC_C1_COMP12_CLK_ENABLE

__HAL_RCC_C1_VREF_CLK_ENABLE

__HAL_RCC_C1_RTC_CLK_ENABLE

__HAL_RCC_C1_SAI4_CLK_ENABLE

__HAL_RCC_C1_SYSCFG_CLK_DISABLE

__HAL_RCC_C1_LPUART1_CLK_DISABLE

__HAL_RCC_C1_SPI6_CLK_DISABLE

__HAL_RCC_C1_I2C4_CLK_DISABLE

__HAL_RCC_C1_LPTIM2_CLK_DISABLE

__HAL_RCC_C1_LPTIM3_CLK_DISABLE

__HAL_RCC_C1_LPTIM4_CLK_DISABLE

__HAL_RCC_C1_LPTIM5_CLK_DISABLE

__HAL_RCC_C1_COMP12_CLK_DISABLE

__HAL_RCC_C1_VREF_CLK_DISABLE

__HAL_RCC_C1_RTC_CLK_DISABLE

__HAL_RCC_C1_SAI4_CLK_DISABLE

__HAL_RCC_C2_MDMA_CLK_ENABLE

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_C2_DMA2D_CLK_ENABLE

__HAL_RCC_C2_JPGDECEN_CLK_ENABLE

__HAL_RCC_FLASH_C2_ALLOCATE

__HAL_RCC_DTCM1_C2_ALLOCATE

__HAL_RCC_DTCM2_C2_ALLOCATE

__HAL_RCC_ITCM_C2_ALLOCATE

__HAL_RCC_D1SRAM1_C2_ALLOCATE

__HAL_RCC_C2_FMC_CLK_ENABLE

__HAL_RCC_C2_QSPI_CLK_ENABLE

__HAL_RCC_C2_SDMMC1_CLK_ENABLE

__HAL_RCC_C2_MDMA_CLK_DISABLE

__HAL_RCC_C2_DMA2D_CLK_DISABLE

__HAL_RCC_C2_JPGDECEN_CLK_DISABLE

__HAL_RCC_C2_FMC_CLK_DISABLE

__HAL_RCC_C2_QSPI_CLK_DISABLE

__HAL_RCC_C2_SDMMC1_CLK_DISABLE

__HAL_RCC_FLASH_C2_DEALLOCATE

__HAL_RCC_DTCM1_C2_DEALLOCATE

__HAL_RCC_DTCM2_C2_DEALLOCATE

__HAL_RCC_ITCM_C2_DEALLOCATE

__HAL_RCC_D1SRAM1_C2_DEALLOCATE

__HAL_RCC_C2_DMA1_CLK_ENABLE

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_C2_DMA2_CLK_ENABLE

__HAL_RCC_C2_ADC12_CLK_ENABLE

__HAL_RCC_C2_ART_CLK_ENABLE

__HAL_RCC_C2_ETH1MAC_CLK_ENABLE

__HAL_RCC_C2_ETH1TX_CLK_ENABLE

__HAL_RCC_C2_ETH1RX_CLK_ENABLE

__HAL_RCC_C2_USB1_OTG_HS_CLK_ENABLE

__HAL_RCC_C2_USB1_OTG_HS_ULPI_CLK_ENABLE

__HAL_RCC_C2_USB2_OTG_FS_CLK_ENABLE

__HAL_RCC_C2_USB2_OTG_FS_ULPI_CLK_ENABLE

__HAL_RCC_C2_DMA1_CLK_DISABLE

__HAL_RCC_C2_DMA2_CLK_DISABLE

__HAL_RCC_C2_ADC12_CLK_DISABLE

__HAL_RCC_C2_ART_CLK_DISABLE

__HAL_RCC_C2_ETH1MAC_CLK_DISABLE

__HAL_RCC_C2_ETH1TX_CLK_DISABLE

__HAL_RCC_C2_ETH1RX_CLK_DISABLE

__HAL_RCC_C2_USB1_OTG_HS_CLK_DISABLE

__HAL_RCC_C2_USB1_OTG_HS_ULPI_CLK_DISABLE

__HAL_RCC_C2_USB2_OTG_FS_CLK_DISABLE

__HAL_RCC_C2_USB2_OTG_FS_ULPI_CLK_DISABLE

__HAL_RCC_C2_DCMI_CLK_ENABLE

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_C2_CRYP_CLK_ENABLE

__HAL_RCC_C2_HASH_CLK_ENABLE

__HAL_RCC_C2_RNG_CLK_ENABLE

__HAL_RCC_C2_SDMMC2_CLK_ENABLE

__HAL_RCC_C2_D2SRAM1_CLK_ENABLE

__HAL_RCC_C2_D2SRAM2_CLK_ENABLE

__HAL_RCC_C2_D2SRAM3_CLK_ENABLE

__HAL_RCC_C2_DCMI_CLK_DISABLE

__HAL_RCC_C2_CRYP_CLK_DISABLE

__HAL_RCC_C2_HASH_CLK_DISABLE

__HAL_RCC_C2_RNG_CLK_DISABLE

__HAL_RCC_C2_SDMMC2_CLK_DISABLE

__HAL_RCC_C2_D2SRAM1_CLK_DISABLE

__HAL_RCC_C2_D2SRAM2_CLK_DISABLE

__HAL_RCC_C2_D2SRAM3_CLK_DISABLE

__HAL_RCC_C2_GPIOA_CLK_ENABLE

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_C2_GPIOB_CLK_ENABLE

__HAL_RCC_C2_GPIOC_CLK_ENABLE

__HAL_RCC_C2_GPIOD_CLK_ENABLE

__HAL_RCC_C2_GPIOE_CLK_ENABLE

__HAL_RCC_C2_GPIOF_CLK_ENABLE

__HAL_RCC_C2_GPIOG_CLK_ENABLE

__HAL_RCC_C2_GPIOH_CLK_ENABLE

__HAL_RCC_C2_GPIOI_CLK_ENABLE

__HAL_RCC_C2_GPIOJ_CLK_ENABLE

__HAL_RCC_C2_GPIOK_CLK_ENABLE

__HAL_RCC_C2_CRC_CLK_ENABLE

__HAL_RCC_C2_BDMA_CLK_ENABLE

__HAL_RCC_C2_ADC3_CLK_ENABLE

__HAL_RCC_C2_HSEM_CLK_ENABLE

__HAL_RCC_C2_BKPRAM_CLK_ENABLE

__HAL_RCC_C2_GPIOA_CLK_DISABLE

__HAL_RCC_C2_GPIOB_CLK_DISABLE

__HAL_RCC_C2_GPIOC_CLK_DISABLE

__HAL_RCC_C2_GPIOD_CLK_DISABLE

__HAL_RCC_C2_GPIOE_CLK_DISABLE

__HAL_RCC_C2_GPIOF_CLK_DISABLE

__HAL_RCC_C2_GPIOG_CLK_DISABLE

__HAL_RCC_C2_GPIOH_CLK_DISABLE

__HAL_RCC_C2_GPIOI_CLK_DISABLE

__HAL_RCC_C2_GPIOJ_CLK_DISABLE

__HAL_RCC_C2_GPIOK_CLK_DISABLE

__HAL_RCC_C2_CRC_CLK_DISABLE

__HAL_RCC_C2_BDMA_CLK_DISABLE

__HAL_RCC_C2_ADC3_CLK_DISABLE

__HAL_RCC_C2_HSEM_CLK_DISABLE

__HAL_RCC_C2_BKPRAM_CLK_DISABLE

__HAL_RCC_C2_LTDC_CLK_ENABLE

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_C2_DSI_CLK_ENABLE

__HAL_RCC_C2_WWDG1_CLK_ENABLE

__HAL_RCC_C2_LTDC_CLK_DISABLE

__HAL_RCC_C2_DSI_CLK_DISABLE

__HAL_RCC_C2_WWDG1_CLK_DISABLE

__HAL_RCC_C2_TIM2_CLK_ENABLE

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_C2_TIM3_CLK_ENABLE

__HAL_RCC_C2_TIM4_CLK_ENABLE

__HAL_RCC_C2_TIM5_CLK_ENABLE

__HAL_RCC_C2_TIM6_CLK_ENABLE

__HAL_RCC_C2_TIM7_CLK_ENABLE

__HAL_RCC_C2_TIM12_CLK_ENABLE

__HAL_RCC_C2_TIM13_CLK_ENABLE

__HAL_RCC_C2_TIM14_CLK_ENABLE

__HAL_RCC_C2_LPTIM1_CLK_ENABLE

__HAL_RCC_C2_WWDG2_CLK_ENABLE

__HAL_RCC_C2_SPI2_CLK_ENABLE

__HAL_RCC_C2_SPI3_CLK_ENABLE

__HAL_RCC_C2_SPDIFRX_CLK_ENABLE

__HAL_RCC_C2_USART2_CLK_ENABLE

__HAL_RCC_C2_USART3_CLK_ENABLE

__HAL_RCC_C2_UART4_CLK_ENABLE

__HAL_RCC_C2_UART5_CLK_ENABLE

__HAL_RCC_C2_I2C1_CLK_ENABLE

__HAL_RCC_C2_I2C2_CLK_ENABLE

__HAL_RCC_C2_I2C3_CLK_ENABLE

__HAL_RCC_C2_CEC_CLK_ENABLE

__HAL_RCC_C2_DAC12_CLK_ENABLE

__HAL_RCC_C2_UART7_CLK_ENABLE

__HAL_RCC_C2_UART8_CLK_ENABLE

__HAL_RCC_C2_CRS_CLK_ENABLE

__HAL_RCC_C2_SWPMI_CLK_ENABLE

__HAL_RCC_C2_OPAMP_CLK_ENABLE

__HAL_RCC_C2_MDIOS_CLK_ENABLE

__HAL_RCC_C2_FDCAN_CLK_ENABLE

__HAL_RCC_C2_TIM2_CLK_DISABLE

__HAL_RCC_C2_TIM3_CLK_DISABLE

__HAL_RCC_C2_TIM4_CLK_DISABLE

__HAL_RCC_C2_TIM5_CLK_DISABLE

__HAL_RCC_C2_TIM6_CLK_DISABLE

__HAL_RCC_C2_TIM7_CLK_DISABLE

__HAL_RCC_C2_TIM12_CLK_DISABLE

__HAL_RCC_C2_TIM13_CLK_DISABLE

__HAL_RCC_C2_TIM14_CLK_DISABLE

__HAL_RCC_C2_LPTIM1_CLK_DISABLE

__HAL_RCC_C2_WWDG2_CLK_DISABLE

__HAL_RCC_C2_SPI2_CLK_DISABLE

__HAL_RCC_C2_SPI3_CLK_DISABLE

__HAL_RCC_C2_SPDIFRX_CLK_DISABLE

__HAL_RCC_C2_USART2_CLK_DISABLE

__HAL_RCC_C2_USART3_CLK_DISABLE

__HAL_RCC_C2_UART4_CLK_DISABLE

__HAL_RCC_C2_UART5_CLK_DISABLE

__HAL_RCC_C2_I2C1_CLK_DISABLE

__HAL_RCC_C2_I2C2_CLK_DISABLE

__HAL_RCC_C2_I2C3_CLK_DISABLE

__HAL_RCC_C2_CEC_CLK_DISABLE

__HAL_RCC_C2_DAC12_CLK_DISABLE

__HAL_RCC_C2_UART7_CLK_DISABLE

__HAL_RCC_C2_UART8_CLK_DISABLE

__HAL_RCC_C2_CRS_CLK_DISABLE

__HAL_RCC_C2_SWPMI_CLK_DISABLE

__HAL_RCC_C2_OPAMP_CLK_DISABLE

__HAL_RCC_C2_MDIOS_CLK_DISABLE

__HAL_RCC_C2_FDCAN_CLK_DISABLE

__HAL_RCC_C2_TIM1_CLK_ENABLE

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_C2_TIM8_CLK_ENABLE

__HAL_RCC_C2_USART1_CLK_ENABLE

__HAL_RCC_C2_USART6_CLK_ENABLE

__HAL_RCC_C2_SPI1_CLK_ENABLE

__HAL_RCC_C2_SPI4_CLK_ENABLE

__HAL_RCC_C2_TIM15_CLK_ENABLE

__HAL_RCC_C2_TIM16_CLK_ENABLE

__HAL_RCC_C2_TIM17_CLK_ENABLE

__HAL_RCC_C2_SPI5_CLK_ENABLE

__HAL_RCC_C2_SAI1_CLK_ENABLE

__HAL_RCC_C2_SAI2_CLK_ENABLE

__HAL_RCC_C2_SAI3_CLK_ENABLE

__HAL_RCC_C2_DFSDM1_CLK_ENABLE

__HAL_RCC_C2_HRTIM1_CLK_ENABLE

__HAL_RCC_C2_TIM1_CLK_DISABLE

__HAL_RCC_C2_TIM8_CLK_DISABLE

__HAL_RCC_C2_USART1_CLK_DISABLE

__HAL_RCC_C2_USART6_CLK_DISABLE

__HAL_RCC_C2_SPI1_CLK_DISABLE

__HAL_RCC_C2_SPI4_CLK_DISABLE

__HAL_RCC_C2_TIM15_CLK_DISABLE

__HAL_RCC_C2_TIM16_CLK_DISABLE

__HAL_RCC_C2_TIM17_CLK_DISABLE

__HAL_RCC_C2_SPI5_CLK_DISABLE

__HAL_RCC_C2_SAI1_CLK_DISABLE

__HAL_RCC_C2_SAI2_CLK_DISABLE

__HAL_RCC_C2_SAI3_CLK_DISABLE

__HAL_RCC_C2_DFSDM1_CLK_DISABLE

__HAL_RCC_C2_HRTIM1_CLK_DISABLE

__HAL_RCC_C2_SYSCFG_CLK_ENABLE

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_C2_LPUART1_CLK_ENABLE

__HAL_RCC_C2_SPI6_CLK_ENABLE

__HAL_RCC_C2_I2C4_CLK_ENABLE

__HAL_RCC_C2_LPTIM2_CLK_ENABLE

__HAL_RCC_C2_LPTIM3_CLK_ENABLE

__HAL_RCC_C2_LPTIM4_CLK_ENABLE

__HAL_RCC_C2_LPTIM5_CLK_ENABLE

__HAL_RCC_C2_COMP12_CLK_ENABLE

__HAL_RCC_C2_VREF_CLK_ENABLE

__HAL_RCC_C2_RTC_CLK_ENABLE

__HAL_RCC_C2_SAI4_CLK_ENABLE

__HAL_RCC_C2_SYSCFG_CLK_DISABLE

__HAL_RCC_C2_LPUART1_CLK_DISABLE

__HAL_RCC_C2_SPI6_CLK_DISABLE

__HAL_RCC_C2_I2C4_CLK_DISABLE

__HAL_RCC_C2_LPTIM2_CLK_DISABLE

__HAL_RCC_C2_LPTIM3_CLK_DISABLE

__HAL_RCC_C2_LPTIM4_CLK_DISABLE

__HAL_RCC_C2_LPTIM5_CLK_DISABLE

__HAL_RCC_C2_COMP12_CLK_DISABLE

__HAL_RCC_C2_VREF_CLK_DISABLE

__HAL_RCC_C2_RTC_CLK_DISABLE

__HAL_RCC_C2_SAI4_CLK_DISABLE

__HAL_RCC_AHB3_FORCE_RESET

__HAL_RCC_MDMA_FORCE_RESET

__HAL_RCC_DMA2D_FORCE_RESET

__HAL_RCC_JPGDECRST_FORCE_RESET

__HAL_RCC_FMC_FORCE_RESET

__HAL_RCC_QSPI_FORCE_RESET

__HAL_RCC_SDMMC1_FORCE_RESET

__HAL_RCC_AHB3_RELEASE_RESET

__HAL_RCC_MDMA_RELEASE_RESET

__HAL_RCC_DMA2D_RELEASE_RESET

__HAL_RCC_JPGDECRST_RELEASE_RESET

__HAL_RCC_FMC_RELEASE_RESET

__HAL_RCC_QSPI_RELEASE_RESET

__HAL_RCC_SDMMC1_RELEASE_RESET

__HAL_RCC_AHB1_FORCE_RESET

__HAL_RCC_DMA1_FORCE_RESET

__HAL_RCC_DMA2_FORCE_RESET

__HAL_RCC_ADC12_FORCE_RESET

__HAL_RCC_ART_FORCE_RESET

__HAL_RCC_ETH1MAC_FORCE_RESET

__HAL_RCC_USB1_OTG_HS_FORCE_RESET

__HAL_RCC_USB2_OTG_FS_FORCE_RESET

__HAL_RCC_AHB1_RELEASE_RESET

__HAL_RCC_DMA1_RELEASE_RESET

__HAL_RCC_DMA2_RELEASE_RESET

__HAL_RCC_ADC12_RELEASE_RESET

__HAL_RCC_ART_RELEASE_RESET

__HAL_RCC_ETH1MAC_RELEASE_RESET

__HAL_RCC_USB1_OTG_HS_RELEASE_RESET

__HAL_RCC_USB2_OTG_FS_RELEASE_RESET

__HAL_RCC_AHB2_FORCE_RESET

__HAL_RCC_DCMI_FORCE_RESET

__HAL_RCC_CRYP_FORCE_RESET

__HAL_RCC_HASH_FORCE_RESET

__HAL_RCC_RNG_FORCE_RESET

__HAL_RCC_SDMMC2_FORCE_RESET

__HAL_RCC_AHB2_RELEASE_RESET

__HAL_RCC_DCMI_RELEASE_RESET

__HAL_RCC_CRYP_RELEASE_RESET

__HAL_RCC_HASH_RELEASE_RESET

__HAL_RCC_RNG_RELEASE_RESET

__HAL_RCC_SDMMC2_RELEASE_RESET

__HAL_RCC_AHB4_FORCE_RESET

__HAL_RCC_GPIOA_FORCE_RESET

__HAL_RCC_GPIOB_FORCE_RESET

__HAL_RCC_GPIOC_FORCE_RESET

__HAL_RCC_GPIOD_FORCE_RESET

__HAL_RCC_GPIOE_FORCE_RESET

__HAL_RCC_GPIOF_FORCE_RESET

__HAL_RCC_GPIOG_FORCE_RESET

__HAL_RCC_GPIOH_FORCE_RESET

__HAL_RCC_GPIOI_FORCE_RESET

__HAL_RCC_GPIOJ_FORCE_RESET

__HAL_RCC_GPIOK_FORCE_RESET

__HAL_RCC_CRC_FORCE_RESET

__HAL_RCC_BDMA_FORCE_RESET

__HAL_RCC_ADC3_FORCE_RESET

__HAL_RCC_HSEM_FORCE_RESET

__HAL_RCC_AHB4_RELEASE_RESET

__HAL_RCC_GPIOA_RELEASE_RESET

__HAL_RCC_GPIOB_RELEASE_RESET

__HAL_RCC_GPIOC_RELEASE_RESET

__HAL_RCC_GPIOD_RELEASE_RESET

__HAL_RCC_GPIOE_RELEASE_RESET

__HAL_RCC_GPIOF_RELEASE_RESET

__HAL_RCC_GPIOG_RELEASE_RESET

__HAL_RCC_GPIOH_RELEASE_RESET

__HAL_RCC_GPIOI_RELEASE_RESET

__HAL_RCC_GPIOJ_RELEASE_RESET

__HAL_RCC_GPIOK_RELEASE_RESET

__HAL_RCC_CRC_RELEASE_RESET

__HAL_RCC_BDMA_RELEASE_RESET

__HAL_RCC_ADC3_RELEASE_RESET

__HAL_RCC_HSEM_RELEASE_RESET

__HAL_RCC_APB3_FORCE_RESET

__HAL_RCC_LTDC_FORCE_RESET

__HAL_RCC_APB3_RELEASE_RESET

__HAL_RCC_LTDC_RELEASE_RESET

__HAL_RCC_APB1L_FORCE_RESET

__HAL_RCC_APB1H_FORCE_RESET

__HAL_RCC_TIM2_FORCE_RESET

__HAL_RCC_TIM3_FORCE_RESET

__HAL_RCC_TIM4_FORCE_RESET

__HAL_RCC_TIM5_FORCE_RESET

__HAL_RCC_TIM6_FORCE_RESET

__HAL_RCC_TIM7_FORCE_RESET

__HAL_RCC_TIM12_FORCE_RESET

__HAL_RCC_TIM13_FORCE_RESET

__HAL_RCC_TIM14_FORCE_RESET

__HAL_RCC_LPTIM1_FORCE_RESET

__HAL_RCC_SPI2_FORCE_RESET

__HAL_RCC_SPI3_FORCE_RESET

__HAL_RCC_SPDIFRX_FORCE_RESET

__HAL_RCC_USART2_FORCE_RESET

__HAL_RCC_USART3_FORCE_RESET

__HAL_RCC_UART4_FORCE_RESET

__HAL_RCC_UART5_FORCE_RESET

__HAL_RCC_I2C1_FORCE_RESET

__HAL_RCC_I2C2_FORCE_RESET

__HAL_RCC_I2C3_FORCE_RESET

__HAL_RCC_CEC_FORCE_RESET

__HAL_RCC_DAC12_FORCE_RESET

__HAL_RCC_UART7_FORCE_RESET

__HAL_RCC_UART8_FORCE_RESET

__HAL_RCC_CRS_FORCE_RESET

__HAL_RCC_SWPMI1_FORCE_RESET

__HAL_RCC_OPAMP_FORCE_RESET

__HAL_RCC_MDIOS_FORCE_RESET

__HAL_RCC_FDCAN_FORCE_RESET

__HAL_RCC_APB1L_RELEASE_RESET

__HAL_RCC_APB1H_RELEASE_RESET

__HAL_RCC_TIM2_RELEASE_RESET

__HAL_RCC_TIM3_RELEASE_RESET

__HAL_RCC_TIM4_RELEASE_RESET

__HAL_RCC_TIM5_RELEASE_RESET

__HAL_RCC_TIM6_RELEASE_RESET

__HAL_RCC_TIM7_RELEASE_RESET

__HAL_RCC_TIM12_RELEASE_RESET

__HAL_RCC_TIM13_RELEASE_RESET

__HAL_RCC_TIM14_RELEASE_RESET

__HAL_RCC_LPTIM1_RELEASE_RESET

__HAL_RCC_SPI2_RELEASE_RESET

__HAL_RCC_SPI3_RELEASE_RESET

__HAL_RCC_SPDIFRX_RELEASE_RESET

__HAL_RCC_USART2_RELEASE_RESET

__HAL_RCC_USART3_RELEASE_RESET

__HAL_RCC_UART4_RELEASE_RESET

__HAL_RCC_UART5_RELEASE_RESET

__HAL_RCC_I2C1_RELEASE_RESET

__HAL_RCC_I2C2_RELEASE_RESET

__HAL_RCC_I2C3_RELEASE_RESET

__HAL_RCC_CEC_RELEASE_RESET

__HAL_RCC_DAC12_RELEASE_RESET

__HAL_RCC_UART7_RELEASE_RESET

__HAL_RCC_UART8_RELEASE_RESET

__HAL_RCC_CRS_RELEASE_RESET

__HAL_RCC_SWPMI1_RELEASE_RESET

__HAL_RCC_OPAMP_RELEASE_RESET

__HAL_RCC_MDIOS_RELEASE_RESET

__HAL_RCC_FDCAN_RELEASE_RESET

__HAL_RCC_APB2_FORCE_RESET

__HAL_RCC_TIM1_FORCE_RESET

__HAL_RCC_TIM8_FORCE_RESET

__HAL_RCC_USART1_FORCE_RESET

__HAL_RCC_USART6_FORCE_RESET

__HAL_RCC_SPI1_FORCE_RESET

__HAL_RCC_SPI4_FORCE_RESET

__HAL_RCC_TIM15_FORCE_RESET

__HAL_RCC_TIM16_FORCE_RESET

__HAL_RCC_TIM17_FORCE_RESET

__HAL_RCC_SPI5_FORCE_RESET

__HAL_RCC_SAI1_FORCE_RESET

__HAL_RCC_SAI2_FORCE_RESET

__HAL_RCC_SAI3_FORCE_RESET

__HAL_RCC_DFSDM1_FORCE_RESET

__HAL_RCC_HRTIM1_FORCE_RESET

__HAL_RCC_APB2_RELEASE_RESET

__HAL_RCC_TIM1_RELEASE_RESET

__HAL_RCC_TIM8_RELEASE_RESET

__HAL_RCC_USART1_RELEASE_RESET

__HAL_RCC_USART6_RELEASE_RESET

__HAL_RCC_SPI1_RELEASE_RESET

__HAL_RCC_SPI4_RELEASE_RESET

__HAL_RCC_TIM15_RELEASE_RESET

__HAL_RCC_TIM16_RELEASE_RESET

__HAL_RCC_TIM17_RELEASE_RESET

__HAL_RCC_SPI5_RELEASE_RESET

__HAL_RCC_SAI1_RELEASE_RESET

__HAL_RCC_SAI2_RELEASE_RESET

__HAL_RCC_SAI3_RELEASE_RESET

__HAL_RCC_DFSDM1_RELEASE_RESET

__HAL_RCC_HRTIM1_RELEASE_RESET

__HAL_RCC_APB4_FORCE_RESET

__HAL_RCC_SYSCFG_FORCE_RESET

__HAL_RCC_LPUART1_FORCE_RESET

__HAL_RCC_SPI6_FORCE_RESET

__HAL_RCC_I2C4_FORCE_RESET

__HAL_RCC_LPTIM2_FORCE_RESET

__HAL_RCC_LPTIM3_FORCE_RESET

__HAL_RCC_LPTIM4_FORCE_RESET

__HAL_RCC_LPTIM5_FORCE_RESET

__HAL_RCC_COMP12_FORCE_RESET

__HAL_RCC_VREF_FORCE_RESET

__HAL_RCC_SAI4_FORCE_RESET

__HAL_RCC_APB4_RELEASE_RESET

__HAL_RCC_SYSCFG_RELEASE_RESET

__HAL_RCC_LPUART1_RELEASE_RESET

__HAL_RCC_SPI6_RELEASE_RESET

__HAL_RCC_I2C4_RELEASE_RESET

__HAL_RCC_LPTIM2_RELEASE_RESET

__HAL_RCC_LPTIM3_RELEASE_RESET

__HAL_RCC_LPTIM4_RELEASE_RESET

__HAL_RCC_LPTIM5_RELEASE_RESET

__HAL_RCC_COMP12_RELEASE_RESET

__HAL_RCC_VREF_RELEASE_RESET

__HAL_RCC_SAI4_RELEASE_RESET

__HAL_RCC_MDMA_CLK_SLEEP_ENABLE

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

__HAL_RCC_DMA2D_CLK_SLEEP_ENABLE

__HAL_RCC_JPGDEC_CLK_SLEEP_ENABLE

__HAL_RCC_FLASH_CLK_SLEEP_ENABLE

__HAL_RCC_FMC_CLK_SLEEP_ENABLE

__HAL_RCC_QSPI_CLK_SLEEP_ENABLE

__HAL_RCC_SDMMC1_CLK_SLEEP_ENABLE

__HAL_RCC_DTCM1_CLK_SLEEP_ENABLE

__HAL_RCC_DTCM2_CLK_SLEEP_ENABLE

__HAL_RCC_ITCM_CLK_SLEEP_ENABLE

__HAL_RCC_D1SRAM1_CLK_SLEEP_ENABLE

__HAL_RCC_AXISRAM_CLK_SLEEP_ENABLE

__HAL_RCC_MDMA_CLK_SLEEP_DISABLE

__HAL_RCC_DMA2D_CLK_SLEEP_DISABLE

__HAL_RCC_JPGDEC_CLK_SLEEP_DISABLE

__HAL_RCC_FLASH_CLK_SLEEP_DISABLE

__HAL_RCC_FMC_CLK_SLEEP_DISABLE

__HAL_RCC_QSPI_CLK_SLEEP_DISABLE

__HAL_RCC_SDMMC1_CLK_SLEEP_DISABLE

__HAL_RCC_DTCM1_CLK_SLEEP_DISABLE

__HAL_RCC_DTCM2_CLK_SLEEP_DISABLE

__HAL_RCC_ITCM_CLK_SLEEP_DISABLE

__HAL_RCC_D1SRAM1_CLK_SLEEP_DISABLE

__HAL_RCC_AXISRAM_CLK_SLEEP_DISABLE

__HAL_RCC_MDMA_IS_CLK_SLEEP_ENABLED

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

__HAL_RCC_DMA2D_IS_CLK_SLEEP_ENABLED

__HAL_RCC_JPGDEC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_FLASH_IS_CLK_SLEEP_ENABLED

__HAL_RCC_FMC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_QSPI_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SDMMC1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_DTCM1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_DTCM2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_ITCM_IS_CLK_SLEEP_ENABLED

__HAL_RCC_D1SRAM1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_MDMA_IS_CLK_SLEEP_DISABLED

__HAL_RCC_DMA2D_IS_CLK_SLEEP_DISABLED

__HAL_RCC_JPGDEC_IS_CLK_SLEEP_DISABLED

__HAL_RCC_FLASH_IS_CLK_SLEEP_DISABLED

__HAL_RCC_FMC_IS_CLK_SLEEP_DISABLED

__HAL_RCC_QSPI_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SDMMC1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_DTCM1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_DTCM2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_ITCM_IS_CLK_SLEEP_DISABLED

__HAL_RCC_D1SRAM1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_DMA1_CLK_SLEEP_ENABLE

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is ENABLEd again. By default, all peripheral clocks are ENABLEd during SLEEP mode.

__HAL_RCC_DMA2_CLK_SLEEP_ENABLE

__HAL_RCC_ADC12_CLK_SLEEP_ENABLE

__HAL_RCC_ETH1MAC_CLK_SLEEP_ENABLE

__HAL_RCC_ART_CLK_SLEEP_ENABLE

__HAL_RCC_ETH1TX_CLK_SLEEP_ENABLE

__HAL_RCC_ETH1RX_CLK_SLEEP_ENABLE

__HAL_RCC_USB1_OTG_HS_CLK_SLEEP_ENABLE

__HAL_RCC_USB1_OTG_HS_ULPI_CLK_SLEEP_ENABLE

__HAL_RCC_USB2_OTG_FS_CLK_SLEEP_ENABLE

__HAL_RCC_USB2_OTG_FS_ULPI_CLK_SLEEP_ENABLE

__HAL_RCC_DMA1_CLK_SLEEP_DISABLE

__HAL_RCC_DMA2_CLK_SLEEP_DISABLE

__HAL_RCC_ADC12_CLK_SLEEP_DISABLE

__HAL_RCC_ETH1MAC_CLK_SLEEP_DISABLE

__HAL_RCC_ART_CLK_SLEEP_DISABLE

__HAL_RCC_ETH1TX_CLK_SLEEP_DISABLE

__HAL_RCC_ETH1RX_CLK_SLEEP_DISABLE

__HAL_RCC_USB1_OTG_HS_CLK_SLEEP_DISABLE

__HAL_RCC_USB1_OTG_HS_ULPI_CLK_SLEEP_DISABLE

__HAL_RCC_USB2_OTG_FS_CLK_SLEEP_DISABLE

__HAL_RCC_USB2_OTG_FS_ULPI_CLK_SLEEP_DISABLE

__HAL_RCC_DMA1_IS_CLK_SLEEP_ENABLED

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

__HAL_RCC_DMA2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_ADC12_IS_CLK_SLEEP_ENABLED

__HAL_RCC_ETH1MAC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_ART_IS_CLK_SLEEP_ENABLED

__HAL_RCC_ETH1TX_IS_CLK_SLEEP_ENABLED

__HAL_RCC_ETH1RX_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USB1_OTG_HS_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USB1_OTG_HS_ULPI_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USB2_OTG_FS_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USB2_OTG_FS_ULPI_IS_CLK_SLEEP_ENABLED

__HAL_RCC_DMA1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_DMA2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_ADC12_IS_CLK_SLEEP_DISABLED

__HAL_RCC_ETH1MAC_IS_CLK_SLEEP_DISABLED

__HAL_RCC_ART_IS_CLK_SLEEP_DISABLED

__HAL_RCC_ETH1TX_IS_CLK_SLEEP_DISABLED

__HAL_RCC_ETH1RX_IS_CLK_SLEEP_DISABLED

__HAL_RCC_USB1_OTG_HS_IS_CLK_SLEEP_DISABLED

__HAL_RCC_USB1_OTG_HS_ULPI_IS_CLK_SLEEP_DISABLED

__HAL_RCC_USB2_OTG_FS_IS_CLK_SLEEP_DISABLED

__HAL_RCC_USB2_OTG_FS_ULPI_IS_CLK_SLEEP_DISABLED

__HAL_RCC_DCMI_CLK_SLEEP_ENABLE

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is ENABLEd again. By default, all peripheral clocks are ENABLEd during SLEEP mode.

__HAL_RCC_CRYP_CLK_SLEEP_ENABLE

__HAL_RCC_HASH_CLK_SLEEP_ENABLE

__HAL_RCC_RNG_CLK_SLEEP_ENABLE

__HAL_RCC_SDMMC2_CLK_SLEEP_ENABLE

__HAL_RCC_D2SRAM1_CLK_SLEEP_ENABLE

__HAL_RCC_D2SRAM2_CLK_SLEEP_ENABLE

__HAL_RCC_D2SRAM3_CLK_SLEEP_ENABLE

__HAL_RCC_DCMI_CLK_SLEEP_DISABLE

__HAL_RCC_CRYP_CLK_SLEEP_DISABLE

__HAL_RCC_HASH_CLK_SLEEP_DISABLE

__HAL_RCC_RNG_CLK_SLEEP_DISABLE

__HAL_RCC_SDMMC2_CLK_SLEEP_DISABLE

__HAL_RCC_D2SRAM1_CLK_SLEEP_DISABLE

__HAL_RCC_D2SRAM2_CLK_SLEEP_DISABLE

__HAL_RCC_D2SRAM3_CLK_SLEEP_DISABLE

__HAL_RCC_DCMI_IS_CLK_SLEEP_ENABLED

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

__HAL_RCC_CRYP_IS_CLK_SLEEP_ENABLED

__HAL_RCC_HASH_IS_CLK_SLEEP_ENABLED

__HAL_RCC_RNG_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SDMMC2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_D2SRAM1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_D2SRAM2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_D2SRAM3_IS_CLK_SLEEP_ENABLED

__HAL_RCC_DCMI_IS_CLK_SLEEP_DISABLED

__HAL_RCC_CRYP_IS_CLK_SLEEP_DISABLED

__HAL_RCC_HASH_IS_CLK_SLEEP_DISABLED

__HAL_RCC_RNG_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SDMMC2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_D2SRAM1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_D2SRAM2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_D2SRAM3_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOA_CLK_SLEEP_ENABLE

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is ENABLEd again. By default, all peripheral clocks are ENABLEd during SLEEP mode.

__HAL_RCC_GPIOB_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOC_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOD_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOE_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOF_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOG_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOH_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOI_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOJ_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOK_CLK_SLEEP_ENABLE

__HAL_RCC_CRC_CLK_SLEEP_ENABLE

__HAL_RCC_BDMA_CLK_SLEEP_ENABLE

__HAL_RCC_ADC3_CLK_SLEEP_ENABLE

__HAL_RCC_BKPRAM_CLK_SLEEP_ENABLE

__HAL_RCC_D3SRAM1_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOA_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOB_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOC_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOD_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOE_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOF_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOG_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOH_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOI_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOJ_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOK_CLK_SLEEP_DISABLE

__HAL_RCC_CRC_CLK_SLEEP_DISABLE

__HAL_RCC_BDMA_CLK_SLEEP_DISABLE

__HAL_RCC_ADC3_CLK_SLEEP_DISABLE

__HAL_RCC_BKPRAM_CLK_SLEEP_DISABLE

__HAL_RCC_D3SRAM1_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOA_IS_CLK_SLEEP_ENABLED

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

__HAL_RCC_GPIOB_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOD_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOE_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOF_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOG_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOH_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOI_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOJ_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOK_IS_CLK_SLEEP_ENABLED

__HAL_RCC_CRC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_BDMA_IS_CLK_SLEEP_ENABLED

__HAL_RCC_ADC3_IS_CLK_SLEEP_ENABLED

__HAL_RCC_BKPRAM_IS_CLK_SLEEP_ENABLED

__HAL_RCC_D3SRAM1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOA_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOB_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOC_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOD_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOE_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOF_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOG_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOH_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOI_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOJ_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOK_IS_CLK_SLEEP_DISABLED

__HAL_RCC_CRC_IS_CLK_SLEEP_DISABLED

__HAL_RCC_BDMA_IS_CLK_SLEEP_DISABLED

__HAL_RCC_ADC3_IS_CLK_SLEEP_DISABLED

__HAL_RCC_BKPRAM_IS_CLK_SLEEP_DISABLED

__HAL_RCC_D3SRAM1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_LTDC_CLK_SLEEP_ENABLE

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is ENABLEd again. By default, all peripheral clocks are ENABLEd during SLEEP mode.

__HAL_RCC_WWDG1_CLK_SLEEP_ENABLE

__HAL_RCC_LTDC_CLK_SLEEP_DISABLE

__HAL_RCC_WWDG1_CLK_SLEEP_DISABLE

__HAL_RCC_LTDC_IS_CLK_SLEEP_ENABLED

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

__HAL_RCC_WWDG1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_LTDC_IS_CLK_SLEEP_DISABLED

__HAL_RCC_WWDG1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM2_CLK_SLEEP_ENABLE

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is ENABLEd again. By default, all peripheral clocks are ENABLEd during SLEEP mode.

__HAL_RCC_TIM3_CLK_SLEEP_ENABLE

__HAL_RCC_TIM4_CLK_SLEEP_ENABLE

__HAL_RCC_TIM5_CLK_SLEEP_ENABLE

__HAL_RCC_TIM6_CLK_SLEEP_ENABLE

__HAL_RCC_TIM7_CLK_SLEEP_ENABLE

__HAL_RCC_TIM12_CLK_SLEEP_ENABLE

__HAL_RCC_TIM13_CLK_SLEEP_ENABLE

__HAL_RCC_TIM14_CLK_SLEEP_ENABLE

__HAL_RCC_LPTIM1_CLK_SLEEP_ENABLE

__HAL_RCC_WWDG2_CLK_SLEEP_ENABLE

__HAL_RCC_SPI2_CLK_SLEEP_ENABLE

__HAL_RCC_SPI3_CLK_SLEEP_ENABLE

__HAL_RCC_SPDIFRX_CLK_SLEEP_ENABLE

__HAL_RCC_USART2_CLK_SLEEP_ENABLE

__HAL_RCC_USART3_CLK_SLEEP_ENABLE

__HAL_RCC_UART4_CLK_SLEEP_ENABLE

__HAL_RCC_UART5_CLK_SLEEP_ENABLE

__HAL_RCC_I2C1_CLK_SLEEP_ENABLE

__HAL_RCC_I2C2_CLK_SLEEP_ENABLE

__HAL_RCC_I2C3_CLK_SLEEP_ENABLE

__HAL_RCC_CEC_CLK_SLEEP_ENABLE

__HAL_RCC_DAC12_CLK_SLEEP_ENABLE

__HAL_RCC_UART7_CLK_SLEEP_ENABLE

__HAL_RCC_UART8_CLK_SLEEP_ENABLE

__HAL_RCC_CRS_CLK_SLEEP_ENABLE

__HAL_RCC_SWPMI1_CLK_SLEEP_ENABLE

__HAL_RCC_OPAMP_CLK_SLEEP_ENABLE

__HAL_RCC_MDIOS_CLK_SLEEP_ENABLE

__HAL_RCC_FDCAN_CLK_SLEEP_ENABLE

__HAL_RCC_TIM2_CLK_SLEEP_DISABLE

__HAL_RCC_TIM3_CLK_SLEEP_DISABLE

__HAL_RCC_TIM4_CLK_SLEEP_DISABLE

__HAL_RCC_TIM5_CLK_SLEEP_DISABLE

__HAL_RCC_TIM6_CLK_SLEEP_DISABLE

__HAL_RCC_TIM7_CLK_SLEEP_DISABLE

__HAL_RCC_TIM12_CLK_SLEEP_DISABLE

__HAL_RCC_TIM13_CLK_SLEEP_DISABLE

__HAL_RCC_TIM14_CLK_SLEEP_DISABLE

__HAL_RCC_LPTIM1_CLK_SLEEP_DISABLE

__HAL_RCC_WWDG2_CLK_SLEEP_DISABLE

__HAL_RCC_SPI2_CLK_SLEEP_DISABLE

__HAL_RCC_SPI3_CLK_SLEEP_DISABLE

__HAL_RCC_SPDIFRX_CLK_SLEEP_DISABLE

__HAL_RCC_USART2_CLK_SLEEP_DISABLE

__HAL_RCC_USART3_CLK_SLEEP_DISABLE

__HAL_RCC_UART4_CLK_SLEEP_DISABLE

__HAL_RCC_UART5_CLK_SLEEP_DISABLE

__HAL_RCC_I2C1_CLK_SLEEP_DISABLE

__HAL_RCC_I2C2_CLK_SLEEP_DISABLE

__HAL_RCC_I2C3_CLK_SLEEP_DISABLE

__HAL_RCC_CEC_CLK_SLEEP_DISABLE

__HAL_RCC_DAC12_CLK_SLEEP_DISABLE

__HAL_RCC_UART7_CLK_SLEEP_DISABLE

__HAL_RCC_UART8_CLK_SLEEP_DISABLE

__HAL_RCC_CRS_CLK_SLEEP_DISABLE

__HAL_RCC_SWPMI1_CLK_SLEEP_DISABLE

__HAL_RCC_OPAMP_CLK_SLEEP_DISABLE

__HAL_RCC_MDIOS_CLK_SLEEP_DISABLE

__HAL_RCC_FDCAN_CLK_SLEEP_DISABLE

__HAL_RCC_TIM2_IS_CLK_SLEEP_ENABLED

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

__HAL_RCC_TIM3_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM4_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM5_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM6_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM7_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM12_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM13_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM14_IS_CLK_SLEEP_ENABLED

__HAL_RCC_LPTIM1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_WWDG2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SPI2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SPI3_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SPDIFRX_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USART2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USART3_IS_CLK_SLEEP_ENABLED

__HAL_RCC_UART4_IS_CLK_SLEEP_ENABLED

__HAL_RCC_UART5_IS_CLK_SLEEP_ENABLED

__HAL_RCC_I2C1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_I2C2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_I2C3_IS_CLK_SLEEP_ENABLED

__HAL_RCC_CEC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_DAC12_IS_CLK_SLEEP_ENABLED

__HAL_RCC_UART7_IS_CLK_SLEEP_ENABLED

__HAL_RCC_UART8_IS_CLK_SLEEP_ENABLED

__HAL_RCC_CRS_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SWPMI1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_OPAMP_IS_CLK_SLEEP_ENABLED

__HAL_RCC_MDIOS_IS_CLK_SLEEP_ENABLED

__HAL_RCC_FDCAN_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM3_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM4_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM5_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM6_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM7_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM12_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM13_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM14_IS_CLK_SLEEP_DISABLED

__HAL_RCC_LPTIM1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_WWDG2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SPI2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SPI3_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SPDIFRX_IS_CLK_SLEEP_DISABLED

__HAL_RCC_USART2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_USART3_IS_CLK_SLEEP_DISABLED

__HAL_RCC_UART4_IS_CLK_SLEEP_DISABLED

__HAL_RCC_UART5_IS_CLK_SLEEP_DISABLED

__HAL_RCC_I2C1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_I2C2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_I2C3_IS_CLK_SLEEP_DISABLED

__HAL_RCC_CEC_IS_CLK_SLEEP_DISABLED

__HAL_RCC_DAC12_IS_CLK_SLEEP_DISABLED

__HAL_RCC_UART7_IS_CLK_SLEEP_DISABLED

__HAL_RCC_UART8_IS_CLK_SLEEP_DISABLED

__HAL_RCC_CRS_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SWPMI1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_OPAMP_IS_CLK_SLEEP_DISABLED

__HAL_RCC_MDIOS_IS_CLK_SLEEP_DISABLED

__HAL_RCC_FDCAN_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM1_CLK_SLEEP_ENABLE

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is ENABLEd again. By default, all peripheral clocks are ENABLEd during SLEEP mode.

__HAL_RCC_TIM8_CLK_SLEEP_ENABLE

__HAL_RCC_USART1_CLK_SLEEP_ENABLE

__HAL_RCC_USART6_CLK_SLEEP_ENABLE

__HAL_RCC_SPI1_CLK_SLEEP_ENABLE

__HAL_RCC_SPI4_CLK_SLEEP_ENABLE

__HAL_RCC_TIM15_CLK_SLEEP_ENABLE

__HAL_RCC_TIM16_CLK_SLEEP_ENABLE

__HAL_RCC_TIM17_CLK_SLEEP_ENABLE

__HAL_RCC_SPI5_CLK_SLEEP_ENABLE

__HAL_RCC_SAI1_CLK_SLEEP_ENABLE

__HAL_RCC_SAI2_CLK_SLEEP_ENABLE

__HAL_RCC_SAI3_CLK_SLEEP_ENABLE

__HAL_RCC_DFSDM1_CLK_SLEEP_ENABLE

__HAL_RCC_HRTIM1_CLK_SLEEP_ENABLE

__HAL_RCC_TIM1_CLK_SLEEP_DISABLE

__HAL_RCC_TIM8_CLK_SLEEP_DISABLE

__HAL_RCC_USART1_CLK_SLEEP_DISABLE

__HAL_RCC_USART6_CLK_SLEEP_DISABLE

__HAL_RCC_SPI1_CLK_SLEEP_DISABLE

__HAL_RCC_SPI4_CLK_SLEEP_DISABLE

__HAL_RCC_TIM15_CLK_SLEEP_DISABLE

__HAL_RCC_TIM16_CLK_SLEEP_DISABLE

__HAL_RCC_TIM17_CLK_SLEEP_DISABLE

__HAL_RCC_SPI5_CLK_SLEEP_DISABLE

__HAL_RCC_SAI1_CLK_SLEEP_DISABLE

__HAL_RCC_SAI2_CLK_SLEEP_DISABLE

__HAL_RCC_SAI3_CLK_SLEEP_DISABLE

__HAL_RCC_DFSDM1_CLK_SLEEP_DISABLE

__HAL_RCC_HRTIM1_CLK_SLEEP_DISABLE

__HAL_RCC_TIM1_IS_CLK_SLEEP_ENABLED

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

__HAL_RCC_TIM8_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USART1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USART6_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SPI1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SPI4_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM15_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM16_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM17_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SPI5_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SAI1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SAI2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SAI3_IS_CLK_SLEEP_ENABLED

__HAL_RCC_DFSDM1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_HRTIM1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM8_IS_CLK_SLEEP_DISABLED

__HAL_RCC_USART1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_USART6_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SPI1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SPI4_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM15_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM16_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM17_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SPI5_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SAI1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SAI2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SAI3_IS_CLK_SLEEP_DISABLED

__HAL_RCC_DFSDM1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_HRTIM1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SYSCFG_CLK_SLEEP_ENABLE

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is ENABLEd again. By default, all peripheral clocks are ENABLEd during SLEEP mode.

__HAL_RCC_LPUART1_CLK_SLEEP_ENABLE

__HAL_RCC_SPI6_CLK_SLEEP_ENABLE

__HAL_RCC_I2C4_CLK_SLEEP_ENABLE

__HAL_RCC_LPTIM2_CLK_SLEEP_ENABLE

__HAL_RCC_LPTIM3_CLK_SLEEP_ENABLE

__HAL_RCC_LPTIM4_CLK_SLEEP_ENABLE

__HAL_RCC_LPTIM5_CLK_SLEEP_ENABLE

__HAL_RCC_COMP12_CLK_SLEEP_ENABLE

__HAL_RCC_VREF_CLK_SLEEP_ENABLE

__HAL_RCC_RTC_CLK_SLEEP_ENABLE

__HAL_RCC_SAI4_CLK_SLEEP_ENABLE

__HAL_RCC_SYSCFG_CLK_SLEEP_DISABLE

__HAL_RCC_LPUART1_CLK_SLEEP_DISABLE

__HAL_RCC_SPI6_CLK_SLEEP_DISABLE

__HAL_RCC_I2C4_CLK_SLEEP_DISABLE

__HAL_RCC_LPTIM2_CLK_SLEEP_DISABLE

__HAL_RCC_LPTIM3_CLK_SLEEP_DISABLE

__HAL_RCC_LPTIM4_CLK_SLEEP_DISABLE

__HAL_RCC_LPTIM5_CLK_SLEEP_DISABLE

__HAL_RCC_COMP12_CLK_SLEEP_DISABLE

__HAL_RCC_VREF_CLK_SLEEP_DISABLE

__HAL_RCC_RTC_CLK_SLEEP_DISABLE

__HAL_RCC_SAI4_CLK_SLEEP_DISABLE

__HAL_RCC_SYSCFG_IS_CLK_SLEEP_ENABLED

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

__HAL_RCC_LPUART1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SPI6_IS_CLK_SLEEP_ENABLED

__HAL_RCC_I2C4_IS_CLK_SLEEP_ENABLED

__HAL_RCC_LPTIM2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_LPTIM3_IS_CLK_SLEEP_ENABLED

__HAL_RCC_LPTIM4_IS_CLK_SLEEP_ENABLED

__HAL_RCC_LPTIM5_IS_CLK_SLEEP_ENABLED

__HAL_RCC_COMP12_IS_CLK_SLEEP_ENABLED

__HAL_RCC_VREF_IS_CLK_SLEEP_ENABLED

__HAL_RCC_RTC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SAI4_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SYSCFG_IS_CLK_SLEEP_DISABLED

__HAL_RCC_LPUART1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SPI6_IS_CLK_SLEEP_DISABLED

__HAL_RCC_I2C4_IS_CLK_SLEEP_DISABLED

__HAL_RCC_LPTIM2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_LPTIM3_IS_CLK_SLEEP_DISABLED

__HAL_RCC_LPTIM4_IS_CLK_SLEEP_DISABLED

__HAL_RCC_LPTIM5_IS_CLK_SLEEP_DISABLED

__HAL_RCC_COMP12_IS_CLK_SLEEP_DISABLED

__HAL_RCC_VREF_IS_CLK_SLEEP_DISABLED

__HAL_RCC_RTC_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SAI4_IS_CLK_SLEEP_DISABLED

__HAL_RCC_C1_MDMA_CLK_SLEEP_ENABLE

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

__HAL_RCC_C1_DMA2D_CLK_SLEEP_ENABLE

__HAL_RCC_C1_JPGDEC_CLK_SLEEP_ENABLE

__HAL_RCC_C1_FLASH_CLK_SLEEP_ENABLE

__HAL_RCC_C1_FMC_CLK_SLEEP_ENABLE

__HAL_RCC_C1_QSPI_CLK_SLEEP_ENABLE

__HAL_RCC_C1_SDMMC1_CLK_SLEEP_ENABLE

__HAL_RCC_C1_DTCM1_CLK_SLEEP_ENABLE

__HAL_RCC_C1_DTCM2_CLK_SLEEP_ENABLE

__HAL_RCC_C1_ITCM_CLK_SLEEP_ENABLE

__HAL_RCC_C1_D1SRAM1_CLK_SLEEP_ENABLE

__HAL_RCC_C1_MDMA_CLK_SLEEP_DISABLE

__HAL_RCC_C1_DMA2D_CLK_SLEEP_DISABLE

__HAL_RCC_C1_JPGDEC_CLK_SLEEP_DISABLE

__HAL_RCC_C1_FLASH_CLK_SLEEP_DISABLE

__HAL_RCC_C1_FMC_CLK_SLEEP_DISABLE

__HAL_RCC_C1_QSPI_CLK_SLEEP_DISABLE

__HAL_RCC_C1_SDMMC1_CLK_SLEEP_DISABLE

__HAL_RCC_C1_DTCM1_CLK_SLEEP_DISABLE

__HAL_RCC_C1_DTCM2_CLK_SLEEP_DISABLE

__HAL_RCC_C1_ITCM_CLK_SLEEP_DISABLE

__HAL_RCC_C1_D1SRAM1_CLK_SLEEP_DISABLE

__HAL_RCC_C1_DMA1_CLK_SLEEP_ENABLE

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is ENABLEd again. By default, all peripheral clocks are ENABLEd during SLEEP mode.

__HAL_RCC_C1_DMA2_CLK_SLEEP_ENABLE

__HAL_RCC_C1_ADC12_CLK_SLEEP_ENABLE

__HAL_RCC_C1_ETH1MAC_CLK_SLEEP_ENABLE

__HAL_RCC_C1_ETH1TX_CLK_SLEEP_ENABLE

__HAL_RCC_C1_ETH1RX_CLK_SLEEP_ENABLE

__HAL_RCC_C1_USB1_OTG_HS_CLK_SLEEP_ENABLE

__HAL_RCC_C1_USB1_OTG_HS_ULPI_CLK_SLEEP_ENABLE

__HAL_RCC_C1_USB2_OTG_FS_CLK_SLEEP_ENABLE

__HAL_RCC_C1_USB2_OTG_FS_ULPI_CLK_SLEEP_ENABLE

__HAL_RCC_C1_DMA1_CLK_SLEEP_DISABLE

__HAL_RCC_C1_DMA2_CLK_SLEEP_DISABLE

__HAL_RCC_C1_ADC12_CLK_SLEEP_DISABLE

__HAL_RCC_C1_ETH1MAC_CLK_SLEEP_DISABLE

__HAL_RCC_C1_ETH1TX_CLK_SLEEP_DISABLE

__HAL_RCC_C1_ETH1RX_CLK_SLEEP_DISABLE

__HAL_RCC_C1_USB1_OTG_HS_CLK_SLEEP_DISABLE

__HAL_RCC_C1_USB1_OTG_HS_ULPI_CLK_SLEEP_DISABLE

__HAL_RCC_C1_USB2_OTG_FS_CLK_SLEEP_DISABLE

__HAL_RCC_C1_USB2_OTG_FS_ULPI_CLK_SLEEP_DISABLE

__HAL_RCC_C1_DCMI_CLK_SLEEP_ENABLE

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is ENABLEd again. By default, all peripheral clocks are ENABLEd during SLEEP mode.

__HAL_RCC_C1_CRYP_CLK_SLEEP_ENABLE

__HAL_RCC_C1_HASH_CLK_SLEEP_ENABLE

__HAL_RCC_C1_RNG_CLK_SLEEP_ENABLE

__HAL_RCC_C1_SDMMC2_CLK_SLEEP_ENABLE

__HAL_RCC_C1_D2SRAM1_CLK_SLEEP_ENABLE

__HAL_RCC_C1_D2SRAM2_CLK_SLEEP_ENABLE

__HAL_RCC_C1_D2SRAM3_CLK_SLEEP_ENABLE

__HAL_RCC_C1_DCMI_CLK_SLEEP_DISABLE

__HAL_RCC_C1_CRYP_CLK_SLEEP_DISABLE

__HAL_RCC_C1_HASH_CLK_SLEEP_DISABLE

__HAL_RCC_C1_RNG_CLK_SLEEP_DISABLE

__HAL_RCC_C1_SDMMC2_CLK_SLEEP_DISABLE

__HAL_RCC_C1_D2SRAM1_CLK_SLEEP_DISABLE

__HAL_RCC_C1_D2SRAM2_CLK_SLEEP_DISABLE

__HAL_RCC_C1_D2SRAM3_CLK_SLEEP_DISABLE

__HAL_RCC_C1_GPIOA_CLK_SLEEP_ENABLE

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is ENABLEd again. By default, all peripheral clocks are ENABLEd during SLEEP mode.

__HAL_RCC_C1_GPIOB_CLK_SLEEP_ENABLE

__HAL_RCC_C1_GPIOC_CLK_SLEEP_ENABLE

__HAL_RCC_C1_GPIOD_CLK_SLEEP_ENABLE

__HAL_RCC_C1_GPIOE_CLK_SLEEP_ENABLE

__HAL_RCC_C1_GPIOF_CLK_SLEEP_ENABLE

__HAL_RCC_C1_GPIOG_CLK_SLEEP_ENABLE

__HAL_RCC_C1_GPIOH_CLK_SLEEP_ENABLE

__HAL_RCC_C1_GPIOI_CLK_SLEEP_ENABLE

__HAL_RCC_C1_GPIOJ_CLK_SLEEP_ENABLE

__HAL_RCC_C1_GPIOK_CLK_SLEEP_ENABLE

__HAL_RCC_C1_CRC_CLK_SLEEP_ENABLE

__HAL_RCC_C1_BDMA_CLK_SLEEP_ENABLE

__HAL_RCC_C1_ADC3_CLK_SLEEP_ENABLE

__HAL_RCC_C1_BKPRAM_CLK_SLEEP_ENABLE

__HAL_RCC_C1_D3SRAM1_CLK_SLEEP_ENABLE

__HAL_RCC_C1_GPIOA_CLK_SLEEP_DISABLE

__HAL_RCC_C1_GPIOB_CLK_SLEEP_DISABLE

__HAL_RCC_C1_GPIOC_CLK_SLEEP_DISABLE

__HAL_RCC_C1_GPIOD_CLK_SLEEP_DISABLE

__HAL_RCC_C1_GPIOE_CLK_SLEEP_DISABLE

__HAL_RCC_C1_GPIOF_CLK_SLEEP_DISABLE

__HAL_RCC_C1_GPIOG_CLK_SLEEP_DISABLE

__HAL_RCC_C1_GPIOH_CLK_SLEEP_DISABLE

__HAL_RCC_C1_GPIOI_CLK_SLEEP_DISABLE

__HAL_RCC_C1_GPIOJ_CLK_SLEEP_DISABLE

__HAL_RCC_C1_GPIOK_CLK_SLEEP_DISABLE

__HAL_RCC_C1_CRC_CLK_SLEEP_DISABLE

__HAL_RCC_C1_BDMA_CLK_SLEEP_DISABLE

__HAL_RCC_C1_ADC3_CLK_SLEEP_DISABLE

__HAL_RCC_C1_BKPRAM_CLK_SLEEP_DISABLE

__HAL_RCC_C1_D3SRAM1_CLK_SLEEP_DISABLE

__HAL_RCC_C1_LTDC_CLK_SLEEP_ENABLE

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is ENABLEd again. By default, all peripheral clocks are ENABLEd during SLEEP mode.

__HAL_RCC_C1_DSI_CLK_SLEEP_ENABLE

__HAL_RCC_C1_WWDG1_CLK_SLEEP_ENABLE

__HAL_RCC_C1_LTDC_CLK_SLEEP_DISABLE

__HAL_RCC_C1_DSI_CLK_SLEEP_DISABLE

__HAL_RCC_C1_WWDG1_CLK_SLEEP_DISABLE

__HAL_RCC_C1_TIM2_CLK_SLEEP_ENABLE

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is ENABLEd again. By default, all peripheral clocks are ENABLEd during SLEEP mode.

__HAL_RCC_C1_TIM3_CLK_SLEEP_ENABLE

__HAL_RCC_C1_TIM4_CLK_SLEEP_ENABLE

__HAL_RCC_C1_TIM5_CLK_SLEEP_ENABLE

__HAL_RCC_C1_TIM6_CLK_SLEEP_ENABLE

__HAL_RCC_C1_TIM7_CLK_SLEEP_ENABLE

__HAL_RCC_C1_TIM12_CLK_SLEEP_ENABLE

__HAL_RCC_C1_TIM13_CLK_SLEEP_ENABLE

__HAL_RCC_C1_TIM14_CLK_SLEEP_ENABLE

__HAL_RCC_C1_LPTIM1_CLK_SLEEP_ENABLE

__HAL_RCC_C1_WWDG2_CLK_SLEEP_ENABLE

__HAL_RCC_C1_SPI2_CLK_SLEEP_ENABLE

__HAL_RCC_C1_SPI3_CLK_SLEEP_ENABLE

__HAL_RCC_C1_SPDIFRX_CLK_SLEEP_ENABLE

__HAL_RCC_C1_USART2_CLK_SLEEP_ENABLE

__HAL_RCC_C1_USART3_CLK_SLEEP_ENABLE

__HAL_RCC_C1_UART4_CLK_SLEEP_ENABLE

__HAL_RCC_C1_UART5_CLK_SLEEP_ENABLE

__HAL_RCC_C1_I2C1_CLK_SLEEP_ENABLE

__HAL_RCC_C1_I2C2_CLK_SLEEP_ENABLE

__HAL_RCC_C1_I2C3_CLK_SLEEP_ENABLE

__HAL_RCC_C1_CEC_CLK_SLEEP_ENABLE

__HAL_RCC_C1_DAC12_CLK_SLEEP_ENABLE

__HAL_RCC_C1_UART7_CLK_SLEEP_ENABLE

__HAL_RCC_C1_UART8_CLK_SLEEP_ENABLE

__HAL_RCC_C1_CRS_CLK_SLEEP_ENABLE

__HAL_RCC_C1_SWPMI_CLK_SLEEP_ENABLE

__HAL_RCC_C1_OPAMP_CLK_SLEEP_ENABLE

__HAL_RCC_C1_MDIOS_CLK_SLEEP_ENABLE

__HAL_RCC_C1_FDCAN_CLK_SLEEP_ENABLE

__HAL_RCC_C1_TIM2_CLK_SLEEP_DISABLE

__HAL_RCC_C1_TIM3_CLK_SLEEP_DISABLE

__HAL_RCC_C1_TIM4_CLK_SLEEP_DISABLE

__HAL_RCC_C1_TIM5_CLK_SLEEP_DISABLE

__HAL_RCC_C1_TIM6_CLK_SLEEP_DISABLE

__HAL_RCC_C1_TIM7_CLK_SLEEP_DISABLE

__HAL_RCC_C1_TIM12_CLK_SLEEP_DISABLE

__HAL_RCC_C1_TIM13_CLK_SLEEP_DISABLE

__HAL_RCC_C1_TIM14_CLK_SLEEP_DISABLE

__HAL_RCC_C1_LPTIM1_CLK_SLEEP_DISABLE

__HAL_RCC_C1_WWDG2_CLK_SLEEP_DISABLE

__HAL_RCC_C1_SPI2_CLK_SLEEP_DISABLE

__HAL_RCC_C1_SPI3_CLK_SLEEP_DISABLE

__HAL_RCC_C1_SPDIFRX_CLK_SLEEP_DISABLE

__HAL_RCC_C1_USART2_CLK_SLEEP_DISABLE

__HAL_RCC_C1_USART3_CLK_SLEEP_DISABLE

__HAL_RCC_C1_UART4_CLK_SLEEP_DISABLE

__HAL_RCC_C1_UART5_CLK_SLEEP_DISABLE

__HAL_RCC_C1_I2C1_CLK_SLEEP_DISABLE

__HAL_RCC_C1_I2C2_CLK_SLEEP_DISABLE

__HAL_RCC_C1_I2C3_CLK_SLEEP_DISABLE

__HAL_RCC_C1_CEC_CLK_SLEEP_DISABLE

__HAL_RCC_C1_DAC12_CLK_SLEEP_DISABLE

__HAL_RCC_C1_UART7_CLK_SLEEP_DISABLE

__HAL_RCC_C1_UART8_CLK_SLEEP_DISABLE

__HAL_RCC_C1_CRS_CLK_SLEEP_DISABLE

__HAL_RCC_C1_SWPMI_CLK_SLEEP_DISABLE

__HAL_RCC_C1_OPAMP_CLK_SLEEP_DISABLE

__HAL_RCC_C1_MDIOS_CLK_SLEEP_DISABLE

__HAL_RCC_C1_FDCAN_CLK_SLEEP_DISABLE

__HAL_RCC_C1_TIM1_CLK_SLEEP_ENABLE

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is ENABLEd again. By default, all peripheral clocks are ENABLEd during SLEEP mode.

__HAL_RCC_C1_TIM8_CLK_SLEEP_ENABLE

__HAL_RCC_C1_USART1_CLK_SLEEP_ENABLE

__HAL_RCC_C1_USART6_CLK_SLEEP_ENABLE

__HAL_RCC_C1_SPI1_CLK_SLEEP_ENABLE

__HAL_RCC_C1_SPI4_CLK_SLEEP_ENABLE

__HAL_RCC_C1_TIM15_CLK_SLEEP_ENABLE

__HAL_RCC_C1_TIM16_CLK_SLEEP_ENABLE

__HAL_RCC_C1_TIM17_CLK_SLEEP_ENABLE

__HAL_RCC_C1_SPI5_CLK_SLEEP_ENABLE

__HAL_RCC_C1_SAI1_CLK_SLEEP_ENABLE

__HAL_RCC_C1_SAI2_CLK_SLEEP_ENABLE

__HAL_RCC_C1_SAI3_CLK_SLEEP_ENABLE

__HAL_RCC_C1_DFSDM1_CLK_SLEEP_ENABLE

__HAL_RCC_C1_HRTIM1_CLK_SLEEP_ENABLE

__HAL_RCC_C1_TIM1_CLK_SLEEP_DISABLE

__HAL_RCC_C1_TIM8_CLK_SLEEP_DISABLE

__HAL_RCC_C1_USART1_CLK_SLEEP_DISABLE

__HAL_RCC_C1_USART6_CLK_SLEEP_DISABLE

__HAL_RCC_C1_SPI1_CLK_SLEEP_DISABLE

__HAL_RCC_C1_SPI4_CLK_SLEEP_DISABLE

__HAL_RCC_C1_TIM15_CLK_SLEEP_DISABLE

__HAL_RCC_C1_TIM16_CLK_SLEEP_DISABLE

__HAL_RCC_C1_TIM17_CLK_SLEEP_DISABLE

__HAL_RCC_C1_SPI5_CLK_SLEEP_DISABLE

__HAL_RCC_C1_SAI1_CLK_SLEEP_DISABLE

__HAL_RCC_C1_SAI2_CLK_SLEEP_DISABLE

__HAL_RCC_C1_SAI3_CLK_SLEEP_DISABLE

__HAL_RCC_C1_DFSDM1_CLK_SLEEP_DISABLE

__HAL_RCC_C1_HRTIM1_CLK_SLEEP_DISABLE

__HAL_RCC_C1_SYSCFG_CLK_SLEEP_ENABLE

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is ENABLEd again. By default, all peripheral clocks are ENABLEd during SLEEP mode.

__HAL_RCC_C1_LPUART1_CLK_SLEEP_ENABLE

__HAL_RCC_C1_SPI6_CLK_SLEEP_ENABLE

__HAL_RCC_C1_I2C4_CLK_SLEEP_ENABLE

__HAL_RCC_C1_LPTIM2_CLK_SLEEP_ENABLE

__HAL_RCC_C1_LPTIM3_CLK_SLEEP_ENABLE

__HAL_RCC_C1_LPTIM4_CLK_SLEEP_ENABLE

__HAL_RCC_C1_LPTIM5_CLK_SLEEP_ENABLE

__HAL_RCC_C1_COMP12_CLK_SLEEP_ENABLE

__HAL_RCC_C1_VREF_CLK_SLEEP_ENABLE

__HAL_RCC_C1_SAI4_CLK_SLEEP_ENABLE

__HAL_RCC_C1_RTC_CLK_SLEEP_ENABLE

__HAL_RCC_C1_SYSCFG_CLK_SLEEP_DISABLE

__HAL_RCC_C1_LPUART1_CLK_SLEEP_DISABLE

__HAL_RCC_C1_SPI6_CLK_SLEEP_DISABLE

__HAL_RCC_C1_I2C4_CLK_SLEEP_DISABLE

__HAL_RCC_C1_LPTIM2_CLK_SLEEP_DISABLE

__HAL_RCC_C1_LPTIM3_CLK_SLEEP_DISABLE

__HAL_RCC_C1_LPTIM4_CLK_SLEEP_DISABLE

__HAL_RCC_C1_LPTIM5_CLK_SLEEP_DISABLE

__HAL_RCC_C1_COMP12_CLK_SLEEP_DISABLE

__HAL_RCC_C1_VREF_CLK_SLEEP_DISABLE

__HAL_RCC_C1_SAI4_CLK_SLEEP_DISABLE

__HAL_RCC_C1_RTC_CLK_SLEEP_DISABLE

__HAL_RCC_C2_MDMA_CLK_SLEEP_ENABLE

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

__HAL_RCC_C2_DMA2D_CLK_SLEEP_ENABLE

__HAL_RCC_C2_JPGDEC_CLK_SLEEP_ENABLE

__HAL_RCC_C2_FLASH_CLK_SLEEP_ENABLE

__HAL_RCC_C2_FMC_CLK_SLEEP_ENABLE

__HAL_RCC_C2_QSPI_CLK_SLEEP_ENABLE

__HAL_RCC_C2_SDMMC1_CLK_SLEEP_ENABLE

__HAL_RCC_C2_DTCM1_CLK_SLEEP_ENABLE

__HAL_RCC_C2_DTCM2_CLK_SLEEP_ENABLE

__HAL_RCC_C2_ITCM_CLK_SLEEP_ENABLE

__HAL_RCC_C2_D1SRAM1_CLK_SLEEP_ENABLE

__HAL_RCC_C2_MDMA_CLK_SLEEP_DISABLE

__HAL_RCC_C2_DMA2D_CLK_SLEEP_DISABLE

__HAL_RCC_C2_JPGDEC_CLK_SLEEP_DISABLE

__HAL_RCC_C2_FLASH_CLK_SLEEP_DISABLE

__HAL_RCC_C2_FMC_CLK_SLEEP_DISABLE

__HAL_RCC_C2_QSPI_CLK_SLEEP_DISABLE

__HAL_RCC_C2_SDMMC1_CLK_SLEEP_DISABLE

__HAL_RCC_C2_DTCM1_CLK_SLEEP_DISABLE

__HAL_RCC_C2_DTCM2_CLK_SLEEP_DISABLE

__HAL_RCC_C2_ITCM_CLK_SLEEP_DISABLE

__HAL_RCC_C2_D1SRAM1_CLK_SLEEP_DISABLE

__HAL_RCC_C2_DMA1_CLK_SLEEP_ENABLE

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is ENABLEd again. By default, all peripheral clocks are ENABLEd during SLEEP mode.

__HAL_RCC_C2_DMA2_CLK_SLEEP_ENABLE

__HAL_RCC_C2_ADC12_CLK_SLEEP_ENABLE

__HAL_RCC_C2_ETH1MAC_CLK_SLEEP_ENABLE

__HAL_RCC_C2_ETH1TX_CLK_SLEEP_ENABLE

__HAL_RCC_C2_ETH1RX_CLK_SLEEP_ENABLE

__HAL_RCC_C2_USB1_OTG_HS_CLK_SLEEP_ENABLE

__HAL_RCC_C2_USB1_OTG_HS_ULPI_CLK_SLEEP_ENABLE

__HAL_RCC_C2_USB2_OTG_FS_CLK_SLEEP_ENABLE

__HAL_RCC_C2_USB2_OTG_FS_ULPI_CLK_SLEEP_ENABLE

__HAL_RCC_C2_DMA1_CLK_SLEEP_DISABLE

__HAL_RCC_C2_DMA2_CLK_SLEEP_DISABLE

__HAL_RCC_C2_ADC12_CLK_SLEEP_DISABLE

__HAL_RCC_C2_ETH1MAC_CLK_SLEEP_DISABLE

__HAL_RCC_C2_ETH1TX_CLK_SLEEP_DISABLE

__HAL_RCC_C2_ETH1RX_CLK_SLEEP_DISABLE

__HAL_RCC_C2_USB1_OTG_HS_CLK_SLEEP_DISABLE

__HAL_RCC_C2_USB1_OTG_HS_ULPI_CLK_SLEEP_DISABLE

__HAL_RCC_C2_USB2_OTG_FS_CLK_SLEEP_DISABLE

__HAL_RCC_C2_USB2_OTG_FS_ULPI_CLK_SLEEP_DISABLE

__HAL_RCC_C2_DCMI_CLK_SLEEP_ENABLE

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is ENABLEd again. By default, all peripheral clocks are ENABLEd during SLEEP mode.

__HAL_RCC_C2_CRYP_CLK_SLEEP_ENABLE

__HAL_RCC_C2_HASH_CLK_SLEEP_ENABLE

__HAL_RCC_C2_RNG_CLK_SLEEP_ENABLE

__HAL_RCC_C2_SDMMC2_CLK_SLEEP_ENABLE

__HAL_RCC_C2_D2SRAM1_CLK_SLEEP_ENABLE

__HAL_RCC_C2_D2SRAM2_CLK_SLEEP_ENABLE

__HAL_RCC_C2_D2SRAM3_CLK_SLEEP_ENABLE

__HAL_RCC_C2_DCMI_CLK_SLEEP_DISABLE

__HAL_RCC_C2_CRYP_CLK_SLEEP_DISABLE

__HAL_RCC_C2_HASH_CLK_SLEEP_DISABLE

__HAL_RCC_C2_RNG_CLK_SLEEP_DISABLE

__HAL_RCC_C2_SDMMC2_CLK_SLEEP_DISABLE

__HAL_RCC_C2_D2SRAM1_CLK_SLEEP_DISABLE

__HAL_RCC_C2_D2SRAM2_CLK_SLEEP_DISABLE

__HAL_RCC_C2_D2SRAM3_CLK_SLEEP_DISABLE

__HAL_RCC_C2_GPIOA_CLK_SLEEP_ENABLE

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is ENABLEd again. By default, all peripheral clocks are ENABLEd during SLEEP mode.

__HAL_RCC_C2_GPIOB_CLK_SLEEP_ENABLE

__HAL_RCC_C2_GPIOC_CLK_SLEEP_ENABLE

__HAL_RCC_C2_GPIOD_CLK_SLEEP_ENABLE

__HAL_RCC_C2_GPIOE_CLK_SLEEP_ENABLE

__HAL_RCC_C2_GPIOF_CLK_SLEEP_ENABLE

__HAL_RCC_C2_GPIOG_CLK_SLEEP_ENABLE

__HAL_RCC_C2_GPIOH_CLK_SLEEP_ENABLE

__HAL_RCC_C2_GPIOI_CLK_SLEEP_ENABLE

__HAL_RCC_C2_GPIOJ_CLK_SLEEP_ENABLE

__HAL_RCC_C2_GPIOK_CLK_SLEEP_ENABLE

__HAL_RCC_C2_CRC_CLK_SLEEP_ENABLE

__HAL_RCC_C2_BDMA_CLK_SLEEP_ENABLE

__HAL_RCC_C2_ADC3_CLK_SLEEP_ENABLE

__HAL_RCC_C2_BKPRAM_CLK_SLEEP_ENABLE

__HAL_RCC_C2_D3SRAM1_CLK_SLEEP_ENABLE

__HAL_RCC_C2_GPIOA_CLK_SLEEP_DISABLE

__HAL_RCC_C2_GPIOB_CLK_SLEEP_DISABLE

__HAL_RCC_C2_GPIOC_CLK_SLEEP_DISABLE

__HAL_RCC_C2_GPIOD_CLK_SLEEP_DISABLE

__HAL_RCC_C2_GPIOE_CLK_SLEEP_DISABLE

__HAL_RCC_C2_GPIOF_CLK_SLEEP_DISABLE

__HAL_RCC_C2_GPIOG_CLK_SLEEP_DISABLE

__HAL_RCC_C2_GPIOH_CLK_SLEEP_DISABLE

__HAL_RCC_C2_GPIOI_CLK_SLEEP_DISABLE

__HAL_RCC_C2_GPIOJ_CLK_SLEEP_DISABLE

__HAL_RCC_C2_GPIOK_CLK_SLEEP_DISABLE

__HAL_RCC_C2_CRC_CLK_SLEEP_DISABLE

__HAL_RCC_C2_BDMA_CLK_SLEEP_DISABLE

__HAL_RCC_C2_ADC3_CLK_SLEEP_DISABLE

__HAL_RCC_C2_BKPRAM_CLK_SLEEP_DISABLE

__HAL_RCC_C2_D3SRAM1_CLK_SLEEP_DISABLE

__HAL_RCC_C2_LTDC_CLK_SLEEP_ENABLE

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is ENABLEd again. By default, all peripheral clocks are ENABLEd during SLEEP mode.

__HAL_RCC_C2_DSI_CLK_SLEEP_ENABLE

__HAL_RCC_C2_WWDG1_CLK_SLEEP_ENABLE

__HAL_RCC_C2_LTDC_CLK_SLEEP_DISABLE

__HAL_RCC_C2_DSI_CLK_SLEEP_DISABLE

__HAL_RCC_C2_WWDG1_CLK_SLEEP_DISABLE

__HAL_RCC_C2_TIM2_CLK_SLEEP_ENABLE

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is ENABLEd again. By default, all peripheral clocks are ENABLEd during SLEEP mode.

__HAL_RCC_C2_TIM3_CLK_SLEEP_ENABLE

__HAL_RCC_C2_TIM4_CLK_SLEEP_ENABLE

__HAL_RCC_C2_TIM5_CLK_SLEEP_ENABLE

__HAL_RCC_C2_TIM6_CLK_SLEEP_ENABLE

__HAL_RCC_C2_TIM7_CLK_SLEEP_ENABLE

__HAL_RCC_C2_TIM12_CLK_SLEEP_ENABLE

__HAL_RCC_C2_TIM13_CLK_SLEEP_ENABLE

__HAL_RCC_C2_TIM14_CLK_SLEEP_ENABLE

__HAL_RCC_C2_LPTIM1_CLK_SLEEP_ENABLE

__HAL_RCC_C2_WWDG2_CLK_SLEEP_ENABLE

__HAL_RCC_C2_SPI2_CLK_SLEEP_ENABLE

__HAL_RCC_C2_SPI3_CLK_SLEEP_ENABLE

__HAL_RCC_C2_SPDIFRX_CLK_SLEEP_ENABLE

__HAL_RCC_C2_USART2_CLK_SLEEP_ENABLE

__HAL_RCC_C2_USART3_CLK_SLEEP_ENABLE

__HAL_RCC_C2_UART4_CLK_SLEEP_ENABLE

__HAL_RCC_C2_UART5_CLK_SLEEP_ENABLE

__HAL_RCC_C2_I2C1_CLK_SLEEP_ENABLE

__HAL_RCC_C2_I2C2_CLK_SLEEP_ENABLE

__HAL_RCC_C2_I2C3_CLK_SLEEP_ENABLE

__HAL_RCC_C2_CEC_CLK_SLEEP_ENABLE

__HAL_RCC_C2_DAC12_CLK_SLEEP_ENABLE

__HAL_RCC_C2_UART7_CLK_SLEEP_ENABLE

__HAL_RCC_C2_UART8_CLK_SLEEP_ENABLE

__HAL_RCC_C2_CRS_CLK_SLEEP_ENABLE

__HAL_RCC_C2_SWPMI_CLK_SLEEP_ENABLE

__HAL_RCC_C2_OPAMP_CLK_SLEEP_ENABLE

__HAL_RCC_C2_MDIOS_CLK_SLEEP_ENABLE

__HAL_RCC_C2_FDCAN_CLK_SLEEP_ENABLE

__HAL_RCC_C2_TIM2_CLK_SLEEP_DISABLE

__HAL_RCC_C2_TIM3_CLK_SLEEP_DISABLE

__HAL_RCC_C2_TIM4_CLK_SLEEP_DISABLE

__HAL_RCC_C2_TIM5_CLK_SLEEP_DISABLE

__HAL_RCC_C2_TIM6_CLK_SLEEP_DISABLE

__HAL_RCC_C2_TIM7_CLK_SLEEP_DISABLE

__HAL_RCC_C2_TIM12_CLK_SLEEP_DISABLE

__HAL_RCC_C2_TIM13_CLK_SLEEP_DISABLE

__HAL_RCC_C2_TIM14_CLK_SLEEP_DISABLE

__HAL_RCC_C2_LPTIM1_CLK_SLEEP_DISABLE

__HAL_RCC_C2_WWDG2_CLK_SLEEP_DISABLE

__HAL_RCC_C2_SPI2_CLK_SLEEP_DISABLE

__HAL_RCC_C2_SPI3_CLK_SLEEP_DISABLE

__HAL_RCC_C2_SPDIFRX_CLK_SLEEP_DISABLE

__HAL_RCC_C2_USART2_CLK_SLEEP_DISABLE

__HAL_RCC_C2_USART3_CLK_SLEEP_DISABLE

__HAL_RCC_C2_UART4_CLK_SLEEP_DISABLE

__HAL_RCC_C2_UART5_CLK_SLEEP_DISABLE

__HAL_RCC_C2_I2C1_CLK_SLEEP_DISABLE

__HAL_RCC_C2_I2C2_CLK_SLEEP_DISABLE

__HAL_RCC_C2_I2C3_CLK_SLEEP_DISABLE

__HAL_RCC_C2_CEC_CLK_SLEEP_DISABLE

__HAL_RCC_C2_DAC12_CLK_SLEEP_DISABLE

__HAL_RCC_C2_UART7_CLK_SLEEP_DISABLE

__HAL_RCC_C2_UART8_CLK_SLEEP_DISABLE

__HAL_RCC_C2_CRS_CLK_SLEEP_DISABLE

__HAL_RCC_C2_SWPMI_CLK_SLEEP_DISABLE

__HAL_RCC_C2_OPAMP_CLK_SLEEP_DISABLE

__HAL_RCC_C2_MDIOS_CLK_SLEEP_DISABLE

__HAL_RCC_C2_FDCAN_CLK_SLEEP_DISABLE

__HAL_RCC_C2_TIM1_CLK_SLEEP_ENABLE

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is ENABLEd again. By default, all peripheral clocks are ENABLEd during SLEEP mode.

__HAL_RCC_C2_TIM8_CLK_SLEEP_ENABLE

__HAL_RCC_C2_USART1_CLK_SLEEP_ENABLE

__HAL_RCC_C2_USART6_CLK_SLEEP_ENABLE

__HAL_RCC_C2_SPI1_CLK_SLEEP_ENABLE

__HAL_RCC_C2_SPI4_CLK_SLEEP_ENABLE

__HAL_RCC_C2_TIM15_CLK_SLEEP_ENABLE

__HAL_RCC_C2_TIM16_CLK_SLEEP_ENABLE

__HAL_RCC_C2_TIM17_CLK_SLEEP_ENABLE

__HAL_RCC_C2_SPI5_CLK_SLEEP_ENABLE

__HAL_RCC_C2_SAI1_CLK_SLEEP_ENABLE

__HAL_RCC_C2_SAI2_CLK_SLEEP_ENABLE

__HAL_RCC_C2_SAI3_CLK_SLEEP_ENABLE

__HAL_RCC_C2_DFSDM1_CLK_SLEEP_ENABLE

__HAL_RCC_C2_HRTIM1_CLK_SLEEP_ENABLE

__HAL_RCC_C2_TIM1_CLK_SLEEP_DISABLE

__HAL_RCC_C2_TIM8_CLK_SLEEP_DISABLE

__HAL_RCC_C2_USART1_CLK_SLEEP_DISABLE

__HAL_RCC_C2_USART6_CLK_SLEEP_DISABLE

__HAL_RCC_C2_SPI1_CLK_SLEEP_DISABLE

__HAL_RCC_C2_SPI4_CLK_SLEEP_DISABLE

__HAL_RCC_C2_TIM15_CLK_SLEEP_DISABLE

__HAL_RCC_C2_TIM16_CLK_SLEEP_DISABLE

__HAL_RCC_C2_TIM17_CLK_SLEEP_DISABLE

__HAL_RCC_C2_SPI5_CLK_SLEEP_DISABLE

__HAL_RCC_C2_SAI1_CLK_SLEEP_DISABLE

__HAL_RCC_C2_SAI2_CLK_SLEEP_DISABLE

__HAL_RCC_C2_SAI3_CLK_SLEEP_DISABLE

__HAL_RCC_C2_DFSDM1_CLK_SLEEP_DISABLE

__HAL_RCC_C2_HRTIM1_CLK_SLEEP_DISABLE

__HAL_RCC_C2_SYSCFG_CLK_SLEEP_ENABLE

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is ENABLEd again. By default, all peripheral clocks are ENABLEd during SLEEP mode.

__HAL_RCC_C2_LPUART1_CLK_SLEEP_ENABLE

__HAL_RCC_C2_SPI6_CLK_SLEEP_ENABLE

__HAL_RCC_C2_I2C4_CLK_SLEEP_ENABLE

__HAL_RCC_C2_LPTIM2_CLK_SLEEP_ENABLE

__HAL_RCC_C2_LPTIM3_CLK_SLEEP_ENABLE

__HAL_RCC_C2_LPTIM4_CLK_SLEEP_ENABLE

__HAL_RCC_C2_LPTIM5_CLK_SLEEP_ENABLE

__HAL_RCC_C2_COMP12_CLK_SLEEP_ENABLE

__HAL_RCC_C2_VREF_CLK_SLEEP_ENABLE

__HAL_RCC_C2_SAI4_CLK_SLEEP_ENABLE

__HAL_RCC_C2_RTC_CLK_SLEEP_ENABLE

__HAL_RCC_C2_SYSCFG_CLK_SLEEP_DISABLE

__HAL_RCC_C2_LPUART1_CLK_SLEEP_DISABLE

__HAL_RCC_C2_SPI6_CLK_SLEEP_DISABLE

__HAL_RCC_C2_I2C4_CLK_SLEEP_DISABLE

__HAL_RCC_C2_LPTIM2_CLK_SLEEP_DISABLE

__HAL_RCC_C2_LPTIM3_CLK_SLEEP_DISABLE

__HAL_RCC_C2_LPTIM4_CLK_SLEEP_DISABLE

__HAL_RCC_C2_LPTIM5_CLK_SLEEP_DISABLE

__HAL_RCC_C2_COMP12_CLK_SLEEP_DISABLE

__HAL_RCC_C2_VREF_CLK_SLEEP_DISABLE

__HAL_RCC_C2_SAI4_CLK_SLEEP_DISABLE

__HAL_RCC_C2_RTC_CLK_SLEEP_DISABLE

__HAL_RCC_BDMA_CLKAM_ENABLE

Notes:

- After reset (default config), peripheral clock is disabled when both CPUs are in CSTOP

__HAL_RCC_LPUART1_CLKAM_ENABLE

__HAL_RCC_SPI6_CLKAM_ENABLE

__HAL_RCC_I2C4_CLKAM_ENABLE

__HAL_RCC_LPTIM2_CLKAM_ENABLE

__HAL_RCC_LPTIM3_CLKAM_ENABLE

__HAL_RCC_LPTIM4_CLKAM_ENABLE

__HAL_RCC_LPTIM5_CLKAM_ENABLE

__HAL_RCC_COMP12_CLKAM_ENABLE

__HAL_RCC_VREF_CLKAM_ENABLE

__HAL_RCC_RTC_CLKAM_ENABLE

__HAL_RCC_CRC_CLKAM_ENABLE

__HAL_RCC_SAI4_CLKAM_ENABLE

__HAL_RCC_ADC3_CLKAM_ENABLE

__HAL_RCC_BKPRAM_CLKAM_ENABLE

__HAL_RCC_D3SRAM1_CLKAM_ENABLE

__HAL_RCC_BDMA_CLKAM_DISABLE

__HAL_RCC_LPUART1_CLKAM_DISABLE

__HAL_RCC_SPI6_CLKAM_DISABLE

__HAL_RCC_I2C4_CLKAM_DISABLE

__HAL_RCC_LPTIM2_CLKAM_DISABLE

__HAL_RCC_LPTIM3_CLKAM_DISABLE

__HAL_RCC_LPTIM4_CLKAM_DISABLE

__HAL_RCC_LPTIM5_CLKAM_DISABLE

__HAL_RCC_COMP12_CLKAM_DISABLE

__HAL_RCC_VREF_CLKAM_DISABLE

__HAL_RCC_RTC_CLKAM_DISABLE

__HAL_RCC_CRC_CLKAM_DISABLE

__HAL_RCC_SAI4_CLKAM_DISABLE

__HAL_RCC_ADC3_CLKAM_DISABLE

__HAL_RCC_BKPRAM_CLKAM_DISABLE

__HAL_RCC_D3SRAM1_CLKAM_DISABLE

__HAL_RCC_HSI_CONFIG

Description:

- Macro to enable or disable the Internal High Speed oscillator (HSI).

Parameters:

- __STATE__: specifies the new state of the HSI. This parameter can be one of the following values:
  - RCC_HSI_OFF turn OFF the HSI oscillator
  - RCC_HSI_ON turn ON the HSI oscillator
  - RCC_HSI_DIV1 turn ON the HSI oscillator and divide it by 1 (default after reset)
  - RCC_HSI_DIV2 turn ON the HSI oscillator and divide it by 2
  - RCC_HSI_DIV4 turn ON the HSI oscillator and divide it by 4
  - RCC_HSI_DIV8 turn ON the HSI oscillator and divide it by 8

Notes:

- After enabling the HSI, the application software should wait on HSIRDY flag to be set indicating that HSI clock is stable and can be used to clock the PLL and/or system clock. HSI can not be stopped if it is used directly or through the PLL as system clock. In this case, you have to select another source of the system clock then stop the HSI. The HSI is stopped by hardware when entering STOP and STANDBY modes.
- When the HSI is stopped, HSIRDY flag goes low after 6 HSI oscillator clock cycles.

__HAL_RCC_GET_HSI_DIVIDER

Description:

- Macro to get the HSI divider.

Return value:

- The: HSI divider. The returned value can be one of the following:
  - RCC_CR_HSIDIV_1 HSI oscillator divided by 1 (default after reset)
  - RCC_CR_HSIDIV_2 HSI oscillator divided by 2
  - RCC_CR_HSIDIV_4 HSI oscillator divided by 4
  - RCC_CR_HSIDIV_8 HSI oscillator divided by 8

__HAL_RCC_HSI_ENABLE

Notes:

- The HSI is stopped by hardware when entering STOP and STANDBY modes. It is used (enabled by hardware) as system clock source after start-up from Reset, wakeup from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled). HSI can not be stopped if it is used as system clock source. In this case, you have to select another source of the system clock then stop the HSI. After enabling the HSI, the application software should wait on HSIRDY flag to be set indicating that HSI clock is stable and can be used as system clock source. This parameter can be: ENABLE or DISABLE. When the HSI is stopped, HSIRDY flag goes low after 6 HSI oscillator clock cycles.

__HAL_RCC_HSI_DISABLE

__HAL_RCC_HSI_CALIBRATIONVALUE_ADJUST

Description:

- Macro to adjust the Internal High Speed oscillator (HSI) calibration value.

Parameters:

- __HSICalibrationValue__: specifies the calibration trimming value. This parameter must be a number between 0 and 0x7F (3F for Rev Y device).

Notes:

- The calibration is used to compensate for the variations in voltage and temperature that influence the frequency of the internal HSI RC.

__HAL_RCC_HSISTOP_ENABLE

Description:

- Macros to enable or disable the force of the Internal High Speed oscillator (HSI) in STOP mode to be quickly available as kernel clock for some peripherals.

Return value:

- None

Notes:

- Keeping the HSI ON in STOP mode allows to avoid slowing down the communication speed because of the HSI start-up time. The enable of this function has not effect on the HSION bit. This parameter can be: ENABLE or DISABLE.

__HAL_RCC_HSISTOP_DISABLE

__HAL_RCC_HSI48_ENABLE

Notes:

- After enabling the HSI48, the application software should wait on HSI48RDY flag to be set indicating that HSI48 clock is stable and can be used to clock the USB. The HSI48 is stopped by hardware when entering STOP and STANDBY modes.

__HAL_RCC_HSI48_DISABLE

__HAL_RCC_CSI_ENABLE

Notes:

- The CSI is stopped by hardware when entering STOP and STANDBY modes. It is used (enabled by hardware) as system clock source after start-up from Reset, wakeup from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled). CSI can not be stopped if it is used as system clock source. In this case, you have to select another source of the system clock then stop the CSI. After enabling the CSI, the application software should wait on CSIRDY flag to be set indicating that CSI clock is stable and can be used as system clock source. When the CSI is stopped, CSIRDY flag goes low after 6 CSI oscillator clock cycles.

__HAL_RCC_CSI_DISABLE

__HAL_RCC_CSI_CALIBRATIONVALUE_ADJUST

Description:

- Macro Adjusts the Internal oscillator (CSI) calibration value.

Parameters:

- __CSICalibrationValue__: specifies the calibration trimming value. This parameter must be a number between 0 and 0x1F.

Notes:

- The calibration is used to compensate for the variations in voltage and temperature that influence the frequency of the internal CSI RC.

__HAL_RCC_CSISTOP_ENABLE

Description:

- Macros to enable or disable the force of the Low-power Internal oscillator (CSI) in STOP mode to be quickly available as kernel clock for USARTs and I2Cs.

Return value:

- None

Notes:

- Keeping the CSI ON in STOP mode allows to avoid slowing down the communication speed because of the CSI start-up time. The enable of this function has not effect on the CSION bit. This parameter can be: ENABLE or DISABLE.

__HAL_RCC_CSISTOP_DISABLE

__HAL_RCC_LSI_ENABLE

Notes:

- After enabling the LSI, the application software should wait on LSIRDY flag to be set indicating that LSI clock is stable and can be used to clock the IWDG and/or the RTC. LSI can not be disabled if the IWDG is running. When the LSI is stopped, LSIRDY flag goes low after 6 LSI oscillator clock cycles.

__HAL_RCC_LSI_DISABLE

__HAL_RCC_HSE_CONFIG

Description:

- Macro to configure the External High Speed oscillator (__HSE__).

Parameters:

- __STATE__: specifies the new state of the HSE. This parameter can be one of the following values:
  - RCC_HSE_OFF: turn OFF the HSE oscillator, HSERDY flag goes low after 6 HSE oscillator clock cycles.
  - RCC_HSE_ON: turn ON the HSE oscillator.
  - RCC_HSE_BYPASS: HSE oscillator bypassed with external clock.
  - RCC_HSE_BYPASS_DIGITAL: HSE oscillator bypassed with digital external clock. (*)

Notes:

- After enabling the HSE (RCC_HSE_ON, RCC_HSE_BYPASS or RCC_HSE_BYPASS_DIGITAL), the application software should wait on HSERDY flag to be set indicating that HSE clock is stable and can be used to clock the PLL and/or system clock. HSE state can not be changed if it is used directly or through the PLL as system clock. In this case, you have to select another source of the system clock then change the HSE state (ex. disable it). The HSE is stopped by hardware when entering STOP and STANDBY modes. This function reset the CSSON bit, so if the clock security system(CSS) was previously enabled you have to enable it again after calling this function.

__HAL_RCC_RTC_ENABLE

Notes:

- These macros must be used only after the RTC clock source was selected.

__HAL_RCC_RTC_DISABLE

__HAL_RCC_RTC_CLKPRESCALER

Description:

- Macros to configure the RTC clock (RTCCLK).

Parameters:

- __RTCCLKSource__: specifies the RTC clock source. This parameter can be one of the following values:
  - RCC_RTCCLKSOURCE_LSE: LSE selected as RTC clock.
  - RCC_RTCCLKSOURCE_LSI: LSI selected as RTC clock.
  - RCC_RTCCLKSOURCE_HSE_DIVx: HSE clock divided by x selected as RTC clock, where x:[2,31]

Notes:

- As the RTC clock configuration bits are in the Backup domain and write access is denied to this domain after reset, you have to enable write access using the Power Backup Access macro before to configure the RTC clock source (to be done once after reset). Once the RTC clock is configured it can't be changed unless the Backup domain is reset using __HAL_RCC_BackupReset_RELEASE() macro, or by a Power On Reset (POR).
- If the LSE or LSI is used as RTC clock source, the RTC continues to work in STOP and STANDBY modes, and can be used as wakeup source. However, when the HSE clock is used as RTC clock source, the RTC cannot be used in STOP and STANDBY modes. The maximum input clock frequency for RTC is 1MHz (when using HSE as RTC clock source).

__HAL_RCC_RTC_CONFIG

__HAL_RCC_GET_RTC_SOURCE

__HAL_RCC_BACKUPRESET_FORCE

Notes:

- This function resets the RTC peripheral (including the backup registers) and the RTC clock source selection in RCC_BDCR register. The BKPSRAM is not affected by this reset.

__HAL_RCC_BACKUPRESET_RELEASE

__HAL_RCC_PLL_ENABLE

Notes:

- After enabling the main PLL, the application software should wait on PLLRDY flag to be set indicating that PLL clock is stable and can be used as system clock source. The main PLL can not be disabled if it is used as system clock source The main PLL is disabled by hardware when entering STOP and STANDBY modes.

__HAL_RCC_PLL_DISABLE

__HAL_RCC_PLLCLKOUT_ENABLE

Description:

- Enables or disables each clock output (PLL_P_CLK, PLL_Q_CLK, PLL_R_CLK)

Parameters:

- __RCC_PLL1ClockOut__: specifies the PLL clock to be outputted This parameter can be one of the following values:
  - RCC_PLL1_DIVP: This clock is used to generate system clock up to 550MHZ(*), 480MHZ(**) or 280MHZ(***)
  - RCC_PLL1_DIVQ: This clock is used to generate peripherals clock up to 550MHZ(*), 480MHZ(**) or 280MHZ(***)
  - RCC_PLL1_DIVR: This clock is used to generate peripherals clock up to 550MHZ(*), 480MHZ(**) or 280MHZ(***)

Return value:

- None

Notes:

- Enabling/disabling those Clocks can be done only when the PLL is disabled. This is mainly used to save Power. (The ck_pll_p of the System PLL cannot be stopped if used as System Clock).

__HAL_RCC_PLLCLKOUT_DISABLE

__HAL_RCC_PLLFRACN_ENABLE

Description:

- Enables or disables Fractional Part Of The Multiplication Factor of PLL1 VCO.

Return value:

- None

Notes:

- Enabling/disabling Fractional Part can be any time without the need to stop the PLL1

__HAL_RCC_PLLFRACN_DISABLE

__HAL_RCC_PLL_CONFIG

Description:

- Macro to configures the main PLL clock source, multiplication and division factors.

Parameters:

- __RCC_PLLSOURCE__: specifies the PLL entry clock source. This parameter can be one of the following values:
  - RCC_PLLSOURCE_CSI: CSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_HSI: HSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_HSE: HSE oscillator clock selected as PLL clock entry
- __PLLM1__: specifies the division factor for PLL VCO input clock This parameter must be a number between 1 and 63.
- __PLLN1__: specifies the multiplication factor for PLL VCO output clock This parameter must be a number between 4 and 512 or between 8 and 420(*).
- __PLLP1__: specifies the division factor for system clock. This parameter must be a number between 2 or 1(**) and 128 (where odd numbers are not allowed)
- __PLLQ1__: specifies the division factor for peripheral kernel clocks This parameter must be a number between 1 and 128
- __PLLR1__: specifies the division factor for peripheral kernel clocks This parameter must be a number between 1 and 128

Return value:

- None: (*) : For stm32h7a3xx and stm32h7b3xx family lines. (**): For stm32h72xxx and stm32h73xxx family lines.

Notes:

- This function must be used only when the main PLL is disabled.
- This clock source (__RCC_PLLSource__) is common for the main PLL1 (main PLL) and PLL2 & PLL3 .
- You have to set the PLLM parameter correctly to ensure that the VCO input frequency ranges from 1 to 16 MHz.
- You have to set the PLLN parameter correctly to ensure that the VCO output frequency is between 150 and 420 MHz (when in medium VCO range) or between 192 and 836 MHZ or between 128 and 560 MHZ(*) (when in wide VCO range)
- To insure an optimal behavior of the PLL when one of the post-divider (DIVP, DIVQ or DIVR) is not used, application shall clear the enable bit (DIVyEN) and assign lowest possible value to __PLL1P__, __PLL1Q__ or __PLL1R__ parameters.

__HAL_RCC_PLL_PLLSOURCE_CONFIG

Description:

- Macro to configure the PLLs clock source.

Parameters:

- __PLLSOURCE__: specifies the PLLs entry clock source. This parameter can be one of the following values:
  - RCC_PLLSOURCE_CSI: CSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_HSI: HSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_HSE: HSE oscillator clock selected as PLL clock entry

Notes:

- This function must be used only when all PLLs are disabled.

__HAL_RCC_PLLFRACN_CONFIG

Description:

- Macro to configures the main PLL clock Fractional Part Of The Multiplication Factor.

Parameters:

- __RCC_PLL1FRACN__: specifies Fractional Part Of The Multiplication Factor for PLL1 VCO It should be a value between 0 and 8191

Return value:

- None

Notes:

- These bits can be written at any time, allowing dynamic fine-tuning of the PLL1 VCO
- Warning: The software has to set correctly these bits to insure that the VCO output frequency is between its valid frequency range, which is: 192 to 836 MHz or 128 to 560 MHz(*) if PLL1VCOSEL = 0 150 to 420 MHz if PLL1VCOSEL = 1.

__HAL_RCC_PLL_VCIRANGE

Description:

- Macro to select the PLL1 reference frequency range.

Parameters:

- __RCC_PLL1VCIRange__: specifies the PLL1 input frequency range This parameter can be one of the following values:
  - RCC_PLL1VCIRANGE_0: Range frequency is between 1 and 2 MHz
  - RCC_PLL1VCIRANGE_1: Range frequency is between 2 and 4 MHz
  - RCC_PLL1VCIRANGE_2: Range frequency is between 4 and 8 MHz
  - RCC_PLL1VCIRANGE_3: Range frequency is between 8 and 16 MHz

Return value:

- None

__HAL_RCC_PLL_VCORANGE

Description:

- Macro to select the PLL1 reference frequency range.

Parameters:

- __RCC_PLL1VCORange__: specifies the PLL1 input frequency range This parameter can be one of the following values:
  - RCC_PLL1VCOWIDE: Range frequency is between 192 and 836 MHz or between 128 to 560 MHz(*)
  - RCC_PLL1VCOMEDIUM: Range frequency is between 150 and 420 MHz

Return value:

- None

__HAL_RCC_GET_SYSCLK_SOURCE

Description:

- Macro to get the clock source used as system clock.

Return value:

- The: clock source used as system clock. The returned value can be one of the following:
  - RCC_CFGR_SWS_CSI: CSI used as system clock.
  - RCC_CFGR_SWS_HSI: HSI used as system clock.
  - RCC_CFGR_SWS_HSE: HSE used as system clock.
  - RCC_CFGR_SWS_PLL: PLL used as system clock.

__HAL_RCC_SYSCLK_CONFIG

Description:

- Macro to configure the system clock source.

Parameters:

- __RCC_SYSCLKSOURCE__: specifies the system clock source. This parameter can be one of the following values:
  - RCC_SYSCLKSOURCE_HSI: HSI oscillator is used as system clock source.
  - RCC_SYSCLKSOURCE_CSI: CSI oscillator is used as system clock source.
  - RCC_SYSCLKSOURCE_HSE: HSE oscillator is used as system clock source.
  - RCC_SYSCLKSOURCE_PLLCLK: PLL output is used as system clock source.

__HAL_RCC_GET_PLL_OSCSOURCE

Description:

- Macro to get the oscillator used as PLL clock source.

Return value:

- The: oscillator used as PLL clock source. The returned value can be one of the following:
  - RCC_PLLSOURCE_NONE: No oscillator is used as PLL clock source.
  - RCC_PLLSOURCE_CSI: CSI oscillator is used as PLL clock source.
  - RCC_PLLSOURCE_HSI: HSI oscillator is used as PLL clock source.
  - RCC_PLLSOURCE_HSE: HSE oscillator is used as PLL clock source.

__HAL_RCC_LSEDRIVE_CONFIG

Description:

- Macro to configure the External Low Speed oscillator (LSE) drive capability.

Parameters:

- __LSEDRIVE__: specifies the new state of the LSE drive capability. This parameter can be one of the following values:
  - RCC_LSEDRIVE_LOW: LSE oscillator low drive capability.
  - RCC_LSEDRIVE_MEDIUMLOW: LSE oscillator medium low drive capability.
  - RCC_LSEDRIVE_MEDIUMHIGH: LSE oscillator medium high drive capability.
  - RCC_LSEDRIVE_HIGH: LSE oscillator high drive capability.

Return value:

- None

Notes:

- As the LSE is in the Backup domain and write access is denied to this domain after reset, you have to enable write access using HAL_PWR_EnableBkUpAccess() function before to configure the LSE (to be done once after reset). On STM32H7 Rev.B and above devices this can't be updated while LSE is ON.

__HAL_RCC_WAKEUPSTOP_CLK_CONFIG

Description:

- Macro to configure the wake up from stop clock.

Parameters:

- __RCC_STOPWUCLK__: specifies the clock source used after wake up from stop This parameter can be one of the following values:
  - RCC_STOP_WAKEUPCLOCK_CSI: CSI selected as system clock source
  - RCC_STOP_WAKEUPCLOCK_HSI: HSI selected as system clock source

Return value:

- None

__HAL_RCC_KERWAKEUPSTOP_CLK_CONFIG

Description:

- Macro to configure the Kernel wake up from stop clock.

Parameters:

- __RCC_STOPKERWUCLK__: specifies the Kernel clock source used after wake up from stop This parameter can be one of the following values:
  - RCC_STOP_KERWAKEUPCLOCK_CSI: CSI selected as Kernel clock source
  - RCC_STOP_KERWAKEUPCLOCK_HSI: HSI selected as Kernel clock source

Return value:

- None

RCC_GET_PLL_OSCSOURCE

RCC Flag

RCC_FLAG_HSIRDY

RCC_FLAG_HSIDIV

RCC_FLAG_CSIRDY

RCC_FLAG_HSI48RDY

RCC_FLAG_D1CKRDY

RCC_FLAG_D2CKRDY

RCC_FLAG_HSERDY

RCC_FLAG_PLLRDY

RCC_FLAG_PLL2RDY

RCC_FLAG_PLL3RDY

RCC_FLAG_LSERDY

RCC_FLAG_LSIRDY

RCC_FLAG_D1RST

RCC_FLAG_D2RST

RCC_FLAG_BORRST

RCC_FLAG_PINRST

RCC_FLAG_PORRST

RCC_FLAG_SFTRST

RCC_FLAG_IWDG1RST

RCC_FLAG_WWDG1RST

RCC_FLAG_LPWR1RST

RCC_FLAG_LPWR2RST

RCC_FLAG_C1RST

RCC_FLAG_C2RST

RCC_FLAG_SFTR1ST

RCC_FLAG_SFTR2ST

RCC_FLAG_WWDG2RST

RCC_FLAG_IWDG2RST

Flags Interrupts Management

__HAL_RCC_ENABLE_IT

Description:

- Enable RCC interrupt.

Parameters:

- __INTERRUPT__: specifies the RCC interrupt sources to be enabled. This parameter can be any combination of the following values:
  - RCC_IT_LSIRDY: LSI ready interrupt
  - RCC_IT_LSERDY: LSE ready interrupt
  - RCC_IT_CSIRDY: HSI ready interrupt
  - RCC_IT_HSIRDY: HSI ready interrupt
  - RCC_IT_HSERDY: HSE ready interrupt
  - RCC_IT_HSI48RDY: HSI48 ready interrupt
  - RCC_IT_PLLRDY: main PLL ready interrupt
  - RCC_IT_PLL2RDY: PLL2 ready interrupt
  - RCC_IT_PLL3RDY: PLL3 ready interrupt
  - RCC_IT_LSECSS: Clock security system interrupt

__HAL_RCC_DISABLE_IT

Description:

- Disable RCC interrupt.

Parameters:

- __INTERRUPT__: specifies the RCC interrupt sources to be disabled. This parameter can be any combination of the following values:
  - RCC_IT_LSIRDY: LSI ready interrupt
  - RCC_IT_LSERDY: LSE ready interrupt
  - RCC_IT_CSIRDY: HSI ready interrupt
  - RCC_IT_HSIRDY: HSI ready interrupt
  - RCC_IT_HSERDY: HSE ready interrupt
  - RCC_IT_HSI48RDY: HSI48 ready interrupt
  - RCC_IT_PLLRDY: main PLL ready interrupt
  - RCC_IT_PLL2RDY: PLL2 ready interrupt
  - RCC_IT_PLL3RDY: PLL3 ready interrupt
  - RCC_IT_LSECSS: Clock security system interrupt

__HAL_RCC_CLEAR_IT

Description:

- Clear the RCC's interrupt pending bits.

Parameters:

- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:
  - RCC_IT_LSIRDY: LSI ready interrupt
  - RCC_IT_LSERDY: LSE ready interrupt
  - RCC_IT_CSIRDY: CSI ready interrupt
  - RCC_IT_HSIRDY: HSI ready interrupt
  - RCC_IT_HSERDY: HSE ready interrupt
  - RCC_IT_HSI48RDY: HSI48 ready interrupt
  - RCC_IT_PLLRDY: main PLL ready interrupt
  - RCC_IT_PLL2RDY: PLL2 ready interrupt
  - RCC_IT_PLL3RDY: PLL3 ready interrupt
  - RCC_IT_HSECSS: HSE Clock Security interrupt
  - RCC_IT_LSECSS: Clock security system interrupt

__HAL_RCC_GET_IT

Description:

- Check the RCC's interrupt has occurred or not.

Parameters:

- __INTERRUPT__: specifies the RCC interrupt source to check. This parameter can be any combination of the following values:
  - RCC_IT_LSIRDY: LSI ready interrupt
  - RCC_IT_LSERDY: LSE ready interrupt
  - RCC_IT_CSIRDY: CSI ready interrupt
  - RCC_IT_HSIRDY: HSI ready interrupt
  - RCC_IT_HSERDY: HSE ready interrupt
  - RCC_IT_HSI48RDY: HSI48 ready interrupt
  - RCC_IT_PLLRDY: main PLL ready interrupt
  - RCC_IT_PLL2RDY: PLL2 ready interrupt
  - RCC_IT_PLL3RDY: PLL3 ready interrupt
  - RCC_IT_HSECSS: HSE Clock Security interrupt
  - RCC_IT_LSECSS: Clock security system interrupt

Return value:

- The: new state of __INTERRUPT__ (TRUE or FALSE).

__HAL_RCC_CLEAR_RESET_FLAGS

__HAL_RCC_C1_CLEAR_RESET_FLAGS

__HAL_RCC_C2_CLEAR_RESET_FLAGS

RCC_FLAG_MASK

Description:

- Check RCC flag is set or not.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - RCC_FLAG_HSIRDY: HSI oscillator clock ready
  - RCC_FLAG_HSIDIV: HSI divider flag
  - RCC_FLAG_CSIRDY: CSI oscillator clock ready
  - RCC_FLAG_HSI48RDY: HSI48 oscillator clock ready
  - RCC_FLAG_HSERDY: HSE oscillator clock ready
  - RCC_FLAG_D1CKRDY: Domain1 clock ready
  - RCC_FLAG_D2CKRDY: Domain2 clock ready
  - RCC_FLAG_PLLRDY: PLL1 clock ready
  - RCC_FLAG_PLL2RDY: PLL2 clock ready
  - RCC_FLAG_PLL3RDY: PLL3 clock ready
  - RCC_FLAG_LSERDY: LSE oscillator clock ready
  - RCC_FLAG_LSIRDY: LSI oscillator clock ready
  - RCC_FLAG_C1RST: CPU reset flag
  - RCC_FLAG_C2RST: CPU2 reset flag
  - RCC_FLAG_D1RST: D1 domain power switch reset flag
  - RCC_FLAG_D2RST: D2 domain power switch reset flag
  - RCC_FLAG_BORRST: BOR reset flag
  - RCC_FLAG_PINRST: Pin reset
  - RCC_FLAG_PORRST: POR/PDR reset
  - RCC_FLAG_SFTR1ST: System reset from CPU reset flag
  - RCC_FLAG_SFTR2ST: System reset from CPU2 reset flag
  - RCC_FLAG_BORRST: D2 domain power switch reset flag
  - RCC_FLAG_IWDG1RST: CPU Independent Watchdog reset
  - RCC_FLAG_IWDG2RST: CPU2 Independent Watchdog reset
  - RCC_FLAG_WWDG2RST: Window Watchdog2 reset
  - RCC_FLAG_WWDG1RST: Window Watchdog1 reset
  - RCC_FLAG_LPWR1RST: Reset due to illegal D1 DSTANDBY or CPU CSTOP flag
  - RCC_FLAG_LPWR2RST: Reset due to illegal D2 DSTANDBY or CPU2 CSTOP flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_RCC_GET_FLAG

__HAL_RCC_C1_GET_FLAG

__HAL_RCC_C2_GET_FLAG

RCC HCLK Clock Source

RCC_HCLK_DIV1

RCC_HCLK_DIV2

RCC_HCLK_DIV4

RCC_HCLK_DIV8

RCC_HCLK_DIV16

RCC_HCLK_DIV64

RCC_HCLK_DIV128

RCC_HCLK_DIV256

RCC_HCLK_DIV512

RCC HSE Config

RCC_HSE_OFF

RCC_HSE_ON

RCC_HSE_BYPASS

RCC HSI48 Config

RCC_HSI48_OFF

RCC_HSI48_ON

RCC HSI Config

RCC_HSI_OFF
: HSI clock deactivation

RCC_HSI_ON
: HSI clock activation

RCC_HSI_DIV1
: HSI_DIV1 clock activation

RCC_HSI_DIV2
: HSI_DIV2 clock activation

RCC_HSI_DIV4
: HSI_DIV4 clock activation

RCC_HSI_DIV8
: HSI_DIV8 clock activation

RCC_HSICALIBRATION_DEFAULT

RCC Interrupt

RCC_IT_LSIRDY

RCC_IT_LSERDY

RCC_IT_HSIRDY

RCC_IT_HSERDY

RCC_IT_CSIRDY

RCC_IT_HSI48RDY

RCC_IT_PLLRDY

RCC_IT_PLL2RDY

RCC_IT_PLL3RDY

RCC_IT_LSECSS

RCC_IT_CSS

RCC Private macros to check input parameters

IS_RCC_OSCILLATORTYPE

IS_RCC_HSE

IS_RCC_LSE

IS_RCC_HSI

IS_RCC_HSI48

IS_RCC_LSI

IS_RCC_CSI

IS_RCC_PLL

IS_RCC_PLLSOURCE

IS_RCC_PLLRGE_VALUE

IS_RCC_PLLVCO_VALUE

IS_RCC_PLLFRACN_VALUE

IS_RCC_PLLM_VALUE

IS_RCC_PLLN_VALUE

IS_RCC_PLLP_VALUE

IS_RCC_PLLQ_VALUE

IS_RCC_PLLR_VALUE

IS_RCC_PLLCLOCKOUT_VALUE

IS_RCC_CLOCKTYPE

IS_RCC_SYSCLKSOURCE

IS_RCC_SYSCLK

IS_RCC_HCLK

IS_RCC_CDPCLK1

IS_RCC_D1PCLK1

IS_RCC_PCLK1

IS_RCC_PCLK2

IS_RCC_SRDPCLK1

IS_RCC_D3PCLK1

IS_RCC_RTCCLKSOURCE

IS_RCC_MCO

IS_RCC_MCO1SOURCE

IS_RCC_MCO2SOURCE

IS_RCC_MCODIV

IS_RCC_FLAG

IS_RCC_HSICALIBRATION_VALUE

IS_RCC_CSICALIBRATION_VALUE

IS_RCC_STOP_WAKEUPCLOCK

IS_RCC_STOP_KERWAKEUPCLOCK

LSE Drive Config

RCC_LSEDRIVE_LOW

LSE low drive capability

RCC_LSEDRIVE_MEDIUMLOW

LSE medium low drive capability

RCC_LSEDRIVE_MEDIUMHIGH

LSE medium high drive capability

RCC_LSEDRIVE_HIGH

LSE high drive capability

RCC LSE Config

RCC_LSE_OFF

RCC_LSE_ON

RCC_LSE_BYPASS

LSE Configuration

__HAL_RCC_LSE_CONFIG

Description:

- Macro to configure the External Low Speed oscillator (LSE).

Parameters:

- __STATE__: specifies the new state of the LSE. This parameter can be one of the following values:
  - RCC_LSE_OFF: turn OFF the LSE oscillator, LSERDY flag goes low after 6 LSE oscillator clock cycles.
  - RCC_LSE_ON: turn ON the LSE oscillator.
  - RCC_LSE_BYPASS: LSE oscillator bypassed with external clock.
  - RCC_LSE_BYPASS_DIGITAL: LSE oscillator bypassed with external digital clock. (*)

Notes:

- Transition LSE Bypass to LSE On and LSE On to LSE Bypass are not supported by this macro. User should request a transition to LSE Off first and then LSE On or LSE Bypass. The external input clock can have a frequency up to 1 MHz and be low swing (analog) or digital(*). A duty cycle close to 50% is recommended. As the LSE is in the Backup domain and write access is denied to this domain after reset, you have to enable write access using HAL_PWR_EnableBkUpAccess() function before to configure the LSE (to be done once after reset). After enabling the LSE (RCC_LSE_ON, RCC_LSE_BYPASS or RCC_LSE_BYPASS_DIGITAL*), the application software should wait on LSERDY flag to be set indicating that LSE clock is stable and can be used to clock the RTC. If the RTC is used, the LSE bypass must not be configured in digital mode but in low swing analog mode (*)

RCC LSI Config

RCC_LSI_OFF

RCC_LSI_ON

RCC MCO1 Clock Source

RCC_MCO1SOURCE_HSI

RCC_MCO1SOURCE_LSE

RCC_MCO1SOURCE_HSE

RCC_MCO1SOURCE_PLL1QCLK

RCC_MCO1SOURCE_HSI48

RCC MCO2 Clock Source

RCC_MCO2SOURCE_SYSCLK

RCC_MCO2SOURCE_PLL2PCLK

RCC_MCO2SOURCE_HSE

RCC_MCO2SOURCE_PLLCLK

RCC_MCO2SOURCE_CSICLK

RCC_MCO2SOURCE_LSICLK

RCC MCOx Clock Prescaler

RCC_MCODIV_1

RCC_MCODIV_2

RCC_MCODIV_3

RCC_MCODIV_4

RCC_MCODIV_5

RCC_MCODIV_6

RCC_MCODIV_7

RCC_MCODIV_8

RCC_MCODIV_9

RCC_MCODIV_10

RCC_MCODIV_11

RCC_MCODIV_12

RCC_MCODIV_13

RCC_MCODIV_14

RCC_MCODIV_15

RCC MCO Index

RCC_MCO1

RCC_MCO2

RCC Oscillator Type

RCC_OSCILLATORTYPE_NONE

RCC_OSCILLATORTYPE_HSE

RCC_OSCILLATORTYPE_HSI

RCC_OSCILLATORTYPE_LSE

RCC_OSCILLATORTYPE_LSI

RCC_OSCILLATORTYPE_CSI

RCC_OSCILLATORTYPE_HSI48

RCC PLL1 VCI Range

RCC_PLL1VCIRANGE_0

Clock range frequency between 1 and 2 MHz

RCC_PLL1VCIRANGE_1

Clock range frequency between 2 and 4 MHz

RCC_PLL1VCIRANGE_2

Clock range frequency between 4 and 8 MHz

RCC_PLL1VCIRANGE_3

Clock range frequency between 8 and 16 MHz

RCC PLL1 VCO Range

RCC_PLL1VCOWIDE

RCC_PLL1VCOMEDIUM

RCC PLL2 Clock Output

RCC_PLL2_DIVP

RCC_PLL2_DIVQ

RCC_PLL2_DIVR

RCC PLL2 VCI Range

RCC_PLL2VCIRANGE_0

Clock range frequency between 1 and 2 MHz

RCC_PLL2VCIRANGE_1

Clock range frequency between 2 and 4 MHz

RCC_PLL2VCIRANGE_2

Clock range frequency between 4 and 8 MHz

RCC_PLL2VCIRANGE_3

Clock range frequency between 8 and 16 MHz

RCC PLL2 VCO Range

RCC_PLL2VCOWIDE

RCC_PLL2VCOMEDIUM

RCC PLL3 Clock Output

RCC_PLL3_DIVP

RCC_PLL3_DIVQ

RCC_PLL3_DIVR

RCC PLL3 VCI Range

RCC_PLL3VCIRANGE_0

Clock range frequency between 1 and 2 MHz

RCC_PLL3VCIRANGE_1

Clock range frequency between 2 and 4 MHz

RCC_PLL3VCIRANGE_2

Clock range frequency between 4 and 8 MHz

RCC_PLL3VCIRANGE_3

Clock range frequency between 8 and 16 MHz

RCC PLL3 VCO Range

RCC_PLL3VCOWIDE

RCC_PLL3VCOMEDIUM

RCC PLL Clock Output

RCC_PLL1_DIVP

RCC_PLL1_DIVQ

RCC_PLL1_DIVR

RCC PLL Clock Source

RCC_PLLSOURCE_HSI

RCC_PLLSOURCE_CSI

RCC_PLLSOURCE_HSE

RCC_PLLSOURCE_NONE

RCC PLL Config

RCC_PLL_NONE

RCC_PLL_OFF

RCC_PLL_ON

RCC RTC Clock Source

RCC_RTCCLKSOURCE_NO_CLK

RCC_RTCCLKSOURCE_LSE

RCC_RTCCLKSOURCE_LSI

RCC_RTCCLKSOURCE_HSE_DIV2

RCC_RTCCLKSOURCE_HSE_DIV3

RCC_RTCCLKSOURCE_HSE_DIV4

RCC_RTCCLKSOURCE_HSE_DIV5

RCC_RTCCLKSOURCE_HSE_DIV6

RCC_RTCCLKSOURCE_HSE_DIV7

RCC_RTCCLKSOURCE_HSE_DIV8

RCC_RTCCLKSOURCE_HSE_DIV9

RCC_RTCCLKSOURCE_HSE_DIV10

RCC_RTCCLKSOURCE_HSE_DIV11

RCC_RTCCLKSOURCE_HSE_DIV12

RCC_RTCCLKSOURCE_HSE_DIV13

RCC_RTCCLKSOURCE_HSE_DIV14

RCC_RTCCLKSOURCE_HSE_DIV15

RCC_RTCCLKSOURCE_HSE_DIV16

RCC_RTCCLKSOURCE_HSE_DIV17

RCC_RTCCLKSOURCE_HSE_DIV18

RCC_RTCCLKSOURCE_HSE_DIV19

RCC_RTCCLKSOURCE_HSE_DIV20

RCC_RTCCLKSOURCE_HSE_DIV21

RCC_RTCCLKSOURCE_HSE_DIV22

RCC_RTCCLKSOURCE_HSE_DIV23

RCC_RTCCLKSOURCE_HSE_DIV24

RCC_RTCCLKSOURCE_HSE_DIV25

RCC_RTCCLKSOURCE_HSE_DIV26

RCC_RTCCLKSOURCE_HSE_DIV27

RCC_RTCCLKSOURCE_HSE_DIV28

RCC_RTCCLKSOURCE_HSE_DIV29

RCC_RTCCLKSOURCE_HSE_DIV30

RCC_RTCCLKSOURCE_HSE_DIV31

RCC_RTCCLKSOURCE_HSE_DIV32

RCC_RTCCLKSOURCE_HSE_DIV33

RCC_RTCCLKSOURCE_HSE_DIV34

RCC_RTCCLKSOURCE_HSE_DIV35

RCC_RTCCLKSOURCE_HSE_DIV36

RCC_RTCCLKSOURCE_HSE_DIV37

RCC_RTCCLKSOURCE_HSE_DIV38

RCC_RTCCLKSOURCE_HSE_DIV39

RCC_RTCCLKSOURCE_HSE_DIV40

RCC_RTCCLKSOURCE_HSE_DIV41

RCC_RTCCLKSOURCE_HSE_DIV42

RCC_RTCCLKSOURCE_HSE_DIV43

RCC_RTCCLKSOURCE_HSE_DIV44

RCC_RTCCLKSOURCE_HSE_DIV45

RCC_RTCCLKSOURCE_HSE_DIV46

RCC_RTCCLKSOURCE_HSE_DIV47

RCC_RTCCLKSOURCE_HSE_DIV48

RCC_RTCCLKSOURCE_HSE_DIV49

RCC_RTCCLKSOURCE_HSE_DIV50

RCC_RTCCLKSOURCE_HSE_DIV51

RCC_RTCCLKSOURCE_HSE_DIV52

RCC_RTCCLKSOURCE_HSE_DIV53

RCC_RTCCLKSOURCE_HSE_DIV54

RCC_RTCCLKSOURCE_HSE_DIV55

RCC_RTCCLKSOURCE_HSE_DIV56

RCC_RTCCLKSOURCE_HSE_DIV57

RCC_RTCCLKSOURCE_HSE_DIV58

RCC_RTCCLKSOURCE_HSE_DIV59

RCC_RTCCLKSOURCE_HSE_DIV60

RCC_RTCCLKSOURCE_HSE_DIV61

RCC_RTCCLKSOURCE_HSE_DIV62

RCC_RTCCLKSOURCE_HSE_DIV63

RCC Stop KernelWakeUpClock

RCC_STOP_KERWAKEUPCLOCK_HSI

RCC_STOP_KERWAKEUPCLOCK_CSI

RCC Stop WakeUpClock

RCC_STOP_WAKEUPCLOCK_HSI

RCC_STOP_WAKEUPCLOCK_CSI

RCC System Clock Source

RCC_SYSCLKSOURCE_CSI

RCC_SYSCLKSOURCE_HSI

RCC_SYSCLKSOURCE_HSE

RCC_SYSCLKSOURCE_PLLCLK

System Clock Source Status

RCC_SYSCLKSOURCE_STATUS_CSI

CSI used as system clock

RCC_SYSCLKSOURCE_STATUS_HSI

HSI used as system clock

RCC_SYSCLKSOURCE_STATUS_HSE

HSE used as system clock

RCC_SYSCLKSOURCE_STATUS_PLLCLK

PLL1 used as system clock

RCC System Clock Type

RCC_CLOCKTYPE_SYSCLK

RCC_CLOCKTYPE_HCLK

RCC_CLOCKTYPE_D1PCLK1

RCC_CLOCKTYPE_PCLK1

RCC_CLOCKTYPE_PCLK2

RCC_CLOCKTYPE_D3PCLK1

RCC SYS Clock Source

RCC_SYSCLK_DIV1

RCC_SYSCLK_DIV2

RCC_SYSCLK_DIV4

RCC_SYSCLK_DIV8

RCC_SYSCLK_DIV16

RCC_SYSCLK_DIV64

RCC_SYSCLK_DIV128

RCC_SYSCLK_DIV256

RCC_SYSCLK_DIV512

69 HAL RCC Extension Driver

69.1 RCCEx Firmware driver registers structures

69.1.1 RCC_PLL2InitTypeDef

RCC_PLL2InitTypeDef is defined in the stm32h7xx_hal_rcc_ex.h

Data Fields

- ***uint32_t PLL2M***
- ***uint32_t PLL2N***
- ***uint32_t PLL2P***
- ***uint32_t PLL2Q***
- ***uint32_t PLL2R***
- ***uint32_t PLL2RGE***
- ***uint32_t PLL2VCOSEL***
- ***uint32_t PLL2FRACN***

Field Documentation

- ***uint32_t RCC_PLL2InitTypeDef::PLL2M***

  PLL2M: Division factor for PLL2 VCO input clock. This parameter must be a number between Min_Data = 1 and Max_Data = 63
- ***uint32_t RCC_PLL2InitTypeDef::PLL2N***

  PLL2N: Multiplication factor for PLL2 VCO output clock. This parameter must be a number between Min_Data = 4 and Max_Data = 512 or between Min_Data = 8 and Max_Data = 420(*) (*) : For stm32h7a3xx and stm32h7b3xx family lines.
- ***uint32_t RCC_PLL2InitTypeDef::PLL2P***

  PLL2P: Division factor for system clock. This parameter must be a number between Min_Data = 2 and Max_Data = 128 odd division factors are not allowed
- ***uint32_t RCC_PLL2InitTypeDef::PLL2Q***

  PLL2Q: Division factor for peripheral clocks. This parameter must be a number between Min_Data = 1 and Max_Data = 128
- ***uint32_t RCC_PLL2InitTypeDef::PLL2R***

  PLL2R: Division factor for peripheral clocks. This parameter must be a number between Min_Data = 1 and Max_Data = 128
- ***uint32_t RCC_PLL2InitTypeDef::PLL2RGE***

  PLL2RGE: PLL2 clock Input range This parameter must be a value of ***RCC_PLL2_VCI_Range***
- ***uint32_t RCC_PLL2InitTypeDef::PLL2VCOSEL***

  PLL2VCOSEL: PLL2 clock Output range This parameter must be a value of ***RCC_PLL2_VCO_Range***
- ***uint32_t RCC_PLL2InitTypeDef::PLL2FRACN***

  PLL2FRACN: Specifies Fractional Part Of The Multiplication Factor for PLL2 VCO It should be a value between 0 and 8191

69.1.2 RCC_PLL3InitTypeDef

RCC_PLL3InitTypeDef is defined in the stm32h7xx_hal_rcc_ex.h

Data Fields

- ***uint32_t PLL3M***
- ***uint32_t PLL3N***
- ***uint32_t PLL3P***
- ***uint32_t PLL3Q***
- ***uint32_t PLL3R***

- ***uint32_t PLL3RGE***
- ***uint32_t PLL3VCOSEL***
- ***uint32_t PLL3FRACN***

Field Documentation

- ***uint32_t RCC_PLL3InitTypeDef::PLL3M***

  PLL3M: Division factor for PLL3 VCO input clock. This parameter must be a number between Min_Data = 1 and Max_Data = 63
- ***uint32_t RCC_PLL3InitTypeDef::PLL3N***

  PLL3N: Multiplication factor for PLL3 VCO output clock. This parameter must be a number between Min_Data = 4 and Max_Data = 512 or between Min_Data = 8 and Max_Data = 420(*) (*) : For stm32h7a3xx and stm32h7b3xx family lines.
- ***uint32_t RCC_PLL3InitTypeDef::PLL3P***

  PLL3P: Division factor for system clock. This parameter must be a number between Min_Data = 2 and Max_Data = 128 odd division factors are not allowed
- ***uint32_t RCC_PLL3InitTypeDef::PLL3Q***

  PLL3Q: Division factor for peripheral clocks. This parameter must be a number between Min_Data = 1 and Max_Data = 128
- ***uint32_t RCC_PLL3InitTypeDef::PLL3R***

  PLL3R: Division factor for peripheral clocks. This parameter must be a number between Min_Data = 1 and Max_Data = 128
- ***uint32_t RCC_PLL3InitTypeDef::PLL3RGE***

  PLL3RGE: PLL3 clock Input range This parameter must be a value of ***RCC_PLL3_VCI_Range***
- ***uint32_t RCC_PLL3InitTypeDef::PLL3VCOSEL***

  PLL3VCOSEL: PLL3 clock Output range This parameter must be a value of ***RCC_PLL3_VCO_Range***
- ***uint32_t RCC_PLL3InitTypeDef::PLL3FRACN***

  PLL3FRACN: Specifies Fractional Part Of The Multiplication Factor for PLL3 VCO It should be a value between 0 and 8191

69.1.3 PLL1_ClocksTypeDef

PLL1_ClocksTypeDef is defined in the stm32h7xx_hal_rcc_ex.h

Data Fields

- ***uint32_t PLL1_P_Frequency***
- ***uint32_t PLL1_Q_Frequency***
- ***uint32_t PLL1_R_Frequency***

Field Documentation

- ***uint32_t PLL1_ClocksTypeDef::PLL1_P_Frequency***
- ***uint32_t PLL1_ClocksTypeDef::PLL1_Q_Frequency***
- ***uint32_t PLL1_ClocksTypeDef::PLL1_R_Frequency***

69.1.4 PLL2_ClocksTypeDef

PLL2_ClocksTypeDef is defined in the stm32h7xx_hal_rcc_ex.h

Data Fields

- ***uint32_t PLL2_P_Frequency***
- ***uint32_t PLL2_Q_Frequency***
- ***uint32_t PLL2_R_Frequency***

Field Documentation

- ***uint32_t PLL2_ClocksTypeDef::PLL2_P_Frequency***
- ***uint32_t PLL2_ClocksTypeDef::PLL2_Q_Frequency***
- ***uint32_t PLL2_ClocksTypeDef::PLL2_R_Frequency***

69.1.5 PLL3_ClocksTypeDef

PLL3_ClocksTypeDef is defined in the stm32h7xx_hal_rcc_ex.h

Data Fields

- ***uint32_t PLL3_P_Frequency***
- ***uint32_t PLL3_Q_Frequency***
- ***uint32_t PLL3_R_Frequency***

Field Documentation

- ***uint32_t PLL3_ClocksTypeDef::PLL3_P_Frequency***
- ***uint32_t PLL3_ClocksTypeDef::PLL3_Q_Frequency***
- ***uint32_t PLL3_ClocksTypeDef::PLL3_R_Frequency***

69.1.6 RCC_PeriphCLKInitTypeDef

RCC_PeriphCLKInitTypeDef is defined in the stm32h7xx_hal_rcc_ex.h

Data Fields

- ***uint64_t PeriphClockSelection***
- ***RCC_PLL2InitTypeDef PLL2***
- ***RCC_PLL3InitTypeDef PLL3***
- ***uint32_t FmcClockSelection***
- ***uint32_t QspiClockSelection***
- ***uint32_t SdmmcClockSelection***
- ***uint32_t CkperClockSelection***
- ***uint32_t Sai1ClockSelection***
- ***uint32_t Sai23ClockSelection***
- ***uint32_t Spi123ClockSelection***
- ***uint32_t Spi45ClockSelection***
- ***uint32_t SpdifrxClockSelection***
- ***uint32_t Dfsdm1ClockSelection***
- ***uint32_t FdcanClockSelection***
- ***uint32_t Swpmi1ClockSelection***
- ***uint32_t Usart234578ClockSelection***
- ***uint32_t Usart16ClockSelection***
- ***uint32_t RngClockSelection***
- ***uint32_t I2c123ClockSelection***
- ***uint32_t UsbClockSelection***
- ***uint32_t CecClockSelection***
- ***uint32_t Lptim1ClockSelection***
- ***uint32_t Lpuart1ClockSelection***
- ***uint32_t I2c4ClockSelection***
- ***uint32_t Lptim2ClockSelection***
- ***uint32_t Lptim345ClockSelection***
- ***uint32_t AdcClockSelection***
- ***uint32_t Sai4AClockSelection***
- ***uint32_t Sai4BClockSelection***
- ***uint32_t Spi6ClockSelection***
- ***uint32_t RTCClockSelection***
- ***uint32_t Hrtim1ClockSelection***
- ***uint32_t TIMPresSelection***

Field Documentation

- ***uint64_t RCC_PeriphCLKInitTypeDef::PeriphClockSelection***
  The Extended Clock to be configured. This parameter can be a value of ***RCCEx_Periph_Clock_Selection***
- ***RCC_PLL2InitTypeDef RCC_PeriphCLKInitTypeDef::PLL2***
  PLL2structure parameters. This parameter will be used only when PLL2 is selected as kernel clock Source for some peripherals
- ***RCC_PLL3InitTypeDef RCC_PeriphCLKInitTypeDef::PLL3***
  PLL3 structure parameters. This parameter will be used only when PLL2 is selected as kernel clock Source for some peripherals
- ***uint32_t RCC_PeriphCLKInitTypeDef::FmcClockSelection***
  Specifies FMC clock source This parameter can be a value of ***RCCEx_FMC_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::QspiClockSelection***
  Specifies QSPI clock source This parameter can be a value of ***RCCEx_QSPI_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::SdmmcClockSelection***
  Specifies SDMMC clock source This parameter can be a value of ***RCCEx_SDMMC_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::CkperClockSelection***
  Specifies CKPER clock source This parameter can be a value of ***RCCEx_CLKP_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Sai1ClockSelection***
  Specifies SAI1 clock source This parameter can be a value of ***RCCEx_SAI1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Sai23ClockSelection***
  Specifies SAI2/3 clock source This parameter can be a value of ***RCCEx_SAI23_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Spi123ClockSelection***
  Specifies SPI1/2/3 clock source This parameter can be a value of ***RCCEx_SPI123_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Spi45ClockSelection***
  Specifies SPI4/5 clock source This parameter can be a value of ***RCCEx_SPI45_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::SpdifrxClockSelection***
  Specifies SPDIFRX Clock clock source This parameter can be a value of ***RCCEx_SPDIFRX_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Dfsdm1ClockSelection***
  Specifies DFSDM1 Clock clock source This parameter can be a value of ***RCCEx_DFSDM1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::FdcanClockSelection***
  Specifies FDCAN Clock clock source This parameter can be a value of ***RCCEx_FDCAN_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Swpmi1ClockSelection***
  Specifies SWPMI1 Clock clock source This parameter can be a value of ***RCCEx_SWPMI1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Usart234578ClockSelection***
  Specifies USART2/3/4/5/7/8 clock source This parameter can be a value of ***RCCEx_USART234578_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Usart16ClockSelection***
  Specifies USART1/6 clock source This parameter can be a value of ***RCCEx_USART16_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::RngClockSelection***
  Specifies RNG clock source This parameter can be a value of ***RCCEx_RNG_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2c123ClockSelection***
  Specifies I2C1/2/3 clock source This parameter can be a value of ***RCCEx_I2C1235_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::UsbClockSelection***
  Specifies USB clock source This parameter can be a value of ***RCCEx_USB_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::CecClockSelection***
  Specifies CEC clock source This parameter can be a value of ***RCCEx_CEC_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Lptim1ClockSelection***
  Specifies LPTIM1 clock source This parameter can be a value of ***RCCEx_LPTIM1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Lpuart1ClockSelection***
  Specifies LPUART1 clock source This parameter can be a value of ***RCCEx_LPUART1_Clock_Source***

- ***uint32_t RCC_PeriphCLKInitTypeDef::I2c4ClockSelection***
  Specifies I2C4 clock source This parameter can be a value of ***RCCEx_I2C4_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Lptim2ClockSelection***
  Specifies LPTIM2 clock source This parameter can be a value of ***RCCEx_LPTIM2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Lptim345ClockSelection***
  Specifies LPTIM3/4/5 clock source This parameter can be a value of ***RCCEx_LPTIM345_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::AdcClockSelection***
  Specifies ADC interface clock source This parameter can be a value of ***RCCEx_ADC_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Sai4AClockSelection***
  Specifies SAI4A clock source This parameter can be a value of ***RCCEx_SAI4A_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Sai4BClockSelection***
  Specifies SAI4B clock source This parameter can be a value of ***RCCEx_SAI4B_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Spi6ClockSelection***
  Specifies SPI6 clock source This parameter can be a value of ***RCCEx_SPI6_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::RTCClockSelection***
  Specifies RTC Clock clock source This parameter can be a value of ***RCC_RTC_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Hrtim1ClockSelection***
  Specifies HRTIM1 Clock clock source This parameter can be a value of ***RCCEx_HRTIM1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::TIMPresSelection***
  Specifies TIM Clock Prescalers Selection. This parameter can be a value of ***RCCEx_TIM_Prescaler_Selection***

69.1.7 RCC_CRSInitTypeDef

RCC_CRSInitTypeDef is defined in the stm32h7xx_hal_rcc_ex.h

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t Source***
- ***uint32_t Polarity***
- ***uint32_t ReloadValue***
- ***uint32_t ErrorLimitValue***
- ***uint32_t HSI48CalibrationValue***

Field Documentation

- ***uint32_t RCC_CRSInitTypeDef::Prescaler***
  Specifies the division factor of the SYNC signal. This parameter can be a value of ***RCCEx_CRS_SynchroDivider***
- ***uint32_t RCC_CRSInitTypeDef::Source***
  Specifies the SYNC signal source. This parameter can be a value of ***RCCEx_CRS_SynchroSource***
- ***uint32_t RCC_CRSInitTypeDef::Polarity***
  Specifies the input polarity for the SYNC signal source. This parameter can be a value of ***RCCEx_CRS_SynchroPolarity***
- ***uint32_t RCC_CRSInitTypeDef::ReloadValue***
  Specifies the value to be loaded in the frequency error counter with each SYNC event. It can be calculated in using macro **__HAL_RCC_CRS_RELOADVALUE_CALCULATE(__FTARGET__, __FSYNC__)** This parameter must be a number between 0 and 0xFFFF or a value of ***RCCEx_CRS_ReloadValueDefault*** .
- ***uint32_t RCC_CRSInitTypeDef::ErrorLimitValue***
  Specifies the value to be used to evaluate the captured frequency error value. This parameter must be a number between 0 and 0xFF or a value of ***RCCEx_CRS_ErrorLimitDefault***
- ***uint32_t RCC_CRSInitTypeDef::HSI48CalibrationValue***
  Specifies a user-programmable trimming value to the HSI48 oscillator. This parameter must be a number between 0 and 0x3F or a value of ***RCCEx_CRS_HSI48CalibrationDefault***

69.1.8 RCC_CRSSynchroInfoTypeDef

RCC_CRSSynchroInfoTypeDef is defined in the stm32h7xx_hal_rcc_ex.h

Data Fields

- ***uint32_t ReloadValue***
- ***uint32_t HSI48CalibrationValue***
- ***uint32_t FreqErrorCapture***
- ***uint32_t FreqErrorDirection***

Field Documentation

- ***uint32_t RCC_CRSSynchroInfoTypeDef::ReloadValue***
  Specifies the value loaded in the Counter reload value. This parameter must be a number between 0 and 0xFFFF
- ***uint32_t RCC_CRSSynchroInfoTypeDef::HSI48CalibrationValue***
  Specifies value loaded in HSI48 oscillator smooth trimming. This parameter must be a number between 0 and 0x3F
- ***uint32_t RCC_CRSSynchroInfoTypeDef::FreqErrorCapture***
  Specifies the value loaded in the .FECAP, the frequency error counter value latched in the time of the last SYNC event. This parameter must be a number between 0 and 0xFFFF
- ***uint32_t RCC_CRSSynchroInfoTypeDef::FreqErrorDirection***
  Specifies the value loaded in the .FEDIR, the counting direction of the frequency error counter latched in the time of the last SYNC event. It shows whether the actual frequency is below or above the target. This parameter must be a value of ***RCCEx_CRS_FreqErrorDirection***

69.2 RCCEx Firmware driver API description

The following section lists the various functions of the RCCEx library.

69.2.1 Extended Peripheral Control functions

This subsection provides a set of functions allowing to control the RCC Clocks frequencies.

Note: Important note: Care must be taken when HAL_RCCEx_PeriphCLKConfig() is used to select the RTC clock source; in this case the Backup domain will be reset in order to modify the RTC Clock source, as consequence RTC registers (including the backup registers) and RCC_BDCR register are set to their reset values.

This section contains the following APIs:

- ***HAL_RCCEx_PeriphCLKConfig()***
- ***HAL_RCCEx_GetPeriphCLKConfig()***
- ***HAL_RCCEx_GetPeriphCLKFreq()***
- ***HAL_RCCEx_GetD1PCLK1Freq()***
- ***HAL_RCCEx_GetD3PCLK1Freq()***
- ***HAL_RCCEx_GetPLL2ClockFreq()***
- ***HAL_RCCEx_GetPLL3ClockFreq()***
- ***HAL_RCCEx_GetPLL1ClockFreq()***
- ***HAL_RCCEx_GetD1SysClockFreq()***

69.2.2 Extended Clock Recovery System Control functions

For devices with Clock Recovery System feature (CRS), RCC Extension HAL driver can be used as follows:

1. In System clock config, HSI48 needs to be enabled
2. Enable CRS clock in IP MSP init which will use CRS functions

3. Call CRS functions as follows:
   a. Prepare synchronization configuration necessary for HSI48 calibration
      - Default values can be set for frequency Error Measurement (reload and error limit) and also HSI48 oscillator smooth trimming.
      - Macro __HAL_RCC_CRS_RELOADVALUE_CALCULATE can be also used to calculate directly reload value with target and synchronization frequencies values
   b. Call function HAL_RCCEx_CRSConfig which
      - Resets CRS registers to their default values.
      - Configures CRS registers with synchronization configuration
      - Enables automatic calibration and frequency error counter feature Note: When using USB LPM (Link Power Management) and the device is in Sleep mode, the periodic USB SOF will not be generated by the host. No SYNC signal will therefore be provided to the CRS to calibrate the HSI48 on the run. To guarantee the required clock precision after waking up from Sleep mode, the LSE or reference clock on the GPIOs should be used as SYNC signal.
   c. A polling function is provided to wait for complete synchronization
      - Call function HAL_RCCEx_CRSWaitSynchronization()
      - According to CRS status, user can decide to adjust again the calibration or continue application if synchronization is OK
4. User can retrieve information related to synchronization in calling function HAL_RCCEx_CRSGetSynchronizationInfo()
5. Regarding synchronization status and synchronization information, user can try a new calibration in changing synchronization configuration and call again HAL_RCCEx_CRSConfig. Note: When the SYNC event is detected during the down-counting phase (before reaching the zero value), it means that the actual frequency is lower than the target (and so, that the TRIM value should be incremented), while when it is detected during the up-counting phase it means that the actual frequency is higher (and that the TRIM value should be decremented).
6. In interrupt mode, user can resort to the available macros (__HAL_RCC_CRS_XXX_IT). Interrupts will go through CRS Handler (CRS_IRQn/CRS_IRQHandler)
   – Call function HAL_RCCEx_CRSConfig()
   – Enable CRS_IRQn (thanks to NVIC functions)
   – Enable CRS interrupt (__HAL_RCC_CRS_ENABLE_IT)
   – Implement CRS status management in the following user callbacks called from HAL_RCCEx_CRS_IRQHandler():
      - HAL_RCCEx_CRS_SyncOkCallback()
      - HAL_RCCEx_CRS_SyncWarnCallback()
      - HAL_RCCEx_CRS_ExpectedSyncCallback()
      - HAL_RCCEx_CRS_ErrorCallback()
7. To force a SYNC EVENT, user can use the function HAL_RCCEx_CRSSoftwareSynchronizationGenerate(). This function can be called before calling HAL_RCCEx_CRSConfig (for instance in Systick handler)

This section contains the following APIs:

- ***HAL_RCCEx_CRSConfig()***
- ***HAL_RCCEx_CRSSoftwareSynchronizationGenerate()***
- ***HAL_RCCEx_CRSGetSynchronizationInfo()***
- ***HAL_RCCEx_CRSWaitSynchronization()***
- ***HAL_RCCEx_CRS_IRQHandler()***
- ***HAL_RCCEx_CRS_SyncOkCallback()***
- ***HAL_RCCEx_CRS_SyncWarnCallback()***
- ***HAL_RCCEx_CRS_ExpectedSyncCallback()***
- ***HAL_RCCEx_CRS_ErrorCallback()***

69.2.3 Detailed description of functions

HAL_RCCEx_PeriphCLKConfig

Function name

HAL_StatusTypeDef HAL_RCCEx_PeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)

Function description

Initializes the RCC extended peripherals clocks according to the specified parameters in the RCC_PeriphCLKInitTypeDef.

Parameters

- **PeriphClkInit:** pointer to an RCC_PeriphCLKInitTypeDef structure that contains the configuration information for the Extended Peripherals clocks (SDMMC, CKPER, FMC, QSPI*, OSPI*, DSI, SPI45, SPDIF, DFSDM1, DFSDM2*, FDCAN, SWPMI, SAI23*,SAI2A*, SAI2B*, SAI1, SPI123, USART234578, USART16 (USART16910*), RNG, HRTIM1*, I2C123 (I2C1235*), USB, CEC, LPTIM1, LPUART1, I2C4, LPTIM2, LPTIM345, ADC, SAI4A*, SAI4B*, SPI6, RTC).

Return values

- **HAL:** status

Notes

- Care must be taken when HAL_RCCEx_PeriphCLKConfig() is used to select the RTC clock source; in this case the Backup domain will be reset in order to modify the RTC Clock source, as consequence RTC registers (including the backup registers) are set to their reset values.

HAL_RCCEx_GetPeriphCLKConfig

Function name

void HAL_RCCEx_GetPeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)

Function description

Get the RCC_ClkInitStruct according to the internal RCC configuration registers.

Parameters

- **PeriphClkInit:** pointer to an RCC_PeriphCLKInitTypeDef structure that returns the configuration information for the Extended Peripherals clocks : (SDMMC, CKPER, FMC, QSPI*, OSPI*, DSI*, SPI45, SPDIF, DFSDM1, DFSDM2*, FDCAN, SWPMI, SAI23*, SAI1, SPI123, USART234578, USART16, RNG, HRTIM1*, I2C123 (I2C1235*), USB, CEC, LPTIM1, LPUART1, I2C4, LPTIM2, LPTIM345, ADC. SAI4A*, SAI4B*, SPI6, RTC, TIM).

Return values

- **None:** (*) : Available on some STM32H7 lines only.

HAL_RCCEx_GetPeriphCLKFreq

Function name

uint32_t HAL_RCCEx_GetPeriphCLKFreq (uint32_t PeriphClk)

Function description

Return the peripheral clock frequency for a given peripheral(SAI..)

Parameters

- **PeriphClk:** Peripheral clock identifier This parameter can be one of the following values:
  - RCC_PERIPHCLK_SAI1 : SAI1 peripheral clock
  - RCC_PERIPHCLK_SAI23 : SAI2/3 peripheral clock (*)
  - RCC_PERIPHCLK_SAI2A : SAI2A peripheral clock (*)
  - RCC_PERIPHCLK_SAI2B : SAI2B peripheral clock (*)
  - RCC_PERIPHCLK_SAI4A : SAI4A peripheral clock (*)
  - RCC_PERIPHCLK_SAI4B : SAI4B peripheral clock (*)
  - RCC_PERIPHCLK_SPI123: SPI1/2/3 peripheral clock
  - RCC_PERIPHCLK_ADC : ADC peripheral clock
  - RCC_PERIPHCLK_SDMMC : SDMMC peripheral clock
  - RCC_PERIPHCLK_SPI6 : SPI6 peripheral clock

Return values

- **Frequency:** in KHz

Notes

- Return 0 if peripheral clock identifier not managed by this API

HAL_RCCEx_GetD1PCLK1Freq

Function name

uint32_t HAL_RCCEx_GetD1PCLK1Freq (void)

Function description

Returns the D1PCLK1 frequency.

Return values

- **D1PCLK1:** frequency

Notes

- Each time D1PCLK1 changes, this function must be called to update the right D1PCLK1 value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCCEx_GetD3PCLK1Freq

Function name

uint32_t HAL_RCCEx_GetD3PCLK1Freq (void)

Function description

Returns the D3PCLK1 frequency.

Return values

- **D3PCLK1:** frequency

Notes

- Each time D3PCLK1 changes, this function must be called to update the right D3PCLK1 value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCCEx_GetD1SysClockFreq

Function name

uint32_t HAL_RCCEx_GetD1SysClockFreq (void)

Function description

Returns the main System frequency.

Return values

- **HCLK:** frequency

Notes

- Each time System clock changes, this function must be called to update the right core clock value. Otherwise, any configuration based on this function will be incorrect.
- The SystemCoreClock CMSIS variable is used to store System current Core Clock Frequency and updated within this function

HAL_RCCEx_GetPLL1ClockFreq

Function name

void HAL_RCCEx_GetPLL1ClockFreq (PLL1_ClocksTypeDef * PLL1_Clocks)

Function description

Returns the PLL1 clock frequencies :PLL1_P_Frequency,PLL1_R_Frequency and PLL1_Q_Frequency.

Parameters

- **PLL1_Clocks:** structure.

Return values

- **None:**

Notes

- The PLL1 clock frequencies computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:
- The function returns values based on HSE_VALUE, HSI_VALUE or CSI Value multiplied/divided by the PLL factors.
- This function can be used by the user application to compute the baud-rate for the communication peripherals or configure other parameters.
- Each time PLL1CLK changes, this function must be called to update the right PLL1CLK value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCCEx_GetPLL2ClockFreq

Function name

void HAL_RCCEx_GetPLL2ClockFreq (PLL2_ClocksTypeDef * PLL2_Clocks)

Function description

Returns the PLL2 clock frequencies :PLL2_P_Frequency,PLL2_R_Frequency and PLL2_Q_Frequency.

Parameters

- **PLL2_Clocks:** structure.

Return values

- **None:**

Notes

- The PLL2 clock frequencies computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:
- The function returns values based on HSE_VALUE, HSI_VALUE or CSI Value multiplied/divided by the PLL factors.
- This function can be used by the user application to compute the baud-rate for the communication peripherals or configure other parameters.
- Each time PLL2CLK changes, this function must be called to update the right PLL2CLK value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCCEx_GetPLL3ClockFreq

Function name

void HAL_RCCEx_GetPLL3ClockFreq (PLL3_ClocksTypeDef * PLL3_Clocks)

Function description

Returns the PLL3 clock frequencies :PLL3_P_Frequency,PLL3_R_Frequency and PLL3_Q_Frequency.

Parameters

- **PLL3_Clocks:** structure.

Return values

- **None:**

Notes

- The PLL3 clock frequencies computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:
- The function returns values based on HSE_VALUE, HSI_VALUE or CSI Value multiplied/divided by the PLL factors.
- This function can be used by the user application to compute the baud-rate for the communication peripherals or configure other parameters.
- Each time PLL3CLK changes, this function must be called to update the right PLL3CLK value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCCEx_WakeUpStopCLKConfig

Function name

void HAL_RCCEx_WakeUpStopCLKConfig (uint32_t WakeUpClk)

Function description

Configure the oscillator clock source for wakeup from Stop and CSS backup clock.

Parameters

- **WakeUpClk:** Wakeup clock This parameter can be one of the following values:
  - RCC_STOP_WAKEUPCLOCK_CSI: CSI oscillator selection
  - RCC_STOP_WAKEUPCLOCK_HSI: HSI oscillator selection

Return values

- **None:**

Notes

- This function shall not be called after the Clock Security System on HSE has been enabled.

HAL_RCCEx_KerWakeUpStopCLKConfig

Function name

void HAL_RCCEx_KerWakeUpStopCLKConfig (uint32_t WakeUpClk)

Function description

Configure the oscillator Kernel clock source for wakeup from Stop.

Parameters

- **WakeUpClk:** Kernel Wakeup clock This parameter can be one of the following values:
  - RCC_STOP_KERWAKEUPCLOCK_CSI: CSI oscillator selection
  - RCC_STOP_KERWAKEUPCLOCK_HSI: HSI oscillator selection

Return values

- **None:**

HAL_RCCEx_EnableLSECSS

Function name

void HAL_RCCEx_EnableLSECSS (void)

Function description

Enables the LSE Clock Security System.

Return values

- **None:**

Notes

- Prior to enable the LSE Clock Security System, LSE oscillator is to be enabled with HAL_RCC_OscConfig() and the LSE oscillator clock is to be selected as RTC clock with HAL_RCCEx_PeriphCLKConfig().

HAL_RCCEx_DisableLSECSS

Function name

void HAL_RCCEx_DisableLSECSS (void)

Function description

Disables the LSE Clock Security System.

Return values

- **None:**

Notes

- LSE Clock Security System can only be disabled after a LSE failure detection.

HAL_RCCEx_EnableLSECSS_IT

Function name

void HAL_RCCEx_EnableLSECSS_IT (void)

Function description

Enable the LSE Clock Security System Interrupt & corresponding EXTI line.

Return values

- **None:**

Notes

- LSE Clock Security System Interrupt is mapped on EXTI line 18

HAL_RCCEx_LSECSS_IRQHandler

Function name

void HAL_RCCEx_LSECSS_IRQHandler (void)

Function description

Handle the RCC LSE Clock Security System interrupt request.

Return values

- **None:**

HAL_RCCEx_LSECSS_Callback

Function name

void HAL_RCCEx_LSECSS_Callback (void)

Function description

RCCEx LSE Clock Security System interrupt callback.

Return values

- **none:**

HAL_RCCEx_EnableBootCore

Function name

void HAL_RCCEx_EnableBootCore (uint32_t RCC_BootCx)

Function description

Enable COREx boot independently of CMx_B option byte value.

Parameters

- **RCC_BootCx:** Boot Core to be enabled This parameter can be one of the following values:
  - RCC_BOOT_C1: CM7 core selection
  - RCC_BOOT_C2: CM4 core selection

Return values

- **None:**

Notes

- This bit can be set by software but is cleared by hardware after a system reset or STANDBY

HAL_RCCEx_WWDGxSysResetConfig

Function name

void HAL_RCCEx_WWDGxSysResetConfig (uint32_t RCC_WWDGx)

Function description

Configure WWDGx to generate a system reset not only CPUx reset(default) when a time-out occurs.

Parameters

- **RCC_WWDGx:** WWDGx to be configured This parameter can be one of the following values:
  - RCC_WWDG1: WWDG1 generates system reset
  - RCC_WWDG2: WWDG2 generates system reset

Return values

- **None:**

Notes

- This bit can be set by software but is cleared by hardware during a system reset

HAL_RCCEx_CRSConfig

Function name

void HAL_RCCEx_CRSConfig (RCC_CRSInitTypeDef * pInit)

Function description

Start automatic synchronization for polling mode.

Parameters

- **pInit:** Pointer on RCC_CRSInitTypeDef structure

Return values

- **None:**

HAL_RCCEx_CRSSoftwareSynchronizationGenerate

Function name

void HAL_RCCEx_CRSSoftwareSynchronizationGenerate (void)

Function description

Generate the software synchronization event.

Return values

- **None:**

HAL_RCCEx_CRSGetSynchronizationInfo

Function name

void HAL_RCCEx_CRSGetSynchronizationInfo (RCC_CRSSynchroInfoTypeDef * pSynchroInfo)

Function description

Return synchronization info.

Parameters

- **pSynchroInfo:** Pointer on RCC_CRSSynchroInfoTypeDef structure

Return values

- **None:**

HAL_RCCEx_CRSWaitSynchronization

Function name

uint32_t HAL_RCCEx_CRSWaitSynchronization (uint32_t Timeout)

Function description

Wait for CRS Synchronization status.

Parameters

- **Timeout:** Duration of the time-out

Return values

- **Combination:** of Synchronization status This parameter can be a combination of the following values:
  - RCC_CRS_TIMEOUT
  - RCC_CRS_SYNCOK
  - RCC_CRS_SYNCWARN
  - RCC_CRS_SYNCERR
  - RCC_CRS_SYNCMISS
  - RCC_CRS_TRIMOVF

Notes

- Timeout is based on the maximum time to receive a SYNC event based on synchronization frequency.
- If Time-out set to HAL_MAX_DELAY, HAL_TIMEOUT will be never returned.

HAL_RCCEx_CRS_IRQHandler

Function name

void HAL_RCCEx_CRS_IRQHandler (void)

Function description

Handle the Clock Recovery System interrupt request.

Return values

- **None:**

HAL_RCCEx_CRS_SyncOkCallback

Function name

void HAL_RCCEx_CRS_SyncOkCallback (void)

Function description

RCCEx Clock Recovery System SYNCOK interrupt callback.

Return values

- **none:**

HAL_RCCEx_CRS_SyncWarnCallback

Function name

void HAL_RCCEx_CRS_SyncWarnCallback (void)

Function description

RCCEx Clock Recovery System SYNCWARN interrupt callback.

Return values

- **none:**

HAL_RCCEx_CRS_ExpectedSyncCallback

Function name

void HAL_RCCEx_CRS_ExpectedSyncCallback (void)

Function description

RCCEx Clock Recovery System Expected SYNC interrupt callback.

Return values

- **none:**

HAL_RCCEx_CRS_ErrorCallback

Function name

void HAL_RCCEx_CRS_ErrorCallback (uint32_t Error)

Function description

RCCEx Clock Recovery System Error interrupt callback.

Parameters

- **Error:** Combination of Error status. This parameter can be a combination of the following values:
  - RCC_CRS_SYNCERR
  - RCC_CRS_SYNCMISS
  - RCC_CRS_TRIMOVF

Return values

- **none:**

69.3 RCCEx Firmware driver defines

The following section lists the various define and macros of the module.

69.3.1 RCCEx

RCCEx

RCCEx ADC Clock Source

RCC_ADCCLKSOURCE_PLL2

RCC_ADCCLKSOURCE_PLL3

RCC_ADCCLKSOURCE_CLKP

RCCEx CEC Clock Source

RCC_CECCLKSOURCE_LSE

RCC_CECCLKSOURCE_LSI

RCC_CECCLKSOURCE_CSI

RCCEx CLKP Clock Source

RCC_CLKPSOURCE_HSI

RCC_CLKPSOURCE_CSI

RCC_CLKPSOURCE_HSE

RCCEx CRS ErrorLimitDefault

RCC_CRS_ERRORLIMIT_DEFAULT

Default Frequency error limit

RCCEx CRS Extended Features

__HAL_RCC_CRS_FREQ_ERROR_COUNTER_ENABLE

Description:

- Enable the oscillator clock for frequency error counter.

Return value:

- None

Notes:

- when the CEN bit is set the CRS_CFGR register becomes write-protected.

__HAL_RCC_CRS_FREQ_ERROR_COUNTER_DISABLE

Description:

- Disable the oscillator clock for frequency error counter.

Return value:

- None

__HAL_RCC_CRS_AUTOMATIC_CALIB_ENABLE

Description:

- Enable the automatic hardware adjustment of TRIM bits.

Return value:

- None

Notes:

- When the AUTOTRIMEN bit is set the CRS_CFGR register becomes write-protected.

__HAL_RCC_CRS_AUTOMATIC_CALIB_DISABLE

Description:

- Enable or disable the automatic hardware adjustment of TRIM bits.

Return value:

- None

__HAL_RCC_CRS_RELOADVALUE_CALCULATE

Description:

- Macro to calculate reload value to be set in CRS register according to target and sync frequencies.

Parameters:

- __FTARGET__: Target frequency (value in Hz)
- __FSYNC__: Synchronization signal frequency (value in Hz)

Return value:

- None

Notes:

- The RELOAD value should be selected according to the ratio between the target frequency and the frequency of the synchronization source after pre-scaling. It is then decreased by one in order to reach the expected synchronization on the zero value. The formula is the following: RELOAD = (fTARGET / fSYNC) -1

RCCEx CRS Flags

RCC_CRS_FLAG_SYNCOK

SYNC event OK flag

RCC_CRS_FLAG_SYNCWARN

SYNC warning flag

RCC_CRS_FLAG_ERR

Error flag

RCC_CRS_FLAG_ESYNC

Expected SYNC flag

RCC_CRS_FLAG_SYNCERR

SYNC error

RCC_CRS_FLAG_SYNCMISS

SYNC missed

RCC_CRS_FLAG_TRIMOVF

Trimming overflow or underflow

RCCEx CRS FreqErrorDirection

RCC_CRS_FREQERRORDIR_UP

Upcounting direction, the actual frequency is above the target

RCC_CRS_FREQERRORDIR_DOWN

Downcounting direction, the actual frequency is below the target

RCCEx CRS HSI48CalibrationDefault

RCC_CRS_HSI48CALIBRATION_DEFAULT

The default value is 32, which corresponds to the middle of the trimming interval. The trimming step is around 67 kHz between two consecutive TRIM steps. A higher TRIM value corresponds to a higher output frequency

RCCEx CRS Interrupt Sources

RCC_CRS_IT_SYNCOK

SYNC event OK

RCC_CRS_IT_SYNCWARN

SYNC warning

RCC_CRS_IT_ERR

Error

RCC_CRS_IT_ESYNC

Expected SYNC

RCC_CRS_IT_SYNCERR

SYNC error

RCC_CRS_IT_SYNCMISS

SYNC missed

RCC_CRS_IT_TRIMOVF

Trimming overflow or underflow

RCCEx CRS ReloadValueDefault

RCC_CRS_RELOADVALUE_DEFAULT

The reset value of the RELOAD field corresponds to a target frequency of 48 MHz and a synchronization signal frequency of 1 kHz (SOF signal from USB).

RCCEx CRS Status

RCC_CRS_NONE

RCC_CRS_TIMEOUT

RCC_CRS_SYNCOK

RCC_CRS_SYNCWARN

RCC_CRS_SYNCERR

RCC_CRS_SYNCMISS

RCC_CRS_TRIMOVF

RCCEx CRS SynchroDivider

RCC_CRS_SYNC_DIV1

Synchro Signal not divided (default)

RCC_CRS_SYNC_DIV2

Synchro Signal divided by 2

RCC_CRS_SYNC_DIV4

Synchro Signal divided by 4

RCC_CRS_SYNC_DIV8

Synchro Signal divided by 8

RCC_CRS_SYNC_DIV16

Synchro Signal divided by 16

RCC_CRS_SYNC_DIV32

Synchro Signal divided by 32

RCC_CRS_SYNC_DIV64

Synchro Signal divided by 64

RCC_CRS_SYNC_DIV128

Synchro Signal divided by 128

RCCEx CRS SynchroPolarity

RCC_CRS_SYNC_POLARITY_RISING

Synchro Active on rising edge (default)

RCC_CRS_SYNC_POLARITY_FALLING

Synchro Active on falling edge

RCCEx CRS SynchroSource

RCC_CRS_SYNC_SOURCE_PIN

Synchro Signal source external pin, Available on STM32H7 Rev.B and above devices only

RCC_CRS_SYNC_SOURCE_LSE

Synchro Signal source LSE

RCC_CRS_SYNC_SOURCE_USB1

Synchro Signal source USB1 SOF (default)

RCC_CRS_SYNC_SOURCE_USB2

Synchro Signal source USB2 SOF

RCCEx DFSDM1 Clock Source

RCC_DFSDM1CLKSOURCE_D2PCLK1

RCC_DFSDM1CLKSOURCE_SYS

RCCEx Exported Macros

__HAL_RCC_PLL2_ENABLE

Notes:

- After enabling PLL2, the application software should wait on PLL2RDY flag to be set indicating that PLL2 clock is stable and can be used as kernel clock source. PLL2 is disabled by hardware when entering STOP and STANDBY modes.

__HAL_RCC_PLL2_DISABLE

__HAL_RCC_PLL2CLKOUT_ENABLE

Description:

- Enables or disables each clock output (PLL2_P_CLK, PLL2_Q_CLK, PLL2_R_CLK)

Parameters:

- __RCC_PLL2ClockOut__: Specifies the PLL2 clock to be outputted This parameter can be one of the following values:
  - RCC_PLL2_DIVP: This clock is used to generate peripherals clock up to 550MHZ(*), 480MHZ(**) or 280MHZ(***)
  - RCC_PLL2_DIVQ: This clock is used to generate peripherals clock up to 550MHZ(*), 480MHZ(**) or 280MHZ(***)
  - RCC_PLL2_DIVR: This clock is used to generate peripherals clock up to 550MHZ(*), 480MHZ(**) or 280MHZ(***)

Return value:

- None

Notes:

- Enabling/disabling those Clocks can be done only when the PLL2 is disabled, This is mainly used to save Power.

__HAL_RCC_PLL2CLKOUT_DISABLE

__HAL_RCC_PLL2FRACN_ENABLE

Description:

- Enables or disables Fractional Part Of The Multiplication Factor of PLL2 VCO.

Return value:

- None

Notes:

- Enabling/disabling Fractional Part can be any time without the need to stop the PLL2

__HAL_RCC_PLL2FRACN_DISABLE

__HAL_RCC_PLL2_CONFIG

Description:

- Macro to configures the PLL2 multiplication and division factors.

Parameters:

- __PLL2M__: specifies the division factor for PLL2 VCO input clock This parameter must be a number between 1 and 63.
- __PLL2N__: specifies the multiplication factor for PLL2 VCO output clock This parameter must be a number between 4 and 512 or between 8 and 420(*).
- __PLL2P__: specifies the division factor for peripheral kernel clocks This parameter must be a number between 1 and 128.
- __PLL2Q__: specifies the division factor for peripheral kernel clocks This parameter must be a number between 1 and 128.
- __PLL2R__: specifies the division factor for peripheral kernel clocks This parameter must be a number between 1 and 128.

Return value:

- None: (*) : For stm32h7a3xx and stm32h7b3xx family lines.

Notes:

- This function must be used only when PLL2 is disabled.
- You have to set the PLLM parameter correctly to ensure that the VCO input frequency ranges from 1 to 16 MHz.
- You have to set the PLL2N parameter correctly to ensure that the VCO output frequency is between 150 and 420 MHz (when in medium VCO range) or between 192 and 836 MHZ or between 128 and 560 MHZ(*) (when in wide VCO range)
- To insure an optimal behavior of the PLL when one of the post-divider (DIVP, DIVQ or DIVR) is not used, application shall clear the enable bit (DIVyEN) and assign lowest possible value to __PLL2P__, __PLL2Q__ or __PLL2R__ parameters.

__HAL_RCC_PLL2FRACN_CONFIG

Description:

- Macro to configures PLL2 clock Fractional Part Of The Multiplication Factor.

Parameters:

- __RCC_PLL2FRACN__: Specifies Fractional Part Of The Multiplication factor for PLL2 VCO It should be a value between 0 and 8191

Return value:

- None

Notes:

- These bits can be written at any time, allowing dynamic fine-tuning of the PLL2 VCO
- Warning: the software has to set correctly these bits to insure that the VCO output frequency is between its valid frequency range, which is: 192 to 836 MHz or 128 to 560 MHz(*) if PLL2VCOSEL = 0 150 to 420 MHz if PLL2VCOSEL = 1.

__HAL_RCC_PLL2_VCIRANGE

Description:

- Macro to select the PLL2 reference frequency range.

Parameters:

- __RCC_PLL2VCIRange__: specifies the PLL2 input frequency range This parameter can be one of the following values:
  - RCC_PLL2VCIRANGE_0: Range frequency is between 1 and 2 MHz
  - RCC_PLL2VCIRANGE_1: Range frequency is between 2 and 4 MHz
  - RCC_PLL2VCIRANGE_2: Range frequency is between 4 and 8 MHz
  - RCC_PLL2VCIRANGE_3: Range frequency is between 8 and 16 MHz

Return value:

- None

__HAL_RCC_PLL2_VCORANGE

Description:

- Macro to select the PLL2 reference frequency range.

Parameters:

- __RCC_PLL2VCORange__: Specifies the PLL2 input frequency range This parameter can be one of the following values:
  - RCC_PLL2VCOWIDE: Range frequency is between 192 and 836 MHz or between 128 to 560 MHz(*)
  - RCC_PLL2VCOMEDIUM: Range frequency is between 150 and 420 MHz

Return value:

- None

__HAL_RCC_PLL3_ENABLE

Notes:

- After enabling PLL3, the application software should wait on PLL3RDY flag to be set indicating that PLL3 clock is stable and can be used as kernel clock source. PLL3 is disabled by hardware when entering STOP and STANDBY modes.

__HAL_RCC_PLL3_DISABLE

__HAL_RCC_PLL3FRACN_ENABLE

Description:

- Enables or disables Fractional Part Of The Multiplication Factor of PLL3 VCO.

Return value:

- None

Notes:

- Enabling/disabling Fractional Part can be any time without the need to stop the PLL3

__HAL_RCC_PLL3FRACN_DISABLE

__HAL_RCC_PLL3CLKOUT_ENABLE

Description:

- Enables or disables each clock output (PLL3_P_CLK, PLL3_Q_CLK, PLL3_R_CLK)

Parameters:

- __RCC_PLL3ClockOut__: specifies the PLL3 clock to be outputted This parameter can be one of the following values:
  - RCC_PLL3_DIVP: This clock is used to generate peripherals clock up to 550MHZ(*), 480MHZ(**) or 280MHZ(***)
  - RCC_PLL3_DIVQ: This clock is used to generate peripherals clock up to 550MHZ(*), 480MHZ(**) or 280MHZ(***)
  - RCC_PLL3_DIVR: This clock is used to generate peripherals clock up to 550MHZ(*), 480MHZ(**) or 280MHZ(***)

Return value:

- None

Notes:

- Enabling/disabling those Clocks can be done only when the PLL3 is disabled, This is mainly used to save Power.

__HAL_RCC_PLL3CLKOUT_DISABLE

__HAL_RCC_PLL3_CONFIG

Description:

- Macro to configures the PLL3 multiplication and division factors.

Parameters:

- __PLL3M__: specifies the division factor for PLL3 VCO input clock This parameter must be a number between 1 and 63.
- __PLL3N__: specifies the multiplication factor for PLL3 VCO output clock This parameter must be a number between 4 and 512.
- __PLL3P__: specifies the division factor for peripheral kernel clocks This parameter must be a number between 2 and 128 (where odd numbers not allowed)
- __PLL3Q__: specifies the division factor for peripheral kernel clocks This parameter must be a number between 1 and 128
- __PLL3R__: specifies the division factor for peripheral kernel clocks This parameter must be a number between 1 and 128

Return value:

- None: (*) : For stm32h7a3xx and stm32h7b3xx family lines.

Notes:

- This function must be used only when PLL3 is disabled.
- You have to set the PLLM parameter correctly to ensure that the VCO input frequency ranges from 1 to 16 MHz.
- You have to set the PLL3N parameter correctly to ensure that the VCO output frequency is between 150 and 420 MHz (when in medium VCO range) or between 192 and 836 MHZ or between 128 and 560 MHZ(*) (when in wide VCO range)
- To insure an optimal behavior of the PLL when one of the post-divider (DIVP, DIVQ or DIVR) is not used, application shall clear the enable bit (DIVyEN) and assign lowest possible value to __PLL3P__, __PLL3Q__ or __PLL3R__ parameters.

__HAL_RCC_PLL3FRACN_CONFIG

Description:

- Macro to configures PLL3 clock Fractional Part of The Multiplication Factor.

Parameters:

- __RCC_PLL3FRACN__: specifies Fractional Part Of The Multiplication Factor for PLL3 VCO It should be a value between 0 and 8191

Return value:

- None

Notes:

- These bits can be written at any time, allowing dynamic fine-tuning of the PLL3 VCO
- Warning: the software has to set correctly these bits to insure that the VCO output frequency is between its valid frequency range, which is: 192 to 836 MHz or 128 to 560 MHz(*) if PLL3VCOSEL = 0 150 to 420 MHz if PLL3VCOSEL = 1.

__HAL_RCC_PLL3_VCIRANGE

Description:

- Macro to select the PLL3 reference frequency range.

Parameters:

- __RCC_PLL3VCIRange__: specifies the PLL1 input frequency range This parameter can be one of the following values:
  - RCC_PLL3VCIRANGE_0: Range frequency is between 1 and 2 MHz
  - RCC_PLL3VCIRANGE_1: Range frequency is between 2 and 4 MHz
  - RCC_PLL3VCIRANGE_2: Range frequency is between 4 and 8 MHz
  - RCC_PLL3VCIRANGE_3: Range frequency is between 8 and 16 MHz

Return value:

- None

__HAL_RCC_PLL3_VCORANGE

Description:

- Macro to select the PLL3 reference frequency range.

Parameters:

- __RCC_PLL3VCORange__: specifies the PLL1 input frequency range This parameter can be one of the following values:
  - RCC_PLL3VCOWIDE: Range frequency is between 192 and 836 MHz or between 128 to 560 MHz(*)
  - RCC_PLL3VCOMEDIUM: Range frequency is between 150 and 420 MHz

Return value:

- None

__HAL_RCC_SAI1_CONFIG

Description:

- Macro to Configure the SAI1 clock source.

Parameters:

- __RCC_SAI1CLKSource__: defines the SAI1 clock source. This clock is derived from system PLL, PLL2, PLL3, OSC or external clock (through a dedicated PIN) This parameter can be one of the following values:
  - RCC_SAI1CLKSOURCE_PLL: SAI1 clock = PLL
  - RCC_SAI1CLKSOURCE_PLL2: SAI1 clock = PLL2
  - RCC_SAI1CLKSOURCE_PLL3: SAI1 clock = PLL3
  - RCC_SAI1CLKSOURCE_OSC: SAI1 clock = OSC
  - RCC_SAI1CLKSOURCE_PIN: SAI1 clock = External Clock

Return value:

- None

__HAL_RCC_GET_SAI1_SOURCE

Description:

- Macro to get the SAI1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SAI1CLKSOURCE_PLL: SAI1 clock = PLL
  - RCC_SAI1CLKSOURCE_PLL2: SAI1 clock = PLL2
  - RCC_SAI1CLKSOURCE_PLL3: SAI1 clock = PLL3
  - RCC_SAI1CLKSOURCE_CLKP: SAI1 clock = CLKP
  - RCC_SAI1CLKSOURCE_PIN: SAI1 clock = External Clock

__HAL_RCC_SPDIFRX_CONFIG

Description:

- Macro to Configure the SPDIFRX clock source.

Parameters:

- __RCC_SPDIFCLKSource__: defines the SPDIFRX clock source. This clock is derived from system PLL, PLL2, PLL3, or internal OSC clock This parameter can be one of the following values:
  - RCC_SPDIFRXCLKSOURCE_PLL: SPDIFRX clock = PLL
  - RCC_SPDIFRXCLKSOURCE_PLL2: SPDIFRX clock = PLL2
  - RCC_SPDIFRXCLKSOURCE_PLL3: SPDIFRX clock = PLL3
  - RCC_SPDIFRXCLKSOURCE_HSI: SPDIFRX clock = HSI

Return value:

- None

__HAL_RCC_GET_SPDIFRX_SOURCE

Description:

- Macro to get the SPDIFRX clock source.

Return value:

- None

__HAL_RCC_SAI23_CONFIG

Description:

- Macro to Configure the SAI2/3 clock source.

Parameters:

- __RCC_SAI23CLKSource__: defines the SAI2/3 clock source. This clock is derived from system PLL, PLL2, PLL3, OSC or external clock (through a dedicated PIN) This parameter can be one of the following values:
  - RCC_SAI23CLKSOURCE_PLL: SAI2/3 clock = PLL
  - RCC_SAI23CLKSOURCE_PLL2: SAI2/3 clock = PLL2
  - RCC_SAI23CLKSOURCE_PLL3: SAI2/3 clock = PLL3
  - RCC_SAI23CLKSOURCE_CLKP: SAI2/3 clock = CLKP
  - RCC_SAI23CLKSOURCE_PIN: SAI2/3 clock = External Clock

Return value:

- None

__HAL_RCC_GET_SAI23_SOURCE

Description:

- Macro to get the SAI2/3 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SAI23CLKSOURCE_PLL: SAI2/3 clock = PLL
  - RCC_SAI23CLKSOURCE_PLL2: SAI2/3 clock = PLL2
  - RCC_SAI23CLKSOURCE_PLL3: SAI2/3 clock = PLL3
  - RCC_SAI23CLKSOURCE_CLKP: SAI2/3 clock = CLKP
  - RCC_SAI23CLKSOURCE_PIN: SAI2/3 clock = External Clock

__HAL_RCC_SAI2_CONFIG

Description:

- Macro to Configure the SAI2 clock source.

Parameters:

- __RCC_SAI2CLKSource__: defines the SAI2 clock source. This clock is derived from system PLL, PLL2, PLL3, OSC or external clock (through a dedicated PIN) This parameter can be one of the following values:
  - RCC_SAI2CLKSOURCE_PLL: SAI2 clock = PLL
  - RCC_SAI2CLKSOURCE_PLL2: SAI2 clock = PLL2
  - RCC_SAI2CLKSOURCE_PLL3: SAI2 clock = PLL3
  - RCC_SAI2CLKSOURCE_CLKP: SAI2 clock = CLKP
  - RCC_SAI2CLKSOURCE_PIN: SAI2 clock = External Clock

Return value:

- None

__HAL_RCC_GET_SAI2_SOURCE

Description:

- Macro to get the SAI2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SAI2CLKSOURCE_PLL: SAI2 clock = PLL
  - RCC_SAI2CLKSOURCE_PLL2: SAI2 clock = PLL2
  - RCC_SAI2CLKSOURCE_PLL3: SAI2 clock = PLL3
  - RCC_SAI2CLKSOURCE_CLKP: SAI2 clock = CLKP
  - RCC_SAI2CLKSOURCE_PIN: SAI2 clock = External Clock

__HAL_RCC_SAI3_CONFIG

Description:

- Macro to Configure the SAI3 clock source.

Parameters:

- __RCC_SAI3CLKSource__: defines the SAI3 clock source. This clock is derived from system PLL, PLL2, PLL3, OSC or external clock (through a dedicated PIN) This parameter can be one of the following values:
  - RCC_SAI3CLKSOURCE_PLL: SAI3 clock = PLL
  - RCC_SAI3CLKSOURCE_PLL2: SAI3 clock = PLL2
  - RCC_SAI3CLKSOURCE_PLL3: SAI3 clock = PLL3
  - RCC_SAI3CLKSOURCE_CLKP: SAI3 clock = CLKP
  - RCC_SAI3CLKSOURCE_PIN: SAI3 clock = External Clock

Return value:

- None

__HAL_RCC_GET_SAI3_SOURCE

Description:

- Macro to get the SAI3 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SAI3CLKSOURCE_PLL: SAI3 clock = PLL
  - RCC_SAI3CLKSOURCE_PLL2: SAI3 clock = PLL2
  - RCC_SAI3CLKSOURCE_PLL3: SAI3 clock = PLL3
  - RCC_SAI3CLKSOURCE_CLKP: SAI3 clock = CLKP
  - RCC_SAI3CLKSOURCE_PIN: SAI3 clock = External Clock

__HAL_RCC_SAI4A_CONFIG

Description:

- Macro to Configure the SAI4A clock source.

Parameters:

- __RCC_SAI4ACLKSource__: defines the SAI4A clock source. This clock is derived from system PLL, PLL2, PLL3, OSC or external clock (through a dedicated PIN) This parameter can be one of the following values:
  - RCC_SAI4ACLKSOURCE_PLL: SAI4A clock = PLL
  - RCC_SAI4ACLKSOURCE_PLL2: SAI4A clock = PLL2
  - RCC_SAI4ACLKSOURCE_PLL3: SAI4A clock = PLL3
  - RCC_SAI4ACLKSOURCE_CLKP: SAI4A clock = CLKP
  - RCC_SAI4ACLKSOURCE_PIN: SAI4A clock = External Clock

Return value:

- None

__HAL_RCC_GET_SAI4A_SOURCE

Description:

- Macro to get the SAI4A clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SAI4ACLKSOURCE_PLL: SAI4B clock = PLL
  - RCC_SAI4ACLKSOURCE_PLL2: SAI4B clock = PLL2
  - RCC_SAI4ACLKSOURCE_PLL3: SAI4B clock = PLL3
  - RCC_SAI4ACLKSOURCE_CLKP: SAI4B clock = CLKP
  - RCC_SAI4ACLKSOURCE_PIN: SAI4B clock = External Clock

__HAL_RCC_SAI4B_CONFIG

Description:

- Macro to Configure the SAI4B clock source.

Parameters:

- __RCC_SAI4BCLKSource__: defines the SAI4B clock source. This clock is derived from system PLL, PLL2, PLL3, OSC or external clock (through a dedicated PIN) This parameter can be one of the following values:
  - RCC_SAI4BCLKSOURCE_PLL: SAI4B clock = PLL
  - RCC_SAI4BCLKSOURCE_PLL2: SAI4B clock = PLL2
  - RCC_SAI4BCLKSOURCE_PLL3: SAI4B clock = PLL3
  - RCC_SAI4BCLKSOURCE_CLKP: SAI4B clock = CLKP
  - RCC_SAI4BCLKSOURCE_PIN: SAI4B clock = External Clock

Return value:

- None

__HAL_RCC_GET_SAI4B_SOURCE

Description:

- Macro to get the SAI4B clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SAI4BCLKSOURCE_PLL: SAI4B clock = PLL
  - RCC_SAI4BCLKSOURCE_PLL2: SAI4B clock = PLL2
  - RCC_SAI4BCLKSOURCE_PLL3: SAI4B clock = PLL3
  - RCC_SAI4BCLKSOURCE_CLKP: SAI4B clock = CLKP
  - RCC_SAI4BCLKSOURCE_PIN: SAI4B clock = External Clock

__HAL_RCC_I2C123_CONFIG

Description:

- macro to configure the I2C1/2/3/5* clock (I2C123CLK).

Parameters:

- __I2C1235CLKSource__: specifies the I2C1/2/3/5* clock source. This parameter can be one of the following values:
  - RCC_I2C123CLKSOURCE_D2PCLK1: D2PCLK1 selected as I2C1/2/3/5* clock
  - RCC_I2C123CLKSOURCE_PLL3: PLL3 selected as I2C1/2/3/5* clock
  - RCC_I2C123CLKSOURCE_HSI: HSI selected as I2C1/2/3/5* clock
  - RCC_I2C123CLKSOURCE_CSI: CSI selected as I2C1/2/3/5* clock

__HAL_RCC_GET_I2C123_SOURCE

Description:

- macro to get the I2C1/2/3/5* clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I2C123CLKSOURCE_D2PCLK1: D2PCLK1 selected as I2C1/2/3/5* clock
  - RCC_I2C123CLKSOURCE_PLL3: PLL3 selected as I2C1/2/3/5* clock
  - RCC_I2C123CLKSOURCE_HSI: HSI selected as I2C1/2/3/5* clock
  - RCC_I2C123CLKSOURCE_CSI: CSI selected as I2C1/2/3/5* clock

__HAL_RCC_I2C1_CONFIG

Description:

- macro to configure the I2C1 clock (I2C1CLK).

Parameters:

- __I2C1CLKSource__: specifies the I2C1 clock source. This parameter can be one of the following values:
  - RCC_I2C1CLKSOURCE_D2PCLK1: D2PCLK1 selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_PLL3: PLL3 selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_HSI: HSI selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_CSI: CSI selected as I2C1 clock

__HAL_RCC_GET_I2C1_SOURCE

Description:

- macro to get the I2C1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I2C1CLKSOURCE_D2PCLK1: D2PCLK1 selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_PLL3: PLL3 selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_HSI: HSI selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_CSI: CSI selected as I2C1 clock

__HAL_RCC_I2C2_CONFIG

Description:

- macro to configure the I2C2 clock (I2C2CLK).

Parameters:

- __I2C2CLKSource__: specifies the I2C2 clock source. This parameter can be one of the following values:
  - RCC_I2C2CLKSOURCE_D2PCLK1: D2PCLK1 selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_PLL3: PLL3 selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_HSI: HSI selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_CSI: CSI selected as I2C2 clock

__HAL_RCC_GET_I2C2_SOURCE

Description:

- macro to get the I2C2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I2C2CLKSOURCE_D2PCLK1: D2PCLK1 selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_PLL3: PLL3 selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_HSI: HSI selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_CSI: CSI selected as I2C2 clock

__HAL_RCC_I2C3_CONFIG

Description:

- macro to configure the I2C3 clock (I2C3CLK).

Parameters:

- __I2C3CLKSource__: specifies the I2C3 clock source. This parameter can be one of the following values:
  - RCC_I2C3CLKSOURCE_D2PCLK1: D2PCLK1 selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_PLL3: PLL3 selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_HSI: HSI selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_CSI: CSI selected as I2C3 clock

__HAL_RCC_GET_I2C3_SOURCE

Description:

- macro to get the I2C3 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I2C3CLKSOURCE_D2PCLK1: D2PCLK1 selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_PLL3: PLL3 selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_HSI: HSI selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_CSI: CSI selected as I2C3 clock

__HAL_RCC_I2C4_CONFIG

Description:

- macro to configure the I2C4 clock (I2C4CLK).

Parameters:

- __I2C4CLKSource__: specifies the I2C4 clock source. This parameter can be one of the following values:
  - RCC_I2C4CLKSOURCE_D3PCLK1: D3PCLK1 selected as I2C4 clock
  - RCC_I2C4CLKSOURCE_PLL3: PLL3 selected as I2C4 clock
  - RCC_I2C4CLKSOURCE_HSI: HSI selected as I2C4 clock
  - RCC_I2C4CLKSOURCE_CSI: CSI selected as I2C4 clock

__HAL_RCC_GET_I2C4_SOURCE

Description:

- macro to get the I2C4 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I2C4CLKSOURCE_D3PCLK1: D3PCLK1 selected as I2C4 clock
  - RCC_I2C4CLKSOURCE_PLL3: PLL3 selected as I2C4 clock
  - RCC_I2C4CLKSOURCE_HSI: HSI selected as I2C4 clock
  - RCC_I2C4CLKSOURCE_CSI: CSI selected as I2C4 clock

__HAL_RCC_USART16_CONFIG

Description:

- macro to configure the USART1/6/9* /10* clock (USART16CLK).

Parameters:

- __USART16910CLKSource__: specifies the USART1/6/9* /10* clock source. This parameter can be one of the following values:
  - RCC_USART16CLKSOURCE_D2PCLK2: APB2 Clock selected as USART1/6/9* /10* clock
  - RCC_USART16CLKSOURCE_PLL2: PLL2_Q Clock selected as USART1/6/9* /10* clock
  - RCC_USART16CLKSOURCE_PLL3: PLL3_Q Clock selected as USART1/6/9* /10* clock
  - RCC_USART16CLKSOURCE_HSI: HSI selected as USART1/6/9* /10* clock
  - RCC_USART16CLKSOURCE_CSI: CSI Clock selected as USART1/6/9* /10* clock
  - RCC_USART16CLKSOURCE_LSE: LSE selected as USART1/6/9* /10* clock

__HAL_RCC_GET_USART16_SOURCE

Description:

- macro to get the USART1/6/9* /10* clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USART16CLKSOURCE_D2PCLK2: APB2 Clock selected as USART1/6/9* /10* clock
  - RCC_USART16CLKSOURCE_PLL2: PLL2_Q Clock selected as USART1/6/9* /10* clock
  - RCC_USART16CLKSOURCE_PLL3: PLL3_Q Clock selected as USART1/6/9* /10* clock
  - RCC_USART16CLKSOURCE_HSI: HSI selected as USART1/6/9* /10* clock
  - RCC_USART16CLKSOURCE_CSI: CSI Clock selected as USART1/6/9* /10* clock
  - RCC_USART16CLKSOURCE_LSE: LSE selected as USART1/6/9* /10* clock

__HAL_RCC_USART234578_CONFIG

Description:

- macro to configure the USART234578 clock (USART234578CLK).

Parameters:

- __USART234578CLKSource__: specifies the USART2/3/4/5/7/8 clock source. This parameter can be one of the following values:
  - RCC_USART234578CLKSOURCE_D2PCLK1: APB1 Clock selected as USART2/3/4/5/7/8 clock
  - RCC_USART234578CLKSOURCE_PLL2: PLL2_Q Clock selected as USART2/3/4/5/7/8 clock
  - RCC_USART234578CLKSOURCE_PLL3: PLL3_Q Clock selected as USART2/3/4/5/7/8 clock
  - RCC_USART234578CLKSOURCE_HSI: HSI selected as USART2/3/4/5/7/8 clock
  - RCC_USART234578CLKSOURCE_CSI: CSI Clock selected as USART2/3/4/5/7/8 clock
  - RCC_USART234578CLKSOURCE_LSE: LSE selected as USART2/3/4/5/7/8 clock

__HAL_RCC_GET_USART234578_SOURCE

Description:

- macro to get the USART2/3/4/5/7/8 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USART234578CLKSOURCE_D2PCLK1: APB1 Clock selected as USART2/3/4/5/7/8 clock
  - RCC_USART234578CLKSOURCE_PLL2: PLL2_Q Clock selected as USART2/3/4/5/7/8 clock
  - RCC_USART234578CLKSOURCE_PLL3: PLL3_Q Clock selected as USART2/3/4/5/7/8 clock
  - RCC_USART234578CLKSOURCE_HSI: HSI selected as USART2/3/4/5/7/8 clock
  - RCC_USART234578CLKSOURCE_CSI: CSI Clock selected as USART2/3/4/5/7/8 clock
  - RCC_USART234578CLKSOURCE_LSE: LSE selected as USART2/3/4/5/7/8 clock

__HAL_RCC_USART1_CONFIG

Description:

- macro to configure the USART1 clock (USART1CLK).

Parameters:

- __USART1CLKSource__: specifies the USART1 clock source. This parameter can be one of the following values:
  - RCC_USART1CLKSOURCE_D2PCLK2: APB2 Clock selected as USART1 clock
  - RCC_USART1CLKSOURCE_PLL2: PLL2_Q Clock selected as USART1 clock
  - RCC_USART1CLKSOURCE_PLL3: PLL3_Q Clock selected as USART1 clock
  - RCC_USART1CLKSOURCE_HSI: HSI selected as USART1 clock
  - RCC_USART1CLKSOURCE_CSI: CSI Clock selected as USART1 clock
  - RCC_USART1CLKSOURCE_LSE: LSE selected as USART1 clock

__HAL_RCC_GET_USART1_SOURCE

Description:

- macro to get the USART1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USART1CLKSOURCE_D2PCLK2: APB2 Clock selected as USART1 clock
  - RCC_USART1CLKSOURCE_PLL2: PLL2_Q Clock selected as USART1 clock
  - RCC_USART1CLKSOURCE_PLL3: PLL3_Q Clock selected as USART1 clock
  - RCC_USART1CLKSOURCE_HSI: HSI selected as USART1 clock
  - RCC_USART1CLKSOURCE_CSI: CSI Clock selected as USART1 clock
  - RCC_USART1CLKSOURCE_LSE: LSE selected as USART1 clock

__HAL_RCC_USART2_CONFIG

Description:

- macro to configure the USART2 clock (USART2CLK).

Parameters:

- __USART2CLKSource__: specifies the USART2 clock source. This parameter can be one of the following values:
  - RCC_USART2CLKSOURCE_D2PCLK1: APB1 Clock selected as USART2 clock
  - RCC_USART2CLKSOURCE_PLL2: PLL2_Q Clock selected as USART2 clock
  - RCC_USART2CLKSOURCE_PLL3: PLL3_Q Clock selected as USART2 clock
  - RCC_USART2CLKSOURCE_HSI: HSI selected as USART2 clock
  - RCC_USART2CLKSOURCE_CSI: CSI Clock selected as USART2 clock
  - RCC_USART2CLKSOURCE_LSE: LSE selected as USART2 clock

__HAL_RCC_GET_USART2_SOURCE

Description:

- macro to get the USART2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USART2CLKSOURCE_D2PCLK1: APB1 Clock selected as USART2 clock
  - RCC_USART2CLKSOURCE_PLL2: PLL2_Q Clock selected as USART2 clock
  - RCC_USART2CLKSOURCE_PLL3: PLL3_Q Clock selected as USART2 clock
  - RCC_USART2CLKSOURCE_HSI: HSI selected as USART2 clock
  - RCC_USART2CLKSOURCE_CSI: CSI Clock selected as USART2 clock
  - RCC_USART2CLKSOURCE_LSE: LSE selected as USART2 clock

__HAL_RCC_USART3_CONFIG

Description:

- macro to configure the USART3 clock (USART3CLK).

Parameters:

- __USART3CLKSource__: specifies the USART3 clock source. This parameter can be one of the following values:
  - RCC_USART3CLKSOURCE_D2PCLK1: APB1 Clock selected as USART3 clock
  - RCC_USART3CLKSOURCE_PLL2: PLL2_Q Clock selected as USART3 clock
  - RCC_USART3CLKSOURCE_PLL3: PLL3_Q Clock selected as USART3 clock
  - RCC_USART3CLKSOURCE_HSI: HSI selected as USART3 clock
  - RCC_USART3CLKSOURCE_CSI: CSI Clock selected as USART3 clock
  - RCC_USART3CLKSOURCE_LSE: LSE selected as USART3 clock

__HAL_RCC_GET_USART3_SOURCE

Description:

- macro to get the USART3 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USART2CLKSOURCE_D2PCLK1: APB1 Clock selected as USART3 clock
  - RCC_USART3CLKSOURCE_PLL2: PLL2_Q Clock selected as USART3 clock
  - RCC_USART3CLKSOURCE_PLL3: PLL3_Q Clock selected as USART3 clock
  - RCC_USART3CLKSOURCE_HSI: HSI selected as USART3 clock
  - RCC_USART3CLKSOURCE_CSI: CSI Clock selected as USART3 clock
  - RCC_USART3CLKSOURCE_LSE: LSE selected as USART3 clock

__HAL_RCC_UART4_CONFIG

Description:

- macro to configure the UART4 clock (UART4CLK).

Parameters:

- __UART4CLKSource__: specifies the UART4 clock source. This parameter can be one of the following values:
  - RCC_UART4CLKSOURCE_D2PCLK1: APB1 Clock selected as UART4 clock
  - RCC_UART4CLKSOURCE_PLL2: PLL2_Q Clock selected as UART4 clock
  - RCC_UART4CLKSOURCE_PLL3: PLL3_Q Clock selected as UART4 clock
  - RCC_UART4CLKSOURCE_HSI: HSI selected as UART4 clock
  - RCC_UART4CLKSOURCE_CSI: CSI Clock selected as UART4 clock
  - RCC_UART4CLKSOURCE_LSE: LSE selected as UART4 clock

__HAL_RCC_GET_UART4_SOURCE

Description:

- macro to get the UART4 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_UART4CLKSOURCE_D2PCLK1: APB1 Clock selected as UART4 clock
  - RCC_UART4CLKSOURCE_PLL2: PLL2_Q Clock selected as UART4 clock
  - RCC_UART4CLKSOURCE_PLL3: PLL3_Q Clock selected as UART4 clock
  - RCC_UART4CLKSOURCE_HSI: HSI selected as UART4 clock
  - RCC_UART4CLKSOURCE_CSI: CSI Clock selected as UART4 clock
  - RCC_UART4CLKSOURCE_LSE: LSE selected as UART4 clock

__HAL_RCC_UART5_CONFIG

Description:

- macro to configure the UART5 clock (UART5CLK).

Parameters:

- __UART5CLKSource__: specifies the UART5 clock source. This parameter can be one of the following values:
  - RCC_UART5CLKSOURCE_D2PCLK1: APB1 Clock selected as UART5 clock
  - RCC_UART5CLKSOURCE_PLL2: PLL2_Q Clock selected as UART5 clock
  - RCC_UART5CLKSOURCE_PLL3: PLL3_Q Clock selected as UART5 clock
  - RCC_UART5CLKSOURCE_HSI: HSI selected as UART5 clock
  - RCC_UART5CLKSOURCE_CSI: CSI Clock selected as UART5 clock
  - RCC_UART5CLKSOURCE_LSE: LSE selected as UART5 clock

__HAL_RCC_GET_UART5_SOURCE

Description:

- macro to get the UART5 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_UART5CLKSOURCE_D2PCLK1: APB1 Clock selected as UART5 clock
  - RCC_UART5CLKSOURCE_PLL2: PLL2_Q Clock selected as UART5 clock
  - RCC_UART5CLKSOURCE_PLL3: PLL3_Q Clock selected as UART5 clock
  - RCC_UART5CLKSOURCE_HSI: HSI selected as UART5 clock
  - RCC_UART5CLKSOURCE_CSI: CSI Clock selected as UART5 clock
  - RCC_UART5CLKSOURCE_LSE: LSE selected as UART5 clock

__HAL_RCC_USART6_CONFIG

Description:

- macro to configure the USART6 clock (USART6CLK).

Parameters:

- __USART6CLKSource__: specifies the USART6 clock source. This parameter can be one of the following values:
  - RCC_USART6CLKSOURCE_D2PCLK2: APB2 Clock selected as USART6 clock
  - RCC_USART6CLKSOURCE_PLL2: PLL2_Q Clock selected as USART6 clock
  - RCC_USART6CLKSOURCE_PLL3: PLL3_Q Clock selected as USART6 clock
  - RCC_USART6CLKSOURCE_HSI: HSI selected as USART6 clock
  - RCC_USART6CLKSOURCE_CSI: CSI Clock selected as USART6 clock
  - RCC_USART6CLKSOURCE_LSE: LSE selected as USART6 clock

__HAL_RCC_GET_USART6_SOURCE

Description:

- macro to get the USART6 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USART6CLKSOURCE_D2PCLK2: APB2 Clock selected as USART6 clock
  - RCC_USART6CLKSOURCE_PLL2: PLL2_Q Clock selected as USART6 clock
  - RCC_USART6CLKSOURCE_PLL3: PLL3_Q Clock selected as USART6 clock
  - RCC_USART6CLKSOURCE_HSI: HSI selected as USART6 clock
  - RCC_USART6CLKSOURCE_CSI: CSI Clock selected as USART6 clock
  - RCC_USART6CLKSOURCE_LSE: LSE selected as USART6 clock

__HAL_RCC_UART7_CONFIG

Description:

- macro to configure the UART5 clock (UART7CLK).

Parameters:

- __UART7CLKSource__: specifies the UART7 clock source. This parameter can be one of the following values:
  - RCC_UART7CLKSOURCE_D2PCLK1: APB1 Clock selected as UART7 clock
  - RCC_UART7CLKSOURCE_PLL2: PLL2_Q Clock selected as UART7 clock
  - RCC_UART7CLKSOURCE_PLL3: PLL3_Q Clock selected as UART7 clock
  - RCC_UART7CLKSOURCE_HSI: HSI selected as UART7 clock
  - RCC_UART7CLKSOURCE_CSI: CSI Clock selected as UART7 clock
  - RCC_UART7CLKSOURCE_LSE: LSE selected as UART7 clock

__HAL_RCC_GET_UART7_SOURCE

Description:

- macro to get the UART7 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_UART7CLKSOURCE_D2PCLK1: APB1 Clock selected as UART7 clock
  - RCC_UART7CLKSOURCE_PLL2: PLL2_Q Clock selected as UART7 clock
  - RCC_UART7CLKSOURCE_PLL3: PLL3_Q Clock selected as UART7 clock
  - RCC_UART7CLKSOURCE_HSI: HSI selected as UART7 clock
  - RCC_UART7CLKSOURCE_CSI: CSI Clock selected as UART7 clock
  - RCC_UART7CLKSOURCE_LSE: LSE selected as UART7 clock

__HAL_RCC_UART8_CONFIG

Description:

- macro to configure the UART8 clock (UART8CLK).

Parameters:

- __UART8CLKSource__: specifies the UART8 clock source. This parameter can be one of the following values:
  - RCC_UART8CLKSOURCE_D2PCLK1: APB1 Clock selected as UART8 clock
  - RCC_UART8CLKSOURCE_PLL2: PLL2_Q Clock selected as UART8 clock
  - RCC_UART8CLKSOURCE_PLL3: PLL3_Q Clock selected as UART8 clock
  - RCC_UART8CLKSOURCE_HSI: HSI selected as UART8 clock
  - RCC_UART8CLKSOURCE_CSI: CSI Clock selected as UART8 clock
  - RCC_UART8CLKSOURCE_LSE: LSE selected as UART8 clock

__HAL_RCC_GET_UART8_SOURCE

Description:

- macro to get the UART8 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_UART8CLKSOURCE_D2PCLK1: APB1 Clock selected as UART8 clock
  - RCC_UART8CLKSOURCE_PLL2: PLL2_Q Clock selected as UART8 clock
  - RCC_UART8CLKSOURCE_PLL3: PLL3_Q Clock selected as UART8 clock
  - RCC_UART8CLKSOURCE_HSI: HSI selected as UART8 clock
  - RCC_UART8CLKSOURCE_CSI: CSI Clock selected as UART8 clock
  - RCC_UART8CLKSOURCE_LSE: LSE selected as UART8 clock

__HAL_RCC_LPUART1_CONFIG

Description:

- macro to configure the LPUART1 clock (LPUART1CLK).

Parameters:

- __LPUART1CLKSource__: specifies the LPUART1 clock source. This parameter can be one of the following values:
  - RCC_LPUART1CLKSOURCE_D3PCLK1: APB4 Clock selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_PLL2: PLL2_Q Clock selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_PLL3: PLL3_Q Clock selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_HSI: HSI selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_CSI: CSI Clock selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_LSE: LSE selected as LPUART1 clock

__HAL_RCC_GET_LPUART1_SOURCE

Description:

- macro to get the LPUART1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPUART1CLKSOURCE_D3PCLK1: APB4 Clock selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_PLL2: PLL2_Q Clock selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_PLL3: PLL3_Q Clock selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_HSI: HSI selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_CSI: CSI Clock selected as LPUART1 clock
  - RCC_LPUART1CLKSOURCE_LSE: LSE selected as LPUART1 clock

__HAL_RCC_LPTIM1_CONFIG

Description:

- macro to configure the LPTIM1 clock source.

Parameters:

- __LPTIM1CLKSource__: specifies the LPTIM1 clock source. This parameter can be one of the following values:
  - RCC_LPTIM1CLKSOURCE_D2PCLK1: APB1 Clock selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_PLL2: PLL2_P Clock selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_PLL3: PLL3_R Clock selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_LSE: LSE selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_LSI: LSI Clock selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_CLKP: CLKP selected as LPTIM1 clock

__HAL_RCC_GET_LPTIM1_SOURCE

Description:

- macro to get the LPTIM1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPTIM1CLKSOURCE_D2PCLK1: APB1 Clock selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_PLL2: PLL2_P Clock selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_PLL3: PLL3_R Clock selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_LSE: LSE selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_LSI: LSI Clock selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_CLKP: CLKP selected as LPTIM1 clock

__HAL_RCC_LPTIM2_CONFIG

Description:

- macro to configure the LPTIM2 clock source.

Parameters:

- __LPTIM2CLKSource__: specifies the LPTIM2 clock source. This parameter can be one of the following values:
  - RCC_LPTIM2CLKSOURCE_D3PCLK1: APB4 Clock selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_PLL2: PLL2_P Clock selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_PLL3: PLL3_R Clock selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_LSE: LSE selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_LSI: LSI Clock selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_CLKP: CLKP selected as LPTIM2 clock

__HAL_RCC_GET_LPTIM2_SOURCE

Description:

- macro to get the LPTIM2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPTIM2CLKSOURCE_D3PCLK1: APB4 Clock selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_PLL2: PLL2_P Clock selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_PLL3: PLL3_R Clock selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_LSE: LSE selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_LSI: LSI Clock selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_CLKP: CLKP selected as LPTIM2 clock

__HAL_RCC_LPTIM345_CONFIG

Description:

- macro to configure the LPTIM3/4/5 clock source.

Parameters:

- __LPTIM345CLKSource__: specifies the LPTIM3/4/5 clock source.
  - RCC_LPTIM345CLKSOURCE_D3PCLK1: APB4 Clock selected as LPTIM3/4/5 clock
  - RCC_LPTIM345CLKSOURCE_PLL2: PLL2_P Clock selected as LPTIM3/4/5 clock
  - RCC_LPTIM345CLKSOURCE_PLL3: PLL3_R Clock selected as LPTIM3/4/5 clock
  - RCC_LPTIM345CLKSOURCE_LSE: LSE selected as LPTIM3/4/5 clock
  - RCC_LPTIM345CLKSOURCE_LSI: LSI Clock selected as LPTIM3/4/5 clock
  - RCC_LPTIM345CLKSOURCE_CLKP: CLKP selected as LPTIM3/4/5 clock

__HAL_RCC_GET_LPTIM345_SOURCE

Description:

- macro to get the LPTIM3/4/5 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPTIM345CLKSOURCE_D3PCLK1: APB4 Clock selected as LPTIM3/4/5 clock
  - RCC_LPTIM345CLKSOURCE_PLL2: PLL2_P Clock selected as LPTIM3/4/5 clock
  - RCC_LPTIM345CLKSOURCE_PLL3: PLL3_R Clock selected as LPTIM3/4/5 clock
  - RCC_LPTIM345CLKSOURCE_LSE: LSE selected as LPTIM3/4/5 clock
  - RCC_LPTIM345CLKSOURCE_LSI: LSI Clock selected as LPTIM3/4/5 clock
  - RCC_LPTIM345CLKSOURCE_CLKP: CLKP selected as LPTIM3/4/5 clock

__HAL_RCC_LPTIM3_CONFIG

Description:

- macro to configure the LPTIM3 clock source.

Parameters:

- __LPTIM3CLKSource__: specifies the LPTIM3 clock source.
  - RCC_LPTIM3CLKSOURCE_D3PCLK1: APB4 Clock selected as LPTIM3 clock
  - RCC_LPTIM3CLKSOURCE_PLL2: PLL2_P Clock selected as LPTIM3 clock
  - RCC_LPTIM3CLKSOURCE_PLL3: PLL3_R Clock selected as LPTIM3 clock
  - RCC_LPTIM3CLKSOURCE_LSE: LSE selected as LPTIM3 clock
  - RCC_LPTIM3CLKSOURCE_LSI: LSI Clock selected as LPTIM3 clock
  - RCC_LPTIM3CLKSOURCE_CLKP: CLKP selected as LPTIM3 clock

__HAL_RCC_GET_LPTIM3_SOURCE

Description:

- macro to get the LPTIM3 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPTIM3CLKSOURCE_D3PCLK1: APB4 Clock selected as LPTIM3 clock
  - RCC_LPTIM3CLKSOURCE_PLL2: PLL2_P Clock selected as LPTIM3 clock
  - RCC_LPTIM3CLKSOURCE_PLL3: PLL3_R Clock selected as LPTIM3 clock
  - RCC_LPTIM3CLKSOURCE_LSE: LSE selected as LPTIM3 clock
  - RCC_LPTIM3CLKSOURCE_LSI: LSI Clock selected as LPTIM3 clock
  - RCC_LPTIM3CLKSOURCE_CLKP: CLKP selected as LPTIM3 clock

__HAL_RCC_LPTIM4_CONFIG

Description:

- macro to configure the LPTIM4 clock source.

Parameters:

- __LPTIM4CLKSource__: specifies the LPTIM4 clock source.
  - RCC_LPTIM4CLKSOURCE_D3PCLK1: APB4 Clock selected as LPTIM4 clock
  - RCC_LPTIM4CLKSOURCE_PLL2: PLL2_P Clock selected as LPTIM4 clock
  - RCC_LPTIM4CLKSOURCE_PLL3: PLL3_R Clock selected as LPTIM4 clock
  - RCC_LPTIM4CLKSOURCE_LSE: LSE selected as LPTIM4 clock
  - RCC_LPTIM4CLKSOURCE_LSI: LSI Clock selected as LPTIM4 clock
  - RCC_LPTIM4CLKSOURCE_CLKP: CLKP selected as LPTIM4 clock

__HAL_RCC_GET_LPTIM4_SOURCE

Description:

- macro to get the LPTIM4 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPTIM4CLKSOURCE_D3PCLK1: APB4 Clock selected as LPTIM4 clock
  - RCC_LPTIM4CLKSOURCE_PLL2: PLL2_P Clock selected as LPTIM4 clock
  - RCC_LPTIM4CLKSOURCE_PLL3: PLL3_R Clock selected as LPTIM4 clock
  - RCC_LPTIM4CLKSOURCE_LSE: LSE selected as LPTIM4 clock
  - RCC_LPTIM4CLKSOURCE_LSI: LSI Clock selected as LPTIM4 clock
  - RCC_LPTIM4CLKSOURCE_CLKP: CLKP selected as LPTIM4 clock

__HAL_RCC_LPTIM5_CONFIG

Description:

- macro to configure the LPTIM5 clock source.

Parameters:

- __LPTIM5CLKSource__: specifies the LPTIM5 clock source.
  - RCC_LPTIM5CLKSOURCE_D3PCLK1: APB4 Clock selected as LPTIM5 clock
  - RCC_LPTIM5CLKSOURCE_PLL2: PLL2_P Clock selected as LPTIM5 clock
  - RCC_LPTIM5CLKSOURCE_PLL3: PLL3_R Clock selected as LPTIM5 clock
  - RCC_LPTIM5CLKSOURCE_LSE: LSE selected as LPTIM5 clock
  - RCC_LPTIM5CLKSOURCE_LSI: LSI Clock selected as LPTIM5 clock
  - RCC_LPTIM5CLKSOURCE_CLKP: CLKP selected as LPTIM5 clock

__HAL_RCC_GET_LPTIM5_SOURCE

Description:

- macro to get the LPTIM5 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPTIM5CLKSOURCE_D3PCLK1: APB4 Clock selected as LPTIM5 clock
  - RCC_LPTIM5CLKSOURCE_PLL2: PLL2_P Clock selected as LPTIM5 clock
  - RCC_LPTIM5CLKSOURCE_PLL3: PLL3_R Clock selected as LPTIM5 clock
  - RCC_LPTIM5CLKSOURCE_LSE: LSE selected as LPTIM5 clock
  - RCC_LPTIM5CLKSOURCE_LSI: LSI Clock selected as LPTIM5 clock
  - RCC_LPTIM5CLKSOURCE_CLKP: CLKP selected as LPTIM5 clock

__HAL_RCC_QSPI_CONFIG

Description:

- macro to configure the QSPI clock source.

Parameters:

- __QSPICLKSource__: specifies the QSPI clock source.
  - RCC_RCC_QSPICLKSOURCE_D1HCLK: Domain1 HCLK Clock selected as QSPI clock
  - RCC_RCC_QSPICLKSOURCE_PLL : PLL1_Q Clock selected as QSPI clock
  - RCC_RCC_QSPICLKSOURCE_PLL2 : PLL2_R Clock selected as QSPI clock
  - RCC_RCC_QSPICLKSOURCE_CLKP CLKP selected as QSPI clock

__HAL_RCC_GET_QSPI_SOURCE

Description:

- macro to get the QSPI clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_RCC_QSPICLKSOURCE_D1HCLK: Domain1 HCLK Clock selected as QSPI clock
  - RCC_RCC_QSPICLKSOURCE_PLL : PLL1_Q Clock selected as QSPI clock
  - RCC_RCC_QSPICLKSOURCE_PLL2 : PLL2_R Clock selected as QSPI clock
  - RCC_RCC_QSPICLKSOURCE_CLKP CLKP selected as QSPI clock

__HAL_RCC_FMC_CONFIG

Description:

- macro to configure the FMC clock source.

Parameters:

- __FMCCLKSource__: specifies the FMC clock source.
  - RCC_RCC_FMCCLKSOURCE_D1HCLK: Domain1 HCLK Clock selected as FMC clock
  - RCC_RCC_FMCCLKSOURCE_PLL : PLL1_Q Clock selected as FMC clock
  - RCC_RCC_FMCCLKSOURCE_PLL2 : PLL2_R Clock selected as FMC clock
  - RCC_RCC_FMCCLKSOURCE_CLKP CLKP selected as FMC clock

__HAL_RCC_GET_FMC_SOURCE

Description:

- macro to get the FMC clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_RCC_FMCCLKSOURCE_D1HCLK: Domain1 HCLK Clock selected as FMC clock
  - RCC_RCC_FMCCLKSOURCE_PLL : PLL1_Q Clock selected as FMC clock
  - RCC_RCC_FMCCLKSOURCE_PLL2 : PLL2_R Clock selected as FMC clock
  - RCC_RCC_FMCCLKSOURCE_CLKP CLKP selected as FMC clock

__HAL_RCC_USB_CONFIG

Description:

- Macro to configure the USB clock (USBCLK).

Parameters:

- __USBCLKSource__: specifies the USB clock source. This parameter can be one of the following values:
  - RCC_USBCLKSOURCE_PLL: PLL1Q selected as USB clock
  - RCC_USBCLKSOURCE_PLL3: PLL3Q Clock selected as USB clock
  - RCC_USBCLKSOURCE_HSI48: RC48 MHZ Clock selected as USB clock

__HAL_RCC_GET_USB_SOURCE

Description:

- Macro to get the USB clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USBCLKSOURCE_PLL: PLL1Q selected as USB clock
  - RCC_USBCLKSOURCE_PLL3: PLL3Q Clock selected as USB clock
  - RCC_USBCLKSOURCE_HSI48: RC48 MHZ Clock selected as USB clock

__HAL_RCC_ADC_CONFIG

Description:

- Macro to configure the ADC clock.

Parameters:

- __ADCCLKSource__: specifies the ADC digital interface clock source. This parameter can be one of the following values:
  - RCC_ADCCLKSOURCE_PLL2: PLL2_P Clock selected as ADC clock
  - RCC_ADCCLKSOURCE_PLL3: PLL3_R Clock selected as ADC clock
  - RCC_ADCCLKSOURCE_CLKP: CLKP Clock selected as ADC clock

__HAL_RCC_GET_ADC_SOURCE

Description:

- Macro to get the ADC clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_ADCCLKSOURCE_PLL2: PLL2_P Clock selected as ADC clock
  - RCC_ADCCLKSOURCE_PLL3: PLL3_R Clock selected as ADC clock
  - RCC_ADCCLKSOURCE_CLKP: CLKP Clock selected as ADC clock

__HAL_RCC_SWPMI1_CONFIG

Description:

- Macro to configure the SWPMI1 clock.

Parameters:

- __SWPMI1CLKSource__: specifies the SWPMI1 clock source. This parameter can be one of the following values:
  - RCC_SWPMI1CLKSOURCE_D2PCLK1: D2PCLK1 Clock selected as SWPMI1 clock
  - RCC_SWPMI1CLKSOURCE_HSI: HSI Clock selected as SWPMI1 clock

__HAL_RCC_GET_SWPMI1_SOURCE

Description:

- Macro to get the SWPMI1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SWPMI1CLKSOURCE_D2PCLK1: D2PCLK1 Clock selected as SWPMI1 clock
  - RCC_SWPMI1CLKSOURCE_HSI: HSI Clock selected as SWPMI1 clock

__HAL_RCC_DFSDM1_CONFIG

Description:

- Macro to configure the DFSDM1 clock.

Parameters:

- __DFSDM1CLKSource__: specifies the DFSDM1 clock source. This parameter can be one of the following values:
  - RCC_DFSDM1CLKSOURCE_D2PCLK: D2PCLK Clock selected as DFSDM1 clock
  - RCC_DFSDM1CLKSOURCE_SYS: System Clock selected as DFSDM1 clock

__HAL_RCC_GET_DFSDM1_SOURCE

Description:

- Macro to get the DFSDM1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_DFSDM1CLKSOURCE_D2PCLK: D2PCLK Clock selected as DFSDM1 clock
  - RCC_DFSDM1CLKSOURCE_SYS: System Clock selected as DFSDM1 clock

__HAL_RCC_CEC_CONFIG

Description:

- macro to configure the CEC clock (CECCLK).

Parameters:

- __CECCLKSource__: specifies the CEC clock source. This parameter can be one of the following values:
  - RCC_CECCLKSOURCE_LSE: LSE selected as CEC clock
  - RCC_CECCLKSOURCE_LSI: LSI selected as CEC clock
  - RCC_CECCLKSOURCE_CSI: CSI Clock selected as CEC clock

__HAL_RCC_GET_CEC_SOURCE

Description:

- macro to get the CEC clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_CECCLKSOURCE_LSE: LSE selected as CEC clock
  - RCC_CECCLKSOURCE_LSI: LSI selected as CEC clock
  - RCC_CECCLKSOURCE_CSI: CSI Clock selected as CEC clock

__HAL_RCC_CLKP_CONFIG

Description:

- Macro to configure the CLKP : Oscillator clock for peripheral.

Parameters:

- __CLKPSource__: specifies Oscillator clock for peripheral This parameter can be one of the following values:
  - RCC_CLKPSOURCE_HSI: HSI selected Oscillator clock for peripheral
  - RCC_CLKPSOURCE_CSI: CSI selected Oscillator clock for peripheral
  - RCC_CLKPSOURCE_HSE: HSE selected Oscillator clock for peripheral

__HAL_RCC_GET_CLKP_SOURCE

Description:

- Macro to get the Oscillator clock for peripheral source.

Return value:

- The: clock source can be one of the following values:
  - RCC_CLKPSOURCE_HSI: HSI selected Oscillator clock for peripheral
  - RCC_CLKPSOURCE_CSI: CSI selected Oscillator clock for peripheral
  - RCC_CLKPSOURCE_HSE: HSE selected Oscillator clock for peripheral

__HAL_RCC_FDCAN_CONFIG

Description:

- Macro to configure the FDCAN clock.

Parameters:

- __FDCANCLKSource__: specifies clock source for FDCAN This parameter can be one of the following values:
  - RCC_FDCANCLKSOURCE_HSE: HSE selected as FDCAN clock
  - RCC_FDCANCLKSOURCE_PLL: PLL selected as FDCAN clock
  - RCC_FDCANCLKSOURCE_PLL2: PLL2 selected as FDCAN clock

__HAL_RCC_GET_FDCAN_SOURCE

Description:

- Macro to get the FDCAN clock.

Return value:

- The: clock source can be one of the following values:
  - RCC_FDCANCLKSOURCE_HSE: HSE selected as FDCAN clock
  - RCC_FDCANCLKSOURCE_PLL: PLL selected as FDCAN clock
  - RCC_FDCANCLKSOURCE_PLL2: PLL2 selected as FDCAN clock

__HAL_RCC_SPI123_CONFIG

Description:

- Macro to Configure the SPI1/2/3 clock source.

Parameters:

- __RCC_SPI123CLKSource__: defines the SPI1/2/3 clock source. This clock is derived from system PLL, PLL2, PLL3, OSC or external clock (through a dedicated PIN) This parameter can be one of the following values:
  - RCC_SPI123CLKSOURCE_PLL: SPI1/2/3 clock = PLL
  - RCC_SPI123CLKSOURCE_PLL2: SPI1/2/3 clock = PLL2
  - RCC_SPI123CLKSOURCE_PLL3: SPI1/2/3 clock = PLL3
  - RCC_SPI123CLKSOURCE_CLKP: SPI1/2/3 clock = CLKP
  - RCC_SPI123CLKSOURCE_PIN: SPI1/2/3 clock = External Clock

Return value:

- None

__HAL_RCC_GET_SPI123_SOURCE

Description:

- Macro to get the SPI1/2/3 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SPI123CLKSOURCE_PLL: SPI1/2/3 clock = PLL
  - RCC_SPI123CLKSOURCE_PLL2: SPI1/2/3 clock = PLL2
  - RCC_SPI123CLKSOURCE_PLL3: SPI1/2/3 clock = PLL3
  - RCC_SPI123CLKSOURCE_CLKP: SPI1/2/3 clock = CLKP
  - RCC_SPI123CLKSOURCE_PIN: SPI1/2/3 clock = External Clock

__HAL_RCC_SPI1_CONFIG

Description:

- Macro to Configure the SPI1 clock source.

Parameters:

- __RCC_SPI1CLKSource__: defines the SPI1 clock source. This clock is derived from system PLL, PLL2, PLL3, OSC or external clock (through a dedicated PIN) This parameter can be one of the following values:
  - RCC_SPI1CLKSOURCE_PLL: SPI1 clock = PLL
  - RCC_SPI1CLKSOURCE_PLL2: SPI1 clock = PLL2
  - RCC_SPI1CLKSOURCE_PLL3: SPI1 clock = PLL3
  - RCC_SPI1CLKSOURCE_CLKP: SPI1 clock = CLKP
  - RCC_SPI1CLKSOURCE_PIN: SPI1 clock = External Clock

Return value:

- None

__HAL_RCC_GET_SPI1_SOURCE

Description:

- Macro to get the SPI1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SPI1CLKSOURCE_PLL: SPI1 clock = PLL
  - RCC_SPI1CLKSOURCE_PLL2: SPI1 clock = PLL2
  - RCC_SPI1CLKSOURCE_PLL3: SPI1 clock = PLL3
  - RCC_SPI1CLKSOURCE_CLKP: SPI1 clock = CLKP
  - RCC_SPI1CLKSOURCE_PIN: SPI1 clock = External Clock

__HAL_RCC_SPI2_CONFIG

Description:

- Macro to Configure the SPI2 clock source.

Parameters:

- __RCC_SPI2CLKSource__: defines the SPI2 clock source. This clock is derived from system PLL, PLL2, PLL3, OSC or external clock (through a dedicated PIN) This parameter can be one of the following values:
  - RCC_SPI2CLKSOURCE_PLL: SPI2 clock = PLL
  - RCC_SPI2CLKSOURCE_PLL2: SPI2 clock = PLL2
  - RCC_SPI2CLKSOURCE_PLL3: SPI2 clock = PLL3
  - RCC_SPI2CLKSOURCE_CLKP: SPI2 clock = CLKP
  - RCC_SPI2CLKSOURCE_PIN: SPI2 clock = External Clock

Return value:

- None

__HAL_RCC_GET_SPI2_SOURCE

Description:

- Macro to get the SPI2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SPI2CLKSOURCE_PLL: SPI2 clock = PLL
  - RCC_SPI2CLKSOURCE_PLL2: SPI2 clock = PLL2
  - RCC_SPI2CLKSOURCE_PLL3: SPI2 clock = PLL3
  - RCC_SPI2CLKSOURCE_CLKP: SPI2 clock = CLKP
  - RCC_SPI2CLKSOURCE_PIN: SPI2 clock = External Clock

__HAL_RCC_SPI3_CONFIG

Description:

- Macro to Configure the SPI3 clock source.

Parameters:

- __RCC_SPI3CLKSource__: defines the SPI3 clock source. This clock is derived from system PLL, PLL2, PLL3, OSC or external clock (through a dedicated PIN) This parameter can be one of the following values:
  - RCC_SPI3CLKSOURCE_PLL: SPI3 clock = PLL
  - RCC_SPI3CLKSOURCE_PLL2: SPI3 clock = PLL2
  - RCC_SPI3CLKSOURCE_PLL3: SPI3 clock = PLL3
  - RCC_SPI3CLKSOURCE_CLKP: SPI3 clock = CLKP
  - RCC_SPI3CLKSOURCE_PIN: SPI3 clock = External Clock

Return value:

- None

__HAL_RCC_GET_SPI3_SOURCE

Description:

- Macro to get the SPI3 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SPI3CLKSOURCE_PLL: SPI3 clock = PLL
  - RCC_SPI3CLKSOURCE_PLL2: SPI3 clock = PLL2
  - RCC_SPI3CLKSOURCE_PLL3: SPI3 clock = PLL3
  - RCC_SPI3CLKSOURCE_CLKP: SPI3 clock = CLKP
  - RCC_SPI3CLKSOURCE_PIN: SPI3 clock = External Clock

__HAL_RCC_SPI45_CONFIG

Description:

- Macro to Configure the SPI4/5 clock source.

Parameters:

- __RCC_SPI45CLKSource__: defines the SPI4/5 clock source. This clock is derived from system PCLK, PLL2, PLL3, OSC This parameter can be one of the following values:
  - RCC_SPI45CLKSOURCE_D2PCLK2:SPI4/5 clock = D2PCLK2
  - RCC_SPI45CLKSOURCE_PLL2: SPI4/5 clock = PLL2
  - RCC_SPI45CLKSOURCE_PLL3: SPI4/5 clock = PLL3
  - RCC_SPI45CLKSOURCE_HSI: SPI4/5 clock = HSI
  - RCC_SPI45CLKSOURCE_CSI: SPI4/5 clock = CSI
  - RCC_SPI45CLKSOURCE_HSE: SPI4/5 clock = HSE

Return value:

- None

__HAL_RCC_GET_SPI45_SOURCE

Description:

- Macro to get the SPI4/5 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SPI45CLKSOURCE_D2PCLK2:SPI4/5 clock = D2PCLK2
  - RCC_SPI45CLKSOURCE_PLL2: SPI4/5 clock = PLL2
  - RCC_SPI45CLKSOURCE_PLL3: SPI4/5 clock = PLL3
  - RCC_SPI45CLKSOURCE_HSI: SPI4/5 clock = HSI
  - RCC_SPI45CLKSOURCE_CSI: SPI4/5 clock = CSI
  - RCC_SPI45CLKSOURCE_HSE: SPI4/5 clock = HSE

__HAL_RCC_SPI4_CONFIG

Description:

- Macro to Configure the SPI4 clock source.

Parameters:

- __RCC_SPI4CLKSource__: defines the SPI4 clock source. This clock is derived from system PCLK, PLL2, PLL3, OSC This parameter can be one of the following values:
  - RCC_SPI4CLKSOURCE_D2PCLK2:SPI4 clock = D2PCLK2
  - RCC_SPI4CLKSOURCE_PLL2: SPI4 clock = PLL2
  - RCC_SPI4CLKSOURCE_PLL3: SPI4 clock = PLL3
  - RCC_SPI4CLKSOURCE_HSI: SPI4 clock = HSI
  - RCC_SPI4CLKSOURCE_CSI: SPI4 clock = CSI
  - RCC_SPI4CLKSOURCE_HSE: SPI4 clock = HSE

Return value:

- None

__HAL_RCC_GET_SPI4_SOURCE

Description:

- Macro to get the SPI4 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SPI4CLKSOURCE_D2PCLK2:SPI4 clock = D2PCLK2
  - RCC_SPI4CLKSOURCE_PLL2: SPI4 clock = PLL2
  - RCC_SPI4CLKSOURCE_PLL3: SPI4 clock = PLL3
  - RCC_SPI4CLKSOURCE_HSI: SPI4 clock = HSI
  - RCC_SPI4CLKSOURCE_CSI: SPI4 clock = CSI
  - RCC_SPI4CLKSOURCE_HSE: SPI4 clock = HSE

__HAL_RCC_SPI5_CONFIG

Description:

- Macro to Configure the SPI5 clock source.

Parameters:

- __RCC_SPI5CLKSource__: defines the SPI5 clock source. This clock is derived from system PCLK, PLL2, PLL3, OSC This parameter can be one of the following values:
  - RCC_SPI5CLKSOURCE_D2PCLK2:SPI5 clock = D2PCLK2
  - RCC_SPI5CLKSOURCE_PLL2: SPI5 clock = PLL2
  - RCC_SPI5CLKSOURCE_PLL3: SPI5 clock = PLL3
  - RCC_SPI5CLKSOURCE_HSI: SPI5 clock = HSI
  - RCC_SPI5CLKSOURCE_CSI: SPI5 clock = CSI
  - RCC_SPI5CLKSOURCE_HSE: SPI5 clock = HSE

Return value:

- None

__HAL_RCC_GET_SPI5_SOURCE

Description:

- Macro to get the SPI5 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SPI5CLKSOURCE_D2PCLK2:SPI5 clock = D2PCLK2
  - RCC_SPI5CLKSOURCE_PLL2: SPI5 clock = PLL2
  - RCC_SPI5CLKSOURCE_PLL3: SPI5 clock = PLL3
  - RCC_SPI5CLKSOURCE_HSI: SPI5 clock = HSI
  - RCC_SPI5CLKSOURCE_CSI: SPI5 clock = CSI
  - RCC_SPI5CLKSOURCE_HSE: SPI5 clock = HSE

__HAL_RCC_SPI6_CONFIG

Description:

- Macro to Configure the SPI6 clock source.

Parameters:

- __RCC_SPI6CLKSource__: defines the SPI6 clock source. This clock is derived from system PCLK, PLL2, PLL3, OSC This parameter can be one of the following values:
  - RCC_SPI6CLKSOURCE_D3PCLK1:SPI6 clock = D2PCLK1
  - RCC_SPI6CLKSOURCE_PLL2: SPI6 clock = PLL2
  - RCC_SPI6CLKSOURCE_PLL3: SPI6 clock = PLL3
  - RCC_SPI6CLKSOURCE_HSI: SPI6 clock = HSI
  - RCC_SPI6CLKSOURCE_CSI: SPI6 clock = CSI
  - RCC_SPI6CLKSOURCE_HSE: SPI6 clock = HSE
  - RCC_SPI6CLKSOURCE_PIN: SPI6 clock = I2S_CKIN (*)

Return value:

- None: (*) : Available on stm32h7a3xx and stm32h7b3xx family lines.

__HAL_RCC_GET_SPI6_SOURCE

Description:

- Macro to get the SPI6 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SPI6CLKSOURCE_D3PCLK1:SPI6 clock = D2PCLK1
  - RCC_SPI6CLKSOURCE_PLL2: SPI6 clock = PLL2
  - RCC_SPI6CLKSOURCE_PLL3: SPI6 clock = PLL3
  - RCC_SPI6CLKSOURCE_HSI: SPI6 clock = HSI
  - RCC_SPI6CLKSOURCE_CSI: SPI6 clock = CSI
  - RCC_SPI6CLKSOURCE_HSE: SPI6 clock = HSE
  - RCC_SPI6CLKSOURCE_PIN: SPI6 clock = I2S_CKIN

__HAL_RCC_SDMMC_CONFIG

Description:

- Macro to configure the SDMMC clock.

Parameters:

- __SDMMCCLKSource__: specifies clock source for SDMMC This parameter can be one of the following values:
  - RCC_SDMMCCLKSOURCE_PLL: PLLQ selected as SDMMC clock
  - RCC_SDMMCCLKSOURCE_PLL2: PLL2R selected as SDMMC clock

__HAL_RCC_GET_SDMMC_SOURCE

__HAL_RCC_RNG_CONFIG

Description:

- macro to configure the RNG clock (RNGCLK).

Parameters:

- __RNGCLKSource__: specifies the RNG clock source. This parameter can be one of the following values:
  - RCC_RNGCLKSOURCE_HSI48: HSI48 selected as RNG clock
  - RCC_RNGCLKSOURCE_PLL: PLL1Q selected as RNG clock
  - RCC_RNGCLKSOURCE_LSE: LSE selected as RNG clock
  - RCC_RNGCLKSOURCE_LSI: LSI selected as RNG clock

__HAL_RCC_GET_RNG_SOURCE

Description:

- macro to get the RNG clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_RNGCLKSOURCE_HSI48: HSI48 selected as RNG clock
  - RCC_RNGCLKSOURCE_PLL: PLL1Q selected as RNG clock
  - RCC_RNGCLKSOURCE_LSE: LSE selected as RNG clock
  - RCC_RNGCLKSOURCE_LSI: LSI selected as RNG clock

__HAL_RCC_HRTIM1_CONFIG

Description:

- Macro to configure the HRTIM1 prescaler clock source.

Parameters:

- __HRTIM1CLKSource__: specifies the HRTIM1 prescaler clock source. This parameter can be one of the following values:
  - RCC_HRTIM1CLK_TIMCLK Timers clock selected as HRTIM1 prescaler clock
  - RCC_HRTIM1CLK_CPUCLK CPU Clock selected as HRTIM1 clock

__HAL_RCC_GET_HRTIM1_SOURCE

Description:

- Macro to get the HRTIM1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_HRTIM1CLK_TIMCLK Timers clock selected as HRTIM1 prescaler clock
  - RCC_HRTIM1CLK_CPUCLK CPU Clock selected as HRTIM1 clock

__HAL_RCC_TIMCLKPRESCALER

Description:

- Macro to configure the Timers clocks prescalers.

Parameters:

- __PRESC__: specifies the Timers clocks prescalers selection This parameter can be one of the following values:
  - RCC_TIMPRES_DESACTIVATED: The Timers kernels clocks prescaler is equal to rcc_hclk1 if D2PPREx is corresponding to division by 1 or 2, else it is equal to 2 x Frcc_pclkx_d2 (default after reset)
  - RCC_TIMPRES_ACTIVATED: The Timers kernels clocks prescaler is equal to rcc_hclk1 if D2PPREx is corresponding to division by 1, 2 or 4, else it is equal to 4 x Frcc_pclkx_d2

__HAL_RCC_LSECSS_EXTI_ENABLE_IT

Description:

- Enable the RCC LSE CSS Extended Interrupt Line.

Return value:

- None

__HAL_RCC_LSECSS_EXTI_DISABLE_IT

Description:

- Disable the RCC LSE CSS Extended Interrupt Line.

Return value:

- None

__HAL_RCC_LSECSS_EXTI_ENABLE_EVENT

Description:

- Enable the RCC LSE CSS Event Line.

Return value:

- None.

__HAL_RCC_LSECSS_EXTI_DISABLE_EVENT

Description:

- Disable the RCC LSE CSS Event Line.

Return value:

- None.

__HAL_RCC_C2_LSECSS_EXTI_ENABLE_IT

Description:

- Enable the RCC LSE CSS Extended Interrupt Line for CM4.

Return value:

- None

__HAL_RCC_C2_LSECSS_EXTI_DISABLE_IT

Description:

- Disable the RCC LSE CSS Extended Interrupt Line for CM4.

Return value:

- None

__HAL_RCC_C2_LSECSS_EXTI_ENABLE_EVENT

Description:

- Enable the RCC LSE CSS Event Line for CM4.

Return value:

- None.

__HAL_RCC_C2_LSECSS_EXTI_DISABLE_EVENT

Description:

- Disable the RCC LSE CSS Event Line for CM4.

Return value:

- None.

__HAL_RCC_LSECSS_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the RCC LSE CSS Extended Interrupt Falling Trigger.

Return value:

- None.

__HAL_RCC_LSECSS_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable the RCC LSE CSS Extended Interrupt Falling Trigger.

Return value:

- None.

__HAL_RCC_LSECSS_EXTI_ENABLE_RISING_EDGE

Description:

- Enable the RCC LSE CSS Extended Interrupt Rising Trigger.

Return value:

- None.

__HAL_RCC_LSECSS_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the RCC LSE CSS Extended Interrupt Rising Trigger.

Return value:

- None.

__HAL_RCC_LSECSS_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable the RCC LSE CSS Extended Interrupt Rising & Falling Trigger.

Return value:

- None.

__HAL_RCC_LSECSS_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable the RCC LSE CSS Extended Interrupt Rising & Falling Trigger.

Return value:

- None.

__HAL_RCC_LSECSS_EXTI_GET_FLAG

Description:

- Check whether the specified RCC LSE CSS EXTI interrupt flag is set or not.

Return value:

- EXTI: RCC LSE CSS Line Status.

__HAL_RCC_LSECSS_EXTI_CLEAR_FLAG

Description:

- Clear the RCC LSE CSS EXTI flag.

Return value:

- None.

__HAL_RCC_C2_LSECSS_EXTI_GET_FLAG

Description:

- Check whether the specified RCC LSE CSS EXTI interrupt flag is set or not for CM4.

Return value:

- EXTI: RCC LSE CSS Line Status.

__HAL_RCC_C2_LSECSS_EXTI_CLEAR_FLAG

Description:

- Clear the RCC LSE CSS EXTI flag or not for CM4.

Return value:

- None.

__HAL_RCC_LSECSS_EXTI_GENERATE_SWIT

Description:

- Generate a Software interrupt on the RCC LSE CSS EXTI line.

Return value:

- None.

__HAL_RCC_CRS_ENABLE_IT

Description:

- Enable the specified CRS interrupts.

Parameters:

- __INTERRUPT__: specifies the CRS interrupt sources to be enabled. This parameter can be any combination of the following values:
  - RCC_CRS_IT_SYNCOK SYNC event OK interrupt
  - RCC_CRS_IT_SYNCWARN SYNC warning interrupt
  - RCC_CRS_IT_ERR Synchronization or trimming error interrupt
  - RCC_CRS_IT_ESYNC Expected SYNC interrupt

Return value:

- None

__HAL_RCC_CRS_DISABLE_IT

Description:

- Disable the specified CRS interrupts.

Parameters:

- __INTERRUPT__: specifies the CRS interrupt sources to be disabled. This parameter can be any combination of the following values:
  - RCC_CRS_IT_SYNCOK SYNC event OK interrupt
  - RCC_CRS_IT_SYNCWARN SYNC warning interrupt
  - RCC_CRS_IT_ERR Synchronization or trimming error interrupt
  - RCC_CRS_IT_ESYNC Expected SYNC interrupt

Return value:

- None

__HAL_RCC_CRS_GET_IT_SOURCE

Description:

- Check whether the CRS interrupt has occurred or not.

Parameters:

- __INTERRUPT__: specifies the CRS interrupt source to check. This parameter can be one of the following values:
  - RCC_CRS_IT_SYNCOK SYNC event OK interrupt
  - RCC_CRS_IT_SYNCWARN SYNC warning interrupt
  - RCC_CRS_IT_ERR Synchronization or trimming error interrupt
  - RCC_CRS_IT_ESYNC Expected SYNC interrupt

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

RCC_CRS_IT_ERROR_MASK

Description:

- Clear the CRS interrupt pending bits.

Parameters:

- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:
  - RCC_CRS_IT_SYNCOK SYNC event OK interrupt
  - RCC_CRS_IT_SYNCWARN SYNC warning interrupt
  - RCC_CRS_IT_ERR Synchronization or trimming error interrupt
  - RCC_CRS_IT_ESYNC Expected SYNC interrupt
  - RCC_CRS_IT_TRIMOVF Trimming overflow or underflow interrupt
  - RCC_CRS_IT_SYNCERR SYNC error interrupt
  - RCC_CRS_IT_SYNCMISS SYNC missed interrupt

__HAL_RCC_CRS_CLEAR_IT

__HAL_RCC_CRS_GET_FLAG

Description:

- Check whether the specified CRS flag is set or not.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - RCC_CRS_FLAG_SYNCOK SYNC event OK
  - RCC_CRS_FLAG_SYNCWARN SYNC warning
  - RCC_CRS_FLAG_ERR Error
  - RCC_CRS_FLAG_ESYNC Expected SYNC
  - RCC_CRS_FLAG_TRIMOVF Trimming overflow or underflow
  - RCC_CRS_FLAG_SYNCERR SYNC error
  - RCC_CRS_FLAG_SYNCMISS SYNC missed

Return value:

- The: new state of _FLAG_ (TRUE or FALSE).

RCC_CRS_FLAG_ERROR_MASK

Description:

- Clear the CRS specified FLAG.

Parameters:

- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
  - RCC_CRS_FLAG_SYNCOK SYNC event OK
  - RCC_CRS_FLAG_SYNCWARN SYNC warning
  - RCC_CRS_FLAG_ERR Error
  - RCC_CRS_FLAG_ESYNC Expected SYNC
  - RCC_CRS_FLAG_TRIMOVF Trimming overflow or underflow
  - RCC_CRS_FLAG_SYNCERR SYNC error
  - RCC_CRS_FLAG_SYNCMISS SYNC missed

Return value:

- None

Notes:

- RCC_CRS_FLAG_ERR clears RCC_CRS_FLAG_TRIMOVF, RCC_CRS_FLAG_SYNCERR, RCC_CRS_FLAG_SYNCMISS and consequently RCC_CRS_FLAG_ERR

__HAL_RCC_CRS_CLEAR_FLAG

RCCEx Exported Types

I2c1235ClockSelection

RCC LSE CSS external interrupt line

RCC_EXTI_LINE_LSECSS

External interrupt line 18 connected to the LSE CSS EXTI Line

RCCEx FDCAN Clock Source

RCC_FDCANCLKSOURCE_HSE

RCC_FDCANCLKSOURCE_PLL

RCC_FDCANCLKSOURCE_PLL2

RCCEx FMC Clock Source

RCC_FMCCLKSOURCE_D1HCLK

RCC_FMCCLKSOURCE_HCLK

RCC_FMCCLKSOURCE_PLL

RCC_FMCCLKSOURCE_PLL2

RCC_FMCCLKSOURCE_CLKP

RCC Extended HRTIM1 Clock Source

RCC_HRTIM1CLK_TIMCLK

RCC_HRTIM1CLK_CPUCLK

RCCEx I2C1/2/3/5 Clock Source

RCC_I2C123CLKSOURCE_D2PCLK1

RCC_I2C123CLKSOURCE_PLL3

RCC_I2C123CLKSOURCE_HSI

RCC_I2C123CLKSOURCE_CSI

RCC_I2C1235CLKSOURCE_D2PCLK1

RCC_I2C1235CLKSOURCE_PLL3

RCC_I2C1235CLKSOURCE_HSI

RCC_I2C1235CLKSOURCE_CSI

RCCEx I2C1 Clock Source

RCC_I2C1CLKSOURCE_D2PCLK1

RCC_I2C1CLKSOURCE_PLL3

RCC_I2C1CLKSOURCE_HSI

RCC_I2C1CLKSOURCE_CSI

RCCEx I2C2 Clock Source

RCC_I2C2CLKSOURCE_D2PCLK1

RCC_I2C2CLKSOURCE_PLL3

RCC_I2C2CLKSOURCE_HSI

RCC_I2C2CLKSOURCE_CSI

RCCEx I2C3 Clock Source

RCC_I2C3CLKSOURCE_D2PCLK1

RCC_I2C3CLKSOURCE_PLL3

RCC_I2C3CLKSOURCE_HSI

RCC_I2C3CLKSOURCE_CSI

RCCEx I2C4 Clock Source

RCC_I2C4CLKSOURCE_D3PCLK1

RCC_I2C4CLKSOURCE_PLL3

RCC_I2C4CLKSOURCE_HSI

RCC_I2C4CLKSOURCE_CSI

RCC Private macros to check input parameters

IS_RCC_PLL2CLOCKOUT_VALUE

IS_RCC_PLL3CLOCKOUT_VALUE

IS_RCC_USART16CLKSOURCE

IS_RCC_USART234578CLKSOURCE

IS_RCC_USART1CLKSOURCE

IS_RCC_USART2CLKSOURCE

IS_RCC_USART3CLKSOURCE

IS_RCC_UART4CLKSOURCE

IS_RCC_UART5CLKSOURCE

IS_RCC_USART6CLKSOURCE

IS_RCC_UART7CLKSOURCE

IS_RCC_UART8CLKSOURCE

IS_RCC_LPUART1CLKSOURCE

IS_RCC_I2C123CLKSOURCE

IS_RCC_I2C1CLKSOURCE

IS_RCC_I2C2CLKSOURCE

IS_RCC_I2C3CLKSOURCE

IS_RCC_I2C4CLKSOURCE

IS_RCC_RNGCLKSOURCE

IS_RCC_HRTIM1CLKSOURCE

IS_RCC_USBCLKSOURCE

IS_RCC_SAI1CLK

IS_RCC_SAI23CLK

IS_RCC_SAI2CLK

IS_RCC_SAI3CLK

IS_RCC_SPI123CLK

IS_RCC_SPI1CLK

IS_RCC_SPI2CLK

IS_RCC_SPI3CLK

IS_RCC_SPI45CLK

IS_RCC_SPI4CLK

IS_RCC_SPI5CLK

IS_RCC_SPI6CLK

IS_RCC_SAI4ACLK

IS_RCC_SAI4BCLK

IS_RCC_PLL3M_VALUE

IS_RCC_PLL3N_VALUE

IS_RCC_PLL3P_VALUE

IS_RCC_PLL3Q_VALUE

IS_RCC_PLL3R_VALUE

IS_RCC_PLL2M_VALUE

IS_RCC_PLL2N_VALUE

IS_RCC_PLL2P_VALUE

IS_RCC_PLL2Q_VALUE

IS_RCC_PLL2R_VALUE

IS_RCC_PLL2RGE_VALUE

IS_RCC_PLL3RGE_VALUE

IS_RCC_PLL2VCO_VALUE

IS_RCC_PLL3VCO_VALUE

IS_RCC_LPTIM1CLK

IS_RCC_LPTIM2CLK

IS_RCC_LPTIM345CLK

IS_RCC_LPTIM3CLK

IS_RCC_LPTIM4CLK

IS_RCC_LPTIM5CLK

IS_RCC_QSPICLK

IS_RCC_FMCCLK

IS_RCC_FDCANCLK

IS_RCC_SDMMC

IS_RCC_ADCCLKSOURCE

IS_RCC_SWPMI1CLKSOURCE

IS_RCC_DFSDM1CLKSOURCE

IS_RCC_SPDIFRXCLKSOURCE

IS_RCC_CECCLKSOURCE

IS_RCC_CLKPSOURCE

IS_RCC_TIMPRES

IS_RCC_BOOT_CORE

IS_RCC_SCOPE_WWDG

IS_RCC_CRS_SYNC_SOURCE

IS_RCC_CRS_SYNC_DIV

IS_RCC_CRS_SYNC_POLARITY

IS_RCC_CRS_RELOADVALUE

IS_RCC_CRS_ERRORLIMIT

IS_RCC_CRS_HSI48CALIBRATION

IS_RCC_CRS_FREQERRORDIR

RCCEx LPTIM1 Clock Source

RCC_LPTIM1CLKSOURCE_D2PCLK1

RCC_LPTIM1CLKSOURCE_PCLK1

RCC_LPTIM1CLKSOURCE_PLL2

RCC_LPTIM1CLKSOURCE_PLL3

RCC_LPTIM1CLKSOURCE_LSE

RCC_LPTIM1CLKSOURCE_LSI

RCC_LPTIM1CLKSOURCE_CLKP

RCCEx LPTIM2 Clock Source

RCC_LPTIM2CLKSOURCE_D3PCLK1

RCC_LPTIM2CLKSOURCE_PCLK4

RCC_LPTIM2CLKSOURCE_PLL2

RCC_LPTIM2CLKSOURCE_PLL3

RCC_LPTIM2CLKSOURCE_LSE

RCC_LPTIM2CLKSOURCE_LSI

RCC_LPTIM2CLKSOURCE_CLKP

RCCEx LPTIM3/4/5 Clock Source

RCC_LPTIM345CLKSOURCE_D3PCLK1

RCC_LPTIM345CLKSOURCE_PCLK4

RCC_LPTIM345CLKSOURCE_PLL2

RCC_LPTIM345CLKSOURCE_PLL3

RCC_LPTIM345CLKSOURCE_LSE

RCC_LPTIM345CLKSOURCE_LSI

RCC_LPTIM345CLKSOURCE_CLKP

RCCEx LPTIM3 Clock Source

RCC_LPTIM3CLKSOURCE_D3PCLK1

RCC_LPTIM3CLKSOURCE_PLL2

RCC_LPTIM3CLKSOURCE_PLL3

RCC_LPTIM3CLKSOURCE_LSE

RCC_LPTIM3CLKSOURCE_LSI

RCC_LPTIM3CLKSOURCE_CLKP

RCCEx LPTIM4 Clock Source

RCC_LPTIM4CLKSOURCE_D3PCLK1

RCC_LPTIM4CLKSOURCE_PLL2

RCC_LPTIM4CLKSOURCE_PLL3

RCC_LPTIM4CLKSOURCE_LSE

RCC_LPTIM4CLKSOURCE_LSI

RCC_LPTIM4CLKSOURCE_CLKP

RCCEx LPTIM5 Clock Source

RCC_LPTIM5CLKSOURCE_D3PCLK1

RCC_LPTIM5CLKSOURCE_PLL2

RCC_LPTIM5CLKSOURCE_PLL3

RCC_LPTIM5CLKSOURCE_LSE

RCC_LPTIM5CLKSOURCE_LSI

RCC_LPTIM5CLKSOURCE_CLKP

RCCEx LPUART1 Clock Source

RCC_LPUART1CLKSOURCE_D3PCLK1

RCC_LPUART1CLKSOURCE_PCLK4

RCC_LPUART1CLKSOURCE_PLL2

RCC_LPUART1CLKSOURCE_PLL3

RCC_LPUART1CLKSOURCE_HSI

RCC_LPUART1CLKSOURCE_CSI

RCC_LPUART1CLKSOURCE_LSE

RCC Extended MCOx Clock Config

__HAL_RCC_MCO1_CONFIG

Description:

- Macro to configure the MCO1 clock.

Parameters:

- __MCOCLKSOURCE__: specifies the MCO clock source. This parameter can be one of the following values:
  - RCC_MCO1SOURCE_HSI: HSI clock selected as MCO1 source
  - RCC_MCO1SOURCE_LSE: LSE clock selected as MCO1 source
  - RCC_MCO1SOURCE_HSE: HSE clock selected as MCO1 source
  - RCC_MCO1SOURCE_PLL1QCLK: PLL1Q clock selected as MCO1 source
  - RCC_MCO1SOURCE_HSI48: HSI48 (48MHZ) selected as MCO1 source
- __MCODIV__: specifies the MCO clock prescaler. This parameter can be one of the following values:
  - RCC_MCODIV_1 up to RCC_MCODIV_15 : divider applied to MCO1 clock

__HAL_RCC_MCO2_CONFIG

Description:

- Macro to configure the MCO2 clock.

Parameters:

- __MCOCLKSOURCE__: specifies the MCO clock source. This parameter can be one of the following values:
  - RCC_MCO2SOURCE_SYSCLK: System clock (SYSCLK) selected as MCO2 source
  - RCC_MCO2SOURCE_PLL2PCLK: PLL2P clock selected as MCO2 source
  - RCC_MCO2SOURCE_HSE: HSE clock selected as MCO2 source
  - RCC_MCO2SOURCE_PLLCLK: PLL1P clock selected as MCO2 source
  - RCC_MCO2SOURCE_CSICLK: CSI clock selected as MCO2 source
  - RCC_MCO2SOURCE_LSICLK: LSI clock selected as MCO2 source
- __MCODIV__: specifies the MCO clock prescaler. This parameter can be one of the following values:
  - RCC_MCODIV_1 up to RCC_MCODIV_15 : divider applied to MCO2 clock

RCCEx Periph Clock Selection

RCC_PERIPHCLK_USART16

RCC_PERIPHCLK_USART1

RCC_PERIPHCLK_USART6

RCC_PERIPHCLK_USART16910

RCC_PERIPHCLK_USART234578

RCC_PERIPHCLK_USART2

RCC_PERIPHCLK_USART3

RCC_PERIPHCLK_UART4

RCC_PERIPHCLK_UART5

RCC_PERIPHCLK_UART7

RCC_PERIPHCLK_UART8

RCC_PERIPHCLK_LPUART1

RCC_PERIPHCLK_I2C123

RCC_PERIPHCLK_I2C1

RCC_PERIPHCLK_I2C2

RCC_PERIPHCLK_I2C3

RCC_PERIPHCLK_I2C4

RCC_PERIPHCLK_LPTIM1

RCC_PERIPHCLK_LPTIM2

RCC_PERIPHCLK_LPTIM345

RCC_PERIPHCLK_LPTIM3

RCC_PERIPHCLK_LPTIM4

RCC_PERIPHCLK_LPTIM5

RCC_PERIPHCLK_SAI1

RCC_PERIPHCLK_SAI23

RCC_PERIPHCLK_SAI2

RCC_PERIPHCLK_SAI3

RCC_PERIPHCLK_SAI4A

RCC_PERIPHCLK_SAI4B

RCC_PERIPHCLK_SPI123

RCC_PERIPHCLK_SPI1

RCC_PERIPHCLK_SPI2

RCC_PERIPHCLK_SPI3

RCC_PERIPHCLK_SPI45

RCC_PERIPHCLK_SPI4

RCC_PERIPHCLK_SPI5

RCC_PERIPHCLK_SPI6

RCC_PERIPHCLK_FDCAN

RCC_PERIPHCLK_SDMMC

RCC_PERIPHCLK_RNG

RCC_PERIPHCLK_USB

RCC_PERIPHCLK_ADC

RCC_PERIPHCLK_SWPMI1

RCC_PERIPHCLK_DFSDM1

RCC_PERIPHCLK_RTC

RCC_PERIPHCLK_CEC

RCC_PERIPHCLK_FMC

RCC_PERIPHCLK_QSPI

RCC_PERIPHCLK_DSI

RCC_PERIPHCLK_SPDIFRX

RCC_PERIPHCLK_HRTIM1

RCC_PERIPHCLK_LTDC

RCC_PERIPHCLK_TIM

RCC_PERIPHCLK_CKPER

RCC_PERIPHCLK_PLL2_DIVP

RCC_PERIPHCLK_PLL2_DIVQ

RCC_PERIPHCLK_PLL2_DIVR

RCC_PERIPHCLK_PLL3_DIVP

RCC_PERIPHCLK_PLL3_DIVQ

RCC_PERIPHCLK_PLL3_DIVR

RCCEx QSPI Clock Source

RCC_QSPICLKSOURCE_D1HCLK

RCC_QSPICLKSOURCE_PLL

RCC_QSPICLKSOURCE_PLL2

RCC_QSPICLKSOURCE_CLKP

RCCEx RCC BootCx

RCC_BOOT_C1

RCC_BOOT_C2

RCCEx RCC WWDGx

RCC_WWDG1

RCC_WWDG2

RCCEx RNG Clock Source

RCC_RNGCLKSOURCE_HSI48

RCC_RNGCLKSOURCE_PLL

RCC_RNGCLKSOURCE_LSE

RCC_RNGCLKSOURCE_LSI

SAI1 Clock Source

RCC_SAI1CLKSOURCE_PLL

RCC_SAI1CLKSOURCE_PLL2

RCC_SAI1CLKSOURCE_PLL3

RCC_SAI1CLKSOURCE_PIN

RCC_SAI1CLKSOURCE_CLKP

SAI2/3 Clock Source

RCC_SAI23CLKSOURCE_PLL

RCC_SAI23CLKSOURCE_PLL2

RCC_SAI23CLKSOURCE_PLL3

RCC_SAI23CLKSOURCE_PIN

RCC_SAI23CLKSOURCE_CLKP

SAI2 Clock Source

RCC_SAI2CLKSOURCE_PLL

RCC_SAI2CLKSOURCE_PLL2

RCC_SAI2CLKSOURCE_PLL3

RCC_SAI2CLKSOURCE_PIN

RCC_SAI2CLKSOURCE_CLKP

SAI3 Clock Source

RCC_SAI3CLKSOURCE_PLL

RCC_SAI3CLKSOURCE_PLL2

RCC_SAI3CLKSOURCE_PLL3

RCC_SAI3CLKSOURCE_PIN

RCC_SAI3CLKSOURCE_CLKP

SAI4A Clock Source

RCC_SAI4ACLKSOURCE_PLL

RCC_SAI4ACLKSOURCE_PLL2

RCC_SAI4ACLKSOURCE_PLL3

RCC_SAI4ACLKSOURCE_PIN

RCC_SAI4ACLKSOURCE_CLKP

SAI4B Clock Source

RCC_SAI4BCLKSOURCE_PLL

RCC_SAI4BCLKSOURCE_PLL2

RCC_SAI4BCLKSOURCE_PLL3

RCC_SAI4BCLKSOURCE_PIN

RCC_SAI4BCLKSOURCE_CLKP

RCCEx SDMMC Clock Source

RCC_SDMMCCLKSOURCE_PLL

RCC_SDMMCCLKSOURCE_PLL2

RCCEx SPDIFRX Clock Source

RCC_SPDIFRXCLKSOURCE_PLL

RCC_SPDIFRXCLKSOURCE_PLL2

RCC_SPDIFRXCLKSOURCE_PLL3

RCC_SPDIFRXCLKSOURCE_HSI

SPI1/2/3 Clock Source

RCC_SPI123CLKSOURCE_PLL

RCC_SPI123CLKSOURCE_PLL2

RCC_SPI123CLKSOURCE_PLL3

RCC_SPI123CLKSOURCE_PIN

RCC_SPI123CLKSOURCE_CLKP

SPI1 Clock Source

RCC_SPI1CLKSOURCE_PLL

RCC_SPI1CLKSOURCE_PLL2

RCC_SPI1CLKSOURCE_PLL3

RCC_SPI1CLKSOURCE_PIN

RCC_SPI1CLKSOURCE_CLKP

SPI2 Clock Source

RCC_SPI2CLKSOURCE_PLL

RCC_SPI2CLKSOURCE_PLL2

RCC_SPI2CLKSOURCE_PLL3

RCC_SPI2CLKSOURCE_PIN

RCC_SPI2CLKSOURCE_CLKP

SPI3 Clock Source

RCC_SPI3CLKSOURCE_PLL

RCC_SPI3CLKSOURCE_PLL2

RCC_SPI3CLKSOURCE_PLL3

RCC_SPI3CLKSOURCE_PIN

RCC_SPI3CLKSOURCE_CLKP

SPI4/5 Clock Source

RCC_SPI45CLKSOURCE_D2PCLK2

RCC_SPI45CLKSOURCE_PCLK2

RCC_SPI45CLKSOURCE_PLL2

RCC_SPI45CLKSOURCE_PLL3

RCC_SPI45CLKSOURCE_HSI

RCC_SPI45CLKSOURCE_CSI

RCC_SPI45CLKSOURCE_HSE

SPI4 Clock Source

RCC_SPI4CLKSOURCE_D2PCLK2

RCC_SPI4CLKSOURCE_PLL2

RCC_SPI4CLKSOURCE_PLL3

RCC_SPI4CLKSOURCE_HSI

RCC_SPI4CLKSOURCE_CSI

RCC_SPI4CLKSOURCE_HSE

SPI5 Clock Source

RCC_SPI5CLKSOURCE_D2PCLK2

RCC_SPI5CLKSOURCE_PLL2

RCC_SPI5CLKSOURCE_PLL3

RCC_SPI5CLKSOURCE_HSI

RCC_SPI5CLKSOURCE_CSI

RCC_SPI5CLKSOURCE_HSE

SPI6 Clock Source

RCC_SPI6CLKSOURCE_D3PCLK1

RCC_SPI6CLKSOURCE_PCLK4

RCC_SPI6CLKSOURCE_PLL2

RCC_SPI6CLKSOURCE_PLL3

RCC_SPI6CLKSOURCE_HSI

RCC_SPI6CLKSOURCE_CSI

RCC_SPI6CLKSOURCE_HSE

RCCEx SWPMI1 Clock Source

RCC_SWPMI1CLKSOURCE_D2PCLK1

RCC_SWPMI1CLKSOURCE_HSI

RCCEx TIM Prescaler Selection

RCC_TIMPRES_DESACTIVATED

RCC_TIMPRES_ACTIVATED

RCCEx UART4 Clock Source

RCC_UART4CLKSOURCE_D2PCLK1

RCC_UART4CLKSOURCE_PLL2

RCC_UART4CLKSOURCE_PLL3

RCC_UART4CLKSOURCE_HSI

RCC_UART4CLKSOURCE_CSI

RCC_UART4CLKSOURCE_LSE

RCCEx UART5 Clock Source

RCC_UART5CLKSOURCE_D2PCLK1

RCC_UART5CLKSOURCE_PLL2

RCC_UART5CLKSOURCE_PLL3

RCC_UART5CLKSOURCE_HSI

RCC_UART5CLKSOURCE_CSI

RCC_UART5CLKSOURCE_LSE

RCCEx UART7 Clock Source

RCC_UART7CLKSOURCE_D2PCLK1

RCC_UART7CLKSOURCE_PLL2

RCC_UART7CLKSOURCE_PLL3

RCC_UART7CLKSOURCE_HSI

RCC_UART7CLKSOURCE_CSI

RCC_UART7CLKSOURCE_LSE

RCCEx UART8 Clock Source

RCC_UART8CLKSOURCE_D2PCLK1

RCC_UART8CLKSOURCE_PLL2

RCC_UART8CLKSOURCE_PLL3

RCC_UART8CLKSOURCE_HSI

RCC_UART8CLKSOURCE_CSI

RCC_UART8CLKSOURCE_LSE

RCCEx USART1/6 Clock Source

RCC_USART16CLKSOURCE_D2PCLK2

RCC_USART16CLKSOURCE_PCLK2

RCC_USART16CLKSOURCE_PLL2

RCC_USART16CLKSOURCE_PLL3

RCC_USART16CLKSOURCE_HSI

RCC_USART16CLKSOURCE_CSI

RCC_USART16CLKSOURCE_LSE

RCCEx USART1 Clock Source

RCC_USART1CLKSOURCE_D2PCLK2

RCC_USART1CLKSOURCE_PLL2

RCC_USART1CLKSOURCE_PLL3

RCC_USART1CLKSOURCE_HSI

RCC_USART1CLKSOURCE_CSI

RCC_USART1CLKSOURCE_LSE

RCCEx USART2/3/4/5/7/8 Clock Source

RCC_USART234578CLKSOURCE_D2PCLK1

RCC_USART234578CLKSOURCE_PCLK1

RCC_USART234578CLKSOURCE_PLL2

RCC_USART234578CLKSOURCE_PLL3

RCC_USART234578CLKSOURCE_HSI

RCC_USART234578CLKSOURCE_CSI

RCC_USART234578CLKSOURCE_LSE

RCCEx USART2 Clock Source

RCC_USART2CLKSOURCE_D2PCLK1

RCC_USART2CLKSOURCE_PLL2

RCC_USART2CLKSOURCE_PLL3

RCC_USART2CLKSOURCE_HSI

RCC_USART2CLKSOURCE_CSI

RCC_USART2CLKSOURCE_LSE

RCCEx USART3 Clock Source

RCC_USART3CLKSOURCE_D2PCLK1

RCC_USART3CLKSOURCE_PLL2

RCC_USART3CLKSOURCE_PLL3

RCC_USART3CLKSOURCE_HSI

RCC_USART3CLKSOURCE_CSI

RCC_USART3CLKSOURCE_LSE

RCCEx USART6 Clock Source

RCC_USART6CLKSOURCE_D2PCLK2

RCC_USART6CLKSOURCE_PLL2

RCC_USART6CLKSOURCE_PLL3

RCC_USART6CLKSOURCE_HSI

RCC_USART6CLKSOURCE_CSI

RCC_USART6CLKSOURCE_LSE

RCCEx USB Clock Source

RCC_USBCLKSOURCE_PLL

RCC_USBCLKSOURCE_PLL3

RCC_USBCLKSOURCE_HSI48

70 HAL RNG Generic Driver

70.1 RNG Firmware driver registers structures

70.1.1 RNG_InitTypeDef

RNG_InitTypeDef is defined in the stm32h7xx_hal_rng.h

Data Fields

- ***uint32_t ClockErrorDetection***

Field Documentation

- ***uint32_t RNG_InitTypeDef::ClockErrorDetection***
  CED Clock error detection

70.1.2 __RNG_HandleTypeDef

__RNG_HandleTypeDef is defined in the stm32h7xx_hal_rng.h

Data Fields

- ***RNG_TypeDef * Instance***
- ***RNG_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_RNG_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***uint32_t RandomNumber***
- ***void(* ReadyDataCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***RNG_TypeDef* __RNG_HandleTypeDef::Instance***
  Register base address
- ***RNG_InitTypeDef __RNG_HandleTypeDef::Init***
  RNG configuration parameters
- ***HAL_LockTypeDef __RNG_HandleTypeDef::Lock***
  RNG locking object
- ***__IO HAL_RNG_StateTypeDef __RNG_HandleTypeDef::State***
  RNG communication state
- ***__IO uint32_t __RNG_HandleTypeDef::ErrorCode***
  RNG Error code
- ***uint32_t __RNG_HandleTypeDef::RandomNumber***
  Last Generated RNG Data
- ***void(* __RNG_HandleTypeDef::ReadyDataCallback)(struct __RNG_HandleTypeDef *hrng, uint32_t random32bit)***
  RNG Data Ready Callback
- ***void(* __RNG_HandleTypeDef::ErrorCallback)(struct __RNG_HandleTypeDef *hrng)***
  RNG Error Callback
- ***void(* __RNG_HandleTypeDef::MspInitCallback)(struct __RNG_HandleTypeDef *hrng)***
  RNG Msp Init callback

- ***void(* __RNG_HandleTypeDef::MspDeInitCallback)(struct __RNG_HandleTypeDef *hrng)***
  RNG Msp DeInit callback

70.2 RNG Firmware driver API description

The following section lists the various functions of the RNG library.

70.2.1 How to use this driver

The RNG HAL driver can be used as follows:

1. Enable the RNG controller clock using __HAL_RCC_RNG_CLK_ENABLE() macro in HAL_RNG_MspInit().
2. Activate the RNG peripheral using HAL_RNG_Init() function.
3. Wait until the 32 bit Random Number Generator contains a valid random data using (polling/interrupt) mode.
4. Get the 32 bit random number using HAL_RNG_GenerateRandomNumber() function.

70.2.2 Callback registration

The compilation define USE_HAL_RNG_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_RNG_RegisterCallback() to register a user callback. Function HAL_RNG_RegisterCallback() allows to register following callbacks:

- ErrorCallback : RNG Error Callback.
- MspInitCallback : RNG MspInit.
- MspDeInitCallback : RNG MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_RNG_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. HAL_RNG_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- ErrorCallback : RNG Error Callback.
- MspInitCallback : RNG MspInit.
- MspDeInitCallback : RNG MspDeInit.

For specific callback ReadyDataCallback, use dedicated register callbacks: respectively HAL_RNG_RegisterReadyDataCallback() , HAL_RNG_UnRegisterReadyDataCallback().

By default, after the HAL_RNG_Init() and when the state is HAL_RNG_STATE_RESET all callbacks are set to the corresponding weak (surcharged) functions: example HAL_RNG_ErrorCallback(). Exception done for MspInit and MspDeInit functions that are respectively reset to the legacy weak (surcharged) functions in the HAL_RNG_Init() and HAL_RNG_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_RNG_Init() and HAL_RNG_DeInit() keep and use the user MspInit/ MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_RNG_STATE_READY state only. Exception done MspInit/ MspDeInit that can be registered/unregistered in HAL_RNG_STATE_READY or HAL_RNG_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_RNG_RegisterCallback() before calling HAL_RNG_DeInit() or HAL_RNG_Init() function.

When The compilation define USE_HAL_RNG_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and weak (surcharged) callbacks are used.

70.2.3 Initialization and configuration functions

This section provides functions allowing to:

- Initialize the RNG according to the specified parameters in the RNG_InitTypeDef and create the associated handle
- DeInitialize the RNG peripheral
- Initialize the RNG MSP
- DeInitialize RNG MSP

This section contains the following APIs:

- *HAL_RNG_Init()*
- *HAL_RNG_DeInit()*
- *HAL_RNG_MspInit()*
- *HAL_RNG_MspDeInit()*
- *HAL_RNG_RegisterCallback()*
- *HAL_RNG_UnRegisterCallback()*
- *HAL_RNG_RegisterReadyDataCallback()*
- *HAL_RNG_UnRegisterReadyDataCallback()*

70.2.4 Peripheral Control functions

This section provides functions allowing to:

- Get the 32 bit Random number
- Get the 32 bit Random number with interrupt enabled
- Handle RNG interrupt request

This section contains the following APIs:

- *HAL_RNG_GenerateRandomNumber()*
- *HAL_RNG_GenerateRandomNumber_IT()*
- *HAL_RNG_IRQHandler()*
- *HAL_RNG_ReadLastRandomNumber()*
- *HAL_RNG_ReadyDataCallback()*
- *HAL_RNG_ErrorCallback()*

70.2.5 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- *HAL_RNG_GetState()*
- *HAL_RNG_GetError()*

70.2.6 Detailed description of functions

HAL_RNG_Init

Function name

HAL_StatusTypeDef HAL_RNG_Init (RNG_HandleTypeDef * hrng)

Function description

Initializes the RNG peripheral and creates the associated handle.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **HAL:** status

HAL_RNG_DeInit

Function name

HAL_StatusTypeDef HAL_RNG_DeInit (RNG_HandleTypeDef * hrng)

Function description

DeInitializes the RNG peripheral.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **HAL:** status

HAL_RNG_MspInit

Function name

void HAL_RNG_MspInit (RNG_HandleTypeDef * hrng)

Function description

Initializes the RNG MSP.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **None:**

HAL_RNG_MspDeInit

Function name

void HAL_RNG_MspDeInit (RNG_HandleTypeDef * hrng)

Function description

DeInitializes the RNG MSP.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **None:**

HAL_RNG_RegisterCallback

Function name

HAL_StatusTypeDef HAL_RNG_RegisterCallback (RNG_HandleTypeDef * hrng, HAL_RNG_CallbackIDTypeDef CallbackID, pRNG_CallbackTypeDef pCallback)

Function description

Register a User RNG Callback To be used instead of the weak predefined callback.

Parameters

- **hrng:** RNG handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_RNG_ERROR_CB_ID Error callback ID
  - HAL_RNG_MSPINIT_CB_ID MspInit callback ID
  - HAL_RNG_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_RNG_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_RNG_UnRegisterCallback (RNG_HandleTypeDef * hrng, HAL_RNG_CallbackIDTypeDef CallbackID)

Function description

Unregister an RNG Callback RNG callback is redirected to the weak predefined callback.

Parameters

- **hrng:** RNG handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_RNG_ERROR_CB_ID Error callback ID
  - HAL_RNG_MSPINIT_CB_ID MspInit callback ID
  - HAL_RNG_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_RNG_RegisterReadyDataCallback

Function name

HAL_StatusTypeDef HAL_RNG_RegisterReadyDataCallback (RNG_HandleTypeDef * hrng, pRNG_ReadyDataCallbackTypeDef pCallback)

Function description

Register Data Ready RNG Callback To be used instead of the weak HAL_RNG_ReadyDataCallback() predefined callback.

Parameters

- **hrng:** RNG handle
- **pCallback:** pointer to the Data Ready Callback function

Return values

- **HAL:** status

HAL_RNG_UnRegisterReadyDataCallback

Function name

HAL_StatusTypeDef HAL_RNG_UnRegisterReadyDataCallback (RNG_HandleTypeDef * hrng)

Function description

UnRegister the Data Ready RNG Callback Data Ready RNG Callback is redirected to the weak HAL_RNG_ReadyDataCallback() predefined callback.

Parameters

- **hrng:** RNG handle

Return values

- **HAL:** status

HAL_RNG_GenerateRandomNumber

Function name

HAL_StatusTypeDef HAL_RNG_GenerateRandomNumber (RNG_HandleTypeDef * hrng, uint32_t * random32bit)

Function description

Generates a 32-bit random number.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
- **random32bit:** pointer to generated random number variable if successful.

Return values

- **HAL:** status

Notes

- This function checks value of RNG_FLAG_DRDY flag to know if valid random number is available in the DR register (RNG_FLAG_DRDY flag set whenever a random number is available through the RNG_DR register). After transitioning from 0 to 1 (random number available), RNG_FLAG_DRDY flag remains high until output buffer becomes empty after reading four words from the RNG_DR register, i.e. further function calls will immediately return a new u32 random number (additional words are available and can be read by the application, till RNG_FLAG_DRDY flag remains high).
- When no more random number data is available in DR register, RNG_FLAG_DRDY flag is automatically cleared.

HAL_RNG_GenerateRandomNumber_IT

Function name

HAL_StatusTypeDef HAL_RNG_GenerateRandomNumber_IT (RNG_HandleTypeDef * hrng)

Function description

Generates a 32-bit random number in interrupt mode.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **HAL:** status

HAL_RNG_ReadLastRandomNumber

Function name

uint32_t HAL_RNG_ReadLastRandomNumber (RNG_HandleTypeDef * hrng)

Function description

Read latest generated random number.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **random:** value

HAL_RNG_IRQHandler

Function name

void HAL_RNG_IRQHandler (RNG_HandleTypeDef * hrng)

Function description

Handles RNG interrupt request.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **None:**

Notes

- In the case of a clock error, the RNG is no more able to generate random numbers because the PLL48CLK clock is not correct. User has to check that the clock controller is correctly configured to provide the RNG clock and clear the CEIS bit using __HAL_RNG_CLEAR_IT(). The clock error has no impact on the previously generated random numbers, and the RNG_DR register contents can be used.
- In the case of a seed error, the generation of random numbers is interrupted as long as the SECS bit is '1'. If a number is available in the RNG_DR register, it must not be used because it may not have enough entropy. In this case, it is recommended to clear the SEIS bit using __HAL_RNG_CLEAR_IT(), then disable and enable the RNG peripheral to reinitialize and restart the RNG.
- User-written HAL_RNG_ErrorCallback() API is called once whether SEIS or CEIS are set.

HAL_RNG_ErrorCallback

Function name

void HAL_RNG_ErrorCallback (RNG_HandleTypeDef * hrng)

Function description

RNG error callbacks.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **None:**

HAL_RNG_ReadyDataCallback

Function name

void HAL_RNG_ReadyDataCallback (RNG_HandleTypeDef * hrng, uint32_t random32bit)

Function description

Data Ready callback in non-blocking mode.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
- **random32bit:** generated random number.

Return values

- **None:**

Notes

- When RNG_FLAG_DRDY flag value is set, first random number has been read from DR register in IRQ Handler and is provided as callback parameter. Depending on valid data available in the conditioning output buffer, additional words can be read by the application from DR register till DRDY bit remains high.

HAL_RNG_GetState

Function name

HAL_RNG_StateTypeDef HAL_RNG_GetState (RNG_HandleTypeDef * hrng)

Function description

Returns the RNG state.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **HAL:** state

HAL_RNG_GetError

Function name

uint32_t HAL_RNG_GetError (RNG_HandleTypeDef * hrng)

Function description

Return the RNG handle error code.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure.

Return values

- **RNG:** Error Code

70.3 RNG Firmware driver defines

The following section lists the various define and macros of the module.

70.3.1 RNG

RNG

RNG Error Definition

HAL_RNG_ERROR_NONE

No error

HAL_RNG_ERROR_INVALID_CALLBACK

Invalid Callback error

HAL_RNG_ERROR_TIMEOUT

Timeout error

HAL_RNG_ERROR_BUSY

Busy error

HAL_RNG_ERROR_SEED

Seed error

HAL_RNG_ERROR_CLOCK

Clock error

RNG Interrupt definition

RNG_IT_DRDY

Data Ready interrupt

RNG_IT_CEI

Clock error interrupt

RNG_IT_SEI

Seed error interrupt

RNG Flag definition

RNG_FLAG_DRDY

Data ready

RNG_FLAG_CECS

Clock error current status

RNG_FLAG_SECS

Seed error current status

RNG Clock Error Detection

RNG_CED_ENABLE

Clock error detection Enabled

RNG_CED_DISABLE

Clock error detection Disabled

RNG Exported Macros

__HAL_RNG_RESET_HANDLE_STATE

Description:

- Reset RNG handle state.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_ENABLE

Description:

- Enables the RNG peripheral.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_DISABLE

Description:

- Disables the RNG peripheral.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_GET_FLAG

Description:

- Check the selected RNG flag status.

Parameters:

- __HANDLE__: RNG Handle
- __FLAG__: RNG flag This parameter can be one of the following values:
  - RNG_FLAG_DRDY: Data ready
  - RNG_FLAG_CECS: Clock error current status
  - RNG_FLAG_SECS: Seed error current status

Return value:

- The: new state of __FLAG__ (SET or RESET).

__HAL_RNG_CLEAR_FLAG

Description:

- Clears the selected RNG flag status.

Parameters:

- __HANDLE__: RNG handle
- __FLAG__: RNG flag to clear

Return value:

- None

Notes:

- WARNING: This is a dummy macro for HAL code alignment, flags RNG_FLAG_DRDY, RNG_FLAG_CECS and RNG_FLAG_SECS are read-only.

__HAL_RNG_ENABLE_IT

Description:

- Enables the RNG interrupts.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_DISABLE_IT

Description:

- Disables the RNG interrupts.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_GET_IT

Description:

- Checks whether the specified RNG interrupt has occurred or not.

Parameters:

- __HANDLE__: RNG Handle
- __INTERRUPT__: specifies the RNG interrupt status flag to check. This parameter can be one of the following values:
  - RNG_IT_DRDY: Data ready interrupt
  - RNG_IT_CEI: Clock error interrupt
  - RNG_IT_SEI: Seed error interrupt

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_RNG_CLEAR_IT

Description:

- Clear the RNG interrupt status flags.

Parameters:

- __HANDLE__: RNG Handle
- __INTERRUPT__: specifies the RNG interrupt status flag to clear. This parameter can be one of the following values:
  - RNG_IT_CEI: Clock error interrupt
  - RNG_IT_SEI: Seed error interrupt

Return value:

- None

Notes:

- RNG_IT_DRDY flag is read-only, reading RNG_DR register automatically clears RNG_IT_DRDY.

71 HAL RNG Extension Driver

71.1 RNGEx Firmware driver registers structures

71.1.1 RNG_ConfigTypeDef

RNG_ConfigTypeDef is defined in the stm32h7xx_hal_rng_ex.h

Data Fields

- ***uint32_t Config1***
- ***uint32_t Config2***
- ***uint32_t Config3***
- ***uint32_t ClockDivider***
- ***uint32_t NistCompliance***

Field Documentation

- ***uint32_t RNG_ConfigTypeDef::Config1***
  Config1 must be a value between 0 and 0x3F
- ***uint32_t RNG_ConfigTypeDef::Config2***
  Config2 must be a value between 0 and 0x7
- ***uint32_t RNG_ConfigTypeDef::Config3***
  Config3 must be a value between 0 and 0xF
- ***uint32_t RNG_ConfigTypeDef::ClockDivider***
  Clock Divider factor.This parameter can be a value of ***Section 70.3.1 RNG_Ex_Clock_Divider_Factor***
- ***uint32_t RNG_ConfigTypeDef::NistCompliance***
  NIST compliance.This parameter can be a value of ***Section 70.3.1 RNG_Ex_NIST_Compliance***

71.2 RNGEx Firmware driver defines

The following section lists the various define and macros of the module.

71.2.1 RNGEx

RNGEx

72 HAL RTC Generic Driver

72.1 RTC Firmware driver registers structures

72.1.1 RTC_InitTypeDef

RTC_InitTypeDef is defined in the stm32h7xx_hal_rtc.h

Data Fields

- ***uint32_t HourFormat***
- ***uint32_t AsynchPrediv***
- ***uint32_t SynchPrediv***
- ***uint32_t OutPut***
- ***uint32_t OutPutRemap***
- ***uint32_t OutPutPolarity***
- ***uint32_t OutPutType***

Field Documentation

- ***uint32_t RTC_InitTypeDef::HourFormat***

  Specifies the RTC Hour Format. This parameter can be a value of ***RTC_Hour_Formats_Definitions***
- ***uint32_t RTC_InitTypeDef::AsynchPrediv***

  Specifies the RTC Asynchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7F
- ***uint32_t RTC_InitTypeDef::SynchPrediv***

  Specifies the RTC Synchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7FFF
- ***uint32_t RTC_InitTypeDef::OutPut***

  Specifies which signal will be routed to the RTC output. This parameter can be a value of ***RTC_Output_selection_Definitions***
- ***uint32_t RTC_InitTypeDef::OutPutRemap***

  Specifies the remap for RTC output. This parameter can be a value of ***RTC_Output_ALARM_OUT_Remap***
- ***uint32_t RTC_InitTypeDef::OutPutPolarity***

  Specifies the polarity of the output signal. This parameter can be a value of ***RTC_Output_Polarity_Definitions***
- ***uint32_t RTC_InitTypeDef::OutPutType***

  Specifies the RTC Output Pin mode. This parameter can be a value of ***RTC_Output_Type_ALARM_OUT***

72.1.2 RTC_TimeTypeDef

RTC_TimeTypeDef is defined in the stm32h7xx_hal_rtc.h

Data Fields

- ***uint8_t Hours***
- ***uint8_t Minutes***
- ***uint8_t Seconds***
- ***uint8_t TimeFormat***
- ***uint32_t SubSeconds***
- ***uint32_t SecondFraction***
- ***uint32_t DayLightSaving***
- ***uint32_t StoreOperation***

Field Documentation

- ***uint8_t RTC_TimeTypeDef::Hours***

  Specifies the RTC Time Hour. This parameter must be a number between Min_Data = 0 and Max_Data = 12 if the RTC_HourFormat_12 is selected. This parameter must be a number between Min_Data = 0 and Max_Data = 23 if the RTC_HourFormat_24 is selected
- ***uint8_t RTC_TimeTypeDef::Minutes***

  Specifies the RTC Time Minutes. This parameter must be a number between Min_Data = 0 and Max_Data = 59
- ***uint8_t RTC_TimeTypeDef::Seconds***

  Specifies the RTC Time Seconds. This parameter must be a number between Min_Data = 0 and Max_Data = 59
- ***uint8_t RTC_TimeTypeDef::TimeFormat***

  Specifies the RTC AM/PM Time. This parameter can be a value of ***RTC_AM_PM_Definitions***
- ***uint32_t RTC_TimeTypeDef::SubSeconds***

  Specifies the RTC_SSR RTC Sub Second register content. This parameter corresponds to a time unit range between [0-1] Second with [1 Sec / SecondFraction +1] granularity
- ***uint32_t RTC_TimeTypeDef::SecondFraction***

  Specifies the range or granularity of Sub Second register content corresponding to Synchronous pre-scaler factor value (PREDIV_S) This parameter corresponds to a time unit range between [0-1] Second with [1 Sec / SecondFraction +1] granularity. This field will be used only by HAL_RTC_GetTime function
- ***uint32_t RTC_TimeTypeDef::DayLightSaving***

  Specifies RTC_DayLightSaveOperation: the value of hour adjustment. This parameter can be a value of ***RTC_DayLightSaving_Definitions***
- ***uint32_t RTC_TimeTypeDef::StoreOperation***

  Specifies RTC_StoreOperation value to be written in the BKP bit in CR register to store the operation. This parameter can be a value of ***RTC_StoreOperation_Definitions***

72.1.3 RTC_DateTypeDef

RTC_DateTypeDef is defined in the stm32h7xx_hal_rtc.h

Data Fields

- ***uint8_t WeekDay***
- ***uint8_t Month***
- ***uint8_t Date***
- ***uint8_t Year***

Field Documentation

- ***uint8_t RTC_DateTypeDef::WeekDay***

  Specifies the RTC Date WeekDay. This parameter can be a value of ***RTC_WeekDay_Definitions***
- ***uint8_t RTC_DateTypeDef::Month***

  Specifies the RTC Date Month (in BCD format). This parameter can be a value of ***RTC_Month_Date_Definitions***
- ***uint8_t RTC_DateTypeDef::Date***

  Specifies the RTC Date. This parameter must be a number between Min_Data = 1 and Max_Data = 31
- ***uint8_t RTC_DateTypeDef::Year***

  Specifies the RTC Date Year. This parameter must be a number between Min_Data = 0 and Max_Data = 99

72.1.4 RTC_AlarmTypeDef

RTC_AlarmTypeDef is defined in the stm32h7xx_hal_rtc.h

Data Fields

- ***RTC_TimeTypeDef AlarmTime***
- ***uint32_t AlarmMask***
- ***uint32_t AlarmSubSecondMask***
- ***uint32_t AlarmDateWeekDaySel***

- ***uint8_t AlarmDateWeekDay***
- ***uint32_t Alarm***

Field Documentation

- ***RTC_TimeTypeDef RTC_AlarmTypeDef::AlarmTime***
  Specifies the RTC Alarm Time members
- ***uint32_t RTC_AlarmTypeDef::AlarmMask***
  Specifies the RTC Alarm Masks. This parameter can be a value of ***RTC_AlarmMask_Definitions***
- ***uint32_t RTC_AlarmTypeDef::AlarmSubSecondMask***
  Specifies the RTC Alarm SubSeconds Masks. This parameter can be a value of ***RTC_Alarm_Sub_Seconds_Masks_Definitions***
- ***uint32_t RTC_AlarmTypeDef::AlarmDateWeekDaySel***
  Specifies the RTC Alarm is on Date or WeekDay. This parameter can be a value of ***RTC_AlarmDateWeekDay_Definitions***
- ***uint8_t RTC_AlarmTypeDef::AlarmDateWeekDay***
  Specifies the RTC Alarm Date/WeekDay. If the Alarm Date is selected, this parameter must be set to a value in the 1-31 range. If the Alarm WeekDay is selected, this parameter can be a value of ***RTC_WeekDay_Definitions***
- ***uint32_t RTC_AlarmTypeDef::Alarm***
  Specifies the alarm . This parameter can be a value of ***RTC_Alarms_Definitions***

72.1.5 __RTC_HandleTypeDef

__RTC_HandleTypeDef is defined in the stm32h7xx_hal_rtc.h

Data Fields

- ***RTC_TypeDef * Instance***
- ***RTC_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_RTCStateTypeDef State***
- ***void(* AlarmAEventCallback***
- ***void(* AlarmBEventCallback***
- ***void(* TimeStampEventCallback***
- ***void(* WakeUpTimerEventCallback***
- ***void(* Tamper1EventCallback***
- ***void(* Tamper2EventCallback***
- ***void(* Tamper3EventCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***RTC_TypeDef* __RTC_HandleTypeDef::Instance***
  Register base address
- ***RTC_InitTypeDef __RTC_HandleTypeDef::Init***
  RTC required parameters
- ***HAL_LockTypeDef __RTC_HandleTypeDef::Lock***
  RTC locking object
- ***__IO HAL_RTCStateTypeDef __RTC_HandleTypeDef::State***
  Time communication state
- ***void(* __RTC_HandleTypeDef::AlarmAEventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Alarm A Event callback
- ***void(* __RTC_HandleTypeDef::AlarmBEventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Alarm B Event callback

- ***void(* __RTC_HandleTypeDef::TimeStampEventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC TimeStamp Event callback
- ***void(* __RTC_HandleTypeDef::WakeUpTimerEventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC WakeUpTimer Event callback
- ***void(* __RTC_HandleTypeDef::Tamper1EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Tamper 1 Event callback
- ***void(* __RTC_HandleTypeDef::Tamper2EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Tamper 2 Event callback
- ***void(* __RTC_HandleTypeDef::Tamper3EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Tamper 3 Event callback
- ***void(* __RTC_HandleTypeDef::MspInitCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Msp Init callback
- ***void(* __RTC_HandleTypeDef::MspDeInitCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Msp DeInit callback

72.2 RTC Firmware driver API description

The following section lists the various functions of the RTC library.

72.2.1 RTC Operating Condition

The real-time clock (RTC) and the RTC backup registers can be powered from the VBAT voltage when the main VDD supply is powered off. To retain the content of the RTC backup registers and supply the RTC when VDD is turned off, VBAT pin can be connected to an optional standby voltage supplied by a battery or by another source.

72.2.2 Backup Domain Reset

The backup domain reset sets all RTC registers and the RCC_BDCR register to their reset values. A backup domain reset is generated when one of the following events occurs:

1. Software reset, triggered by setting the BDRST bit in the RCC Backup domain control register (RCC_BDCR).
2. VDD or VBAT power on, if both supplies have previously been powered off.
3. Tamper detection event resets all data backup registers.

72.2.3 Backup Domain Access

After reset, the backup domain (RTC registers, RTC backup data registers and backup SRAM) is protected against possible unwanted write accesses.

To enable access to the RTC Domain and RTC registers, proceed as follows:

1. Call the function HAL_RCCEx_PeriphCLKConfig with RCC_PERIPHCLK_RTC for PeriphClockSelection and select RTCClockSelection (LSE, LSI or HSEdiv32)
2. Enable RTC Clock using the __HAL_RCC_RTC_ENABLE() macro.

72.2.4 How to use RTC Driver

- Enable the RTC domain access (see description in the section above).
- Configure the RTC Prescaler (Asynchronous and Synchronous) and RTC hour format using the HAL_RTC_Init() function.

Time and Date configuration

- To configure the RTC Calendar (Time and Date) use the HAL_RTC_SetTime() and HAL_RTC_SetDate() functions.
- To read the RTC Calendar, use the HAL_RTC_GetTime() and HAL_RTC_GetDate() functions.

Alarm configuration

- To configure the RTC Alarm use the HAL_RTC_SetAlarm() function. You can also configure the RTC Alarm with interrupt mode using the HAL_RTC_SetAlarm_IT() function.
- To read the RTC Alarm, use the HAL_RTC_GetAlarm() function.

72.2.5 RTC and low power modes

The MCU can be woken up from a low power mode by an RTC alternate function.

The RTC alternate functions are the RTC alarms (Alarm A and Alarm B), RTC wakeup, RTC tamper event detection and RTC time stamp event detection. These RTC alternate functions can wake up the system from the Stop and Standby low power modes.

The system can also wake up from low power modes without depending on an external interrupt (Auto-wakeup mode), by using the RTC alarm or the RTC wakeup events.

The RTC provides a programmable time base for waking up from the Stop or Standby mode at regular intervals. Wakeup from STOP and STANDBY modes is possible only when the RTC clock source is LSE or LSI.

Callback registration

72.2.6 Initialization and de-initialization functions

This section provides functions allowing to initialize and configure the RTC Prescaler (Synchronous and Asynchronous), RTC Hour format, disable RTC registers Write protection, enter and exit the RTC initialization mode, RTC registers synchronization check and reference clock detection enable.

1. The RTC Prescaler is programmed to generate the RTC 1Hz time base. It is split into 2 programmable prescalers to minimize power consumption.
   – A 7-bit asynchronous prescaler and a 15-bit synchronous prescaler.
   – When both prescalers are used, it is recommended to configure the asynchronous prescaler to a high value to minimize power consumption.
2. All RTC registers are Write protected. Writing to the RTC registers is enabled by writing a key into the Write Protection register, RTC_WPR.
3. To configure the RTC Calendar, user application should enter initialization mode. In this mode, the calendar counter is stopped and its value can be updated. When the initialization sequence is complete, the calendar restarts counting after 4 RTCCLK cycles.
4. To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers.The HAL_RTC_WaitForSynchro() function implements the above software sequence (RSF clear and RSF check).

This section contains the following APIs:

- ***HAL_RTC_Init()***
- ***HAL_RTC_DeInit()***
- ***HAL_RTC_RegisterCallback()***
- ***HAL_RTC_UnRegisterCallback()***
- ***HAL_RTC_MspInit()***
- ***HAL_RTC_MspDeInit()***

72.2.7 RTC Time and Date functions

This section provides functions allowing to configure Time and Date features

This section contains the following APIs:

- ***HAL_RTC_SetTime()***
- ***HAL_RTC_GetTime()***
- ***HAL_RTC_SetDate()***
- ***HAL_RTC_GetDate()***

72.2.8 RTC Alarm functions

This section provides functions allowing to configure Alarm feature
This section contains the following APIs:

- *HAL_RTC_SetAlarm()*
- *HAL_RTC_SetAlarm_IT()*
- *HAL_RTC_DeactivateAlarm()*
- *HAL_RTC_GetAlarm()*
- *HAL_RTC_AlarmIRQHandler()*
- *HAL_RTC_AlarmAEventCallback()*
- *HAL_RTC_PollForAlarmAEvent()*

72.2.9 Peripheral Control functions

This subsection provides functions allowing to

- Wait for RTC Time and Date Synchronization

This section contains the following APIs:

- *HAL_RTC_WaitForSynchro()*

72.2.10 Peripheral State functions

This subsection provides functions allowing to

- Get RTC state

This section contains the following APIs:

- *HAL_RTC_GetState()*

72.2.11 Detailed description of functions

HAL_RTC_Init

Function name

HAL_StatusTypeDef HAL_RTC_Init (RTC_HandleTypeDef * hrtc)

Function description

Initialize the RTC peripheral.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTC_DeInit

Function name

HAL_StatusTypeDef HAL_RTC_DeInit (RTC_HandleTypeDef * hrtc)

Function description

DeInitialize the RTC peripheral.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- This function doesn't reset the RTC Backup Data registers.

HAL_RTC_MspInit

Function name

void HAL_RTC_MspInit (RTC_HandleTypeDef * hrtc)

Function description

Initialize the RTC MSP.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTC_MspDeInit

Function name

void HAL_RTC_MspDeInit (RTC_HandleTypeDef * hrtc)

Function description

DeInitialize the RTC MSP.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_RTC_RegisterCallback (RTC_HandleTypeDef * hrtc, HAL_RTC_CallbackIDTypeDef CallbackID, pRTC_CallbackTypeDef pCallback)

Function description

Register a User RTC Callback To be used instead of the weak predefined callback.

Parameters

- **hrtc:** RTC handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_RTC_ALARM_A_EVENT_CB_ID Alarm A Event Callback ID
  - HAL_RTC_ALARM_B_EVENT_CB_ID Alarm B Event Callback ID
  - HAL_RTC_TIMESTAMP_EVENT_CB_ID TimeStamp Event Callback ID
  - HAL_RTC_WAKEUPTIMER_EVENT_CB_ID WakeUp Timer Event Callback ID
  - HAL_RTC_TAMPER1_EVENT_CB_ID Tamper 1 Callback ID
  - HAL_RTC_TAMPER2_EVENT_CB_ID Tamper 2 Callback ID
  - HAL_RTC_TAMPER3_EVENT_CB_ID Tamper 3 Callback ID
  - HAL_RTC_INTERNAL_TAMPER1_EVENT_CB_ID Internal Tamper 1 Callback ID
  - HAL_RTC_INTERNAL_TAMPER2_EVENT_CB_ID Internal Tamper 2 Callback ID
  - HAL_RTC_INTERNAL_TAMPER3_EVENT_CB_ID Internal Tamper 3 Callback ID
  - HAL_RTC_INTERNAL_TAMPER4_EVENT_CB_ID Internal Tamper 4 Callback ID
  - HAL_RTC_INTERNAL_TAMPER5_EVENT_CB_ID Internal Tamper 5 Callback ID
  - HAL_RTC_INTERNAL_TAMPER6_EVENT_CB_ID Internal Tamper 6 Callback ID
  - HAL_RTC_INTERNAL_TAMPER8_EVENT_CB_ID Internal Tamper 8 Callback ID
  - HAL_RTC_MSPINIT_CB_ID Msp Init callback ID
  - HAL_RTC_MSPDEINIT_CB_ID Msp DeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_RTC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_RTC_UnRegisterCallback (RTC_HandleTypeDef * hrtc, HAL_RTC_CallbackIDTypeDef CallbackID)

Function description

Unregister an RTC Callback RTC callback is redirected to the weak predefined callback.

Parameters

- **hrtc:** RTC handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_RTC_ALARM_A_EVENT_CB_ID Alarm A Event Callback ID
  - HAL_RTC_ALARM_B_EVENT_CB_ID Alarm B Event Callback ID
  - HAL_RTC_TIMESTAMP_EVENT_CB_ID TimeStamp Event Callback ID
  - HAL_RTC_WAKEUPTIMER_EVENT_CB_ID WakeUp Timer Event Callback ID
  - HAL_RTC_TAMPER1_EVENT_CB_ID Tamper 1 Callback ID
  - HAL_RTC_TAMPER2_EVENT_CB_ID Tamper 2 Callback ID
  - HAL_RTC_TAMPER3_EVENT_CB_ID Tamper 3 Callback ID
  - HAL_RTC_INTERNAL_TAMPER1_EVENT_CB_ID Internal Tamper 1 Callback ID
  - HAL_RTC_INTERNAL_TAMPER2_EVENT_CB_ID Internal Tamper 2 Callback ID
  - HAL_RTC_INTERNAL_TAMPER3_EVENT_CB_ID Internal Tamper 3 Callback ID
  - HAL_RTC_INTERNAL_TAMPER4_EVENT_CB_ID Internal Tamper 4 Callback ID
  - HAL_RTC_INTERNAL_TAMPER5_EVENT_CB_ID Internal Tamper 5 Callback ID
  - HAL_RTC_INTERNAL_TAMPER6_EVENT_CB_ID Internal Tamper 6 Callback ID
  - HAL_RTC_INTERNAL_TAMPER8_EVENT_CB_ID Internal Tamper 8 Callback ID
  - HAL_RTC_MSPINIT_CB_ID Msp Init callback ID
  - HAL_RTC_MSPDEINIT_CB_ID Msp DeInit callback ID

Return values

- **HAL:** status

HAL_RTC_SetTime

Function name

HAL_StatusTypeDef HAL_RTC_SetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)

Function description

Set RTC current time.

Parameters

- **hrtc:** RTC handle
- **sTime:** Pointer to Time structure
- **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values:
  - RTC_FORMAT_BIN: Binary data format
  - RTC_FORMAT_BCD: BCD data format

Return values

- **HAL:** status

HAL_RTC_GetTime

Function name

HAL_StatusTypeDef HAL_RTC_GetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)

Function description

Get RTC current time.

Parameters

- **hrtc:** RTC handle
- **sTime:** Pointer to Time structure with Hours, Minutes and Seconds fields returned with input format (BIN or BCD), also SubSeconds field returning the RTC_SSR register content and SecondFraction field the Synchronous pre-scaler factor to be used for second fraction ratio computation.
- **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values:
  - RTC_FORMAT_BIN: Binary data format
  - RTC_FORMAT_BCD: BCD data format

Return values

- **HAL:** status

Notes

- You can use SubSeconds and SecondFraction (sTime structure fields returned) to convert SubSeconds value in second fraction ratio with time unit following generic formula: Second fraction ratio * time_unit= [(SecondFraction-SubSeconds)/(SecondFraction+1)] * time_unit This conversion can be performed only if no shift operation is pending (ie. SHFP=0) when PREDIV_S >= SS
- You must call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers to ensure consistency between the time and date values. Reading RTC current time locks the values in calendar shadow registers until Current date is read to ensure consistency between the time and date values.

HAL_RTC_SetDate

Function name

HAL_StatusTypeDef HAL_RTC_SetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)

Function description

Set RTC current date.

Parameters

- **hrtc:** RTC handle
- **sDate:** Pointer to date structure
- **Format:** specifies the format of the entered parameters. This parameter can be one of the following values:
  - RTC_FORMAT_BIN: Binary data format
  - RTC_FORMAT_BCD: BCD data format

Return values

- **HAL:** status

HAL_RTC_GetDate

Function name

HAL_StatusTypeDef HAL_RTC_GetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)

Function description

Get RTC current date.

Parameters

- **hrtc:** RTC handle
- **sDate:** Pointer to Date structure
- **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values:
  - RTC_FORMAT_BIN: Binary data format
  - RTC_FORMAT_BCD: BCD data format

Return values

- **HAL:** status

Notes

- You must call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers to ensure consistency between the time and date values. Reading RTC current time locks the values in calendar shadow registers until Current date is read.

HAL_RTC_SetAlarm

Function name

HAL_StatusTypeDef HAL_RTC_SetAlarm (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)

Function description

Set the specified RTC Alarm.

Parameters

- **hrtc:** RTC handle
- **sAlarm:** Pointer to Alarm structure
- **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values:
  - RTC_FORMAT_BIN: Binary data format
  - RTC_FORMAT_BCD: BCD data format

Return values

- **HAL:** status

HAL_RTC_SetAlarm_IT

Function name

HAL_StatusTypeDef HAL_RTC_SetAlarm_IT (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)

Function description

Set the specified RTC Alarm with Interrupt.

Parameters

- **hrtc:** RTC handle
- **sAlarm:** Pointer to Alarm structure
- **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values:
  - RTC_FORMAT_BIN: Binary data format
  - RTC_FORMAT_BCD: BCD data format

Return values

- **HAL:** status

Notes

- The Alarm register can only be written when the corresponding Alarm is disabled (Use the HAL_RTC_DeactivateAlarm()).
- The HAL_RTC_SetTime() must be called before enabling the Alarm feature.

HAL_RTC_DeactivateAlarm

Function name

HAL_StatusTypeDef HAL_RTC_DeactivateAlarm (RTC_HandleTypeDef * hrtc, uint32_t Alarm)

Function description

Deactivate the specified RTC Alarm.

Parameters

- **hrtc:** RTC handle
- **Alarm:** Specifies the Alarm. This parameter can be one of the following values:
  - RTC_ALARM_A: AlarmA
  - RTC_ALARM_B: AlarmB

Return values

- **HAL:** status

HAL_RTC_GetAlarm

Function name

HAL_StatusTypeDef HAL_RTC_GetAlarm (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Alarm, uint32_t Format)

Function description

Get the RTC Alarm value and masks.

Parameters

- **hrtc:** RTC handle
- **sAlarm:** Pointer to Date structure
- **Alarm:** Specifies the Alarm. This parameter can be one of the following values:
  - RTC_ALARM_A: AlarmA
  - RTC_ALARM_B: AlarmB
- **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values:
  - RTC_FORMAT_BIN: Binary data format
  - RTC_FORMAT_BCD: BCD data format

Return values

- **HAL:** status

HAL_RTC_AlarmIRQHandler

Function name

void HAL_RTC_AlarmIRQHandler (RTC_HandleTypeDef * hrtc)

Function description

Handle Alarm interrupt request.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTC_AlarmAEventCallback

Function name

void HAL_RTC_AlarmAEventCallback (RTC_HandleTypeDef * hrtc)

Function description

Alarm A callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTC_PollForAlarmAEvent

Function name

HAL_StatusTypeDef HAL_RTC_PollForAlarmAEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)

Function description

Handle AlarmA Polling request.

Parameters

- **hrtc:** RTC handle
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTC_WaitForSynchro

Function name

HAL_StatusTypeDef HAL_RTC_WaitForSynchro (RTC_HandleTypeDef * hrtc)

Function description

Wait until the RTC Time and Date registers (RTC_TR and RTC_DR) are synchronized with RTC APB clock.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- The RTC Resynchronization mode is write protected, use the __HAL_RTC_WRITEPROTECTION_DISABLE() before calling this function.
- To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers.

HAL_RTC_GetState

Function name

HAL_RTCStateTypeDef HAL_RTC_GetState (RTC_HandleTypeDef * hrtc)

Function description

Return the RTC handle state.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** state

RTC_EnterInitMode

Function name

HAL_StatusTypeDef RTC_EnterInitMode (RTC_HandleTypeDef * hrtc)

Function description

Enter the RTC Initialization mode.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- The RTC Initialization mode is write protected, use the __HAL_RTC_WRITEPROTECTION_DISABLE() before calling this function.

RTC_ExitInitMode

Function name

HAL_StatusTypeDef RTC_ExitInitMode (RTC_HandleTypeDef * hrtc)

Function description

Exit the RTC Initialization mode.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

RTC_ByteToBcd2

Function name

uint8_t RTC_ByteToBcd2 (uint8_t Value)

Function description

Convert a 2 digit decimal to BCD format.

Parameters

- **Value:** Byte to be converted

Return values

- **Converted:** byte

RTC_Bcd2ToByte

Function name

uint8_t RTC_Bcd2ToByte (uint8_t Value)

Function description

Convert from 2 digit BCD to Binary.

Parameters

- **Value:** BCD value to be converted

Return values

- **Converted:** word

72.3 RTC Firmware driver defines

The following section lists the various define and macros of the module.

72.3.1 RTC

RTC

RTC Alarm Date WeekDay Definitions

RTC_ALARMDATEWEEKDAYSEL_DATE

RTC_ALARMDATEWEEKDAYSEL_WEEKDAY

RTC Alarm Mask Definitions

RTC_ALARMMASK_NONE

RTC_ALARMMASK_DATEWEEKDAY

RTC_ALARMMASK_HOURS

RTC_ALARMMASK_MINUTES

RTC_ALARMMASK_SECONDS

RTC_ALARMMASK_ALL

RTC Alarms Definitions

RTC_ALARM_A

RTC_ALARM_B

RTC Alarm Sub Seconds Masks Definitions

RTC_ALARMSUBSECONDMASK_ALL

< All Alarm SS fields are masked. There is no comparison on sub seconds for Alarm SS[14:1] are don't care in Alarm comparison. Only SS[0] is compared.

RTC_ALARMSUBSECONDMASK_SS14_1

SS[14:2] are don't care in Alarm comparison. Only SS[1:0] are compared.

RTC_ALARMSUBSECONDMASK_SS14_2

SS[14:3] are don't care in Alarm comparison. Only SS[2:0] are compared.

RTC_ALARMSUBSECONDMASK_SS14_3

SS[14:4] are don't care in Alarm comparison. Only SS[3:0] are compared.

RTC_ALARMSUBSECONDMASK_SS14_4

SS[14:5] are don't care in Alarm comparison. Only SS[4:0] are compared.

RTC_ALARMSUBSECONDMASK_SS14_5

SS[14:6] are don't care in Alarm comparison. Only SS[5:0] are compared.

RTC_ALARMSUBSECONDMASK_SS14_6

SS[14:7] are don't care in Alarm comparison. Only SS[6:0] are compared.

RTC_ALARMSUBSECONDMASK_SS14_7

SS[14:8] are don't care in Alarm comparison. Only SS[7:0] are compared.

RTC_ALARMSUBSECONDMASK_SS14_8

SS[14:9] are don't care in Alarm comparison. Only SS[8:0] are compared.

RTC_ALARMSUBSECONDMASK_SS14_9

SS[14:10] are don't care in Alarm comparison. Only SS[9:0] are compared.

RTC_ALARMSUBSECONDMASK_SS14_10

SS[14:11] are don't care in Alarm comparison. Only SS[10:0] are compared.

RTC_ALARMSUBSECONDMASK_SS14_11

SS[14:12] are don't care in Alarm comparison. Only SS[11:0] are compared.

RTC_ALARMSUBSECONDMASK_SS14_12

SS[14:13] are don't care in Alarm comparison. Only SS[12:0] are compared.

RTC_ALARMSUBSECONDMASK_SS14_13

SS[14] is don't care in Alarm comparison. Only SS[13:0] are compared.

RTC_ALARMSUBSECONDMASK_SS14

SS[14:0] are compared and must match to activate alarm.

RTC_ALARMSUBSECONDMASK_NONE

RTC AM PM Definitions

RTC_HOURFORMAT12_AM

RTC_HOURFORMAT12_PM

RTC DayLight Saving Definitions

RTC_DAYLIGHTSAVING_SUB1H

RTC_DAYLIGHTSAVING_ADD1H

RTC_DAYLIGHTSAVING_NONE

RTC Exported Macros

__HAL_RTC_RESET_HANDLE_STATE

Description:

- Reset RTC handle state.

Parameters:

- __HANDLE__: RTC handle.

Return value:

- None

__HAL_RTC_WRITEPROTECTION_DISABLE

Description:

- Disable the write protection for RTC registers.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_WRITEPROTECTION_ENABLE

Description:

- Enable the write protection for RTC registers.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_DAYLIGHT_SAVING_TIME_ADD1H

Description:

- Add 1 hour (summer time change).

Parameters:

- __HANDLE__: specifies the RTC handle.
- __BKP__: Backup This parameter can be:
  - RTC_STOREOPERATION_RESET
  - RTC_STOREOPERATION_SET

Return value:

- None

__HAL_RTC_DAYLIGHT_SAVING_TIME_SUB1H

Description:

- Subtract 1 hour (winter time change).

Parameters:

- __HANDLE__: specifies the RTC handle.
- __BKP__: Backup This parameter can be:
  - RTC_STOREOPERATION_RESET
  - RTC_STOREOPERATION_SET

Return value:

- None

__HAL_RTC_ALARMA_ENABLE

Description:

- Enable the RTC ALARMA peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_ALARMA_DISABLE

Description:

- Disable the RTC ALARMA peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_ALARMB_ENABLE

Description:

- Enable the RTC ALARMB peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_ALARMB_DISABLE

Description:

- Disable the RTC ALARMB peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_ALARM_ENABLE_IT

Description:

- Enable the RTC Alarm interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - RTC_IT_ALRA Alarm A interrupt
  - RTC_IT_ALRB Alarm B interrupt

Return value:

- None

__HAL_RTC_ALARM_DISABLE_IT

Description:

- Disable the RTC Alarm interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - RTC_IT_ALRA Alarm A interrupt
  - RTC_IT_ALRB Alarm B interrupt

Return value:

- None

__HAL_RTC_ALARM_GET_IT

Description:

- Check whether the specified RTC Alarm interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to check. This parameter can be:
  - RTC_IT_ALRA Alarm A interrupt
  - RTC_IT_ALRB Alarm B interrupt

Return value:

- None

__HAL_RTC_ALARM_GET_IT_SOURCE

Description:

- Check whether the specified RTC Alarm interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to check. This parameter can be:
  - RTC_IT_ALRA Alarm A interrupt
  - RTC_IT_ALRB Alarm B interrupt

Return value:

- None

__HAL_RTC_ALARM_GET_FLAG

Description:

- Get the selected RTC Alarm's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Alarm Flag sources to check. This parameter can be:
  - RTC_FLAG_ALRAF
  - RTC_FLAG_ALRBF
  - RTC_FLAG_ALRAWF
  - RTC_FLAG_ALRBWF

Return value:

- None

__HAL_RTC_ALARM_CLEAR_FLAG

Description:

- Clear the RTC Alarm's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Alarm Flag sources to clear. This parameter can be:
  - RTC_FLAG_ALRAF
  - RTC_FLAG_ALRBF

Return value:

- None

__HAL_RTC_ALARM_EXTI_ENABLE_IT

Description:

- Enable interrupt on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_DISABLE_IT

Description:

- Disable interrupt on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_ENABLE_EVENT

Description:

- Enable event on the RTC Alarm associated Exti line.

Return value:

- None.

__HAL_RTC_ALARM_EXTI_DISABLE_EVENT

Description:

- Disable event on the RTC Alarm associated Exti line.

Return value:

- None.

__HAL_RTC_ALARM_EXTID2_ENABLE_IT

Description:

- Enable interrupt on the RTC Alarm associated D2 Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTID2_DISABLE_IT

Description:

- Disable interrupt on the RTC Alarm associated D2 Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTID2_ENABLE_EVENT

Description:

- Enable event on the RTC Alarm associated D2 Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTID2_DISABLE_EVENT

Description:

- Disable event on the RTC Alarm associated D2 Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_ENABLE_RISING_EDGE

Description:

- Enable rising edge trigger on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_DISABLE_RISING_EDGE

Description:

- Disable rising edge trigger on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable rising & falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable rising & falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_GET_FLAG

Description:

- Check whether the RTC Alarm associated Exti line interrupt flag is set or not.

Return value:

- Line: Status.

__HAL_RTC_ALARM_EXTI_CLEAR_FLAG

Description:

- Clear the RTC Alarm associated Exti line flag.

Return value:

- None.

__HAL_RTC_ALARM_EXTID2_GET_FLAG

Description:

- Check whether the RTC Alarm associated D2 Exti line interrupt flag is set or not.

Return value:

- Line: Status

__HAL_RTC_ALARM_EXTID2_CLEAR_FLAG

Description:

- Clear the RTC Alarm associated D2 Exti line flag.

Return value:

- None

__HAL_RTC_ALARM_EXTI_GENERATE_SWIT

Description:

- Generate a Software interrupt on RTC Alarm associated Exti line.

Return value:

- None

RTC Flags Definitions

RTC_FLAG_RECALPF

RTC_FLAG_TSOVF

RTC_FLAG_TSF

RTC_FLAG_ITSF

RTC_FLAG_WUTF

RTC_FLAG_ALRBF

RTC_FLAG_ALRAF

RTC_FLAG_INITF

RTC_FLAG_RSF

RTC_FLAG_INITS

RTC_FLAG_SHPF

RTC_FLAG_WUTWF

RTC_FLAG_ALRBWF

RTC_FLAG_ALRAWF

RTC Hour Formats Definitions

RTC_HOURFORMAT_24

RTC_HOURFORMAT_12

RTC Input Parameter Format Definitions

RTC_FORMAT_BIN

RTC_FORMAT_BCD

RTC Interrupts Definitions

RTC_IT_TS

Enable Timestamp Interrupt

RTC_IT_WUT

Enable Wakeup timer Interrupt

RTC_IT_ALRA

Enable Alarm A Interrupt

RTC_IT_ALRB

Enable Alarm B Interrupt

RTC Private macros to check input parameters

IS_RTC_OUTPUT

IS_RTC_HOUR_FORMAT

IS_RTC_OUTPUT_POL

IS_RTC_OUTPUT_TYPE

IS_RTC_OUTPUT_REMAP

IS_RTC_HOURFORMAT12

IS_RTC_DAYLIGHT_SAVING

IS_RTC_STORE_OPERATION

IS_RTC_FORMAT

IS_RTC_YEAR

IS_RTC_MONTH

IS_RTC_DATE

IS_RTC_WEEKDAY

IS_RTC_ALARM_DATE_WEEKDAY_DATE

IS_RTC_ALARM_DATE_WEEKDAY_WEEKDAY

IS_RTC_ALARM_DATE_WEEKDAY_SEL

IS_RTC_ALARM_MASK

IS_RTC_ALARM

IS_RTC_ALARM_SUB_SECOND_VALUE

IS_RTC_ALARM_SUB_SECOND_MASK

IS_RTC_ASYNCH_PREDIV

IS_RTC_SYNCH_PREDIV

IS_RTC_HOUR12

IS_RTC_HOUR24

IS_RTC_MINUTES

IS_RTC_SECONDS

RTC Month Date Definitions (in BCD format)

RTC_MONTH_JANUARY

RTC_MONTH_FEBRUARY

RTC_MONTH_MARCH

RTC_MONTH_APRIL

RTC_MONTH_MAY

RTC_MONTH_JUNE

RTC_MONTH_JULY

RTC_MONTH_AUGUST

RTC_MONTH_SEPTEMBER

RTC_MONTH_OCTOBER

RTC_MONTH_NOVEMBER

RTC_MONTH_DECEMBER

RTC Output ALARM OUT Remap

RTC_OUTPUT_REMAP_NONE

RTC_OUTPUT_REMAP_POS1

RTC Output Polarity Definitions

RTC_OUTPUT_POLARITY_HIGH

RTC_OUTPUT_POLARITY_LOW

RTC Output Selection Definitions

RTC_OUTPUT_DISABLE

RTC_OUTPUT_ALARMA

RTC_OUTPUT_ALARMB

RTC_OUTPUT_WAKEUP

RTC Output Type ALARM OUT

RTC_OUTPUT_TYPE_PUSHPULL

RTC_OUTPUT_TYPE_OPENDRAIN

RTC_ALARM_OUTPUT_TYPE

RTC Store Operation Definitions

RTC_STOREOPERATION_RESET

RTC_STOREOPERATION_SET

RTC Tamper Flags Definitions

RTC_FLAG_TAMP1F

RTC_FLAG_TAMP2F

RTC_FLAG_TAMP3F

RTC WeekDay Definitions

RTC_WEEKDAY_MONDAY

RTC_WEEKDAY_TUESDAY

RTC_WEEKDAY_WEDNESDAY

RTC_WEEKDAY_THURSDAY

RTC_WEEKDAY_FRIDAY

RTC_WEEKDAY_SATURDAY

RTC_WEEKDAY_SUNDAY

73 HAL RTC Extension Driver

73.1 RTCEx Firmware driver registers structures

73.1.1 RTC_TamperTypeDef

RTC_TamperTypeDef is defined in the stm32h7xx_hal_rtc_ex.h

Data Fields

- ***uint32_t Tamper***
- ***uint32_t Interrupt***
- ***uint32_t Trigger***
- ***uint32_t NoErase***
- ***uint32_t MaskFlag***
- ***uint32_t Filter***
- ***uint32_t SamplingFrequency***
- ***uint32_t PrechargeDuration***
- ***uint32_t TamperPullUp***
- ***uint32_t TimeStampOnTamperDetection***

Field Documentation

- ***uint32_t RTC_TamperTypeDef::Tamper***
  Specifies the Tamper Pin. This parameter can be a value of ***RTCEx_Tamper_Pins_Definitions***
- ***uint32_t RTC_TamperTypeDef::Interrupt***
  Specifies the Tamper Interrupt. This parameter can be a value of ***RTCEx_Tamper_Interrupt_Definitions***
- ***uint32_t RTC_TamperTypeDef::Trigger***
  Specifies the Tamper Trigger. This parameter can be a value of ***RTCEx_Tamper_Trigger_Definitions***
- ***uint32_t RTC_TamperTypeDef::NoErase***
  Specifies the Tamper no erase mode. This parameter can be a value of ***RTCEx_Tamper_EraseBackUp_Definitions***
- ***uint32_t RTC_TamperTypeDef::MaskFlag***
  Specifies the Tamper Flag masking. This parameter can be a value of ***RTCEx_Tamper_MaskFlag_Definitions***
- ***uint32_t RTC_TamperTypeDef::Filter***
  Specifies the TAMP Filter Tamper. This parameter can be a value of ***RTCEx_Tamper_Filter_Definitions***
- ***uint32_t RTC_TamperTypeDef::SamplingFrequency***
  Specifies the sampling frequency. This parameter can be a value of ***RTCEx_Tamper_Sampling_Frequencies_Definitions***
- ***uint32_t RTC_TamperTypeDef::PrechargeDuration***
  Specifies the Precharge Duration . This parameter can be a value of ***RTCEx_Tamper_Pin_Precharge_Duration_Definitions***
- ***uint32_t RTC_TamperTypeDef::TamperPullUp***
  Specifies the Tamper PullUp . This parameter can be a value of ***RTCEx_Tamper_Pull_UP_Definitions***
- ***uint32_t RTC_TamperTypeDef::TimeStampOnTamperDetection***
  Specifies the TimeStampOnTamperDetection. This parameter can be a value of ***RTCEx_Tamper_TimeStampOnTamperDetection_Definitions***

73.2 RTCEx Firmware driver API description

The following section lists the various functions of the RTCEx library.

73.2.1 How to use this driver

- Enable the RTC domain access.
- Configure the RTC Prescaler (Asynchronous and Synchronous) and RTC hour format using the HAL_RTC_Init() function.

RTC Wakeup configuration

- To configure the RTC Wakeup Clock source and Counter use the HAL_RTCEx_SetWakeUpTimer() function. You can also configure the RTC Wakeup timer with interrupt mode using the HAL_RTCEx_SetWakeUpTimer_IT() function.
- To read the RTC WakeUp Counter register, use the HAL_RTCEx_GetWakeUpTimer() function.

Outputs configuration

The RTC has 2 different outputs:

- RTC_ALARM: this output is used to manage the RTC Alarm A, Alarm B and WaKeUp signals. To output the selected RTC signal, use the HAL_RTC_Init() function.
- RTC_CALIB: this output is 512Hz signal or 1Hz. To enable the RTC_CALIB, use the HAL_RTCEx_SetCalibrationOutPut() function.
- Two pins can be used as RTC_ALARM or RTC_CALIB (PC13, PB2) managed on the RTC_OR register.
- When the RTC_CALIB or RTC_ALARM output is selected, the RTC_OUT pin is automatically configured in output alternate function.

Smooth digital Calibration configuration

- Configure the RTC Original Digital Calibration Value and the corresponding calibration cycle period (32s,16s and 8s) using the HAL_RTCEx_SetSmoothCalib() function.

TimeStamp configuration

- Enable the RTC TimeStamp using the HAL_RTCEx_SetTimeStamp() function. You can also configure the RTC TimeStamp with interrupt mode using the HAL_RTCEx_SetTimeStamp_IT() function.
- To read the RTC TimeStamp Time and Date register, use the HAL_RTCEx_GetTimeStamp() function.

Internal TimeStamp configuration

- Enable the RTC internal TimeStamp using the HAL_RTCEx_SetInternalTimeStamp() function. User has to check internal timestamp occurrence using __HAL_RTC_INTERNAL_TIMESTAMP_GET_FLAG.
- To read the RTC TimeStamp Time and Date register, use the HAL_RTCEx_GetTimeStamp() function.

Tamper configuration

- Enable the RTC Tamper and configure the Tamper filter count, trigger Edge or Level according to the Tamper filter (if equal to 0 Edge else Level) value, sampling frequency, NoErase, MaskFlag, precharge or discharge and Pull-UP using the HAL_RTCEx_SetTamper() function. You can configure RTC Tamper with interrupt mode using HAL_RTCEx_SetTamper_IT() function.
- The default configuration of the Tamper erases the backup registers. To avoid erase, enable the NoErase field on the RTC_TAMPCR register.

Backup Data Registers configuration

- To write to the RTC Backup Data registers, use the HAL_RTCEx_BKUPWrite() function.
- To read the RTC Backup Data registers, use the HAL_RTCEx_BKUPRead() function.

73.2.2 RTC TimeStamp and Tamper functions

This section provides functions allowing to configure TimeStamp feature

This section contains the following APIs:

- ***HAL_RTCEx_SetTimeStamp()***
- ***HAL_RTCEx_SetTimeStamp_IT()***

- *HAL_RTCEx_DeactivateTimeStamp()*
- *HAL_RTCEx_SetInternalTimeStamp()*
- *HAL_RTCEx_DeactivateInternalTimeStamp()*
- *HAL_RTCEx_GetTimeStamp()*
- *HAL_RTCEx_TamperTimeStampIRQHandler()*
- *HAL_RTCEx_TimeStampEventCallback()*
- *HAL_RTCEx_PollForTimeStampEvent()*

73.2.3 Tamper functions

- Before calling any tamper or internal tamper function, you have to call first HAL_RTC_Init() function.
- In that ine you can select to output tamper event on RTC pin.
- Enable the Tamper and configure the Tamper filter count, trigger Edge or Level according to the Tamper filter (if equal to 0 Edge else Level) value, sampling frequency, NoErase, MaskFlag, precharge or discharge and Pull-UP, timestamp using the HAL_RTCEx_SetTamper() function. You can configure Tamper with interrupt mode using HAL_RTCEx_SetTamper_IT() function.
- The default configuration of the Tamper erases the backup registers. To avoid erase, enable the NoErase field on the TAMP_TAMPCR register.
- Enable Internal Tamper and configure it with interrupt, timestamp using the HAL_RTCEx_SetInternalTamper() function.

This section contains the following APIs:

- *HAL_RTCEx_SetTamper()*
- *HAL_RTCEx_SetTamper_IT()*
- *HAL_RTCEx_DeactivateTamper()*
- *HAL_RTCEx_Tamper1EventCallback()*
- *HAL_RTCEx_Tamper2EventCallback()*
- *HAL_RTCEx_Tamper3EventCallback()*
- *HAL_RTCEx_PollForTamper1Event()*
- *HAL_RTCEx_PollForTamper2Event()*
- *HAL_RTCEx_PollForTamper3Event()*

73.2.4 RTC Wake-up functions

This section provides functions allowing to configure Wake-up feature

This section contains the following APIs:

- *HAL_RTCEx_SetWakeUpTimer()*
- *HAL_RTCEx_SetWakeUpTimer_IT()*
- *HAL_RTCEx_DeactivateWakeUpTimer()*
- *HAL_RTCEx_GetWakeUpTimer()*
- *HAL_RTCEx_WakeUpTimerIRQHandler()*
- *HAL_RTCEx_WakeUpTimerEventCallback()*
- *HAL_RTCEx_PollForWakeUpTimerEvent()*

73.2.5 Extended RTC Backup register functions

- Before calling any tamper or internal tamper function, you have to call first HAL_RTC_Init() function.
- In that ine you can select to output tamper event on RTC pin.

This subsection provides functions allowing to

- Write a data in a specified RTC Backup data register
- Read a data in a specified RTC Backup data register

This section contains the following APIs:

- *HAL_RTCEx_BKUPWrite()*

- *HAL_RTCEx_BKUPRead()*

73.2.6 Extended Peripheral Control functions

This subsection provides functions allowing to

- Write a data in a specified RTC Backup data register
- Read a data in a specified RTC Backup data register
- Set the Smooth calibration parameters.
- Set Low Power calibration parameter (if feature supported).
- Configure the Synchronization Shift Control Settings.
- Configure the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
- Deactivate the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
- Enable the RTC reference clock detection.
- Disable the RTC reference clock detection.
- Enable the Bypass Shadow feature.
- Disable the Bypass Shadow feature.

This section contains the following APIs:

- *HAL_RTCEx_SetSmoothCalib()*
- *HAL_RTCEx_SetSynchroShift()*
- *HAL_RTCEx_SetCalibrationOutPut()*
- *HAL_RTCEx_DeactivateCalibrationOutPut()*
- *HAL_RTCEx_SetRefClock()*
- *HAL_RTCEx_DeactivateRefClock()*
- *HAL_RTCEx_EnableBypassShadow()*
- *HAL_RTCEx_DisableBypassShadow()*

73.2.7 Extended features functions

This section provides functions allowing to:

- RTC Alarm B callback
- RTC Poll for Alarm B request

This section contains the following APIs:

- *HAL_RTCEx_AlarmBEventCallback()*
- *HAL_RTCEx_PollForAlarmBEvent()*

73.2.8 Detailed description of functions

HAL_RTCEx_SetTimeStamp

Function name

HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)

Function description

Set TimeStamp.

Parameters

- **hrtc:** RTC handle
- **TimeStampEdge:** Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following values:
  - RTC_TIMESTAMPEDGE_RISING: the Time stamp event occurs on the rising edge of the related pin.
  - RTC_TIMESTAMPEDGE_FALLING: the Time stamp event occurs on the falling edge of the related pin.
- **RTC_TimeStampPin:** specifies the RTC TimeStamp Pin. This parameter can be one of the following values:
  - RTC_TIMESTAMPPIN_DEFAULT: PC13 is selected as RTC TimeStamp Pin. The RTC TimeStamp Pin is per default PC13, but for reasons of compatibility, this parameter is required.

Return values

- **HAL:** status

Notes

- This API must be called before enabling the TimeStamp feature.

HAL_RTCEx_SetTimeStamp_IT

Function name

HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp_IT (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)

Function description

Set TimeStamp with Interrupt.

Parameters

- **hrtc:** RTC handle
- **TimeStampEdge:** Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following values:
  - RTC_TIMESTAMPEDGE_RISING: the Time stamp event occurs on the rising edge of the related pin.
  - RTC_TIMESTAMPEDGE_FALLING: the Time stamp event occurs on the falling edge of the related pin.
- **RTC_TimeStampPin:** Specifies the RTC TimeStamp Pin. This parameter can be one of the following values:
  - RTC_TIMESTAMPPIN_DEFAULT: PC13 is selected as RTC TimeStamp Pin. The RTC TimeStamp Pin is per default PC13, but for reasons of compatibility, this parameter is required.

Return values

- **HAL:** status

Notes

- This API must be called before enabling the TimeStamp feature.

HAL_RTCEx_DeactivateTimeStamp

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateTimeStamp (RTC_HandleTypeDef * hrtc)

Function description

Deactivate TimeStamp.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_SetInternalTimeStamp

Function name

HAL_StatusTypeDef HAL_RTCEx_SetInternalTimeStamp (RTC_HandleTypeDef * hrtc)

Function description

Set Internal TimeStamp.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- This API must be called before enabling the internal TimeStamp feature.

HAL_RTCEx_DeactivateInternalTimeStamp

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateInternalTimeStamp (RTC_HandleTypeDef * hrtc)

Function description

Deactivate Internal TimeStamp.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_GetTimeStamp

Function name

HAL_StatusTypeDef HAL_RTCEx_GetTimeStamp (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTimeStamp, RTC_DateTypeDef * sTimeStampDate, uint32_t Format)

Function description

Get the RTC TimeStamp value.

Parameters

- **hrtc:** RTC handle
- **sTimeStamp:** Pointer to Time structure
- **sTimeStampDate:** Pointer to Date structure
- **Format:** specifies the format of the entered parameters. This parameter can be one of the following values:
  - RTC_FORMAT_BIN: Binary data format
  - RTC_FORMAT_BCD: BCD data format

Return values

- **HAL:** status

HAL_RTCEx_TamperTimeStampIRQHandler

Function name

void HAL_RTCEx_TamperTimeStampIRQHandler (RTC_HandleTypeDef * hrtc)

Function description

Handle Tamper and TimeStamp interrupt request.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_TimeStampEventCallback

Function name

void HAL_RTCEx_TimeStampEventCallback (RTC_HandleTypeDef * hrtc)

Function description

TimeStamp callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_PollForTimeStampEvent

Function name

HAL_StatusTypeDef HAL_RTCEx_PollForTimeStampEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)

Function description

Handle TimeStamp polling request.

Parameters

- **hrtc:** RTC handle
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTCEx_SetTamper

Function name

HAL_StatusTypeDef HAL_RTCEx_SetTamper (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)

Function description

Set Tamper.

Parameters

- **hrtc:** RTC handle
- **sTamper:** Pointer to Tamper Structure.

Return values

- **HAL:** status

Notes

- By calling this API we disable the tamper interrupt for all tampers.

HAL_RTCEx_SetTamper_IT

Function name

HAL_StatusTypeDef HAL_RTCEx_SetTamper_IT (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)

Function description

Set Tamper with interrupt.

Parameters

- **hrtc:** RTC handle
- **sTamper:** Pointer to Tamper Structure.

Return values

- **HAL:** status

Notes

- By calling this API we force the tamper interrupt for all tampers.

HAL_RTCEx_DeactivateTamper

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateTamper (RTC_HandleTypeDef * hrtc, uint32_t Tamper)

Function description

Deactivate Tamper.

Parameters

- **hrtc:** RTC handle
- **Tamper:** Selected tamper pin. This parameter can be any combination of the following values:
  - RTC_TAMPER_1
  - RTC_TAMPER_2
  - RTC_TAMPER_3

Return values

- **HAL:** status

HAL_RTCEx_PollForTamper1Event

Function name

HAL_StatusTypeDef HAL_RTCEx_PollForTamper1Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)

Function description

Handle Tamper1 Polling.

Parameters

- **hrtc:** RTC handle
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTCEx_PollForTamper2Event

Function name

HAL_StatusTypeDef HAL_RTCEx_PollForTamper2Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)

Function description

Handle Tamper2 Polling.

Parameters

- **hrtc:** RTC handle
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTCEx_PollForTamper3Event

Function name

HAL_StatusTypeDef HAL_RTCEx_PollForTamper3Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)

Function description

Handle Tamper3 Polling.

Parameters

- **hrtc:** RTC handle
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTCEx_Tamper1EventCallback

Function name

void HAL_RTCEx_Tamper1EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Tamper 1 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_Tamper2EventCallback

Function name

void HAL_RTCEx_Tamper2EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Tamper 2 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_Tamper3EventCallback

Function name

void HAL_RTCEx_Tamper3EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Tamper 3 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_SetWakeUpTimer

Function name

HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock)

Function description

Set wake up timer.

Parameters

- **hrtc:** RTC handle
- **WakeUpCounter:** Wake up counter
- **WakeUpClock:** Wake up clock

Return values

- **HAL:** status

HAL_RTCEx_SetWakeUpTimer_IT

Function name

HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer_IT (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock)

Function description

Set wake up timer with interrupt.

Parameters

- **hrtc:** RTC handle
- **WakeUpCounter:** Wake up counter
- **WakeUpClock:** Wake up clock

Return values

- **HAL:** status

HAL_RTCEx_DeactivateWakeUpTimer

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateWakeUpTimer (RTC_HandleTypeDef * hrtc)

Function description

Deactivate wake up timer counter.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_GetWakeUpTimer

Function name

uint32_t HAL_RTCEx_GetWakeUpTimer (RTC_HandleTypeDef * hrtc)

Function description

Get wake up timer counter.

Parameters

- **hrtc:** RTC handle

Return values

- **Counter:** value

HAL_RTCEx_WakeUpTimerIRQHandler

Function name

void HAL_RTCEx_WakeUpTimerIRQHandler (RTC_HandleTypeDef * hrtc)

Function description

Handle Wake Up Timer interrupt request.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_WakeUpTimerEventCallback

Function name

void HAL_RTCEx_WakeUpTimerEventCallback (RTC_HandleTypeDef * hrtc)

Function description

Wake Up Timer callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_PollForWakeUpTimerEvent

Function name

HAL_StatusTypeDef HAL_RTCEx_PollForWakeUpTimerEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)

Function description

Handle Wake Up Timer Polling.

Parameters

- **hrtc:** RTC handle
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTCEx_BKUPWrite

Function name

void HAL_RTCEx_BKUPWrite (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister, uint32_t Data)

Function description

Write a data in a specified RTC Backup data register.

Parameters

- **hrtc:** RTC handle
- **BackupRegister:** RTC Backup data Register number. This parameter can be: RTC_BKP_DRx where x can be from 0 to 31 to specify the register.
- **Data:** Data to be written in the specified Backup data register.

Return values

- **None:**

HAL_RTCEx_BKUPRead

Function name

uint32_t HAL_RTCEx_BKUPRead (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister)

Function description

Read data from the specified RTC Backup data Register.

Parameters

- **hrtc:** RTC handle
- **BackupRegister:** RTC Backup data Register number. This parameter can be: RTC_BKP_DRx where x can be from 0 to 31 to specify the register.

Return values

- **Read:** value

HAL_RTCEx_SetSmoothCalib

Function name

HAL_StatusTypeDef HAL_RTCEx_SetSmoothCalib (RTC_HandleTypeDef * hrtc, uint32_t SmoothCalibPeriod, uint32_t SmoothCalibPlusPulses, uint32_t SmoothCalibMinusPulsesValue)

Function description

Set the Smooth calibration parameters.

Parameters

- **hrtc:** RTC handle
- **SmoothCalibPeriod:** Select the Smooth Calibration Period. This parameter can be can be one of the following values :
  - RTC_SMOOTHCALIB_PERIOD_32SEC: The smooth calibration period is 32s.
  - RTC_SMOOTHCALIB_PERIOD_16SEC: The smooth calibration period is 16s.
  - RTC_SMOOTHCALIB_PERIOD_8SEC: The smooth calibration period is 8s.
- **SmoothCalibPlusPulses:** Select to Set or reset the CALP bit. This parameter can be one of the following values:
  - RTC_SMOOTHCALIB_PLUSPULSES_SET: Add one RTCCLK pulse every 2*11 pulses.
  - RTC_SMOOTHCALIB_PLUSPULSES_RESET: No RTCCLK pulses are added.
- **SmoothCalibMinusPulsesValue:** Select the value of CALM[8:0] bits. This parameter can be one any value from 0 to 0x000001FF.

Return values

- **HAL:** status

Notes

- To deactivate the smooth calibration, the field SmoothCalibPlusPulses must be equal to SMOOTHCALIB_PLUSPULSES_RESET and the field SmoothCalibMinusPulsesValue must be equal to 0.

HAL_RTCEx_SetSynchroShift

Function name

HAL_StatusTypeDef HAL_RTCEx_SetSynchroShift (RTC_HandleTypeDef * hrtc, uint32_t ShiftAdd1S, uint32_t ShiftSubFS)

Function description

Configure the Synchronization Shift Control Settings.

Parameters

- **hrtc:** RTC handle
- **ShiftAdd1S:** Select to add or not 1 second to the time calendar. This parameter can be one of the following values:
  - RTC_SHIFTADD1S_SET: Add one second to the clock calendar.
  - RTC_SHIFTADD1S_RESET: No effect.
- **ShiftSubFS:** Select the number of Second Fractions to substitute. This parameter can be one any value from 0 to 0x7FFF.

Return values

- **HAL:** status

Notes

- When REFCKON is set, firmware must not write to Shift control register.

HAL_RTCEx_SetCalibrationOutPut

Function name

HAL_StatusTypeDef HAL_RTCEx_SetCalibrationOutPut (RTC_HandleTypeDef * hrtc, uint32_t CalibOutput)

Function description

Configure the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).

Parameters

- **hrtc:** RTC handle
- **CalibOutput:** Select the Calibration output Selection. This parameter can be one of the following values:
  - RTC_CALIBOUTPUT_512HZ: A signal has a regular waveform at 512Hz.
  - RTC_CALIBOUTPUT_1HZ: A signal has a regular waveform at 1Hz.

Return values

- **HAL:** status

HAL_RTCEx_DeactivateCalibrationOutPut

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateCalibrationOutPut (RTC_HandleTypeDef * hrtc)

Function description

Deactivate the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_SetRefClock

Function name

HAL_StatusTypeDef HAL_RTCEx_SetRefClock (RTC_HandleTypeDef * hrtc)

Function description

Enable the RTC reference clock detection.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_DeactivateRefClock

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateRefClock (RTC_HandleTypeDef * hrtc)

Function description

Disable the RTC reference clock detection.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_EnableBypassShadow

Function name

HAL_StatusTypeDef HAL_RTCEx_EnableBypassShadow (RTC_HandleTypeDef * hrtc)

Function description

Enable the Bypass Shadow feature.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

HAL_RTCEx_DisableBypassShadow

Function name

HAL_StatusTypeDef HAL_RTCEx_DisableBypassShadow (RTC_HandleTypeDef * hrtc)

Function description

Disable the Bypass Shadow feature.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

HAL_RTCEx_AlarmBEventCallback

Function name

void HAL_RTCEx_AlarmBEventCallback (RTC_HandleTypeDef * hrtc)

Function description

Alarm B callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_PollForAlarmBEvent

Function name

HAL_StatusTypeDef HAL_RTCEx_PollForAlarmBEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)

Function description

Handle Alarm B Polling request.

Parameters

- **hrtc:** RTC handle
- **Timeout:** Timeout duration

Return values

- **HAL:** status

73.3 RTCEx Firmware driver defines

The following section lists the various define and macros of the module.

73.3.1 RTCEx

RTCEx

RTC Active_Tamper_Asynchronous_Prescaler clock Definitions

RTC_ATAMP_ASYNCPRES_RTCCLK

RTCCLK

RTC_ATAMP_ASYNCPRES_RTCCLK_2

RTCCLK/2

RTC_ATAMP_ASYNCPRES_RTCCLK_4

RTCCLK/4

RTC_ATAMP_ASYNCPRES_RTCCLK_8

RTCCLK/8

RTC_ATAMP_ASYNCPRES_RTCCLK_16

RTCCLK/16

RTC_ATAMP_ASYNCPRES_RTCCLK_32

RTCCLK/32

RTC_ATAMP_ASYNCPRES_RTCCLK_64

RTCCLK/64

RTC_ATAMP_ASYNCPRES_RTCCLK_128

RTCCLK/128

RTCEx_ActiveTamper_Enable Definitions

RTC_ATAMP_ENABLE

RTC_ATAMP_DISABLE

RTCEx_ActiveTamper_Filter Definitions

RTC_ATAMP_FILTER_ENABLE

RTC_ATAMP_FILTER_DISABLE

RTCEx_ActiveTamper_Interrupt Definitions

RTC_ATAMP_INTERRUPT_ENABLE

RTC_ATAMP_INTERRUPT_DISABLE

RTC Active Tamper selection Definition

RTC_ATAMP_1

Tamper 1

RTC_ATAMP_2

Tamper 2

RTC_ATAMP_3

Tamper 3

RTC_ATAMP_4

Tamper 4

RTC_ATAMP_5

Tamper 5

RTC_ATAMP_6

Tamper 6

RTC_ATAMP_7

Tamper 7

RTC_ATAMP_8

Tamper 8

RTC Add 1 Second Parameter Definitions

RTC_SHIFTADD1S_RESET

RTC_SHIFTADD1S_SET

RTC Backup Registers Definitions

RTC_BKP_DR0

RTC_BKP_DR1

RTC_BKP_DR2

RTC_BKP_DR3

RTC_BKP_DR4

RTC_BKP_DR5

RTC_BKP_DR6

RTC_BKP_DR7

RTC_BKP_DR8

RTC_BKP_DR9

RTC_BKP_DR10

RTC_BKP_DR11

RTC_BKP_DR12

RTC_BKP_DR13

RTC_BKP_DR14

RTC_BKP_DR15

RTC_BKP_DR16

RTC_BKP_DR17

RTC_BKP_DR18

RTC_BKP_DR19

RTC_BKP_DR20

RTC_BKP_DR21

RTC_BKP_DR22

RTC_BKP_DR23

RTC_BKP_DR24

RTC_BKP_DR25

RTC_BKP_DR26

RTC_BKP_DR27

RTC_BKP_DR28

RTC_BKP_DR29

RTC_BKP_DR30

RTC_BKP_DR31

RTC Backup Registers Number Definitions

BKP_REG_NUMBER

RTC Calib Output Selection Definitions

RTC_CALIBOUTPUT_512HZ

RTC_CALIBOUTPUT_1HZ

RTC Exported Macros

__HAL_RTC_WAKEUPTIMER_ENABLE

Description:

- Enable the RTC WakeUp Timer peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_DISABLE

Description:

- Disable the RTC WakeUp Timer peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_ENABLE_IT

Description:

- Enable the RTC WakeUpTimer interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC WakeUpTimer interrupt sources to be enabled. This parameter can be:
  - RTC_IT_WUT WakeUpTimer interrupt

Return value:

- None

__HAL_RTC_WAKEUPTIMER_DISABLE_IT

Description:

- Disable the RTC WakeUpTimer interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC WakeUpTimer interrupt sources to be disabled. This parameter can be:
  - RTC_IT_WUT WakeUpTimer interrupt

Return value:

- None

__HAL_RTC_WAKEUPTIMER_GET_IT

Description:

- Check whether the specified RTC WakeUpTimer interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC WakeUpTimer interrupt sources to check. This parameter can be:
  - RTC_FLAG_WUTF WakeUpTimer interrupt flag

Return value:

- None

__HAL_RTC_WAKEUPTIMER_GET_IT_SOURCE

Description:

- Check whether the specified RTC Wake Up timer interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Wake Up timer interrupt sources to check. This parameter can be:
  - RTC_IT_WUT WakeUpTimer interrupt

Return value:

- None

__HAL_RTC_WAKEUPTIMER_GET_FLAG

Description:

- Get the selected RTC WakeUpTimer's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC WakeUpTimer Flag sources to check. This parameter can be:
  - RTC_FLAG_WUTF
  - RTC_FLAG_WUTWF

Return value:

- Flag: status

__HAL_RTC_WAKEUPTIMER_CLEAR_FLAG

Description:

- Clear the RTC Wake Up timer's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC WakeUpTimer Flag to clear. This parameter can be:
  - RTC_FLAG_WUTF

Return value:

- None

__HAL_RTC_TAMPER1_ENABLE

Description:

- Enable the RTC Tamper1 input detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPER1_DISABLE

Description:

- Disable the RTC Tamper1 input detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPER2_ENABLE

Description:

- Enable the RTC Tamper2 input detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPER2_DISABLE

Description:

- Disable the RTC Tamper2 input detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPER3_ENABLE

Description:

- Enable the RTC Tamper3 input detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPER3_DISABLE

Description:

- Disable the RTC Tamper3 input detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPER_ENABLE_IT

Description:

- Enable the RTC Tamper interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt sources to be enabled. This parameter can be any combination of the following values:
  - RTC_IT_TAMPALL: All tampers interrupts
  - RTC_IT_TAMP1: Tamper1 interrupt
  - RTC_IT_TAMP2: Tamper2 interrupt
  - RTC_IT_TAMP3: Tamper3 interrupt

Return value:

- None

__HAL_RTC_TAMPER_DISABLE_IT

Description:

- Disable the RTC Tamper interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt sources to be disabled. This parameter can be any combination of the following values:
  - RTC_IT_TAMP: All tampers interrupts
  - RTC_IT_TAMP1: Tamper1 interrupt
  - RTC_IT_TAMP2: Tamper2 interrupt
  - RTC_IT_TAMP3: Tamper3 interrupt

Return value:

- None

__HAL_RTC_TAMPER_GET_IT

Description:

- Check whether the specified RTC Tamper interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt to check. This parameter can be:
  - RTC_FLAG_TAMP1F: Tamper1 interrupt flag
  - RTC_FLAG_TAMP2F: Tamper2 interrupt flag
  - RTC_FLAG_TAMP3F: Tamper3 interrupt flag

Return value:

- Flag: status

__HAL_RTC_TAMPER_GET_IT_SOURCE

Description:

- Check whether the specified RTC Tamper interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt source to check. This parameter can be:
  - RTC_IT_TAMPALL: All tampers interrupts
  - RTC_IT_TAMP1: Tamper1 interrupt
  - RTC_IT_TAMP2: Tamper2 interrupt
  - RTC_IT_TAMP3: Tamper3 interrupt

Return value:

- Flag: status

__HAL_RTC_TAMPER_GET_FLAG

Description:

- Get the selected RTC Tamper's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Tamper Flag is pending or not. This parameter can be:
  - RTC_FLAG_TAMP1F: Tamper1 flag
  - RTC_FLAG_TAMP2F: Tamper2 flag
  - RTC_FLAG_TAMP3F: Tamper3 flag

Return value:

- Flag: status

__HAL_RTC_TAMPER_CLEAR_FLAG

Description:

- Clear the RTC Tamper's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Tamper Flag to clear. This parameter can be:
  - RTC_FLAG_TAMP1F: Tamper1 flag
  - RTC_FLAG_TAMP2F: Tamper2 flag
  - RTC_FLAG_TAMP3F: Tamper3 flag

Return value:

- None

__HAL_RTC_TAMPER_GET_SAMPLING_FREQ

Description:

- Get the frequency at which each of the Tamper inputs are sampled.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- Sampling: frequency This value can be:
  - RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV32768
  - RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV16384
  - RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV8192
  - RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV4096
  - RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV2048
  - RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV512
  - RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV256

__HAL_RTC_TAMPER_GET_SAMPLES_COUNT

Description:

- Get the number of consecutive samples at the specified level needed to activate a Tamper event.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- Number: of consecutive samples This value can be:
  - RTC_TAMPERFILTER_DISABLE
  - RTC_TAMPERFILTER_2SAMPLE
  - RTC_TAMPERFILTER_4SAMPLE
  - RTC_TAMPERFILTER_8SAMPLE

__HAL_RTC_TAMPER_GET_PRCHRG_DURATION

Description:

- Get the pull-up resistors precharge duration.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- Number: of consecutive samples This value can be:
  - RTC_TAMPERPRECHARGEDURATION_1RTCCLK
  - RTC_TAMPERPRECHARGEDURATION_2RTCCLK
  - RTC_TAMPERPRECHARGEDURATION_4RTCCLK
  - RTC_TAMPERPRECHARGEDURATION_8RTCCLK

__HAL_RTC_TAMPER_GET_PULLUP_STATUS

Description:

- Get the pull-up resistors status.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- Pull-up: resistors status This value can be:
  - RTC_TAMPER_PULLUP_ENABLE
  - RTC_TAMPER_PULLUP_DISABLE

__HAL_RTC_TIMESTAMP_ENABLE

Description:

- Enable the RTC TimeStamp peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TIMESTAMP_DISABLE

Description:

- Disable the RTC TimeStamp peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TIMESTAMP_ENABLE_IT

Description:

- Enable the RTC TimeStamp interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC TimeStamp interrupt source to be enabled. This parameter can be:
  - RTC_IT_TS TimeStamp interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_DISABLE_IT

Description:

- Disable the RTC TimeStamp interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC TimeStamp interrupt source to be disabled. This parameter can be:
  - RTC_IT_TS TimeStamp interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_GET_IT

Description:

- Check whether the specified RTC TimeStamp interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC TimeStamp interrupt source to check. This parameter can be:
  - RTC_IT_TS TimeStamp interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_GET_IT_SOURCE

Description:

- Check whether the specified RTC Time Stamp interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Time Stamp interrupt source to check. This parameter can be:
  - RTC_IT_TS TimeStamp interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_GET_FLAG

Description:

- Get the selected RTC TimeStamp's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC TimeStamp Flag is pending or not. This parameter can be:
  - RTC_FLAG_TSF
  - RTC_FLAG_TSOVF

Return value:

- Flag: status

__HAL_RTC_TIMESTAMP_CLEAR_FLAG

Description:

- Clear the RTC Time Stamp's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Alarm Flag sources to clear. This parameter can be:
  - RTC_FLAG_TSF
  - RTC_FLAG_TSOVF

Return value:

- None

__HAL_RTC_INTERNAL_TIMESTAMP_ENABLE

Description:

- Enable the RTC internal TimeStamp peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_INTERNAL_TIMESTAMP_DISABLE

Description:

- Disable the RTC internal TimeStamp peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_INTERNAL_TIMESTAMP_GET_FLAG

Description:

- Get the selected RTC Internal Time Stamp's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Internal Time Stamp Flag is pending or not. This parameter can be:
  - RTC_FLAG_ITSF

Return value:

- Flag: status

__HAL_RTC_INTERNAL_TIMESTAMP_CLEAR_FLAG

Description:

- Clear the RTC Internal Time Stamp's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Internal Time Stamp Flag source to clear. This parameter can be:
  - RTC_FLAG_ITSF

Return value:

- None

Notes:

- This flag must be cleared together with TSF flag.

__HAL_RTC_TAMPTS_ENABLE

Description:

- Enable the RTC TimeStamp on Tamper detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPTS_DISABLE

Description:

- Disable the RTC TimeStamp on Tamper detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPTS_GET_STATUS

Description:

- Get activation status of the RTC TimeStamp on Tamper detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- Activation: status of TimeStamp on Tamper detection This value can be:
  - RTC_TIMESTAMPONTAMPERDETECTION_ENABLE
  - RTC_TIMESTAMPONTAMPERDETECTION_DISABLE

__HAL_RTC_CALIBRATION_OUTPUT_ENABLE

Description:

- Enable the RTC calibration output.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_CALIBRATION_OUTPUT_DISABLE

Description:

- Disable the calibration output.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_CLOCKREF_DETECTION_ENABLE

Description:

- Enable the clock reference detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_CLOCKREF_DETECTION_DISABLE

Description:

- Disable the clock reference detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_SHIFT_GET_FLAG

Description:

- Get the selected RTC shift operation's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC shift operation Flag is pending or not. This parameter can be:
  - RTC_FLAG_SHPF

Return value:

- Flag: status

__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_IT

Description:

- Enable interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_IT

Description:

- Disable interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_EVENT

Description:

- Enable event on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_EVENT

Description:

- Disable event on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTID3_ENABLE_EVENT

Description:

- Enable event on the RTC WakeUp Timer associated D3 Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTID3_DISABLE_EVENT

Description:

- Disable event on the RTC WakeUp Timer associated D3 Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTID2_ENABLE_IT

Description:

- Enable interrupt on the RTC WakeUp Timer associated D2 Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTID2_DISABLE_IT

Description:

- Disable interrupt on the RTC WakeUp Timer associated D2 Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTID2_ENABLE_EVENT

Description:

- Enable event on the RTC WakeUp Timer associated D2 Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTID2_DISABLE_EVENT

Description:

- Disable event on the RTC WakeUp Timer associated D2 Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable falling edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable falling edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_RISING_EDGE

Description:

- Enable rising edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_RISING_EDGE

Description:

- Disable rising edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable rising & falling edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable rising & falling edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTI_GET_FLAG

Description:

- Check whether the RTC WakeUp Timer associated Exti line interrupt flag is set or not.

Return value:

- Line: Status.

__HAL_RTC_WAKEUPTIMER_EXTI_CLEAR_FLAG

Description:

- Clear the RTC WakeUp Timer associated Exti line flag.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_EXTID3_GET_FLAG

Description:

- Check whether the RTC WakeUp Timer associated D3 Exti line interrupt flag is set or not.

Return value:

- Line: Status

__HAL_RTC_WAKEUPTIMER_EXTID3_CLEAR_FLAG

Description:

- Clear the RTC WakeUp Timer associated D3 Exti line flag.

Return value:

- None.

__HAL_RTC_WAKEUPTIMER_EXTI_GENERATE_SWIT

Description:

- Generate a Software interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

__HAL_RTC_WAKEUPTIMER_EXTID2_GET_FLAG

Description:

- Check whether the RTC WakeUp Timer associated D2 Exti line interrupt flag is set or not.

Return value:

- Line: Status.

__HAL_RTC_WAKEUPTIMER_EXTID2_CLEAR_FLAG

Description:

- Clear the RTC WakeUp Timer associated D2 Exti line flag.

Return value:

- None.

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_IT

Description:

- Enable interrupt on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_IT

Description:

- Disable interrupt on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_EVENT

Description:

- Enable event on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_EVENT

Description:

- Disable event on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

__HAL_RTC_TAMPER_TIMESTAMP_EXTID2_ENABLE_IT

Description:

- Enable interrupt on the RTC Tamper and Timestamp associated D2 Exti line.

Return value:

- None

__HAL_RTC_TAMPER_TIMESTAMP_EXTID2_DISABLE_IT

Description:

- Disable interrupt on the RTC Tamper and Timestamp associated D2 Exti line.

Return value:

- None

__HAL_RTC_TAMPER_TIMESTAMP_EXTID2_ENABLE_EVENT

Description:

- Enable event on the RTC Tamper and Timestamp associated D2 Exti line.

Return value:

- None

__HAL_RTC_TAMPER_TIMESTAMP_EXTID2_DISABLE_EVENT

Description:

- Disable event on the RTC Tamper and Timestamp associated D2 Exti line.

Return value:

- None

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_RISING_EDGE

Description:

- Enable rising edge trigger on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_RISING_EDGE

Description:

- Disable rising edge trigger on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable rising & falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable rising & falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_GET_FLAG

Description:

- Check whether the RTC Tamper and Timestamp associated Exti line interrupt flag is set or not.

Return value:

- Line: Status

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_CLEAR_FLAG

Description:

- Clear the RTC Tamper and Timestamp associated Exti line flag.

Return value:

- None

__HAL_RTC_TAMPER_TIMESTAMP_EXTID2_GET_FLAG

Description:

- Check whether the RTC Tamper and Timestamp associated D2 Exti line interrupt flag is set or not.

Return value:

- Line: Status

__HAL_RTC_TAMPER_TIMESTAMP_EXTID2_CLEAR_FLAG

Description:

- Clear the RTC Tamper and Timestamp associated D2 Exti line flag.

Return value:

- None

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_GENERATE_SWIT

Description:

- Generate a Software interrupt on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

Private macros to check input parameters

IS_RTC_BKP

IS_TIMESTAMP_EDGE

IS_RTC_TIMESTAMP_PIN

IS_RTC_WAKEUP_CLOCK

IS_RTC_WAKEUP_COUNTER

IS_RTC_SMOOTH_CALIB_PERIOD

IS_RTC_SMOOTH_CALIB_PLUS

IS_RTC_SMOOTH_CALIB_MINUS

IS_RTC_SHIFT_ADD1S

IS_RTC_SHIFT_SUBFS

IS_RTC_CALIB_OUTPUT

IS_RTC_TAMPER

IS_RTC_TAMPER_INTERRUPT

IS_RTC_TAMPER_TRIGGER

IS_RTC_TAMPER_ERASE_MODE

IS_RTC_TAMPER_MASKFLAG_STATE

IS_RTC_TAMPER_FILTER

IS_RTC_TAMPER_SAMPLING_FREQ

IS_RTC_TAMPER_PRECHARGE_DURATION

IS_RTC_TAMPER_PULLUP_STATE

IS_RTC_TAMPER_TIMESTAMPONTAMPER_DETECTION

IS_RTC_TAMPER_FILTER_CONFIG_CORRECT

IS_RTC_INTERNAL_TAMPER

RTCEx Monotonic Counter Instance Definition

RTC_MONOTONIC_COUNTER_1

Monotonic counter 1

RTC Smooth Calib Period Definitions

RTC_SMOOTHCALIB_PERIOD_32SEC

If RTCCLK = 32768 Hz, Smooth calibration period is 32s, else 2exp20 RTCCLK pulses

RTC_SMOOTHCALIB_PERIOD_16SEC

If RTCCLK = 32768 Hz, Smooth calibration period is 16s, else 2exp19 RTCCLK pulses

RTC_SMOOTHCALIB_PERIOD_8SEC

If RTCCLK = 32768 Hz, Smooth calibration period is 8s, else 2exp18 RTCCLK pulses

RTC Smooth Calib Plus pulses Definitions

RTC_SMOOTHCALIB_PLUSPULSES_SET

The number of RTCCLK pulses added during a X -second window = Y - CALM[8:0] with Y = 512, 256, 128 when X = 32, 16, 8

RTC_SMOOTHCALIB_PLUSPULSES_RESET

The number of RTCCLK pulses subbstited during a 32-second window = CALM[8:0]

RTC Tamper EraseBackUp Definitions

RTC_TAMPER_ERASE_BACKUP_ENABLE

RTC_TAMPER_ERASE_BACKUP_DISABLE

RTC_DISABLE_BKP_ERASE_ON_TAMPER_1

RTC_DISABLE_BKP_ERASE_ON_TAMPER_2

RTC_DISABLE_BKP_ERASE_ON_TAMPER_3

RTC_DISABLE_BKP_ERASE_ON_TAMPER_MASK

RTC Tamper Filter Definitions

RTC_TAMPERFILTER_DISABLE

Tamper filter is disabled

RTC_TAMPERFILTER_2SAMPLE

Tamper is activated after 2 consecutive samples at the active level

RTC_TAMPERFILTER_4SAMPLE

Tamper is activated after 4 consecutive samples at the active level

RTC_TAMPERFILTER_8SAMPLE

Tamper is activated after 8 consecutive samples at the active level.

RTC_TAMPERFILTER_MASK

Masking all bits except those of field TAMPFLT[1:0].

RTC Tamper Interrupts Definitions

RTC_IT_TAMP1

Enable Tamper 1 Interrupt

RTC_IT_TAMP2

Enable Tamper 2 Interrupt

RTC_IT_TAMP3

Enable Tamper 3 Interrupt

RTC_IT_TAMP

Enable all Tamper Interrupts

RTC_IT_TAMPALL

RTC Tamper Mask Flag Definitions

RTC_TAMPERMASK_FLAG_DISABLE

RTC_TAMPERMASK_FLAG_ENABLE

RTC_TAMPER_1_MASK_FLAG

RTC_TAMPER_2_MASK_FLAG

RTC_TAMPER_3_MASK_FLAG

RTC_TAMPER_X_MASK_FLAG

RTC Tamper Pins Definitions

RTC_TAMPER_1

RTC_TAMPER_2

RTC_TAMPER_3

RTC_TAMPER_ALL

RTC Tamper Pin Precharge Duration Definitions

RTC_TAMPERPRECHARGEDURATION_1RTCCLK

Tamper pins are pre-charged before sampling during 1 RTCCLK cycle

RTC_TAMPERPRECHARGEDURATION_2RTCCLK

Tamper pins are pre-charged before sampling during 2 RTCCLK cycles

RTC_TAMPERPRECHARGEDURATION_4RTCCLK

Tamper pins are pre-charged before sampling during 4 RTCCLK cycles

RTC_TAMPERPRECHARGEDURATION_8RTCCLK

Tamper pins are pre-charged before sampling during 8 RTCCLK cycles

RTC_TAMPERPRECHARGEDURATION_MASK

Masking all bits except those of field TAMPPRCH[1:0]

RTC Tamper Pull Up Definitions

RTC_TAMPER_PULLUP_ENABLE

TimeStamp on Tamper Detection event saved

RTC_TAMPER_PULLUP_DISABLE

TimeStamp on Tamper Detection event is not saved

RTC_TAMPER_PULLUP_MASK

Maskin all bits except bit TAMPPUDIS

RTC Tamper Sampling Frequencies Definitions

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV32768

Each of the tamper inputs are sampled with a frequency = RTCCLK / 32768

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV16384

Each of the tamper inputs are sampled with a frequency = RTCCLK / 16384

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV8192

Each of the tamper inputs are sampled with a frequency = RTCCLK / 8192

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV4096

Each of the tamper inputs are sampled with a frequency = RTCCLK / 4096

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV2048

Each of the tamper inputs are sampled with a frequency = RTCCLK / 2048

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV1024

Each of the tamper inputs are sampled with a frequency = RTCCLK / 1024

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV512

Each of the tamper inputs are sampled with a frequency = RTCCLK / 512

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV256

Each of the tamper inputs are sampled with a frequency = RTCCLK / 256

RTC_TAMPERSAMPLINGFREQ_RTCCLK_MASK

Masking all bits except those of field TAMPFREQ[2:0]

RTC Tamper TimeStamp On Tamper Detection Definitions

RTC_TIMESTAMPONTAMPERDETECTION_DISABLE

TimeStamp on Tamper Detection event is not saved

RTC_TIMESTAMPONTAMPERDETECTION_ENABLE

TimeStamp on Tamper Detection event saved

RTC_TIMESTAMPONTAMPERDETECTION_MASK

Masking all bits except bit TAMPTS

RTC Tamper Triggers Definitions

RTC_TAMPERTRIGGER_RISINGEDGE

Warning : Filter must be RTC_TAMPERFILTER_DISABLE

RTC_TAMPERTRIGGER_FALLINGEDGE

Warning : Filter must be RTC_TAMPERFILTER_DISABLE

RTC_TAMPERTRIGGER_LOWLEVEL

Warning : Filter must not be RTC_TAMPERFILTER_DISABLE

RTC_TAMPERTRIGGER_HIGHLEVEL

Warning : Filter must not be RTC_TAMPERFILTER_DISABLE

RTC_TAMPER_1_TRIGGER

RTC_TAMPER_2_TRIGGER

RTC_TAMPER_3_TRIGGER

RTC_TAMPER_X_TRIGGER

RTC TimeStamp Edges Definitions

RTC_TIMESTAMPEDGE_RISING

RTC_TIMESTAMPEDGE_FALLING

RTC TimeStamp Pin Selection

RTC_TIMESTAMPPIN_DEFAULT

RTC Wakeup Timer Definitions

RTC_WAKEUPCLOCK_RTCCLK_DIV16

RTC_WAKEUPCLOCK_RTCCLK_DIV8

RTC_WAKEUPCLOCK_RTCCLK_DIV4

RTC_WAKEUPCLOCK_RTCCLK_DIV2

RTC_WAKEUPCLOCK_CK_SPRE_16BITS

RTC_WAKEUPCLOCK_CK_SPRE_17BITS

74 HAL SAI Generic Driver

74.1 SAI Firmware driver registers structures

74.1.1 SAI_PdmInitTypeDef

SAI_PdmInitTypeDef is defined in the stm32h7xx_hal_sai.h

Data Fields

- ***FunctionalState Activation***
- ***uint32_t MicPairsNbr***
- ***uint32_t ClockEnable***

Field Documentation

- ***FunctionalState SAI_PdmInitTypeDef::Activation***
  Enable/disable PDM interface
- ***uint32_t SAI_PdmInitTypeDef::MicPairsNbr***
  Specifies the number of microphone pairs used. This parameter must be a number between Min_Data = 1 and Max_Data = 3.
- ***uint32_t SAI_PdmInitTypeDef::ClockEnable***
  Specifies which clock must be enabled. This parameter can be a values combination of ***SAI_PDM_ClockEnable***

74.1.2 SAI_InitTypeDef

SAI_InitTypeDef is defined in the stm32h7xx_hal_sai.h

Data Fields

- ***uint32_t AudioMode***
- ***uint32_t Synchro***
- ***uint32_t SynchroExt***
- ***uint32_t MckOutput***
- ***uint32_t OutputDrive***
- ***uint32_t NoDivider***
- ***uint32_t FIFOThreshold***
- ***uint32_t AudioFrequency***
- ***uint32_t Mckdiv***
- ***uint32_t MckOverSampling***
- ***uint32_t MonoStereoMode***
- ***uint32_t CompandingMode***
- ***uint32_t TriState***
- ***SAI_PdmInitTypeDef PdmInit***
- ***uint32_t Protocol***
- ***uint32_t DataSize***
- ***uint32_t FirstBit***
- ***uint32_t ClockStrobing***

Field Documentation

- ***uint32_t SAI_InitTypeDef::AudioMode***
  Specifies the SAI Block audio Mode. This parameter can be a value of ***SAI_Block_Mode***
- ***uint32_t SAI_InitTypeDef::Synchro***
  Specifies SAI Block synchronization This parameter can be a value of ***SAI_Block_Synchronization***

- ***uint32_t SAI_InitTypeDef::SynchroExt***

  Specifies SAI external output synchronization, this setup is common for BlockA and BlockB This parameter can be a value of ***SAI_Block_SyncExt***

  Note:

  – If both audio blocks of same SAI are used, this parameter has to be set to the same value for each audio block
- ***uint32_t SAI_InitTypeDef::MckOutput***

  Specifies whether master clock output will be generated or not. This parameter can be a value of ***SAI_Block_MckOutput***

  Note:

  – This feature is only available on STM32H7xx Rev.B and above
- ***uint32_t SAI_InitTypeDef::OutputDrive***

  Specifies when SAI Block outputs are driven. This parameter can be a value of ***SAI_Block_Output_Drive***

  Note:

  – This value has to be set before enabling the audio block but after the audio block configuration.
- ***uint32_t SAI_InitTypeDef::NoDivider***

  Specifies whether master clock will be divided or not. This parameter can be a value of ***SAI_Block_NoDivider***

  Note:

  – If bit NODIV in the SAI_xCR1 register is cleared, the frame length should be aligned to a number equal to a power of 2, from 8 to 256. If bit NODIV in the SAI_xCR1 register is set, the frame length can take any of the values from 8 to 256.

  – The NODIV bit is the same as NOMCK bit in STM32H7xx rev.Y
- ***uint32_t SAI_InitTypeDef::FIFOThreshold***

  Specifies SAI Block FIFO threshold. This parameter can be a value of ***SAI_Block_Fifo_Threshold***
- ***uint32_t SAI_InitTypeDef::AudioFrequency***

  Specifies the audio frequency sampling. This parameter can be a value of ***SAI_Audio_Frequency***
- ***uint32_t SAI_InitTypeDef::Mckdiv***

  Specifies the master clock divider. This parameter must be a number between Min_Data = 0 and Max_Data = 63.

  Note:

  – This parameter is used only if AudioFrequency is set to SAI_AUDIO_FREQUENCY_MCKDIV otherwise it is internally computed.
- ***uint32_t SAI_InitTypeDef::MckOverSampling***

  Specifies the master clock oversampling. This parameter can be a value of ***SAI_Block_Mck_OverSampling***
- ***uint32_t SAI_InitTypeDef::MonoStereoMode***

  Specifies if the mono or stereo mode is selected. This parameter can be a value of ***SAI_Mono_Stereo_Mode***
- ***uint32_t SAI_InitTypeDef::CompandingMode***

  Specifies the companding mode type. This parameter can be a value of ***SAI_Block_Companding_Mode***
- ***uint32_t SAI_InitTypeDef::TriState***

  Specifies the companding mode type. This parameter can be a value of ***SAI_TRIState_Management***
- ***SAI_PdmInitTypeDef SAI_InitTypeDef::PdmInit***

  Specifies the PDM configuration.
- ***uint32_t SAI_InitTypeDef::Protocol***

  Specifies the SAI Block protocol. This parameter can be a value of ***SAI_Block_Protocol***
- ***uint32_t SAI_InitTypeDef::DataSize***

  Specifies the SAI Block data size. This parameter can be a value of ***SAI_Block_Data_Size***
- ***uint32_t SAI_InitTypeDef::FirstBit***

  Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of ***SAI_Block_MSB_LSB_transmission***

- ***uint32_t SAI_InitTypeDef::ClockStrobing***

  Specifies the SAI Block clock strobing edge sensitivity. This parameter can be a value of ***SAI_Block_Clock_Strobing***

74.1.3 SAI_FrameInitTypeDef

SAI_FrameInitTypeDef is defined in the stm32h7xx_hal_sai.h

Data Fields

- ***uint32_t FrameLength***
- ***uint32_t ActiveFrameLength***
- ***uint32_t FSDefinition***
- ***uint32_t FSPolarity***
- ***uint32_t FSOffset***

Field Documentation

- ***uint32_t SAI_FrameInitTypeDef::FrameLength***

  Specifies the Frame length, the number of SCK clocks for each audio frame. This parameter must be a number between Min_Data = 8 and Max_Data = 256.

  Note:

  – If master clock MCLK_x pin is declared as an output, the frame length should be aligned to a number equal to power of 2 in order to keep in an audio frame, an integer number of MCLK pulses by bit Clock.

- ***uint32_t SAI_FrameInitTypeDef::ActiveFrameLength***

  Specifies the Frame synchronization active level length. This Parameter specifies the length in number of bit clock (SCK + 1) of the active level of FS signal in audio frame. This parameter must be a number between Min_Data = 1 and Max_Data = 128

- ***uint32_t SAI_FrameInitTypeDef::FSDefinition***

  Specifies the Frame synchronization definition. This parameter can be a value of ***SAI_Block_FS_Definition***

- ***uint32_t SAI_FrameInitTypeDef::FSPolarity***

  Specifies the Frame synchronization Polarity. This parameter can be a value of ***SAI_Block_FS_Polarity***

- ***uint32_t SAI_FrameInitTypeDef::FSOffset***

  Specifies the Frame synchronization Offset. This parameter can be a value of ***SAI_Block_FS_Offset***

74.1.4 SAI_SlotInitTypeDef

SAI_SlotInitTypeDef is defined in the stm32h7xx_hal_sai.h

Data Fields

- ***uint32_t FirstBitOffset***
- ***uint32_t SlotSize***
- ***uint32_t SlotNumber***
- ***uint32_t SlotActive***

Field Documentation

- ***uint32_t SAI_SlotInitTypeDef::FirstBitOffset***

  Specifies the position of first data transfer bit in the slot. This parameter must be a number between Min_Data = 0 and Max_Data = 24

- ***uint32_t SAI_SlotInitTypeDef::SlotSize***

  Specifies the Slot Size. This parameter can be a value of ***SAI_Block_Slot_Size***

- ***uint32_t SAI_SlotInitTypeDef::SlotNumber***

  Specifies the number of slot in the audio frame. This parameter must be a number between Min_Data = 1 and Max_Data = 16

- ***uint32_t SAI_SlotInitTypeDef::SlotActive***

  Specifies the slots in audio frame that will be activated. This parameter can be a value of ***SAI_Block_Slot_Active***

74.1.5 __SAI_HandleTypeDef

__SAI_HandleTypeDef is defined in the stm32h7xx_hal_sai.h

Data Fields

- ***SAI_Block_TypeDef * Instance***
- ***SAI_InitTypeDef Init***
- ***SAI_FrameInitTypeDef FrameInit***
- ***SAI_SlotInitTypeDef SlotInit***
- ***uint8_t * pBuffPtr***
- ***uint16_t XferSize***
- ***uint16_t XferCount***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***SAIcallback mutecallback***
- ***void(* InterruptServiceRoutine***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SAI_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* RxCpltCallback***
- ***void(* RxHalfCpltCallback***
- ***void(* TxCpltCallback***
- ***void(* TxHalfCpltCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***SAI_Block_TypeDef* __SAI_HandleTypeDef::Instance***
  SAI Blockx registers base address
- ***SAI_InitTypeDef __SAI_HandleTypeDef::Init***
  SAI communication parameters
- ***SAI_FrameInitTypeDef __SAI_HandleTypeDef::FrameInit***
  SAI Frame configuration parameters
- ***SAI_SlotInitTypeDef __SAI_HandleTypeDef::SlotInit***
  SAI Slot configuration parameters
- ***uint8_t* __SAI_HandleTypeDef::pBuffPtr***
  Pointer to SAI transfer Buffer
- ***uint16_t __SAI_HandleTypeDef::XferSize***
  SAI transfer size
- ***uint16_t __SAI_HandleTypeDef::XferCount***
  SAI transfer counter
- ***DMA_HandleTypeDef* __SAI_HandleTypeDef::hdmatx***
  SAI Tx DMA handle parameters
- ***DMA_HandleTypeDef* __SAI_HandleTypeDef::hdmarx***
  SAI Rx DMA handle parameters
- ***SAIcallback __SAI_HandleTypeDef::mutecallback***
  SAI mute callback
- ***void(* __SAI_HandleTypeDef::InterruptServiceRoutine)(struct __SAI_HandleTypeDef *hsai)***
- ***HAL_LockTypeDef __SAI_HandleTypeDef::Lock***
  SAI locking object

- ***__IO HAL_SAI_StateTypeDef __SAI_HandleTypeDef::State***
  SAI communication state
- ***__IO uint32_t __SAI_HandleTypeDef::ErrorCode***
  SAI Error code
- ***void(* __SAI_HandleTypeDef::RxCpltCallback)(struct __SAI_HandleTypeDef *hsai)***
  SAI receive complete callback
- ***void(* __SAI_HandleTypeDef::RxHalfCpltCallback)(struct __SAI_HandleTypeDef *hsai)***
  SAI receive half complete callback
- ***void(* __SAI_HandleTypeDef::TxCpltCallback)(struct __SAI_HandleTypeDef *hsai)***
  SAI transmit complete callback
- ***void(* __SAI_HandleTypeDef::TxHalfCpltCallback)(struct __SAI_HandleTypeDef *hsai)***
  SAI transmit half complete callback
- ***void(* __SAI_HandleTypeDef::ErrorCallback)(struct __SAI_HandleTypeDef *hsai)***
  SAI error callback
- ***void(* __SAI_HandleTypeDef::MspInitCallback)(struct __SAI_HandleTypeDef *hsai)***
  SAI MSP init callback
- ***void(* __SAI_HandleTypeDef::MspDeInitCallback)(struct __SAI_HandleTypeDef *hsai)***
  SAI MSP de-init callback

74.2 SAI Firmware driver API description

The following section lists the various functions of the SAI library.

74.2.1 How to use this driver

The SAI HAL driver can be used as follows:

1. Declare a SAI_HandleTypeDef handle structure (eg. SAI_HandleTypeDef hsai).
2. Initialize the SAI low level resources by implementing the HAL_SAI_MspInit() API:
   a. Enable the SAI interface clock.
   b. SAI pins configuration:
      - Enable the clock for the SAI GPIOs.
      - Configure these SAI pins as alternate function pull-up.
   c. NVIC configuration if you need to use interrupt process (HAL_SAI_Transmit_IT() and HAL_SAI_Receive_IT() APIs):
      - Configure the SAI interrupt priority.
      - Enable the NVIC SAI IRQ handle.
   d. DMA Configuration if you need to use DMA process (HAL_SAI_Transmit_DMA() and HAL_SAI_Receive_DMA() APIs):
      - Declare a DMA handle structure for the Tx/Rx stream.
      - Enable the DMAx interface clock.
      - Configure the declared DMA handle structure with the required Tx/Rx parameters.
      - Configure the DMA Tx/Rx Stream.
      - Associate the initialized DMA handle to the SAI DMA Tx/Rx handle.
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx Stream.
3. The initialization can be done by two ways
   a. Expert mode : Initialize the structures Init, FrameInit and SlotInit and call HAL_SAI_Init().
   b. Simplified mode : Initialize the high part of Init Structure and call HAL_SAI_InitProtocol().

Note: The specific SAI interrupts (FIFO request and Overrun underrun interrupt) will be managed using the macros __HAL_SAI_ENABLE_IT() and __HAL_SAI_DISABLE_IT() inside the transmit and receive process.

Note: Make sure that either:

- *PLLSAI1CLK output is configured or*
- *PLLSAI2CLK output is configured or*
- *PLLSAI3CLK output is configured or*
- *PLLSAI4ACLK output is configured or*
- *PLLSAI4BCLK output is configured or*
- *External clock source is configured after setting correctly the define constant EXTERNAL_CLOCK_VALUE in the stm32h7xx_hal_conf.h file.*

Note: In master Tx mode: enabling the audio block immediately generates the bit clock for the external slaves even if there is no data in the FIFO, However FS signal generation is conditioned by the presence of data in the FIFO.

Note: In master Rx mode: enabling the audio block immediately generates the bit clock and FS signal for the external slaves.

Note: It is mandatory to respect the following conditions in order to avoid bad SAI behavior:

- *First bit Offset <= (SLOT size - Data size)*
- *Data size <= SLOT size*
- *Number of SLOT x SLOT size = Frame length*
- *The number of slots should be even when SAI_FS_CHANNEL_IDENTIFICATION is selected.*

Note: PDM interface can be activated through HAL_SAI_Init function. Please note that PDM interface is only available for SAI1 or SAI4 sub-block A. PDM microphone delays can be tuned with HAL_SAIEx_ConfigPdmMicDelay function.

Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_SAI_Transmit()
- Receive an amount of data in blocking mode using HAL_SAI_Receive()

Interrupt mode IO operation

- Send an amount of data in non-blocking mode using HAL_SAI_Transmit_IT()
- At transmission end of transfer HAL_SAI_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SAI_TxCpltCallback()
- Receive an amount of data in non-blocking mode using HAL_SAI_Receive_IT()
- At reception end of transfer HAL_SAI_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SAI_RxCpltCallback()
- In case of flag error, HAL_SAI_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SAI_ErrorCallback()

DMA mode IO operation

- Send an amount of data in non-blocking mode (DMA) using HAL_SAI_Transmit_DMA()
- At transmission end of transfer HAL_SAI_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SAI_TxCpltCallback()
- Receive an amount of data in non-blocking mode (DMA) using HAL_SAI_Receive_DMA()
- At reception end of transfer HAL_SAI_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SAI_RxCpltCallback()
- In case of flag error, HAL_SAI_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SAI_ErrorCallback()
- Pause the DMA Transfer using HAL_SAI_DMAPause()
- Resume the DMA Transfer using HAL_SAI_DMAResume()
- Stop the DMA Transfer using HAL_SAI_DMAStop()

SAI HAL driver additional function list

Below the list the others API available SAI HAL driver :

- HAL_SAI_EnableTxMuteMode(): Enable the mute in tx mode
- HAL_SAI_DisableTxMuteMode(): Disable the mute in tx mode
- HAL_SAI_EnableRxMuteMode(): Enable the mute in Rx mode
- HAL_SAI_DisableRxMuteMode(): Disable the mute in Rx mode
- HAL_SAI_FlushRxFifo(): Flush the rx fifo.
- HAL_SAI_Abort(): Abort the current transfer

SAI HAL driver macros list

Below the list of most used macros in SAI HAL driver :

- __HAL_SAI_ENABLE(): Enable the SAI peripheral
- __HAL_SAI_DISABLE(): Disable the SAI peripheral
- __HAL_SAI_ENABLE_IT(): Enable the specified SAI interrupts
- __HAL_SAI_DISABLE_IT(): Disable the specified SAI interrupts
- __HAL_SAI_GET_IT_SOURCE(): Check if the specified SAI interrupt source is enabled or disabled
- __HAL_SAI_GET_FLAG(): Check whether the specified SAI flag is set or not

Callback registration

The compilation define USE_HAL_SAI_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use functions HAL_SAI_RegisterCallback() to register a user callback.

Function HAL_SAI_RegisterCallback() allows to register following callbacks:

- RxCpltCallback : SAI receive complete.
- RxHalfCpltCallback : SAI receive half complete.
- TxCpltCallback : SAI transmit complete.
- TxHalfCpltCallback : SAI transmit half complete.
- ErrorCallback : SAI error.
- MspInitCallback : SAI MspInit.
- MspDeInitCallback : SAI MspDeInit.

This function takes as parameters the HAL peripheral handle, the callback ID and a pointer to the user callback function.

Use function HAL_SAI_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. HAL_SAI_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the callback ID.

This function allows to reset following callbacks:

- RxCpltCallback : SAI receive complete.
- RxHalfCpltCallback : SAI receive half complete.
- TxCpltCallback : SAI transmit complete.
- TxHalfCpltCallback : SAI transmit half complete.
- ErrorCallback : SAI error.
- MspInitCallback : SAI MspInit.
- MspDeInitCallback : SAI MspDeInit.

By default, after the HAL_SAI_Init and if the state is HAL_SAI_STATE_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions: examples HAL_SAI_RxCpltCallback(), HAL_SAI_ErrorCallback(). Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the HAL_SAI_Init and HAL_SAI_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_SAI_Init and HAL_SAI_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_SAI_RegisterCallback before calling HAL_SAI_DeInit or HAL_SAI_Init function.

When the compilation define USE_HAL_SAI_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

74.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the SAIx peripheral:

- User must implement HAL_SAI_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_SAI_Init() to configure the selected device with the selected configuration:
  - Mode (Master/slave TX/RX)
  - Protocol
  - Data Size
  - MCLK Output
  - Audio frequency
  - FIFO Threshold
  - Frame Config
  - Slot Config
  - PDM Config
- Call the function HAL_SAI_DeInit() to restore the default configuration of the selected SAI peripheral.

This section contains the following APIs:

- ***HAL_SAI_InitProtocol()***
- ***HAL_SAI_Init()***
- ***HAL_SAI_DeInit()***
- ***HAL_SAI_MspInit()***
- ***HAL_SAI_MspDeInit()***
- ***HAL_SAI_RegisterCallback()***
- ***HAL_SAI_UnRegisterCallback()***

74.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the SAI data transfers.

- There are two modes of transfer:
  - Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
  - No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated SAI IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
- Blocking mode functions are :
  - HAL_SAI_Transmit()
  - HAL_SAI_Receive()
- Non Blocking mode functions with Interrupt are :
  - HAL_SAI_Transmit_IT()
  - HAL_SAI_Receive_IT()
- Non Blocking mode functions with DMA are :
  - HAL_SAI_Transmit_DMA()
  - HAL_SAI_Receive_DMA()
- A set of Transfer Complete Callbacks are provided in non Blocking mode:
  - HAL_SAI_TxCpltCallback()
  - HAL_SAI_RxCpltCallback()
  - HAL_SAI_ErrorCallback()

This section contains the following APIs:

- ***HAL_SAI_Transmit()***
- ***HAL_SAI_Receive()***
- ***HAL_SAI_Transmit_IT()***

- *HAL_SAI_Receive_IT()*
- *HAL_SAI_DMAPause()*
- *HAL_SAI_DMAResume()*
- *HAL_SAI_DMAStop()*
- *HAL_SAI_Abort()*
- *HAL_SAI_Transmit_DMA()*
- *HAL_SAI_Receive_DMA()*
- *HAL_SAI_EnableTxMuteMode()*
- *HAL_SAI_DisableTxMuteMode()*
- *HAL_SAI_EnableRxMuteMode()*
- *HAL_SAI_DisableRxMuteMode()*
- *HAL_SAI_IRQHandler()*
- *HAL_SAI_TxCpltCallback()*
- *HAL_SAI_TxHalfCpltCallback()*
- *HAL_SAI_RxCpltCallback()*
- *HAL_SAI_RxHalfCpltCallback()*
- *HAL_SAI_ErrorCallback()*

74.2.4 Peripheral State and Errors functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- *HAL_SAI_GetState()*
- *HAL_SAI_GetError()*

74.2.5 Detailed description of functions

HAL_SAI_InitProtocol

Function name

HAL_StatusTypeDef HAL_SAI_InitProtocol (SAI_HandleTypeDef * hsai, uint32_t protocol, uint32_t datasize, uint32_t nbslot)

Function description

Initialize the structure FrameInit, SlotInit and the low part of Init according to the specified parameters and call the function HAL_SAI_Init to initialize the SAI block.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **protocol:** one of the supported protocol SAI Supported protocol
- **datasize:** one of the supported datasize SAI protocol data size the configuration information for SAI module.
- **nbslot:** Number of slot.

Return values

- **HAL:** status

HAL_SAI_Init

Function name

HAL_StatusTypeDef HAL_SAI_Init (SAI_HandleTypeDef * hsai)

Function description

Initialize the SAI according to the specified parameters.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_DeInit

Function name

HAL_StatusTypeDef HAL_SAI_DeInit (SAI_HandleTypeDef * hsai)

Function description

DeInitialize the SAI peripheral.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_MspInit

Function name

void HAL_SAI_MspInit (SAI_HandleTypeDef * hsai)

Function description

Initialize the SAI MSP.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_MspDeInit

Function name

void HAL_SAI_MspDeInit (SAI_HandleTypeDef * hsai)

Function description

DeInitialize the SAI MSP.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SAI_RegisterCallback (SAI_HandleTypeDef * hsai, HAL_SAI_CallbackIDTypeDef CallbackID, pSAI_CallbackTypeDef pCallback)

Function description

Register a user SAI callback to be used instead of the weak predefined callback.

Parameters

- **hsai:** SAI handle.
- **CallbackID:** ID of the callback to be registered. This parameter can be one of the following values:
  - HAL_SAI_RX_COMPLETE_CB_ID receive complete callback ID.
  - HAL_SAI_RX_HALFCOMPLETE_CB_ID receive half complete callback ID.
  - HAL_SAI_TX_COMPLETE_CB_ID transmit complete callback ID.
  - HAL_SAI_TX_HALFCOMPLETE_CB_ID transmit half complete callback ID.
  - HAL_SAI_ERROR_CB_ID error callback ID.
  - HAL_SAI_MSPINIT_CB_ID MSP init callback ID.
  - HAL_SAI_MSPDEINIT_CB_ID MSP de-init callback ID.
- **pCallback:** pointer to the callback function.

Return values

- **HAL:** status.

HAL_SAI_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SAI_UnRegisterCallback (SAI_HandleTypeDef * hsai, HAL_SAI_CallbackIDTypeDef CallbackID)

Function description

Unregister a user SAI callback.

Parameters

- **hsai:** SAI handle.
- **CallbackID:** ID of the callback to be unregistered. This parameter can be one of the following values:
  - HAL_SAI_RX_COMPLETE_CB_ID receive complete callback ID.
  - HAL_SAI_RX_HALFCOMPLETE_CB_ID receive half complete callback ID.
  - HAL_SAI_TX_COMPLETE_CB_ID transmit complete callback ID.
  - HAL_SAI_TX_HALFCOMPLETE_CB_ID transmit half complete callback ID.
  - HAL_SAI_ERROR_CB_ID error callback ID.
  - HAL_SAI_MSPINIT_CB_ID MSP init callback ID.
  - HAL_SAI_MSPDEINIT_CB_ID MSP de-init callback ID.

Return values

- **HAL:** status.

HAL_SAI_Transmit

Function name

HAL_StatusTypeDef HAL_SAI_Transmit (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Transmit an amount of data in blocking mode.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SAI_Receive

Function name

HAL_StatusTypeDef HAL_SAI_Receive (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SAI_Transmit_IT

Function name

HAL_StatusTypeDef HAL_SAI_Transmit_IT (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_SAI_Receive_IT

Function name

HAL_StatusTypeDef HAL_SAI_Receive_IT (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received

Return values

- **HAL:** status

HAL_SAI_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_SAI_Transmit_DMA (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with DMA.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_SAI_Receive_DMA

Function name

HAL_StatusTypeDef HAL_SAI_Receive_DMA (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received

Return values

- **HAL:** status

HAL_SAI_DMAPause

Function name

HAL_StatusTypeDef HAL_SAI_DMAPause (SAI_HandleTypeDef * hsai)

Function description

Pause the audio stream playing from the Media.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_DMAResume

Function name

HAL_StatusTypeDef HAL_SAI_DMAResume (SAI_HandleTypeDef * hsai)

Function description

Resume the audio stream playing from the Media.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_DMAStop

Function name

HAL_StatusTypeDef HAL_SAI_DMAStop (SAI_HandleTypeDef * hsai)

Function description

Stop the audio stream playing from the Media.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_Abort

Function name

HAL_StatusTypeDef HAL_SAI_Abort (SAI_HandleTypeDef * hsai)

Function description

Abort the current transfer and disable the SAI.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_EnableTxMuteMode

Function name

HAL_StatusTypeDef HAL_SAI_EnableTxMuteMode (SAI_HandleTypeDef * hsai, uint16_t val)

Function description

Enable the Tx mute mode.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **val:** value sent during the mute SAI Block Mute Value

Return values

- **HAL:** status

HAL_SAI_DisableTxMuteMode

Function name

HAL_StatusTypeDef HAL_SAI_DisableTxMuteMode (SAI_HandleTypeDef * hsai)

Function description

Disable the Tx mute mode.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_EnableRxMuteMode

Function name

HAL_StatusTypeDef HAL_SAI_EnableRxMuteMode (SAI_HandleTypeDef * hsai, SAIcallback callback, uint16_t counter)

Function description

Enable the Rx mute detection.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **callback:** function called when the mute is detected.
- **counter:** number a data before mute detection max 63.

Return values

- **HAL:** status

HAL_SAI_DisableRxMuteMode

Function name

HAL_StatusTypeDef HAL_SAI_DisableRxMuteMode (SAI_HandleTypeDef * hsai)

Function description

Disable the Rx mute detection.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_IRQHandler

Function name

void HAL_SAI_IRQHandler (SAI_HandleTypeDef * hsai)

Function description

Handle SAI interrupt request.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_TxHalfCpltCallback

Function name

void HAL_SAI_TxHalfCpltCallback (SAI_HandleTypeDef * hsai)

Function description

Tx Transfer Half completed callback.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_TxCpltCallback

Function name

void HAL_SAI_TxCpltCallback (SAI_HandleTypeDef * hsai)

Function description

Tx Transfer completed callback.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_RxHalfCpltCallback

Function name

void HAL_SAI_RxHalfCpltCallback (SAI_HandleTypeDef * hsai)

Function description

Rx Transfer half completed callback.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_RxCpltCallback

Function name

void HAL_SAI_RxCpltCallback (SAI_HandleTypeDef * hsai)

Function description

Rx Transfer completed callback.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_ErrorCallback

Function name

void HAL_SAI_ErrorCallback (SAI_HandleTypeDef * hsai)

Function description

SAI error callback.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_GetState

Function name

HAL_SAI_StateTypeDef HAL_SAI_GetState (const SAI_HandleTypeDef * hsai)

Function description

Return the SAI handle state.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** state

HAL_SAI_GetError

Function name

uint32_t HAL_SAI_GetError (const SAI_HandleTypeDef * hsai)

Function description

Return the SAI error code.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for the specified SAI Block.

Return values

- **SAI:** Error Code

74.3 SAI Firmware driver defines

The following section lists the various define and macros of the module.

74.3.1 SAI

SAI

SAI Audio Frequency

SAI_AUDIO_FREQUENCY_192K

SAI_AUDIO_FREQUENCY_96K

SAI_AUDIO_FREQUENCY_48K

SAI_AUDIO_FREQUENCY_44K

SAI_AUDIO_FREQUENCY_32K

SAI_AUDIO_FREQUENCY_22K

SAI_AUDIO_FREQUENCY_16K

SAI_AUDIO_FREQUENCY_11K

SAI_AUDIO_FREQUENCY_8K

SAI_AUDIO_FREQUENCY_MCKDIV

SAI Block Clock Strobing

SAI_CLOCKSTROBING_FALLINGEDGE

SAI_CLOCKSTROBING_RISINGEDGE

SAI Block Companding Mode

SAI_NOCOMPANDING

SAI_ULAW_1CPL_COMPANDING

SAI_ALAW_1CPL_COMPANDING

SAI_ULAW_2CPL_COMPANDING

SAI_ALAW_2CPL_COMPANDING

SAI Block Data Size

SAI_DATASIZE_8

SAI_DATASIZE_10

SAI_DATASIZE_16

SAI_DATASIZE_20

SAI_DATASIZE_24

SAI_DATASIZE_32

SAI Block Fifo Status Level

SAI_FIFOSTATUS_EMPTY

SAI_FIFOSTATUS_LESS1QUARTERFULL

SAI_FIFOSTATUS_1QUARTERFULL

SAI_FIFOSTATUS_HALFFULL

SAI_FIFOSTATUS_3QUARTERFULL

SAI_FIFOSTATUS_FULL

SAI Block Fifo Threshold

SAI_FIFOTHRESHOLD_EMPTY

SAI_FIFOTHRESHOLD_1QF

SAI_FIFOTHRESHOLD_HF

SAI_FIFOTHRESHOLD_3QF

SAI_FIFOTHRESHOLD_FULL

SAI Block Flags Definition

SAI_FLAG_OVRUDR

SAI_FLAG_MUTEDET

SAI_FLAG_WCKCFG

SAI_FLAG_FREQ

SAI_FLAG_CNRDY

SAI_FLAG_AFSDET

SAI_FLAG_LFSDET

SAI Block FS Definition

SAI_FS_STARTFRAME

SAI_FS_CHANNEL_IDENTIFICATION

SAI Block FS Offset

SAI_FS_FIRSTBIT

SAI_FS_BEFOREFIRSTBIT

SAI Block FS Polarity

SAI_FS_ACTIVE_LOW

SAI_FS_ACTIVE_HIGH

SAI Block Interrupts Definition

SAI_IT_OVRUDR

SAI_IT_MUTEDET

SAI_IT_WCKCFG

SAI_IT_FREQ

SAI_IT_CNRDY

SAI_IT_AFSDET

SAI_IT_LFSDET

SAI Block Master Clock Output

SAI_MCK_OUTPUT_DISABLE

SAI_MCK_OUTPUT_ENABLE

SAI Block Master Clock OverSampling

SAI_MCK_OVERSAMPLING_DISABLE

SAI_MCK_OVERSAMPLING_ENABLE

SAI Block Mode

SAI_MODEMASTER_TX

SAI_MODEMASTER_RX

SAI_MODESLAVE_TX

SAI_MODESLAVE_RX

SAI Block MSB LSB transmission

SAI_FIRSTBIT_MSB

SAI_FIRSTBIT_LSB

SAI Block Mute Value

SAI_ZERO_VALUE

SAI_LAST_SENT_VALUE

SAI Block NoDivider

SAI_MASTERDIVIDER_ENABLE

SAI_MASTERDIVIDER_DISABLE

SAI Block Output Drive

SAI_OUTPUTDRIVE_DISABLE

SAI_OUTPUTDRIVE_ENABLE

SAI Block Protocol

SAI_FREE_PROTOCOL

SAI_SPDIF_PROTOCOL

SAI_AC97_PROTOCOL

SAI Block Slot Active

SAI_SLOT_NOTACTIVE

SAI_SLOTACTIVE_0

SAI_SLOTACTIVE_1

SAI_SLOTACTIVE_2

SAI_SLOTACTIVE_3

SAI_SLOTACTIVE_4

SAI_SLOTACTIVE_5

SAI_SLOTACTIVE_6

SAI_SLOTACTIVE_7

SAI_SLOTACTIVE_8

SAI_SLOTACTIVE_9

SAI_SLOTACTIVE_10

SAI_SLOTACTIVE_11

SAI_SLOTACTIVE_12

SAI_SLOTACTIVE_13

SAI_SLOTACTIVE_14

SAI_SLOTACTIVE_15

SAI_SLOTACTIVE_ALL

SAI Block Slot Size

SAI_SLOTSIZE_DATASIZE

SAI_SLOTSIZE_16B

SAI_SLOTSIZE_32B

SAI External synchronisation

SAI_SYNCEXT_DISABLE

SAI_SYNCEXT_OUTBLOCKA_ENABLE

SAI_SYNCEXT_OUTBLOCKB_ENABLE

SAI Block Synchronization

SAI_ASYNCHRONOUS

Asynchronous

SAI_SYNCHRONOUS

Synchronous with other block of same SAI

SAI_SYNCHRONOUS_EXT_SAI1

Synchronous with other SAI, SAI1

SAI_SYNCHRONOUS_EXT_SAI2

Synchronous with other SAI, SAI2

SAI_SYNCHRONOUS_EXT_SAI3

Synchronous with other SAI, SAI3

SAI_SYNCHRONOUS_EXT_SAI4

Synchronous with other SAI, SAI4

SAI Error Code

HAL_SAI_ERROR_NONE

No error

HAL_SAI_ERROR_OVR

Overrun Error

HAL_SAI_ERROR_UDR

Underrun error

HAL_SAI_ERROR_AFSDET

Anticipated Frame synchronisation detection

HAL_SAI_ERROR_LFSDET

Late Frame synchronisation detection

HAL_SAI_ERROR_CNREADY

codec not ready

HAL_SAI_ERROR_WCKCFG

Wrong clock configuration

HAL_SAI_ERROR_TIMEOUT

Timeout error

HAL_SAI_ERROR_DMA

DMA error

HAL_SAI_ERROR_INVALID_CALLBACK

Invalid callback error

SAI Exported Macros

__HAL_SAI_RESET_HANDLE_STATE

Description:

- Reset SAI handle state.

Parameters:

- __HANDLE__: specifies the SAI Handle.

Return value:

- None

__HAL_SAI_ENABLE_IT

Description:

- Enable the specified SAI interrupts.

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - SAI_IT_OVRUDR: Overrun underrun interrupt enable
  - SAI_IT_MUTEDET: Mute detection interrupt enable
  - SAI_IT_WCKCFG: Wrong Clock Configuration interrupt enable
  - SAI_IT_FREQ: FIFO request interrupt enable
  - SAI_IT_CNRDY: Codec not ready interrupt enable
  - SAI_IT_AFSDET: Anticipated frame synchronization detection interrupt enable
  - SAI_IT_LFSDET: Late frame synchronization detection interrupt enable

Return value:

- None

__HAL_SAI_DISABLE_IT

Description:

- Disable the specified SAI interrupts.

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - SAI_IT_OVRUDR: Overrun underrun interrupt enable
  - SAI_IT_MUTEDET: Mute detection interrupt enable
  - SAI_IT_WCKCFG: Wrong Clock Configuration interrupt enable
  - SAI_IT_FREQ: FIFO request interrupt enable
  - SAI_IT_CNRDY: Codec not ready interrupt enable
  - SAI_IT_AFSDET: Anticipated frame synchronization detection interrupt enable
  - SAI_IT_LFSDET: Late frame synchronization detection interrupt enable

Return value:

- None

__HAL_SAI_GET_IT_SOURCE

Description:

- Check whether the specified SAI interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __INTERRUPT__: specifies the SAI interrupt source to check. This parameter can be one of the following values:
  - SAI_IT_OVRUDR: Overrun underrun interrupt enable
  - SAI_IT_MUTEDET: Mute detection interrupt enable
  - SAI_IT_WCKCFG: Wrong Clock Configuration interrupt enable
  - SAI_IT_FREQ: FIFO request interrupt enable
  - SAI_IT_CNRDY: Codec not ready interrupt enable
  - SAI_IT_AFSDET: Anticipated frame synchronization detection interrupt enable
  - SAI_IT_LFSDET: Late frame synchronization detection interrupt enable

Return value:

- The: new state of __INTERRUPT__ (TRUE or FALSE).

__HAL_SAI_GET_FLAG

Description:

- Check whether the specified SAI flag is set or not.

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SAI_FLAG_OVRUDR: Overrun underrun flag.
  - SAI_FLAG_MUTEDET: Mute detection flag.
  - SAI_FLAG_WCKCFG: Wrong Clock Configuration flag.
  - SAI_FLAG_FREQ: FIFO request flag.
  - SAI_FLAG_CNRDY: Codec not ready flag.
  - SAI_FLAG_AFSDET: Anticipated frame synchronization detection flag.
  - SAI_FLAG_LFSDET: Late frame synchronization detection flag.

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_SAI_CLEAR_FLAG

Description:

- Clear the specified SAI pending flag.

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - SAI_FLAG_OVRUDR: Clear Overrun underrun
  - SAI_FLAG_MUTEDET: Clear Mute detection
  - SAI_FLAG_WCKCFG: Clear Wrong Clock Configuration
  - SAI_FLAG_FREQ: Clear FIFO request
  - SAI_FLAG_CNRDY: Clear Codec not ready
  - SAI_FLAG_AFSDET: Clear Anticipated frame synchronization detection
  - SAI_FLAG_LFSDET: Clear Late frame synchronization detection

Return value:

- None

__HAL_SAI_ENABLE

Description:

- Enable SAI.

Parameters:

- __HANDLE__: specifies the SAI Handle.

Return value:

- None

__HAL_SAI_DISABLE

Description:

- Disable SAI.

Parameters:

- __HANDLE__: specifies the SAI Handle.

Return value:

- None

SAI Mono Stereo Mode

SAI_STEREOMODE

SAI_MONOMODE

SAI PDM Clock Enable

SAI_PDM_CLOCK1_ENABLE

SAI_PDM_CLOCK2_ENABLE

SAI Supported protocol

SAI_I2S_STANDARD

SAI_I2S_MSBJUSTIFIED

SAI_I2S_LSBJUSTIFIED

SAI_PCM_LONG

SAI_PCM_SHORT

SAI protocol data size

SAI_PROTOCOL_DATASIZE_16BIT

SAI_PROTOCOL_DATASIZE_16BITEXTENDED

SAI_PROTOCOL_DATASIZE_24BIT

SAI_PROTOCOL_DATASIZE_32BIT

SAI TRIState Management

SAI_OUTPUT_NOTRELEASED

SAI_OUTPUT_RELEASED

75 HAL SAI Extension Driver

75.1 SAIEx Firmware driver registers structures

75.1.1 SAIEx_PdmMicDelayParamTypeDef

SAIEx_PdmMicDelayParamTypeDef is defined in the stm32h7xx_hal_sai_ex.h

Data Fields

- ***uint32_t MicPair***
- ***uint32_t LeftDelay***
- ***uint32_t RightDelay***

Field Documentation

- ***uint32_t SAIEx_PdmMicDelayParamTypeDef::MicPair***
  Specifies which pair of microphones is selected. This parameter must be a number between Min_Data = 1 and Max_Data = 3.
- ***uint32_t SAIEx_PdmMicDelayParamTypeDef::LeftDelay***
  Specifies the delay in PDM clock unit to apply on left microphone. This parameter must be a number between Min_Data = 0 and Max_Data = 7.
- ***uint32_t SAIEx_PdmMicDelayParamTypeDef::RightDelay***
  Specifies the delay in PDM clock unit to apply on right microphone. This parameter must be a number between Min_Data = 0 and Max_Data = 7.

75.2 SAIEx Firmware driver API description

The following section lists the various functions of the SAIEx library.

75.2.1 Extended features functions

This section provides functions allowing to:

- Modify PDM microphone delays

This section contains the following APIs:

- ***HAL_SAIEx_ConfigPdmMicDelay()***

75.2.2 Detailed description of functions

HAL_SAIEx_ConfigPdmMicDelay

Function name

HAL_StatusTypeDef HAL_SAIEx_ConfigPdmMicDelay (const SAI_HandleTypeDef * hsai, const SAIEx_PdmMicDelayParamTypeDef * pdmMicDelay)

Function description

Configure PDM microphone delays.

Parameters

- **hsai:** SAI handle.
- **pdmMicDelay:** Microphone delays configuration.

Return values

- **HAL:** status

76 HAL SDRAM Generic Driver

76.1 SDRAM Firmware driver registers structures

76.1.1 __SDRAM_HandleTypeDef

__SDRAM_HandleTypeDef is defined in the stm32h7xx_hal_sdram.h

Data Fields

- ***FMC_SDRAM_TypeDef * Instance***
- ***FMC_SDRAM_InitTypeDef Init***
- ***__IO HAL_SDRAM_StateTypeDef State***
- ***HAL_LockTypeDef Lock***
- ***MDMA_HandleTypeDef * hmdma***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***void(* RefreshErrorCallback***
- ***void(* DmaXferCpltCallback***
- ***void(* DmaXferErrorCallback***

Field Documentation

- ***FMC_SDRAM_TypeDef* __SDRAM_HandleTypeDef::Instance***
  Register base address
- ***FMC_SDRAM_InitTypeDef __SDRAM_HandleTypeDef::Init***
  SDRAM device configuration parameters
- ***__IO HAL_SDRAM_StateTypeDef __SDRAM_HandleTypeDef::State***
  SDRAM access state
- ***HAL_LockTypeDef __SDRAM_HandleTypeDef::Lock***
  SDRAM locking object
- ***MDMA_HandleTypeDef* __SDRAM_HandleTypeDef::hmdma***
  Pointer DMA handler
- ***void(* __SDRAM_HandleTypeDef::MspInitCallback)(struct __SDRAM_HandleTypeDef *hsdram)***
  SDRAM Msp Init callback
- ***void(* __SDRAM_HandleTypeDef::MspDeInitCallback)(struct __SDRAM_HandleTypeDef *hsdram)***
  SDRAM Msp DeInit callback
- ***void(* __SDRAM_HandleTypeDef::RefreshErrorCallback)(struct __SDRAM_HandleTypeDef *hsdram)***
  SDRAM Refresh Error callback
- ***void(* __SDRAM_HandleTypeDef::DmaXferCpltCallback)(MDMA_HandleTypeDef *hmdma)***
  SDRAM DMA Xfer Complete callback
- ***void(* __SDRAM_HandleTypeDef::DmaXferErrorCallback)(MDMA_HandleTypeDef *hmdma)***
  SDRAM DMA Xfer Error callback

76.2 SDRAM Firmware driver API description

The following section lists the various functions of the SDRAM library.

76.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control SDRAM memories. It uses the FMC layer functions to interface with SDRAM devices. The following sequence should be followed to configure the FMC to interface with SDRAM memories:

1. Declare a SDRAM_HandleTypeDef handle structure, for example: SDRAM_HandleTypeDef hsdram
   – Fill the SDRAM_HandleTypeDef handle "Init" field with the allowed values of the structure member.
   – Fill the SDRAM_HandleTypeDef handle "Instance" field with a predefined base register instance for NOR or SDRAM device
2. Declare a FMC_SDRAM_TimingTypeDef structure; for example: FMC_SDRAM_TimingTypeDef Timing; and fill its fields with the allowed values of the structure member.
3. Initialize the SDRAM Controller by calling the function HAL_SDRAM_Init(). This function performs the following sequence:
   a. MSP hardware layer configuration using the function HAL_SDRAM_MspInit()
   b. Control register configuration using the FMC SDRAM interface function FMC_SDRAM_Init()
   c. Timing register configuration using the FMC SDRAM interface function FMC_SDRAM_Timing_Init()
   d. Program the SDRAM external device by applying its initialization sequence according to the device plugged in your hardware. This step is mandatory for accessing the SDRAM device.
4. At this stage you can perform read/write accesses from/to the memory connected to the SDRAM Bank. You can perform either polling or DMA transfer using the following APIs:
   – HAL_SDRAM_Read()/HAL_SDRAM_Write() for polling read/write access
   – HAL_SDRAM_Read_DMA()/HAL_SDRAM_Write_DMA() for DMA read/write transfer
5. You can also control the SDRAM device by calling the control APIs HAL_SDRAM_WriteOperation_Enable()/ HAL_SDRAM_WriteOperation_Disable() to respectively enable/disable the SDRAM write operation or the function HAL_SDRAM_SendCommand() to send a specified command to the SDRAM device. The command to be sent must be configured with the FMC_SDRAM_CommandTypeDef structure.
6. You can continuously monitor the SDRAM device HAL state by calling the function HAL_SDRAM_GetState()

Callback registration

The compilation define USE_HAL_SDRAM_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_SDRAM_RegisterCallback() to register a user callback, it allows to register following callbacks:

- MspInitCallback : SDRAM MspInit.
- MspDeInitCallback : SDRAM MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. Use function HAL_SDRAM_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. It allows to reset following callbacks:
- MspInitCallback : SDRAM MspInit.
- MspDeInitCallback : SDRAM MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the HAL_SDRAM_Init and if the state is HAL_SDRAM_STATE_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the HAL_SDRAM_Init and HAL_SDRAM_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_SDRAM_Init and HAL_SDRAM_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/ unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_SDRAM_RegisterCallback before calling HAL_SDRAM_DeInit or HAL_SDRAM_Init function. When The compilation define USE_HAL_SDRAM_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

76.2.2 SDRAM Initialization and de_initialization functions

This section provides functions allowing to initialize/de-initialize the SDRAM memory

This section contains the following APIs:

- ***HAL_SDRAM_Init()***
- ***HAL_SDRAM_DeInit()***
- ***HAL_SDRAM_MspInit()***
- ***HAL_SDRAM_MspDeInit()***
- ***HAL_SDRAM_IRQHandler()***

- *HAL_SDRAM_RefreshErrorCallback()*
- *HAL_SDRAM_DMA_XferCpltCallback()*
- *HAL_SDRAM_DMA_XferErrorCallback()*

76.2.3 SDRAM Input and Output functions

This section provides functions allowing to use and control the SDRAM memory

This section contains the following APIs:

- *HAL_SDRAM_Read_8b()*
- *HAL_SDRAM_Write_8b()*
- *HAL_SDRAM_Read_16b()*
- *HAL_SDRAM_Write_16b()*
- *HAL_SDRAM_Read_32b()*
- *HAL_SDRAM_Write_32b()*
- *HAL_SDRAM_Read_DMA()*
- *HAL_SDRAM_Write_DMA()*
- *HAL_SDRAM_RegisterCallback()*
- *HAL_SDRAM_UnRegisterCallback()*
- *HAL_SDRAM_RegisterDmaCallback()*

76.2.4 SDRAM Control functions

This subsection provides a set of functions allowing to control dynamically the SDRAM interface.

This section contains the following APIs:

- *HAL_SDRAM_WriteProtection_Enable()*
- *HAL_SDRAM_WriteProtection_Disable()*
- *HAL_SDRAM_SendCommand()*
- *HAL_SDRAM_ProgramRefreshRate()*
- *HAL_SDRAM_SetAutoRefreshNumber()*
- *HAL_SDRAM_GetModeStatus()*

76.2.5 SDRAM State functions

This subsection permits to get in run-time the status of the SDRAM controller and the data flow.

This section contains the following APIs:

- *HAL_SDRAM_GetState()*

76.2.6 Detailed description of functions

HAL_SDRAM_Init

Function name

HAL_StatusTypeDef HAL_SDRAM_Init (SDRAM_HandleTypeDef * hsdram, FMC_SDRAM_TimingTypeDef * Timing)

Function description

Performs the SDRAM device initialization sequence.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **Timing:** Pointer to SDRAM control timing structure

Return values

- **HAL:** status

HAL_SDRAM_DeInit

Function name

HAL_StatusTypeDef HAL_SDRAM_DeInit (SDRAM_HandleTypeDef * hsdram)

Function description

Perform the SDRAM device initialization sequence.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values

- **HAL:** status

HAL_SDRAM_MspInit

Function name

void HAL_SDRAM_MspInit (SDRAM_HandleTypeDef * hsdram)

Function description

SDRAM MSP Init.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values

- **None:**

HAL_SDRAM_MspDeInit

Function name

void HAL_SDRAM_MspDeInit (SDRAM_HandleTypeDef * hsdram)

Function description

SDRAM MSP DeInit.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values

- **None:**

HAL_SDRAM_IRQHandler

Function name

void HAL_SDRAM_IRQHandler (SDRAM_HandleTypeDef * hsdram)

Function description

This function handles SDRAM refresh error interrupt request.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values

- **HAL:** status

HAL_SDRAM_RefreshErrorCallback

Function name

void HAL_SDRAM_RefreshErrorCallback (SDRAM_HandleTypeDef * hsdram)

Function description

SDRAM Refresh error callback.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values

- **None:**

HAL_SDRAM_DMA_XferCpltCallback

Function name

void HAL_SDRAM_DMA_XferCpltCallback (MDMA_HandleTypeDef * hmdma)

Function description

DMA transfer complete callback.

Parameters

- **hmdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

HAL_SDRAM_DMA_XferErrorCallback

Function name

void HAL_SDRAM_DMA_XferErrorCallback (MDMA_HandleTypeDef * hmdma)

Function description

DMA transfer complete error callback.

Parameters

- **hmdma:** DMA handle

Return values

- **None:**

HAL_SDRAM_Read_8b

Function name

HAL_StatusTypeDef HAL_SDRAM_Read_8b (SDRAM_HandleTypeDef * hsdram, uint32_t * pAddress, uint8_t * pDstBuffer, uint32_t BufferSize)

Function description

Reads 8-bit data buffer from the SDRAM memory.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SDRAM_Write_8b

Function name

HAL_StatusTypeDef HAL_SDRAM_Write_8b (SDRAM_HandleTypeDef * hsdram, uint32_t * pAddress, uint8_t * pSrcBuffer, uint32_t BufferSize)

Function description

Writes 8-bit data buffer to SDRAM memory.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SDRAM_Read_16b

Function name

HAL_StatusTypeDef HAL_SDRAM_Read_16b (SDRAM_HandleTypeDef * hsdram, uint32_t * pAddress, uint16_t * pDstBuffer, uint32_t BufferSize)

Function description

Reads 16-bit data buffer from the SDRAM memory.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SDRAM_Write_16b

Function name

HAL_StatusTypeDef HAL_SDRAM_Write_16b (SDRAM_HandleTypeDef * hsdram, uint32_t * pAddress, uint16_t * pSrcBuffer, uint32_t BufferSize)

Function description

Writes 16-bit data buffer to SDRAM memory.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SDRAM_Read_32b

Function name

HAL_StatusTypeDef HAL_SDRAM_Read_32b (SDRAM_HandleTypeDef * hsdram, uint32_t * pAddress, uint32_t * pDstBuffer, uint32_t BufferSize)

Function description

Reads 32-bit data buffer from the SDRAM memory.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SDRAM_Write_32b

Function name

HAL_StatusTypeDef HAL_SDRAM_Write_32b (SDRAM_HandleTypeDef * hsdram, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)

Function description

Writes 32-bit data buffer to SDRAM memory.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SDRAM_Read_DMA

Function name

HAL_StatusTypeDef HAL_SDRAM_Read_DMA (SDRAM_HandleTypeDef * hsdram, uint32_t * pAddress, uint32_t * pDstBuffer, uint32_t BufferSize)

Function description

Reads a Words data from the SDRAM memory using DMA transfer.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SDRAM_Write_DMA

Function name

HAL_StatusTypeDef HAL_SDRAM_Write_DMA (SDRAM_HandleTypeDef * hsdram, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)

Function description

Writes a Words data buffer to SDRAM memory using DMA transfer.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SDRAM_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SDRAM_RegisterCallback (SDRAM_HandleTypeDef * hsdram, HAL_SDRAM_CallbackIDTypeDef CallbackId, pSDRAM_CallbackTypeDef pCallback)

Function description

Register a User SDRAM Callback To be used instead of the weak (surcharged) predefined callback.

Parameters

- **hsdram:** : SDRAM handle
- **CallbackId:** : ID of the callback to be registered This parameter can be one of the following values:
  - HAL_SDRAM_MSP_INIT_CB_ID SDRAM MspInit callback ID
  - HAL_SDRAM_MSP_DEINIT_CB_ID SDRAM MspDeInit callback ID
  - HAL_SDRAM_REFRESH_ERR_CB_ID SDRAM Refresh Error callback ID
- **pCallback:** : pointer to the Callback function

Return values

- **status:**

HAL_SDRAM_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SDRAM_UnRegisterCallback (SDRAM_HandleTypeDef * hsdram, HAL_SDRAM_CallbackIDTypeDef CallbackId)

Function description

Unregister a User SDRAM Callback SDRAM Callback is redirected to the weak (surcharged) predefined callback.

Parameters

- **hsdram:** : SDRAM handle
- **CallbackId:** : ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_SDRAM_MSP_INIT_CB_ID SDRAM MspInit callback ID
  - HAL_SDRAM_MSP_DEINIT_CB_ID SDRAM MspDeInit callback ID
  - HAL_SDRAM_REFRESH_ERR_CB_ID SDRAM Refresh Error callback ID
  - HAL_SDRAM_DMA_XFER_CPLT_CB_ID SDRAM DMA Xfer Complete callback ID
  - HAL_SDRAM_DMA_XFER_ERR_CB_ID SDRAM DMA Xfer Error callback ID

Return values

- **status:**

HAL_SDRAM_RegisterDmaCallback

Function name

HAL_StatusTypeDef HAL_SDRAM_RegisterDmaCallback (SDRAM_HandleTypeDef * hsdram, HAL_SDRAM_CallbackIDTypeDef CallbackId, pSDRAM_DmaCallbackTypeDef pCallback)

Function description

Register a User SDRAM Callback for DMA transfers To be used instead of the weak (surcharged) predefined callback.

Parameters

- **hsdram:** : SDRAM handle
- **CallbackId:** : ID of the callback to be registered This parameter can be one of the following values:
  - HAL_SDRAM_DMA_XFER_CPLT_CB_ID SDRAM DMA Xfer Complete callback ID
  - HAL_SDRAM_DMA_XFER_ERR_CB_ID SDRAM DMA Xfer Error callback ID
- **pCallback:** : pointer to the Callback function

Return values

- **status:**

HAL_SDRAM_WriteProtection_Enable

Function name

HAL_StatusTypeDef HAL_SDRAM_WriteProtection_Enable (SDRAM_HandleTypeDef * hsdram)

Function description

Enables dynamically SDRAM write protection.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values

- **HAL:** status

HAL_SDRAM_WriteProtection_Disable

Function name

HAL_StatusTypeDef HAL_SDRAM_WriteProtection_Disable (SDRAM_HandleTypeDef * hsdram)

Function description

Disables dynamically SDRAM write protection.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values

- **HAL:** status

HAL_SDRAM_SendCommand

Function name

HAL_StatusTypeDef HAL_SDRAM_SendCommand (SDRAM_HandleTypeDef * hsdram, FMC_SDRAM_CommandTypeDef * Command, uint32_t Timeout)

Function description

Sends Command to the SDRAM bank.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **Command:** SDRAM command structure
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SDRAM_ProgramRefreshRate

Function name

HAL_StatusTypeDef HAL_SDRAM_ProgramRefreshRate (SDRAM_HandleTypeDef * hsdram, uint32_t RefreshRate)

Function description

Programs the SDRAM Memory Refresh rate.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **RefreshRate:** The SDRAM refresh rate value

Return values

- **HAL:** status

HAL_SDRAM_SetAutoRefreshNumber

Function name

HAL_StatusTypeDef HAL_SDRAM_SetAutoRefreshNumber (SDRAM_HandleTypeDef * hsdram, uint32_t AutoRefreshNumber)

Function description

Sets the Number of consecutive SDRAM Memory auto Refresh commands.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **AutoRefreshNumber:** The SDRAM auto Refresh number

Return values

- **HAL:** status

HAL_SDRAM_GetModeStatus

Function name

uint32_t HAL_SDRAM_GetModeStatus (SDRAM_HandleTypeDef * hsdram)

Function description

Returns the SDRAM memory current mode.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values

- **The:** SDRAM memory mode.

HAL_SDRAM_GetState

Function name

HAL_SDRAM_StateTypeDef HAL_SDRAM_GetState (SDRAM_HandleTypeDef * hsdram)

Function description

Returns the SDRAM state.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values

- **HAL:** state

76.3 SDRAM Firmware driver defines

The following section lists the various define and macros of the module.

76.3.1 SDRAM

SDRAM

SDRAM Exported Macros

__HAL_SDRAM_RESET_HANDLE_STATE

Description:

- Reset SDRAM handle state.

Parameters:

- __HANDLE__: specifies the SDRAM handle.

Return value:

- None

77 HAL SD Generic Driver

77.1 SD Firmware driver registers structures

77.1.1 HAL_SD_CardInfoTypeDef

HAL_SD_CardInfoTypeDef is defined in the stm32h7xx_hal_sd.h

Data Fields

- ***uint32_t CardType***
- ***uint32_t CardVersion***
- ***uint32_t Class***
- ***uint32_t RelCardAdd***
- ***uint32_t BlockNbr***
- ***uint32_t BlockSize***
- ***uint32_t LogBlockNbr***
- ***uint32_t LogBlockSize***
- ***uint32_t CardSpeed***

Field Documentation

- ***uint32_t HAL_SD_CardInfoTypeDef::CardType***
  Specifies the card Type
- ***uint32_t HAL_SD_CardInfoTypeDef::CardVersion***
  Specifies the card version
- ***uint32_t HAL_SD_CardInfoTypeDef::Class***
  Specifies the class of the card class
- ***uint32_t HAL_SD_CardInfoTypeDef::RelCardAdd***
  Specifies the Relative Card Address
- ***uint32_t HAL_SD_CardInfoTypeDef::BlockNbr***
  Specifies the Card Capacity in blocks
- ***uint32_t HAL_SD_CardInfoTypeDef::BlockSize***
  Specifies one block size in bytes
- ***uint32_t HAL_SD_CardInfoTypeDef::LogBlockNbr***
  Specifies the Card logical Capacity in blocks
- ***uint32_t HAL_SD_CardInfoTypeDef::LogBlockSize***
  Specifies logical block size in bytes
- ***uint32_t HAL_SD_CardInfoTypeDef::CardSpeed***
  Specifies the card Speed

77.1.2 __SD_HandleTypeDef

__SD_HandleTypeDef is defined in the stm32h7xx_hal_sd.h

Data Fields

- ***SD_TypeDef * Instance***
- ***SD_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***const uint8_t * pTxBuffPtr***
- ***uint32_t TxXferSize***
- ***uint8_t * pRxBuffPtr***
- ***uint32_t RxXferSize***

- ***__IO uint32_t Context***
- ***__IO HAL_SD_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***HAL_SD_CardInfoTypeDef SdCard***
- ***uint32_t CSD***
- ***uint32_t CID***
- ***void(* TxCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* Read_DMADblBuf0CpltCallback***
- ***void(* Read_DMADblBuf1CpltCallback***
- ***void(* Write_DMADblBuf0CpltCallback***
- ***void(* Write_DMADblBuf1CpltCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***SD_TypeDef* __SD_HandleTypeDef::Instance***
  SD registers base address
- ***SD_InitTypeDef __SD_HandleTypeDef::Init***
  SD required parameters
- ***HAL_LockTypeDef __SD_HandleTypeDef::Lock***
  SD locking object
- ***const uint8_t* __SD_HandleTypeDef::pTxBuffPtr***
  Pointer to SD Tx transfer Buffer
- ***uint32_t __SD_HandleTypeDef::TxXferSize***
  SD Tx Transfer size
- ***uint8_t* __SD_HandleTypeDef::pRxBuffPtr***
  Pointer to SD Rx transfer Buffer
- ***uint32_t __SD_HandleTypeDef::RxXferSize***
  SD Rx Transfer size
- ***__IO uint32_t __SD_HandleTypeDef::Context***
  SD transfer context
- ***__IO HAL_SD_StateTypeDef __SD_HandleTypeDef::State***
  SD card State
- ***__IO uint32_t __SD_HandleTypeDef::ErrorCode***
  SD Card Error codes
- ***HAL_SD_CardInfoTypeDef __SD_HandleTypeDef::SdCard***
  SD Card information
- ***uint32_t __SD_HandleTypeDef::CSD[4]***
  SD card specific data table
- ***uint32_t __SD_HandleTypeDef::CID[4]***
  SD card identification number table
- ***void(* __SD_HandleTypeDef::TxCpltCallback)(struct __SD_HandleTypeDef *hsd)***
- ***void(* __SD_HandleTypeDef::RxCpltCallback)(struct __SD_HandleTypeDef *hsd)***
- ***void(* __SD_HandleTypeDef::ErrorCallback)(struct __SD_HandleTypeDef *hsd)***
- ***void(* __SD_HandleTypeDef::AbortCpltCallback)(struct __SD_HandleTypeDef *hsd)***
- ***void(* __SD_HandleTypeDef::Read_DMADblBuf0CpltCallback)(struct __SD_HandleTypeDef *hsd)***

- *void(* __SD_HandleTypeDef::Read_DMADblBuf1CpltCallback)(struct __SD_HandleTypeDef *hsd)*
- *void(* __SD_HandleTypeDef::Write_DMADblBuf0CpltCallback)(struct __SD_HandleTypeDef *hsd)*
- *void(* __SD_HandleTypeDef::Write_DMADblBuf1CpltCallback)(struct __SD_HandleTypeDef *hsd)*
- *void(* __SD_HandleTypeDef::MspInitCallback)(struct __SD_HandleTypeDef *hsd)*
- *void(* __SD_HandleTypeDef::MspDeInitCallback)(struct __SD_HandleTypeDef *hsd)*

77.1.3 HAL_SD_CardCSDTypeDef

HAL_SD_CardCSDTypeDef is defined in the stm32h7xx_hal_sd.h

Data Fields

- *__IO uint8_t CSDStruct*
- *__IO uint8_t SysSpecVersion*
- *__IO uint8_t Reserved1*
- *__IO uint8_t TAAC*
- *__IO uint8_t NSAC*
- *__IO uint8_t MaxBusClkFrec*
- *__IO uint16_t CardComdClasses*
- *__IO uint8_t RdBlockLen*
- *__IO uint8_t PartBlockRead*
- *__IO uint8_t WrBlockMisalign*
- *__IO uint8_t RdBlockMisalign*
- *__IO uint8_t DSRImpl*
- *__IO uint8_t Reserved2*
- *__IO uint32_t DeviceSize*
- *__IO uint8_t MaxRdCurrentVDDMin*
- *__IO uint8_t MaxRdCurrentVDDMax*
- *__IO uint8_t MaxWrCurrentVDDMin*
- *__IO uint8_t MaxWrCurrentVDDMax*
- *__IO uint8_t DeviceSizeMul*
- *__IO uint8_t EraseGrSize*
- *__IO uint8_t EraseGrMul*
- *__IO uint8_t WrProtectGrSize*
- *__IO uint8_t WrProtectGrEnable*
- *__IO uint8_t ManDeflECC*
- *__IO uint8_t WrSpeedFact*
- *__IO uint8_t MaxWrBlockLen*
- *__IO uint8_t WriteBlockPaPartial*
- *__IO uint8_t Reserved3*
- *__IO uint8_t ContentProtectAppli*
- *__IO uint8_t FileFormatGroup*
- *__IO uint8_t CopyFlag*
- *__IO uint8_t PermWrProtect*
- *__IO uint8_t TempWrProtect*
- *__IO uint8_t FileFormat*
- *__IO uint8_t ECC*
- *__IO uint8_t CSD_CRC*
- *__IO uint8_t Reserved4*

Field Documentation

- ***__IO uint8_t HAL_SD_CardCSDTypeDef::CSDStruct***
  CSD structure
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::SysSpecVersion***
  System specification version
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::Reserved1***
  Reserved
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::TAAC***
  Data read access time 1
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::NSAC***
  Data read access time 2 in CLK cycles
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::MaxBusClkFrec***
  Max. bus clock frequency
- ***__IO uint16_t HAL_SD_CardCSDTypeDef::CardComdClasses***
  Card command classes
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::RdBlockLen***
  Max. read data block length
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::PartBlockRead***
  Partial blocks for read allowed
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::WrBlockMisalign***
  Write block misalignment
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::RdBlockMisalign***
  Read block misalignment
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::DSRImpl***
  DSR implemented
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::Reserved2***
  Reserved
- ***__IO uint32_t HAL_SD_CardCSDTypeDef::DeviceSize***
  Device Size
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::MaxRdCurrentVDDMin***
  Max. read current @ VDD min
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::MaxRdCurrentVDDMax***
  Max. read current @ VDD max
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::MaxWrCurrentVDDMin***
  Max. write current @ VDD min
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::MaxWrCurrentVDDMax***
  Max. write current @ VDD max
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::DeviceSizeMul***
  Device size multiplier
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::EraseGrSize***
  Erase group size
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::EraseGrMul***
  Erase group size multiplier
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::WrProtectGrSize***
  Write protect group size
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::WrProtectGrEnable***
  Write protect group enable
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::ManDeflECC***
  Manufacturer default ECC
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::WrSpeedFact***
  Write speed factor

- ***__IO uint8_t HAL_SD_CardCSDTypeDef::MaxWrBlockLen***
  Max. write data block length
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::WriteBlockPaPartial***
  Partial blocks for write allowed
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::Reserved3***
  Reserved
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::ContentProtectAppli***
  Content protection application
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::FileFormatGroup***
  File format group
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::CopyFlag***
  Copy flag (OTP)
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::PermWrProtect***
  Permanent write protection
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::TempWrProtect***
  Temporary write protection
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::FileFormat***
  File format
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::ECC***
  ECC code
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::CSD_CRC***
  CSD CRC
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::Reserved4***
  Always 1

77.1.4 HAL_SD_CardCIDTypeDef

HAL_SD_CardCIDTypeDef is defined in the stm32h7xx_hal_sd.h

Data Fields

- ***__IO uint8_t ManufacturerID***
- ***__IO uint16_t OEM_AppliID***
- ***__IO uint32_t ProdName1***
- ***__IO uint8_t ProdName2***
- ***__IO uint8_t ProdRev***
- ***__IO uint32_t ProdSN***
- ***__IO uint8_t Reserved1***
- ***__IO uint16_t ManufactDate***
- ***__IO uint8_t CID_CRC***
- ***__IO uint8_t Reserved2***

Field Documentation

- ***__IO uint8_t HAL_SD_CardCIDTypeDef::ManufacturerID***
  Manufacturer ID
- ***__IO uint16_t HAL_SD_CardCIDTypeDef::OEM_AppliID***
  OEM/Application ID
- ***__IO uint32_t HAL_SD_CardCIDTypeDef::ProdName1***
  Product Name part1
- ***__IO uint8_t HAL_SD_CardCIDTypeDef::ProdName2***
  Product Name part2
- ***__IO uint8_t HAL_SD_CardCIDTypeDef::ProdRev***
  Product Revision

- ***__IO uint32_t HAL_SD_CardCIDTypeDef::ProdSN***
  Product Serial Number
- ***__IO uint8_t HAL_SD_CardCIDTypeDef::Reserved1***
  Reserved1
- ***__IO uint16_t HAL_SD_CardCIDTypeDef::ManufactDate***
  Manufacturing Date
- ***__IO uint8_t HAL_SD_CardCIDTypeDef::CID_CRC***
  CID CRC
- ***__IO uint8_t HAL_SD_CardCIDTypeDef::Reserved2***
  Always 1

77.1.5 HAL_SD_CardStatusTypeDef

HAL_SD_CardStatusTypeDef is defined in the stm32h7xx_hal_sd.h

Data Fields

- ***__IO uint8_t DataBusWidth***
- ***__IO uint8_t SecuredMode***
- ***__IO uint16_t CardType***
- ***__IO uint32_t ProtectedAreaSize***
- ***__IO uint8_t SpeedClass***
- ***__IO uint8_t PerformanceMove***
- ***__IO uint8_t AllocationUnitSize***
- ***__IO uint16_t EraseSize***
- ***__IO uint8_t EraseTimeout***
- ***__IO uint8_t EraseOffset***
- ***__IO uint8_t UhsSpeedGrade***
- ***__IO uint8_t UhsAllocationUnitSize***
- ***__IO uint8_t VideoSpeedClass***

Field Documentation

- ***__IO uint8_t HAL_SD_CardStatusTypeDef::DataBusWidth***
  Shows the currently defined data bus width
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::SecuredMode***
  Card is in secured mode of operation
- ***__IO uint16_t HAL_SD_CardStatusTypeDef::CardType***
  Carries information about card type
- ***__IO uint32_t HAL_SD_CardStatusTypeDef::ProtectedAreaSize***
  Carries information about the capacity of protected area
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::SpeedClass***
  Carries information about the speed class of the card
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::PerformanceMove***
  Carries information about the card's performance move
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::AllocationUnitSize***
  Carries information about the card's allocation unit size
- ***__IO uint16_t HAL_SD_CardStatusTypeDef::EraseSize***
  Determines the number of AUs to be erased in one operation
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::EraseTimeout***
  Determines the timeout for any number of AU erase
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::EraseOffset***
  Carries information about the erase offset

- ***__IO uint8_t HAL_SD_CardStatusTypeDef::UhsSpeedGrade***
  Carries information about the speed grade of UHS card
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::UhsAllocationUnitSize***
  Carries information about the UHS card's allocation unit size
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::VideoSpeedClass***
  Carries information about the Video Speed Class of UHS card

77.2 SD Firmware driver API description

The following section lists the various functions of the SD library.

77.2.1 How to use this driver

This driver implements a high level communication layer for read and write from/to this memory. The needed STM32 hardware resources (SDMMC and GPIO) are performed by the user in HAL_SD_MspInit() function (MSP layer). Basically, the MSP layer configuration should be the same as we provide in the examples. You can easily tailor this configuration according to hardware resources.

This driver is a generic layered driver for SDMMC memories which uses the HAL SDMMC driver functions to interface with SD and uSD cards devices. It is used as follows:

1. Initialize the SDMMC low level resources by implementing the HAL_SD_MspInit() API:
   a. Enable the SDMMC interface clock using __HAL_RCC_SDMMC_CLK_ENABLE();
   b. SDMMC pins configuration for SD card
      - Enable the clock for the SDMMC GPIOs using the functions __HAL_RCC_GPIOx_CLK_ENABLE();
      - Configure these SDMMC pins as alternate function pull-up using HAL_GPIO_Init() and according to your pin assignment;
   c. NVIC configuration if you need to use interrupt process (HAL_SD_ReadBlocks_IT() and HAL_SD_WriteBlocks_IT() APIs).
      - Configure the SDMMC interrupt priorities using function HAL_NVIC_SetPriority();
      - Enable the NVIC SDMMC IRQs using function HAL_NVIC_EnableIRQ()
      - SDMMC interrupts are managed using the macros __HAL_SD_ENABLE_IT() and __HAL_SD_DISABLE_IT() inside the communication process.
      - SDMMC interrupts pending bits are managed using the macros __HAL_SD_GET_IT() and __HAL_SD_CLEAR_IT()
   d. No general propose DMA Configuration is needed, an Internal DMA for SDMMC Peripheral are used.
2. At this stage, you can perform SD read/write/erase operations after SD card initialization

SD Card Initialization and configuration

To initialize the SD Card, use the HAL_SD_Init() function. It Initializes SDMMC Peripheral(STM32 side) and the SD Card, and put it into StandBy State (Ready for data transfer). This function provide the following operations:

1. Apply the SD Card initialization process at 400KHz and check the SD Card type (Standard Capacity or High Capacity). You can change or adapt this frequency by adjusting the "ClockDiv" field. The SD Card frequency (SDMMC_CK) is computed as follows: SDMMC_CK = SDMMCCLK / (2 * ClockDiv) In initialization mode and according to the SD Card standard, make sure that the SDMMC_CK frequency doesn't exceed 400KHz. This phase of initialization is done through SDMMC_Init() and SDMMC_PowerState_ON() SDMMC low level APIs.
2. Initialize the SD card. The API used is HAL_SD_InitCard(). This phase allows the card initialization and identification and check the SD Card type (Standard Capacity or High Capacity) The initialization flow is compatible with SD standard. This API (HAL_SD_InitCard()) could be used also to reinitialize the card in case of plug-off plug-in.
3. Configure the SD Card Data transfer frequency. You can change or adapt this frequency by adjusting the "ClockDiv" field. In transfer mode and according to the SD Card standard, make sure that the SDMMC_CK frequency doesn't exceed 25MHz and 100MHz in High-speed mode switch.
4. Select the corresponding SD Card according to the address read with the step 2.
5. Configure the SD Card in wide bus mode: 4-bits data.

SD Card Read operation

- You can read from SD card in polling mode by using function HAL_SD_ReadBlocks(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_SD_GetCardState() function for SD card state.
- You can read from SD card in DMA mode by using function HAL_SD_ReadBlocks_DMA(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_SD_GetCardState() function for SD card state. You could also check the DMA transfer process through the SD Rx interrupt event.
- You can read from SD card in Interrupt mode by using function HAL_SD_ReadBlocks_IT(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_SD_GetCardState() function for SD card state. You could also check the IT transfer process through the SD Rx interrupt event.

SD Card Write operation

- You can write to SD card in polling mode by using function HAL_SD_WriteBlocks(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_SD_GetCardState() function for SD card state.
- You can write to SD card in DMA mode by using function HAL_SD_WriteBlocks_DMA(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_SD_GetCardState() function for SD card state. You could also check the DMA transfer process through the SD Tx interrupt event.
- You can write to SD card in Interrupt mode by using function HAL_SD_WriteBlocks_IT(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_SD_GetCardState() function for SD card state. You could also check the IT transfer process through the SD Tx interrupt event.

SD card status

- The SD Status contains status bits that are related to the SD Memory Card proprietary features. To get SD card status use the HAL_SD_GetCardStatus().

SD card information

- To get SD card information, you can use the function HAL_SD_GetCardInfo(). It returns useful information about the SD card such as block size, card type, block number ...

SD card CSD register

SD card CID register

SD HAL driver macros list

Note: You can refer to the SD HAL driver header file for more useful macros

Callback registration

The compilation define USE_HAL_SD_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_SD_RegisterCallback() to register a user callback, it allows to register following callbacks:

- TxCpltCallback : callback when a transmission transfer is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- Read_DMADblBuf0CpltCallback : callback when the DMA reception of first buffer is completed.
- Read_DMADblBuf1CpltCallback : callback when the DMA reception of second buffer is completed.
- Write_DMADblBuf0CpltCallback : callback when the DMA transmission of first buffer is completed.
- Write_DMADblBuf1CpltCallback : callback when the DMA transmission of second buffer is completed.
- MspInitCallback : SD MspInit.
- MspDeInitCallback : SD MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. For specific callbacks TransceiverCallback use dedicated register callbacks: respectively HAL_SD_RegisterTransceiverCallback(). Use function HAL_SD_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. It allows to reset following callbacks:
- TxCpltCallback : callback when a transmission transfer is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- Read_DMADblBuf0CpltCallback : callback when the DMA reception of first buffer is completed.
- Read_DMADblBuf1CpltCallback : callback when the DMA reception of second buffer is completed.
- Write_DMADblBuf0CpltCallback : callback when the DMA transmission of first buffer is completed.
- Write_DMADblBuf1CpltCallback : callback when the DMA transmission of second buffer is completed.
- MspInitCallback : SD MspInit.
- MspDeInitCallback : SD MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. For specific callbacks TransceiverCallback use dedicated unregister callbacks: respectively HAL_SD_UnRegisterTransceiverCallback(). By default, after the HAL_SD_Init and if the state is HAL_SD_STATE_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the HAL_SD_Init and HAL_SD_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_SD_Init and HAL_SD_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_SD_RegisterCallback before calling HAL_SD_DeInit or HAL_SD_Init function. When The compilation define USE_HAL_SD_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

77.2.2 Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the SD card device to be ready for use.

This section contains the following APIs:

- ***HAL_SD_Init()***
- ***HAL_SD_InitCard()***
- ***HAL_SD_DeInit()***
- ***HAL_SD_MspInit()***
- ***HAL_SD_MspDeInit()***

77.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the data transfer from/to SD card.

This section contains the following APIs:

- *HAL_SD_ReadBlocks()*
- *HAL_SD_WriteBlocks()*
- *HAL_SD_ReadBlocks_IT()*
- *HAL_SD_WriteBlocks_IT()*
- *HAL_SD_ReadBlocks_DMA()*
- *HAL_SD_WriteBlocks_DMA()*
- *HAL_SD_Erase()*
- *HAL_SD_IRQHandler()*
- *HAL_SD_GetState()*
- *HAL_SD_GetError()*
- *HAL_SD_TxCpltCallback()*
- *HAL_SD_RxCpltCallback()*
- *HAL_SD_ErrorCallback()*
- *HAL_SD_AbortCallback()*
- *HAL_SD_RegisterCallback()*
- *HAL_SD_UnRegisterCallback()*

77.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the SD card operations and get the related information

This section contains the following APIs:

- *HAL_SD_GetCardCID()*
- *HAL_SD_GetCardCSD()*
- *HAL_SD_GetCardStatus()*
- *HAL_SD_GetCardInfo()*
- *HAL_SD_ConfigWideBusOperation()*
- *HAL_SD_ConfigSpeedBusOperation()*
- *HAL_SD_GetCardState()*
- *HAL_SD_Abort()*
- *HAL_SD_Abort_IT()*

77.2.5 Detailed description of functions

HAL_SD_Init

Function name

HAL_StatusTypeDef HAL_SD_Init (SD_HandleTypeDef * hsd)

Function description

Initializes the SD according to the specified parameters in the SD_HandleTypeDef and create the associated handle.

Parameters

- **hsd:** Pointer to the SD handle

Return values

- **HAL:** status

HAL_SD_InitCard

Function name

HAL_StatusTypeDef HAL_SD_InitCard (SD_HandleTypeDef * hsd)

Function description

Initializes the SD Card.

Parameters

- **hsd:** Pointer to SD handle

Return values

- **HAL:** status

Notes

- This function initializes the SD card. It could be used when a card re-initialization is needed.

HAL_SD_DeInit

Function name

HAL_StatusTypeDef HAL_SD_DeInit (SD_HandleTypeDef * hsd)

Function description

De-Initializes the SD card.

Parameters

- **hsd:** Pointer to SD handle

Return values

- **HAL:** status

HAL_SD_MspInit

Function name

void HAL_SD_MspInit (SD_HandleTypeDef * hsd)

Function description

Initializes the SD MSP.

Parameters

- **hsd:** Pointer to SD handle

Return values

- **None:**

HAL_SD_MspDeInit

Function name

void HAL_SD_MspDeInit (SD_HandleTypeDef * hsd)

Function description

De-Initialize SD MSP.

Parameters

- **hsd:** Pointer to SD handle

Return values

- **None:**

HAL_SD_ReadBlocks

Function name

HAL_StatusTypeDef HAL_SD_ReadBlocks (SD_HandleTypeDef * hsd, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks, uint32_t Timeout)

Function description

Reads block(s) from a specified address in a card.

Parameters

- **hsd:** Pointer to SD handle
- **pData:** pointer to the buffer that will contain the received data
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Number of SD blocks to read
- **Timeout:** Specify timeout value

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_SD_GetCardState().

HAL_SD_WriteBlocks

Function name

HAL_StatusTypeDef HAL_SD_WriteBlocks (SD_HandleTypeDef * hsd, const uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks, uint32_t Timeout)

Function description

Allows to write block(s) to a specified address in a card.

Parameters

- **hsd:** Pointer to SD handle
- **pData:** pointer to the buffer that will contain the data to transmit
- **BlockAdd:** Block Address where data will be written
- **NumberOfBlocks:** Number of SD blocks to write
- **Timeout:** Specify timeout value

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_SD_GetCardState().

HAL_SD_Erase

Function name

HAL_StatusTypeDef HAL_SD_Erase (SD_HandleTypeDef * hsd, uint32_t BlockStartAdd, uint32_t BlockEndAdd)

Function description

Erases the specified memory area of the given SD card.

Parameters

- **hsd:** Pointer to SD handle
- **BlockStartAdd:** Start Block address
- **BlockEndAdd:** End Block address

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_SD_GetCardState().

HAL_SD_ReadBlocks_IT

Function name

HAL_StatusTypeDef HAL_SD_ReadBlocks_IT (SD_HandleTypeDef * hsd, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Reads block(s) from a specified address in a card.

Parameters

- **hsd:** Pointer to SD handle
- **pData:** Pointer to the buffer that will contain the received data
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Number of blocks to read.

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_SD_GetCardState().
- You could also check the IT transfer process through the SD Rx interrupt event.

HAL_SD_WriteBlocks_IT

Function name

HAL_StatusTypeDef HAL_SD_WriteBlocks_IT (SD_HandleTypeDef * hsd, const uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Writes block(s) to a specified address in a card.

Parameters

- **hsd:** Pointer to SD handle
- **pData:** Pointer to the buffer that will contain the data to transmit
- **BlockAdd:** Block Address where data will be written
- **NumberOfBlocks:** Number of blocks to write

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_SD_GetCardState().
- You could also check the IT transfer process through the SD Tx interrupt event.

HAL_SD_ReadBlocks_DMA

Function name

HAL_StatusTypeDef HAL_SD_ReadBlocks_DMA (SD_HandleTypeDef * hsd, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Reads block(s) from a specified address in a card.

Parameters

- **hsd:** Pointer SD handle
- **pData:** Pointer to the buffer that will contain the received data
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Number of blocks to read.

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_SD_GetCardState().
- You could also check the DMA transfer process through the SD Rx interrupt event.

HAL_SD_WriteBlocks_DMA

Function name

HAL_StatusTypeDef HAL_SD_WriteBlocks_DMA (SD_HandleTypeDef * hsd, const uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Writes block(s) to a specified address in a card.

Parameters

- **hsd:** Pointer to SD handle
- **pData:** Pointer to the buffer that will contain the data to transmit
- **BlockAdd:** Block Address where data will be written
- **NumberOfBlocks:** Number of blocks to write

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_SD_GetCardState().
- You could also check the DMA transfer process through the SD Tx interrupt event.

HAL_SD_IRQHandler

Function name

void HAL_SD_IRQHandler (SD_HandleTypeDef * hsd)

Function description

This function handles SD card interrupt request.

Parameters

- **hsd:** Pointer to SD handle

Return values

- **None:**

HAL_SD_TxCpltCallback

Function name

void HAL_SD_TxCpltCallback (SD_HandleTypeDef * hsd)

Function description

Tx Transfer completed callbacks.

Parameters

- **hsd:** Pointer to SD handle

Return values

- **None:**

HAL_SD_RxCpltCallback

Function name

void HAL_SD_RxCpltCallback (SD_HandleTypeDef * hsd)

Function description

Rx Transfer completed callbacks.

Parameters

- **hsd:** Pointer SD handle

Return values

- **None:**

HAL_SD_ErrorCallback

Function name

void HAL_SD_ErrorCallback (SD_HandleTypeDef * hsd)

Function description

SD error callbacks.

Parameters

- **hsd:** Pointer SD handle

Return values

- **None:**

HAL_SD_AbortCallback

Function name

void HAL_SD_AbortCallback (SD_HandleTypeDef * hsd)

Function description

SD Abort callbacks.

Parameters

- **hsd:** Pointer SD handle

Return values

- **None:**

HAL_SD_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SD_RegisterCallback (SD_HandleTypeDef * hsd, HAL_SD_CallbackIDTypeDef CallbackID, pSD_CallbackTypeDef pCallback)

Function description

Register a User SD Callback To be used instead of the weak (surcharged) predefined callback.

Parameters

- **hsd:** : SD handle
- **CallbackID:** : ID of the callback to be registered This parameter can be one of the following values:
  - HAL_SD_TX_CPLT_CB_ID SD Tx Complete Callback ID
  - HAL_SD_RX_CPLT_CB_ID SD Rx Complete Callback ID
  - HAL_SD_ERROR_CB_ID SD Error Callback ID
  - HAL_SD_ABORT_CB_ID SD Abort Callback ID
  - HAL_SD_READ_DMA_DBL_BUF0_CPLT_CB_ID SD DMA Rx Double buffer 0 Callback ID
  - HAL_SD_READ_DMA_DBL_BUF1_CPLT_CB_ID SD DMA Rx Double buffer 1 Callback ID
  - HAL_SD_WRITE_DMA_DBL_BUF0_CPLT_CB_ID SD DMA Tx Double buffer 0 Callback ID
  - HAL_SD_WRITE_DMA_DBL_BUF1_CPLT_CB_ID SD DMA Tx Double buffer 1 Callback ID
  - HAL_SD_MSP_INIT_CB_ID SD MspInit Callback ID
  - HAL_SD_MSP_DEINIT_CB_ID SD MspDeInit Callback ID
- **pCallback:** : pointer to the Callback function

Return values

- **status:**

Notes

- The HAL_SD_RegisterCallback() may be called before HAL_SD_Init() in HAL_SD_STATE_RESET to register callbacks for HAL_SD_MSP_INIT_CB_ID and HAL_SD_MSP_DEINIT_CB_ID.

HAL_SD_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SD_UnRegisterCallback (SD_HandleTypeDef * hsd, HAL_SD_CallbackIDTypeDef CallbackID)

Function description

Unregister a User SD Callback SD Callback is redirected to the weak (surcharged) predefined callback.

Parameters

- **hsd:** : SD handle
- **CallbackID:** : ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_SD_TX_CPLT_CB_ID SD Tx Complete Callback ID
  - HAL_SD_RX_CPLT_CB_ID SD Rx Complete Callback ID
  - HAL_SD_ERROR_CB_ID SD Error Callback ID
  - HAL_SD_ABORT_CB_ID SD Abort Callback ID
  - HAL_SD_READ_DMA_DBL_BUF0_CPLT_CB_ID SD DMA Rx Double buffer 0 Callback ID
  - HAL_SD_READ_DMA_DBL_BUF1_CPLT_CB_ID SD DMA Rx Double buffer 1 Callback ID
  - HAL_SD_WRITE_DMA_DBL_BUF0_CPLT_CB_ID SD DMA Tx Double buffer 0 Callback ID
  - HAL_SD_WRITE_DMA_DBL_BUF1_CPLT_CB_ID SD DMA Tx Double buffer 1 Callback ID
  - HAL_SD_MSP_INIT_CB_ID SD MspInit Callback ID
  - HAL_SD_MSP_DEINIT_CB_ID SD MspDeInit Callback ID

Return values

- **status:**

Notes

- The HAL_SD_UnRegisterCallback() may be called before HAL_SD_Init() in HAL_SD_STATE_RESET to register callbacks for HAL_SD_MSP_INIT_CB_ID and HAL_SD_MSP_DEINIT_CB_ID.

HAL_SD_ConfigWideBusOperation

Function name

HAL_StatusTypeDef HAL_SD_ConfigWideBusOperation (SD_HandleTypeDef * hsd, uint32_t WideMode)

Function description

Enables wide bus operation for the requested card if supported by card.

Parameters

- **hsd:** Pointer to SD handle
- **WideMode:** Specifies the SD card wide bus mode This parameter can be one of the following values:
  - SDMMC_BUS_WIDE_8B: 8-bit data transfer
  - SDMMC_BUS_WIDE_4B: 4-bit data transfer
  - SDMMC_BUS_WIDE_1B: 1-bit data transfer

Return values

- **HAL:** status

HAL_SD_ConfigSpeedBusOperation

Function name

HAL_StatusTypeDef HAL_SD_ConfigSpeedBusOperation (SD_HandleTypeDef * hsd, uint32_t SpeedMode)

Function description

Configure the speed bus mode.

Parameters

- **hsd:** Pointer to the SD handle
- **SpeedMode:** Specifies the SD card speed bus mode This parameter can be one of the following values:
  - SDMMC_SPEED_MODE_AUTO: Max speed mode supported by the card
  - SDMMC_SPEED_MODE_DEFAULT: Default Speed/SDR12 mode
  - SDMMC_SPEED_MODE_HIGH: High Speed/SDR25 mode
  - SDMMC_SPEED_MODE_ULTRA: Ultra high speed mode

Return values

- **HAL:** status

HAL_SD_GetCardState

Function name

HAL_SD_CardStateTypeDef HAL_SD_GetCardState (SD_HandleTypeDef * hsd)

Function description

Gets the current sd card data state.

Parameters

- **hsd:** pointer to SD handle

Return values

- **Card:** state

HAL_SD_GetCardCID

Function name

HAL_StatusTypeDef HAL_SD_GetCardCID (SD_HandleTypeDef * hsd, HAL_SD_CardCIDTypeDef * pCID)

Function description

Returns information the information of the card which are stored on the CID register.

Parameters

- **hsd:** Pointer to SD handle
- **pCID:** Pointer to a HAL_SD_CardCIDTypeDef structure that contains all CID register parameters

Return values

- **HAL:** status

HAL_SD_GetCardCSD

Function name

HAL_StatusTypeDef HAL_SD_GetCardCSD (SD_HandleTypeDef * hsd, HAL_SD_CardCSDTypeDef * pCSD)

Function description

Returns information the information of the card which are stored on the CSD register.

Parameters

- **hsd:** Pointer to SD handle
- **pCSD:** Pointer to a HAL_SD_CardCSDTypeDef structure that contains all CSD register parameters

Return values

- **HAL:** status

HAL_SD_GetCardStatus

Function name

HAL_StatusTypeDef HAL_SD_GetCardStatus (SD_HandleTypeDef * hsd, HAL_SD_CardStatusTypeDef * pStatus)

Function description

Gets the SD status info.

Parameters

- **hsd:** Pointer to SD handle
- **pStatus:** Pointer to the HAL_SD_CardStatusTypeDef structure that will contain the SD card status information

Return values

- **HAL:** status

HAL_SD_GetCardInfo

Function name

HAL_StatusTypeDef HAL_SD_GetCardInfo (SD_HandleTypeDef * hsd, HAL_SD_CardInfoTypeDef * pCardInfo)

Function description

Gets the SD card info.

Parameters

- **hsd:** Pointer to SD handle
- **pCardInfo:** Pointer to the HAL_SD_CardInfoTypeDef structure that will contain the SD card status information

Return values

- **HAL:** status

HAL_SD_GetState

Function name

HAL_SD_StateTypeDef HAL_SD_GetState (SD_HandleTypeDef * hsd)

Function description

return the SD state

Parameters

- **hsd:** Pointer to sd handle

Return values

- **HAL:** state

HAL_SD_GetError

Function name

uint32_t HAL_SD_GetError (SD_HandleTypeDef * hsd)

Function description

Return the SD error code.

Parameters

- **hsd:** : Pointer to a SD_HandleTypeDef structure that contains the configuration information.

Return values

- **SD:** Error Code

HAL_SD_Abort

Function name

HAL_StatusTypeDef HAL_SD_Abort (SD_HandleTypeDef * hsd)

Function description

Abort the current transfer and disable the SD.

Parameters

- **hsd:** pointer to a SD_HandleTypeDef structure that contains the configuration information for SD module.

Return values

- **HAL:** status

HAL_SD_Abort_IT

Function name

HAL_StatusTypeDef HAL_SD_Abort_IT (SD_HandleTypeDef * hsd)

Function description

Abort the current transfer and disable the SD (IT mode).

Parameters

- **hsd:** pointer to a SD_HandleTypeDef structure that contains the configuration information for SD module.

Return values

- **HAL:** status

77.3 SD Firmware driver defines

The following section lists the various define and macros of the module.

77.3.1 SD

SD

SD Error status enumeration Structure definition

HAL_SD_ERROR_NONE

No error

HAL_SD_ERROR_CMD_CRC_FAIL

Command response received (but CRC check failed)

HAL_SD_ERROR_DATA_CRC_FAIL

Data block sent/received (CRC check failed)

HAL_SD_ERROR_CMD_RSP_TIMEOUT

Command response timeout

HAL_SD_ERROR_DATA_TIMEOUT

Data timeout

HAL_SD_ERROR_TX_UNDERRUN

Transmit FIFO underrun

HAL_SD_ERROR_RX_OVERRUN

Receive FIFO overrun

HAL_SD_ERROR_ADDR_MISALIGNED

Misaligned address

HAL_SD_ERROR_BLOCK_LEN_ERR

Transferred block length is not allowed for the card or the number of transferred bytes does not match the block length

HAL_SD_ERROR_ERASE_SEQ_ERR

An error in the sequence of erase command occurs

HAL_SD_ERROR_BAD_ERASE_PARAM

An invalid selection for erase groups

HAL_SD_ERROR_WRITE_PROT_VIOLATION

Attempt to program a write protect block

HAL_SD_ERROR_LOCK_UNLOCK_FAILED

Sequence or password error has been detected in unlock command or if there was an attempt to access a locked card

HAL_SD_ERROR_COM_CRC_FAILED

CRC check of the previous command failed

HAL_SD_ERROR_ILLEGAL_CMD

Command is not legal for the card state

HAL_SD_ERROR_CARD_ECC_FAILED

Card internal ECC was applied but failed to correct the data

HAL_SD_ERROR_CC_ERR

Internal card controller error

HAL_SD_ERROR_GENERAL_UNKNOWN_ERR

General or unknown error

HAL_SD_ERROR_STREAM_READ_UNDERRUN

The card could not sustain data reading in stream rmode

HAL_SD_ERROR_STREAM_WRITE_OVERRUN

The card could not sustain data programming in stream mode

HAL_SD_ERROR_CID_CSD_OVERWRITE

CID/CSD overwrite error

HAL_SD_ERROR_WP_ERASE_SKIP

Only partial address space was erased

HAL_SD_ERROR_CARD_ECC_DISABLED

Command has been executed without using internal ECC

HAL_SD_ERROR_ERASE_RESET

Erase sequence was cleared before executing because an out of erase sequence command was received

HAL_SD_ERROR_AKE_SEQ_ERR

Error in sequence of authentication

HAL_SD_ERROR_INVALID_VOLTRANGE

Error in case of invalid voltage range

HAL_SD_ERROR_ADDR_OUT_OF_RANGE

Error when addressed block is out of range

HAL_SD_ERROR_REQUEST_NOT_APPLICABLE

Error when command request is not applicable

HAL_SD_ERROR_PARAM

the used parameter is not valid

HAL_SD_ERROR_UNSUPPORTED_FEATURE

Error when feature is not insupported

HAL_SD_ERROR_BUSY

Error when transfer process is busy

HAL_SD_ERROR_DMA

Error while DMA transfer

HAL_SD_ERROR_TIMEOUT

Timeout error

HAL_SD_ERROR_INVALID_CALLBACK

Invalid callback error

SD context enumeration

SD_CONTEXT_NONE

None

SD_CONTEXT_READ_SINGLE_BLOCK

Read single block operation

SD_CONTEXT_READ_MULTIPLE_BLOCK

Read multiple blocks operation

SD_CONTEXT_WRITE_SINGLE_BLOCK

Write single block operation

SD_CONTEXT_WRITE_MULTIPLE_BLOCK

Write multiple blocks operation

SD_CONTEXT_IT

Process in Interrupt mode

SD_CONTEXT_DMA

Process in DMA mode

SD Supported Memory Cards

CARD_NORMAL_SPEED

Normal Speed Card <12.5Mo/s , Spec Version 1.01

CARD_HIGH_SPEED

High Speed Card <25Mo/s , Spec version 2.00

CARD_ULTRA_HIGH_SPEED

UHS-I SD Card <50Mo/s for SDR50, DDR5 Cards and <104Mo/s for SDR104, Spec version 3.01

CARD_SDSC

SD Standard Capacity <2Go

CARD_SDHC_SDXC

SD High Capacity <32Go, SD Extended Capacity <2To

CARD_SECURED

SD Supported Version

CARD_V1_X

CARD_V2_X

SD Exported Constants

BLOCKSIZE

Block size is 512 bytes

SD Exported Macros

__HAL_SD_RESET_HANDLE_STATE

Description:

- Reset SD handle state.

Parameters:

- __HANDLE__: SD Handle.

Return value:

- None

__HAL_SD_ENABLE_IT

Description:

- Enable the SD device interrupt.

Parameters:

- __HANDLE__: SD Handle.
- __INTERRUPT__: specifies the SDMMC interrupt sources to be enabled. This parameter can be one or a combination of the following values:
  - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  - SDMMC_IT_DTIMEOUT: Data timeout interrupt
  - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDMMC_IT_DHOLD: Data transfer Hold interrupt
  - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  - SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
  - SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
  - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  - SDMMC_IT_SDIOIT: SDIO interrupt received interrupt
  - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- None

__HAL_SD_DISABLE_IT

Description:

- Disable the SD device interrupt.

Parameters:

- __HANDLE__: SD Handle.
- __INTERRUPT__: specifies the SDMMC interrupt sources to be disabled. This parameter can be one or a combination of the following values:
  - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  - SDMMC_IT_DTIMEOUT: Data timeout interrupt
  - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDMMC_IT_DHOLD: Data transfer Hold interrupt
  - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  - SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
  - SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
  - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  - SDMMC_IT_SDIOIT: SDIO interrupt received interrupt
  - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- None

__HAL_SD_GET_FLAG

Description:

- Check whether the specified SD flag is set or not.

Parameters:

- __HANDLE__: SD Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SDMMC_FLAG_CCRCFAIL: Command response received (CRC check failed)
  - SDMMC_FLAG_DCRCFAIL: Data block sent/received (CRC check failed)
  - SDMMC_FLAG_CTIMEOUT: Command response timeout
  - SDMMC_FLAG_DTIMEOUT: Data timeout
  - SDMMC_FLAG_TXUNDERR: Transmit FIFO underrun error
  - SDMMC_FLAG_RXOVERR: Received FIFO overrun error
  - SDMMC_FLAG_CMDREND: Command response received (CRC check passed)
  - SDMMC_FLAG_CMDSENT: Command sent (no response required)
  - SDMMC_FLAG_DATAEND: Data end (data counter, DATACOUNT, is zero)
  - SDMMC_FLAG_DHOLD: Data transfer Hold
  - SDMMC_FLAG_DBCKEND: Data block sent/received (CRC check passed)
  - SDMMC_FLAG_DABORT: Data transfer aborted by CMD12
  - SDMMC_FLAG_DPSMACT: Data path state machine active
  - SDMMC_FLAG_CPSMACT: Command path state machine active
  - SDMMC_FLAG_TXFIFOHE: Transmit FIFO Half Empty
  - SDMMC_FLAG_RXFIFOHF: Receive FIFO Half Full
  - SDMMC_FLAG_TXFIFOF: Transmit FIFO full
  - SDMMC_FLAG_RXFIFOF: Receive FIFO full
  - SDMMC_FLAG_TXFIFOE: Transmit FIFO empty
  - SDMMC_FLAG_RXFIFOE: Receive FIFO empty
  - SDMMC_FLAG_BUSYD0: Inverted value of SDMMC_D0 line (Busy)
  - SDMMC_FLAG_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected
  - SDMMC_FLAG_SDIOIT: SDIO interrupt received
  - SDMMC_FLAG_ACKFAIL: Boot Acknowledgment received
  - SDMMC_FLAG_ACKTIMEOUT: Boot Acknowledgment timeout
  - SDMMC_FLAG_VSWEND: Voltage switch critical timing section completion
  - SDMMC_FLAG_CKSTOP: SDMMC_CK stopped in Voltage switch procedure
  - SDMMC_FLAG_IDMATE: IDMA transfer error
  - SDMMC_FLAG_IDMABTC: IDMA buffer transfer complete

Return value:

- The: new state of SD FLAG (SET or RESET).

__HAL_SD_CLEAR_FLAG

Description:

- Clear the SD's pending flags.

Parameters:

- __HANDLE__: SD Handle.
- __FLAG__: specifies the flag to clear. This parameter can be one or a combination of the following values:
  - SDMMC_FLAG_CCRCFAIL: Command response received (CRC check failed)
  - SDMMC_FLAG_DCRCFAIL: Data block sent/received (CRC check failed)
  - SDMMC_FLAG_CTIMEOUT: Command response timeout
  - SDMMC_FLAG_DTIMEOUT: Data timeout
  - SDMMC_FLAG_TXUNDERR: Transmit FIFO underrun error
  - SDMMC_FLAG_RXOVERR: Received FIFO overrun error
  - SDMMC_FLAG_CMDREND: Command response received (CRC check passed)
  - SDMMC_FLAG_CMDSENT: Command sent (no response required)
  - SDMMC_FLAG_DATAEND: Data end (data counter, DATACOUNT, is zero)
  - SDMMC_FLAG_DHOLD: Data transfer Hold
  - SDMMC_FLAG_DBCKEND: Data block sent/received (CRC check passed)
  - SDMMC_FLAG_DABORT: Data transfer aborted by CMD12
  - SDMMC_FLAG_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected
  - SDMMC_FLAG_SDIOIT: SDIO interrupt received
  - SDMMC_FLAG_ACKFAIL: Boot Acknowledgment received
  - SDMMC_FLAG_ACKTIMEOUT: Boot Acknowledgment timeout
  - SDMMC_FLAG_VSWEND: Voltage switch critical timing section completion
  - SDMMC_FLAG_CKSTOP: SDMMC_CK stopped in Voltage switch procedure
  - SDMMC_FLAG_IDMATE: IDMA transfer error
  - SDMMC_FLAG_IDMABTC: IDMA buffer transfer complete

Return value:

- None

__HAL_SD_GET_IT

Description:

- Check whether the specified SD interrupt has occurred or not.

Parameters:

- __HANDLE__: SD Handle.
- __INTERRUPT__: specifies the SDMMC interrupt source to check. This parameter can be one of the following values:
  - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  - SDMMC_IT_DTIMEOUT: Data timeout interrupt
  - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDMMC_IT_DHOLD: Data transfer Hold interrupt
  - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  - SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
  - SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
  - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  - SDMMC_IT_SDIOIT: SDIO interrupt received interrupt
  - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- The: new state of SD IT (SET or RESET).

__HAL_SD_CLEAR_IT

Description:

- Clear the SD's interrupt pending bits.

Parameters:

- __HANDLE__: SD Handle.
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one or a combination of the following values:
  - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  - SDMMC_IT_DTIMEOUT: Data timeout interrupt
  - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDMMC_IT_DHOLD: Data transfer Hold interrupt
  - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  - SDMMC_IT_SDIOIT: SDIO interrupt received interrupt
  - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- None

SD Card State enumeration structure

HAL_SD_CARD_READY

Card state is ready

HAL_SD_CARD_IDENTIFICATION

Card is in identification state

HAL_SD_CARD_STANDBY

Card is in standby state

HAL_SD_CARD_TRANSFER

Card is in transfer state

HAL_SD_CARD_SENDING

Card is sending an operation

HAL_SD_CARD_RECEIVING

Card is receiving operation information

HAL_SD_CARD_PROGRAMMING

Card is in programming state

HAL_SD_CARD_DISCONNECTED

Card is disconnected

HAL_SD_CARD_ERROR

Card response Error

SD Handle Structure definition

SD_InitTypeDef

SD_TypeDef

78 HAL SD Extension Driver

78.1 SDEx Firmware driver API description

The following section lists the various functions of the SDEx library.

78.1.1 How to use this driver

The SD Extension HAL driver can be used as follows:

- Configure Buffer0 and Buffer1 start address and Buffer size using HAL_SDEx_ConfigDMAMultiBuffer() function.
- Start Read and Write for multibuffer mode using HAL_SDEx_ReadBlocksDMAMultiBuffer() and HAL_SDEx_WriteBlocksDMAMultiBuffer() functions.

78.1.2 Multibuffer functions

This section provides functions allowing to configure the multibuffer mode and start read and write multibuffer mode for SD HAL driver.

This section contains the following APIs:

- *HAL_SDEx_ConfigDMAMultiBuffer()*
- *HAL_SDEx_ReadBlocksDMAMultiBuffer()*
- *HAL_SDEx_WriteBlocksDMAMultiBuffer()*
- *HAL_SDEx_ChangeDMABuffer()*
- *HAL_SDEx_Read_DMADoubleBuf0CpltCallback()*
- *HAL_SDEx_Read_DMADoubleBuf1CpltCallback()*
- *HAL_SDEx_Write_DMADoubleBuf0CpltCallback()*
- *HAL_SDEx_Write_DMADoubleBuf1CpltCallback()*

78.1.3 Detailed description of functions

HAL_SDEx_ConfigDMAMultiBuffer

Function name

HAL_StatusTypeDef HAL_SDEx_ConfigDMAMultiBuffer (SD_HandleTypeDef * hsd, uint32_t * pDataBuffer0, uint32_t * pDataBuffer1, uint32_t BufferSize)

Function description

Configure DMA Dual Buffer mode.

Parameters

- **hsd:** SD handle
- **pDataBuffer0:** Pointer to the buffer0 that will contain/receive the transferred data
- **pDataBuffer1:** Pointer to the buffer1 that will contain/receive the transferred data
- **BufferSize:** Size of Buffer0 in Blocks. Buffer0 and Buffer1 must have the same size.

Return values

- **HAL:** status

HAL_SDEx_ReadBlocksDMAMultiBuffer

Function name

HAL_StatusTypeDef HAL_SDEx_ReadBlocksDMAMultiBuffer (SD_HandleTypeDef * hsd, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Reads block(s) from a specified address in a card.

Parameters

- **hsd:** SD handle
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Total number of blocks to read

Return values

- **HAL:** status

HAL_SDEx_WriteBlocksDMAMultiBuffer

Function name

HAL_StatusTypeDef HAL_SDEx_WriteBlocksDMAMultiBuffer (SD_HandleTypeDef * hsd, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Write block(s) to a specified address in a card.

Parameters

- **hsd:** SD handle
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Total number of blocks to read

Return values

- **HAL:** status

HAL_SDEx_ChangeDMABuffer

Function name

HAL_StatusTypeDef HAL_SDEx_ChangeDMABuffer (SD_HandleTypeDef * hsd, HAL_SDEx_DMABuffer_MemoryTypeDef Buffer, uint32_t * pDataBuffer)

Function description

Change the DMA Buffer0 or Buffer1 address on the fly.

Parameters

- **hsd:** pointer to a SD_HandleTypeDef structure.
- **Buffer:** the buffer to be changed, This parameter can be one of the following values: SD_DMA_BUFFER0 or SD_DMA_BUFFER1
- **pDataBuffer:** The new address

Return values

- **HAL:** status

Notes

- The BUFFER0 address can be changed only when the current transfer use BUFFER1 and the BUFFER1 address can be changed only when the current transfer use BUFFER0.

HAL_SDEx_Read_DMADoubleBuf0CpltCallback

Function name

void HAL_SDEx_Read_DMADoubleBuf0CpltCallback (SD_HandleTypeDef * hsd)

Function description

Read DMA Buffer 0 Transfer completed callbacks.

Parameters

- **hsd:** SD handle

Return values

- **None:**

HAL_SDEx_Read_DMADoubleBuf1CpltCallback

Function name

void HAL_SDEx_Read_DMADoubleBuf1CpltCallback (SD_HandleTypeDef * hsd)

Function description

Read DMA Buffer 1 Transfer completed callbacks.

Parameters

- **hsd:** SD handle

Return values

- **None:**

HAL_SDEx_Write_DMADoubleBuf0CpltCallback

Function name

void HAL_SDEx_Write_DMADoubleBuf0CpltCallback (SD_HandleTypeDef * hsd)

Function description

Write DMA Buffer 0 Transfer completed callbacks.

Parameters

- **hsd:** SD handle

Return values

- **None:**

HAL_SDEx_Write_DMADoubleBuf1CpltCallback

Function name

void HAL_SDEx_Write_DMADoubleBuf1CpltCallback (SD_HandleTypeDef * hsd)

Function description

Write DMA Buffer 1 Transfer completed callbacks.

Parameters

- **hsd:** SD handle

Return values

- **None:**

79 HAL SMARTCARD Generic Driver

79.1 SMARTCARD Firmware driver registers structures

79.1.1 SMARTCARD_InitTypeDef

SMARTCARD_InitTypeDef is defined in the stm32h7xx_hal_smartcard.h

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t StopBits***
- ***uint16_t Parity***
- ***uint16_t Mode***
- ***uint16_t CLKPolarity***
- ***uint16_t CLKPhase***
- ***uint16_t CLKLastBit***
- ***uint16_t OneBitSampling***
- ***uint8_t Prescaler***
- ***uint8_t GuardTime***
- ***uint16_t NACKEnable***
- ***uint32_t TimeOutEnable***
- ***uint32_t TimeOutValue***
- ***uint8_t BlockLength***
- ***uint8_t AutoRetryCount***
- ***uint32_t ClockPrescaler***

Field Documentation

- ***uint32_t SMARTCARD_InitTypeDef::BaudRate***

  Configures the SmartCard communication baud rate. The baud rate register is computed using the following formula: Baud Rate Register = ((usart_ker_ckpres) / ((hsmartcard->Init.BaudRate))) where usart_ker_ckpres is the USART input clock divided by a prescaler
- ***uint32_t SMARTCARD_InitTypeDef::WordLength***

  Specifies the number of data bits transmitted or received in a frame. This parameter ***SMARTCARD_Word_Length*** can only be set to 9 (8 data + 1 parity bits).
- ***uint32_t SMARTCARD_InitTypeDef::StopBits***

  Specifies the number of stop bits. This parameter can be a value of ***SMARTCARD_Stop_Bits***.
- ***uint16_t SMARTCARD_InitTypeDef::Parity***

  Specifies the parity mode. This parameter can be a value of ***SMARTCARD_Parity***

  Note:

  – The parity is enabled by default (PCE is forced to 1). Since the WordLength is forced to 8 bits + parity, M is forced to 1 and the parity bit is the 9th bit.
- ***uint16_t SMARTCARD_InitTypeDef::Mode***

  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***SMARTCARD_Mode***
- ***uint16_t SMARTCARD_InitTypeDef::CLKPolarity***

  Specifies the steady state of the serial clock. This parameter can be a value of ***SMARTCARD_Clock_Polarity***
- ***uint16_t SMARTCARD_InitTypeDef::CLKPhase***

  Specifies the clock transition on which the bit capture is made. This parameter can be a value of ***SMARTCARD_Clock_Phase***

- ***uint16_t SMARTCARD_InitTypeDef::CLKLastBit***

  Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of ***SMARTCARD_Last_Bit***
- ***uint16_t SMARTCARD_InitTypeDef::OneBitSampling***

  Specifies whether a single sample or three samples' majority vote is selected. Selecting the single sample method increases the receiver tolerance to clock deviations. This parameter can be a value of ***SMARTCARD_OneBit_Sampling***.
- ***uint8_t SMARTCARD_InitTypeDef::Prescaler***

  Specifies the SmartCard Prescaler. This parameter can be any value from 0x01 to 0x1F. Prescaler value is multiplied by 2 to give the division factor of the source clock frequency
- ***uint8_t SMARTCARD_InitTypeDef::GuardTime***

  Specifies the SmartCard Guard Time applied after stop bits.
- ***uint16_t SMARTCARD_InitTypeDef::NACKEnable***

  Specifies whether the SmartCard NACK transmission is enabled in case of parity error. This parameter can be a value of ***SMARTCARD_NACK_Enable***
- ***uint32_t SMARTCARD_InitTypeDef::TimeOutEnable***

  Specifies whether the receiver timeout is enabled. This parameter can be a value of ***SMARTCARD_Timeout_Enable***
- ***uint32_t SMARTCARD_InitTypeDef::TimeOutValue***

  Specifies the receiver time out value in number of baud blocks: it is used to implement the Character Wait Time (CWT) and Block Wait Time (BWT). It is coded over 24 bits.
- ***uint8_t SMARTCARD_InitTypeDef::BlockLength***

  Specifies the SmartCard Block Length in T=1 Reception mode. This parameter can be any value from 0x0 to 0xFF
- ***uint8_t SMARTCARD_InitTypeDef::AutoRetryCount***

  Specifies the SmartCard auto-retry count (number of retries in receive and transmit mode). When set to 0, retransmission is disabled. Otherwise, its maximum value is 7 (before signalling an error)
- ***uint32_t SMARTCARD_InitTypeDef::ClockPrescaler***

  Specifies the prescaler value used to divide the USART clock source. This parameter can be a value of ***SMARTCARD_ClockPrescaler***.

79.1.2 SMARTCARD_AdvFeatureInitTypeDef

SMARTCARD_AdvFeatureInitTypeDef is defined in the stm32h7xx_hal_smartcard.h

Data Fields

- ***uint32_t AdvFeatureInit***
- ***uint32_t TxPinLevelInvert***
- ***uint32_t RxPinLevelInvert***
- ***uint32_t DataInvert***
- ***uint32_t Swap***
- ***uint32_t OverrunDisable***
- ***uint32_t DMADisableonRxError***
- ***uint32_t MSBFirst***
- ***uint16_t TxCompletionIndication***

Field Documentation

- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::AdvFeatureInit***

  Specifies which advanced SMARTCARD features is initialized. Several advanced features may be initialized at the same time. This parameter can be a value of ***SMARTCARDEx_Advanced_Features_Initialization_Type***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::TxPinLevelInvert***

  Specifies whether the TX pin active level is inverted. This parameter can be a value of ***SMARTCARD_Tx_Inv***

- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::RxPinLevelInvert***
  Specifies whether the RX pin active level is inverted. This parameter can be a value of ***SMARTCARD_Rx_Inv***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::DataInvert***
  Specifies whether data are inverted (positive/direct logic vs negative/inverted logic). This parameter can be a value of ***SMARTCARD_Data_Inv***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::Swap***
  Specifies whether TX and RX pins are swapped. This parameter can be a value of ***SMARTCARD_Rx_Tx_Swap***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::OverrunDisable***
  Specifies whether the reception overrun detection is disabled. This parameter can be a value of ***SMARTCARD_Overrun_Disable***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::DMADisableonRxError***
  Specifies whether the DMA is disabled in case of reception error. This parameter can be a value of ***SMARTCARD_DMA_Disable_on_Rx_Error***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::MSBFirst***
  Specifies whether MSB is sent first on UART line. This parameter can be a value of ***SMARTCARD_MSB_First***
- ***uint16_t SMARTCARD_AdvFeatureInitTypeDef::TxCompletionIndication***
  Specifies which transmission completion indication is used: before (when relevant flag is available) or once guard time period has elapsed. This parameter can be a value of ***SMARTCARDEx_Transmission_Completion_Indication***.

79.1.3 __SMARTCARD_HandleTypeDef

__SMARTCARD_HandleTypeDef is defined in the stm32h7xx_hal_smartcard.h

Data Fields

- ***USART_TypeDef * Instance***
- ***SMARTCARD_InitTypeDef Init***
- ***SMARTCARD_AdvFeatureInitTypeDef AdvancedInit***
- ***const uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***uint16_t NbRxDataToProcess***
- ***uint16_t NbTxDataToProcess***
- ***uint32_t FifoMode***
- ***void(* RxISR***
- ***void(* TxISR***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SMARTCARD_StateTypeDef gState***
- ***__IO HAL_SMARTCARD_StateTypeDef RxState***
- ***__IO uint32_t ErrorCode***
- ***void(* TxCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* AbortTransmitCpltCallback***

- ***void(* AbortReceiveCpltCallback***
- ***void(* RxFifoFullCallback***
- ***void(* TxFifoEmptyCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***USART_TypeDef* __SMARTCARD_HandleTypeDef::Instance***
  USART registers base address
- ***SMARTCARD_InitTypeDef __SMARTCARD_HandleTypeDef::Init***
  SmartCard communication parameters
- ***SMARTCARD_AdvFeatureInitTypeDef __SMARTCARD_HandleTypeDef::AdvancedInit***
  SmartCard advanced features initialization parameters
- ***const uint8_t* __SMARTCARD_HandleTypeDef::pTxBuffPtr***
  Pointer to SmartCard Tx transfer Buffer
- ***uint16_t __SMARTCARD_HandleTypeDef::TxXferSize***
  SmartCard Tx Transfer size
- ***__IO uint16_t __SMARTCARD_HandleTypeDef::TxXferCount***
  SmartCard Tx Transfer Counter
- ***uint8_t* __SMARTCARD_HandleTypeDef::pRxBuffPtr***
  Pointer to SmartCard Rx transfer Buffer
- ***uint16_t __SMARTCARD_HandleTypeDef::RxXferSize***
  SmartCard Rx Transfer size
- ***__IO uint16_t __SMARTCARD_HandleTypeDef::RxXferCount***
  SmartCard Rx Transfer Counter
- ***uint16_t __SMARTCARD_HandleTypeDef::NbRxDataToProcess***
  Number of data to process during RX ISR execution
- ***uint16_t __SMARTCARD_HandleTypeDef::NbTxDataToProcess***
  Number of data to process during TX ISR execution
- ***uint32_t __SMARTCARD_HandleTypeDef::FifoMode***
  Specifies if the FIFO mode will be used. This parameter can be a value of ***SMARTCARDEx_FIFO_mode***.
- ***void(* __SMARTCARD_HandleTypeDef::RxISR)(struct __SMARTCARD_HandleTypeDef *huart)***
  Function pointer on Rx IRQ handler
- ***void(* __SMARTCARD_HandleTypeDef::TxISR)(struct __SMARTCARD_HandleTypeDef *huart)***
  Function pointer on Tx IRQ handler
- ***DMA_HandleTypeDef* __SMARTCARD_HandleTypeDef::hdmatx***
  SmartCard Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __SMARTCARD_HandleTypeDef::hdmarx***
  SmartCard Rx DMA Handle parameters
- ***HAL_LockTypeDef __SMARTCARD_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_SMARTCARD_StateTypeDef __SMARTCARD_HandleTypeDef::gState***
  SmartCard state information related to global Handle management and also related to Tx operations. This parameter can be a value of **HAL_SMARTCARD_StateTypeDef**
- ***__IO HAL_SMARTCARD_StateTypeDef __SMARTCARD_HandleTypeDef::RxState***
  SmartCard state information related to Rx operations. This parameter can be a value of **HAL_SMARTCARD_StateTypeDef**
- ***__IO uint32_t __SMARTCARD_HandleTypeDef::ErrorCode***
  SmartCard Error code

- ***void(* __SMARTCARD_HandleTypeDef::TxCpltCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Tx Complete Callback
- ***void(* __SMARTCARD_HandleTypeDef::RxCpltCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Rx Complete Callback
- ***void(* __SMARTCARD_HandleTypeDef::ErrorCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Error Callback
- ***void(* __SMARTCARD_HandleTypeDef::AbortCpltCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Abort Complete Callback
- ***void(* __SMARTCARD_HandleTypeDef::AbortTransmitCpltCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Abort Transmit Complete Callback
- ***void(* __SMARTCARD_HandleTypeDef::AbortReceiveCpltCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Abort Receive Complete Callback
- ***void(* __SMARTCARD_HandleTypeDef::RxFifoFullCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Rx Fifo Full Callback
- ***void(* __SMARTCARD_HandleTypeDef::TxFifoEmptyCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Tx Fifo Empty Callback
- ***void(* __SMARTCARD_HandleTypeDef::MspInitCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Msp Init callback
- ***void(* __SMARTCARD_HandleTypeDef::MspDeInitCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Msp DeInit callback

79.2 SMARTCARD Firmware driver API description

The following section lists the various functions of the SMARTCARD library.

79.2.1 How to use this driver

The SMARTCARD HAL driver can be used as follows:

1. Declare a SMARTCARD_HandleTypeDef handle structure (eg. SMARTCARD_HandleTypeDef hsmartcard).
2. Associate a USART to the SMARTCARD handle hsmartcard.

3. Initialize the SMARTCARD low level resources by implementing the HAL_SMARTCARD_MspInit() API:
   - Enable the USARTx interface clock.
   - USART pins configuration:
     - Enable the clock for the USART GPIOs.
     - Configure the USART pins (TX as alternate function pull-up, RX as alternate function Input).
   - NVIC configuration if you need to use interrupt process (HAL_SMARTCARD_Transmit_IT() and HAL_SMARTCARD_Receive_IT() APIs):
     - Configure the USARTx interrupt priority.
     - Enable the NVIC USART IRQ handle.
   - DMA Configuration if you need to use DMA process (HAL_SMARTCARD_Transmit_DMA() and HAL_SMARTCARD_Receive_DMA() APIs):
     - Declare a DMA handle structure for the Tx/Rx channel.
     - Enable the DMAx interface clock.
     - Configure the declared DMA handle structure with the required Tx/Rx parameters.
     - Configure the DMA Tx/Rx channel.
     - Associate the initialized DMA handle to the SMARTCARD DMA Tx/Rx handle.
     - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
4. Program the Baud Rate, Parity, Mode(Receiver/Transmitter), clock enabling/disabling and accordingly, the clock parameters (parity, phase, last bit), prescaler value, guard time and NACK on transmission error enabling or disabling in the hsmartcard handle Init structure.
5. If required, program SMARTCARD advanced features (TX/RX pins swap, TimeOut, auto-retry counter,...) in the hsmartcard handle AdvancedInit structure.
6. Initialize the SMARTCARD registers by calling the HAL_SMARTCARD_Init() API:
   - This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_SMARTCARD_MspInit() API.

Note: The specific SMARTCARD interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_SMARTCARD_ENABLE_IT() and __HAL_SMARTCARD_DISABLE_IT() inside the transmit and receive process.

Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_SMARTCARD_Transmit()
- Receive an amount of data in blocking mode using HAL_SMARTCARD_Receive()

Interrupt mode IO operation

- Send an amount of data in non-blocking mode using HAL_SMARTCARD_Transmit_IT()
- At transmission end of transfer HAL_SMARTCARD_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_TxCpltCallback()
- Receive an amount of data in non-blocking mode using HAL_SMARTCARD_Receive_IT()
- At reception end of transfer HAL_SMARTCARD_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_RxCpltCallback()
- In case of transfer Error, HAL_SMARTCARD_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_ErrorCallback()

DMA mode IO operation

- Send an amount of data in non-blocking mode (DMA) using HAL_SMARTCARD_Transmit_DMA()
- At transmission end of transfer HAL_SMARTCARD_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_TxCpltCallback()
- Receive an amount of data in non-blocking mode (DMA) using HAL_SMARTCARD_Receive_DMA()
- At reception end of transfer HAL_SMARTCARD_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_RxCpltCallback()

- In case of transfer Error, HAL_SMARTCARD_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_ErrorCallback()

SMARTCARD HAL driver macros list

Below the list of most used macros in SMARTCARD HAL driver.

- __HAL_SMARTCARD_GET_FLAG : Check whether or not the specified SMARTCARD flag is set
- __HAL_SMARTCARD_CLEAR_FLAG : Clear the specified SMARTCARD pending flag
- __HAL_SMARTCARD_ENABLE_IT: Enable the specified SMARTCARD interrupt
- __HAL_SMARTCARD_DISABLE_IT: Disable the specified SMARTCARD interrupt
- __HAL_SMARTCARD_GET_IT_SOURCE: Check whether or not the specified SMARTCARD interrupt is enabled

Note: You can refer to the SMARTCARD HAL driver header file for more useful macros

79.2.2 Callback registration

The compilation define USE_HAL_SMARTCARD_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_SMARTCARD_RegisterCallback() to register a user callback. Function HAL_SMARTCARD_RegisterCallback() allows to register following callbacks:

- TxCpltCallback : Tx Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- AbortTransmitCpltCallback : Abort Transmit Complete Callback.
- AbortReceiveCpltCallback : Abort Receive Complete Callback.
- RxFifoFullCallback : Rx Fifo Full Callback.
- TxFifoEmptyCallback : Tx Fifo Empty Callback.
- MspInitCallback : SMARTCARD MspInit.
- MspDeInitCallback : SMARTCARD MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_SMARTCARD_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. HAL_SMARTCARD_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- TxCpltCallback : Tx Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- AbortTransmitCpltCallback : Abort Transmit Complete Callback.
- AbortReceiveCpltCallback : Abort Receive Complete Callback.
- RxFifoFullCallback : Rx Fifo Full Callback.
- TxFifoEmptyCallback : Tx Fifo Empty Callback.
- MspInitCallback : SMARTCARD MspInit.
- MspDeInitCallback : SMARTCARD MspDeInit.

By default, after the HAL_SMARTCARD_Init() and when the state is HAL_SMARTCARD_STATE_RESET all callbacks are set to the corresponding weak (surcharged) functions: examples HAL_SMARTCARD_TxCpltCallback(), HAL_SMARTCARD_RxCpltCallback(). Exception done for MspInit and MspDeInit functions that are respectively reset to the legacy weak (surcharged) functions in the HAL_SMARTCARD_Init() and HAL_SMARTCARD_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_SMARTCARD_Init() and HAL_SMARTCARD_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_SMARTCARD_STATE_READY state only. Exception done MspInit/MspDeInit that can be registered/unregistered in HAL_SMARTCARD_STATE_READY or HAL_SMARTCARD_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_SMARTCARD_RegisterCallback() before calling HAL_SMARTCARD_DeInit() or HAL_SMARTCARD_Init() function.

When The compilation define USE_HAL_SMARTCARD_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and weak (surcharged) callbacks are used.

79.2.3 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx associated to the SmartCard.

- These parameters can be configured:
  - Baud Rate
  - Parity: parity should be enabled, frame Length is fixed to 8 bits plus parity
  - Receiver/transmitter modes
  - Synchronous mode (and if enabled, phase, polarity and last bit parameters)
  - Prescaler value
  - Guard bit time
  - NACK enabling or disabling on transmission error
- The following advanced features can be configured as well:
  - TX and/or RX pin level inversion
  - data logical level inversion
  - RX and TX pins swap
  - RX overrun detection disabling
  - DMA disabling on RX error
  - MSB first on communication line
  - Time out enabling (and if activated, timeout value)
  - Block length
  - Auto-retry counter

The HAL_SMARTCARD_Init() API follows the USART synchronous configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- ***HAL_SMARTCARD_Init()***
- ***HAL_SMARTCARD_DeInit()***
- ***HAL_SMARTCARD_MspInit()***
- ***HAL_SMARTCARD_MspDeInit()***
- ***HAL_SMARTCARD_RegisterCallback()***
- ***HAL_SMARTCARD_UnRegisterCallback()***

79.2.4 IO operation functions

This subsection provides a set of functions allowing to manage the SMARTCARD data transfers.

Smartcard is a single wire half duplex communication protocol. The Smartcard interface is designed to support asynchronous protocol Smartcards as defined in the ISO 7816-3 standard. The USART should be configured as:

- 8 bits plus parity: where M=1 and PCE=1 in the USART_CR1 register
- 1.5 stop bits when transmitting and receiving: where STOP=11 in the USART_CR2 register.

1. There are two modes of transfer:
   a. Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
   b. Non-Blocking mode: The communication is performed using Interrupts or DMA, the relevant API's return the HAL status. The end of the data processing will be indicated through the dedicated SMARTCARD IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
   c. The HAL_SMARTCARD_TxCpltCallback(), HAL_SMARTCARD_RxCpltCallback() user callbacks will be executed respectively at the end of the Transmit or Receive process The HAL_SMARTCARD_ErrorCallback() user callback will be executed when a communication error is detected.
2. Blocking mode APIs are :
   a. HAL_SMARTCARD_Transmit()
   b. HAL_SMARTCARD_Receive()
3. Non Blocking mode APIs with Interrupt are :
   a. HAL_SMARTCARD_Transmit_IT()
   b. HAL_SMARTCARD_Receive_IT()
   c. HAL_SMARTCARD_IRQHandler()
4. Non Blocking mode functions with DMA are :
   a. HAL_SMARTCARD_Transmit_DMA()
   b. HAL_SMARTCARD_Receive_DMA()
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
   a. HAL_SMARTCARD_TxCpltCallback()
   b. HAL_SMARTCARD_RxCpltCallback()
   c. HAL_SMARTCARD_ErrorCallback()
1. Non-Blocking mode transfers could be aborted using Abort API's :
   a. HAL_SMARTCARD_Abort()
   b. HAL_SMARTCARD_AbortTransmit()
   c. HAL_SMARTCARD_AbortReceive()
   d. HAL_SMARTCARD_Abort_IT()
   e. HAL_SMARTCARD_AbortTransmit_IT()
   f. HAL_SMARTCARD_AbortReceive_IT()
2. For Abort services based on interrupts (HAL_SMARTCARD_Abortxxx_IT), a set of Abort Complete Callbacks are provided:
   a. HAL_SMARTCARD_AbortCpltCallback()
   b. HAL_SMARTCARD_AbortTransmitCpltCallback()
   c. HAL_SMARTCARD_AbortReceiveCpltCallback()
3. In Non-Blocking mode transfers, possible errors are split into 2 categories. Errors are handled as follows :
   a. Error is considered as Recoverable and non blocking : Transfer could go till end, but error severity is to be evaluated by user : this concerns Frame Error, Parity Error or Noise Error in Interrupt mode reception . Received character is then retrieved and stored in Rx buffer, Error code is set to allow user to identify error type, and HAL_SMARTCARD_ErrorCallback() user callback is executed. Transfer is kept ongoing on SMARTCARD side. If user wants to abort it, Abort services should be called by user.
   b. Error is considered as Blocking : Transfer could not be completed properly and is aborted. This concerns Frame Error in Interrupt mode transmission, Overrun Error in Interrupt mode reception and all errors in DMA mode. Error code is set to allow user to identify error type, and HAL_SMARTCARD_ErrorCallback() user callback is executed.

This section contains the following APIs:

- ***HAL_SMARTCARD_Transmit()***
- ***HAL_SMARTCARD_Receive()***
- ***HAL_SMARTCARD_Transmit_IT()***
- ***HAL_SMARTCARD_Receive_IT()***
- ***HAL_SMARTCARD_Transmit_DMA()***

- *HAL_SMARTCARD_Receive_DMA()*
- *HAL_SMARTCARD_Abort()*
- *HAL_SMARTCARD_AbortTransmit()*
- *HAL_SMARTCARD_AbortReceive()*
- *HAL_SMARTCARD_Abort_IT()*
- *HAL_SMARTCARD_AbortTransmit_IT()*
- *HAL_SMARTCARD_AbortReceive_IT()*
- *HAL_SMARTCARD_IRQHandler()*
- *HAL_SMARTCARD_TxCpltCallback()*
- *HAL_SMARTCARD_RxCpltCallback()*
- *HAL_SMARTCARD_ErrorCallback()*
- *HAL_SMARTCARD_AbortCpltCallback()*
- *HAL_SMARTCARD_AbortTransmitCpltCallback()*
- *HAL_SMARTCARD_AbortReceiveCpltCallback()*

79.2.5 Peripheral State and Errors functions

This subsection provides a set of functions allowing to return the State of SmartCard handle and also return Peripheral Errors occurred during communication process

- HAL_SMARTCARD_GetState() API can be helpful to check in run-time the state of the SMARTCARD peripheral.
- HAL_SMARTCARD_GetError() checks in run-time errors that could occur during communication.

This section contains the following APIs:

- *HAL_SMARTCARD_GetState()*
- *HAL_SMARTCARD_GetError()*

79.2.6 Detailed description of functions

HAL_SMARTCARD_Init

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Init (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Initialize the SMARTCARD mode according to the specified parameters in the SMARTCARD_HandleTypeDef and initialize the associated handle.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

HAL_SMARTCARD_DeInit

Function name

HAL_StatusTypeDef HAL_SMARTCARD_DeInit (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

DeInitialize the SMARTCARD peripheral.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

HAL_SMARTCARD_MspInit

Function name

void HAL_SMARTCARD_MspInit (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Initialize the SMARTCARD MSP.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_MspDeInit

Function name

void HAL_SMARTCARD_MspDeInit (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

DeInitialize the SMARTCARD MSP.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SMARTCARD_RegisterCallback (SMARTCARD_HandleTypeDef * hsmartcard, HAL_SMARTCARD_CallbackIDTypeDef CallbackID, pSMARTCARD_CallbackTypeDef pCallback)

Function description

Register a User SMARTCARD Callback To be used instead of the weak predefined callback.

Parameters

- **hsmartcard:** smartcard handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_SMARTCARD_TX_COMPLETE_CB_ID Tx Complete Callback ID
  - HAL_SMARTCARD_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_SMARTCARD_ERROR_CB_ID Error Callback ID
  - HAL_SMARTCARD_ABORT_COMPLETE_CB_ID Abort Complete Callback ID
  - HAL_SMARTCARD_ABORT_TRANSMIT_COMPLETE_CB_ID Abort Transmit Complete Callback ID
  - HAL_SMARTCARD_ABORT_RECEIVE_COMPLETE_CB_ID Abort Receive Complete Callback ID
  - HAL_SMARTCARD_RX_FIFO_FULL_CB_ID Rx Fifo Full Callback ID
  - HAL_SMARTCARD_TX_FIFO_EMPTY_CB_ID Tx Fifo Empty Callback ID
  - HAL_SMARTCARD_MSPINIT_CB_ID MspInit Callback ID
  - HAL_SMARTCARD_MSPDEINIT_CB_ID MspDeInit Callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

Notes

- The HAL_SMARTCARD_RegisterCallback() may be called before HAL_SMARTCARD_Init() in HAL_SMARTCARD_STATE_RESET to register callbacks for HAL_SMARTCARD_MSPINIT_CB_ID and HAL_SMARTCARD_MSPDEINIT_CB_ID

HAL_SMARTCARD_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SMARTCARD_UnRegisterCallback (SMARTCARD_HandleTypeDef * hsmartcard, HAL_SMARTCARD_CallbackIDTypeDef CallbackID)

Function description

Unregister an SMARTCARD callback SMARTCARD callback is redirected to the weak predefined callback.

Parameters

- **hsmartcard:** smartcard handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_SMARTCARD_TX_COMPLETE_CB_ID Tx Complete Callback ID
  - HAL_SMARTCARD_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_SMARTCARD_ERROR_CB_ID Error Callback ID
  - HAL_SMARTCARD_ABORT_COMPLETE_CB_ID Abort Complete Callback ID
  - HAL_SMARTCARD_ABORT_TRANSMIT_COMPLETE_CB_ID Abort Transmit Complete Callback ID
  - HAL_SMARTCARD_ABORT_RECEIVE_COMPLETE_CB_ID Abort Receive Complete Callback ID
  - HAL_SMARTCARD_RX_FIFO_FULL_CB_ID Rx Fifo Full Callback ID
  - HAL_SMARTCARD_TX_FIFO_EMPTY_CB_ID Tx Fifo Empty Callback ID
  - HAL_SMARTCARD_MSPINIT_CB_ID MspInit Callback ID
  - HAL_SMARTCARD_MSPDEINIT_CB_ID MspDeInit Callback ID

Return values

- **HAL:** status

Notes

- The HAL_SMARTCARD_UnRegisterCallback() may be called before HAL_SMARTCARD_Init() in HAL_SMARTCARD_STATE_RESET to un-register callbacks for HAL_SMARTCARD_MSPINIT_CB_ID and HAL_SMARTCARD_MSPDEINIT_CB_ID

HAL_SMARTCARD_Transmit

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Transmit (SMARTCARD_HandleTypeDef * hsmartcard, const uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Send an amount of data in blocking mode.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be sent.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- When FIFO mode is enabled, writing a data in the TDR register adds one data to the TXFIFO. Write operations to the TDR register are performed when TXFNF flag is set. From hardware perspective, TXFNF flag and TXE are mapped on the same bit-field.

HAL_SMARTCARD_Receive

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Receive (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be received.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- When FIFO mode is enabled, the RXFNE flag is set as long as the RXFIFO is not empty. Read operations from the RDR register are performed when RXFNE flag is set. From hardware perspective, RXFNE flag and RXNE are mapped on the same bit-field.

HAL_SMARTCARD_Transmit_IT

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Transmit_IT (SMARTCARD_HandleTypeDef * hsmartcard, const uint8_t * pData, uint16_t Size)

Function description

Send an amount of data in interrupt mode.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be sent.

Return values

- **HAL:** status

Notes

- When FIFO mode is disabled, USART interrupt is generated whenever USART_TDR register is empty, i.e one interrupt per data to transmit.
- When FIFO mode is enabled, USART interrupt is generated whenever TXFIFO threshold reached. In that case the interrupt rate depends on TXFIFO threshold configuration.
- This function sets the hsmartcard->TxIsr function pointer according to the FIFO mode (data transmission processing depends on FIFO mode).

HAL_SMARTCARD_Receive_IT

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Receive_IT (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in interrupt mode.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be received.

Return values

- **HAL:** status

Notes

- When FIFO mode is disabled, USART interrupt is generated whenever USART_RDR register can be read, i.e one interrupt per data to receive.
- When FIFO mode is enabled, USART interrupt is generated whenever RXFIFO threshold reached. In that case the interrupt rate depends on RXFIFO threshold configuration.
- This function sets the hsmartcard->RxIsr function pointer according to the FIFO mode (data reception processing depends on FIFO mode).

HAL_SMARTCARD_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Transmit_DMA (SMARTCARD_HandleTypeDef * hsmartcard, const uint8_t * pData, uint16_t Size)

Function description

Send an amount of data in DMA mode.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be sent.

Return values

- **HAL:** status

HAL_SMARTCARD_Receive_DMA

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Receive_DMA (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in DMA mode.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be received.

Return values

- **HAL:** status

Notes

- The SMARTCARD-associated USART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position).

HAL_SMARTCARD_Abort

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Abort (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Abort ongoing transfers (blocking mode).

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_SMARTCARD_AbortTransmit

Function name

HAL_StatusTypeDef HAL_SMARTCARD_AbortTransmit (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Abort ongoing Transmit transfer (blocking mode).

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_SMARTCARD_AbortReceive

Function name

HAL_StatusTypeDef HAL_SMARTCARD_AbortReceive (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Abort ongoing Receive transfer (blocking mode).

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_SMARTCARD_Abort_IT

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Abort_IT (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Abort ongoing transfers (Interrupt mode).

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_SMARTCARD_AbortTransmit_IT

Function name

HAL_StatusTypeDef HAL_SMARTCARD_AbortTransmit_IT (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Abort ongoing Transmit transfer (Interrupt mode).

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_SMARTCARD_AbortReceive_IT

Function name

HAL_StatusTypeDef HAL_SMARTCARD_AbortReceive_IT (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Abort ongoing Receive transfer (Interrupt mode).

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_SMARTCARD_IRQHandler

Function name

void HAL_SMARTCARD_IRQHandler (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Handle SMARTCARD interrupt requests.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_TxCpltCallback

Function name

void HAL_SMARTCARD_TxCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Tx Transfer completed callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_RxCpltCallback

Function name

void HAL_SMARTCARD_RxCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Rx Transfer completed callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_ErrorCallback

Function name

void HAL_SMARTCARD_ErrorCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

SMARTCARD error callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_AbortCpltCallback

Function name

void HAL_SMARTCARD_AbortCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

SMARTCARD Abort Complete callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_AbortTransmitCpltCallback

Function name

void HAL_SMARTCARD_AbortTransmitCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

SMARTCARD Abort Complete callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_AbortReceiveCpltCallback

Function name

void HAL_SMARTCARD_AbortReceiveCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

SMARTCARD Abort Receive Complete callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_GetState

Function name

HAL_SMARTCARD_StateTypeDef HAL_SMARTCARD_GetState (const SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Return the SMARTCARD handle state.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **SMARTCARD:** handle state

HAL_SMARTCARD_GetError

Function name

uint32_t HAL_SMARTCARD_GetError (const SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Return the SMARTCARD handle error code.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **SMARTCARD:** handle Error Code

79.3 SMARTCARD Firmware driver defines

The following section lists the various define and macros of the module.

79.3.1 SMARTCARD

SMARTCARD

SMARTCARD Clock Prescaler

SMARTCARD_PRESCALER_DIV1

fclk_pres = fclk

SMARTCARD_PRESCALER_DIV2

fclk_pres = fclk/2

SMARTCARD_PRESCALER_DIV4

fclk_pres = fclk/4

SMARTCARD_PRESCALER_DIV6

fclk_pres = fclk/6

SMARTCARD_PRESCALER_DIV8

fclk_pres = fclk/8

SMARTCARD_PRESCALER_DIV10

fclk_pres = fclk/10

SMARTCARD_PRESCALER_DIV12

fclk_pres = fclk/12

SMARTCARD_PRESCALER_DIV16

fclk_pres = fclk/16

SMARTCARD_PRESCALER_DIV32

fclk_pres = fclk/32

SMARTCARD_PRESCALER_DIV64

fclk_pres = fclk/64

SMARTCARD_PRESCALER_DIV128

fclk_pres = fclk/128

SMARTCARD_PRESCALER_DIV256

fclk_pres = fclk/256

SMARTCARD Clock Phase

SMARTCARD_PHASE_1EDGE

SMARTCARD frame phase on first clock transition

SMARTCARD_PHASE_2EDGE

SMARTCARD frame phase on second clock transition

SMARTCARD Clock Polarity

SMARTCARD_POLARITY_LOW

SMARTCARD frame low polarity

SMARTCARD_POLARITY_HIGH

SMARTCARD frame high polarity

SMARTCARD advanced feature Binary Data inversion

SMARTCARD_ADVFEATURE_DATAINV_DISABLE

Binary data inversion disable

SMARTCARD_ADVFEATURE_DATAINV_ENABLE

Binary data inversion enable

SMARTCARD advanced feature DMA Disable on Rx Error

SMARTCARD_ADVFEATURE_DMA_ENABLEONRXERROR

DMA enable on Reception Error

SMARTCARD_ADVFEATURE_DMA_DISABLEONRXERROR

DMA disable on Reception Error

SMARTCARD Error Code Definition

HAL_SMARTCARD_ERROR_NONE

No error

HAL_SMARTCARD_ERROR_PE

Parity error

HAL_SMARTCARD_ERROR_NE

Noise error

HAL_SMARTCARD_ERROR_FE

frame error

HAL_SMARTCARD_ERROR_ORE

Overrun error

HAL_SMARTCARD_ERROR_DMA

DMA transfer error

HAL_SMARTCARD_ERROR_RTO

Receiver TimeOut error

HAL_SMARTCARD_ERROR_INVALID_CALLBACK

Invalid Callback error

SMARTCARD Exported Macros

__HAL_SMARTCARD_RESET_HANDLE_STATE

Description:

- Reset SMARTCARD handle states.

Parameters:

- __HANDLE__: SMARTCARD handle.

Return value:

- None

__HAL_SMARTCARD_FLUSH_DRREGISTER

Description:

- Flush the Smartcard Data registers.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_FLAG

Description:

- Clear the specified SMARTCARD pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - SMARTCARD_CLEAR_PEF Parity error clear flag
  - SMARTCARD_CLEAR_FEF Framing error clear flag
  - SMARTCARD_CLEAR_NEF Noise detected clear flag
  - SMARTCARD_CLEAR_OREF OverRun error clear flag
  - SMARTCARD_CLEAR_IDLEF Idle line detected clear flag
  - SMARTCARD_CLEAR_TCF Transmission complete clear flag
  - SMARTCARD_CLEAR_TCBGTF Transmission complete before guard time clear flag
  - SMARTCARD_CLEAR_RTOF Receiver timeout clear flag
  - SMARTCARD_CLEAR_EOBF End of block clear flag
  - SMARTCARD_CLEAR_TXFECF TXFIFO empty Clear flag

Return value:

- None

__HAL_SMARTCARD_CLEAR_PEFLAG

Description:

- Clear the SMARTCARD PE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_FEFLAG

Description:

- Clear the SMARTCARD FE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_NEFLAG

Description:

- Clear the SMARTCARD NE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_OREFLAG

Description:

- Clear the SMARTCARD ORE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_IDLEFLAG

Description:

- Clear the SMARTCARD IDLE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_GET_FLAG

Description:

- Check whether the specified Smartcard flag is set or not.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SMARTCARD_FLAG_TCBGT Transmission complete before guard time flag (when flag available)
  - SMARTCARD_FLAG_REACK Receive enable acknowledge flag
  - SMARTCARD_FLAG_TEACK Transmit enable acknowledge flag
  - SMARTCARD_FLAG_BUSY Busy flag
  - SMARTCARD_FLAG_EOBF End of block flag
  - SMARTCARD_FLAG_RTOF Receiver timeout flag
  - SMARTCARD_FLAG_TXE Transmit data register empty flag
  - SMARTCARD_FLAG_TC Transmission complete flag
  - SMARTCARD_FLAG_RXNE Receive data register not empty flag
  - SMARTCARD_FLAG_IDLE Idle line detection flag
  - SMARTCARD_FLAG_ORE Overrun error flag
  - SMARTCARD_FLAG_NE Noise error flag
  - SMARTCARD_FLAG_FE Framing error flag
  - SMARTCARD_FLAG_PE Parity error flag
  - SMARTCARD_FLAG_TXFNF TXFIFO not full flag
  - SMARTCARD_FLAG_RXFNE RXFIFO not empty flag
  - SMARTCARD_FLAG_TXFE TXFIFO Empty flag
  - SMARTCARD_FLAG_RXFF RXFIFO Full flag
  - SMARTCARD_FLAG_RXFT SMARTCARD RXFIFO threshold flag
  - SMARTCARD_FLAG_TXFT SMARTCARD TXFIFO threshold flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_SMARTCARD_ENABLE_IT

Description:

- Enable the specified SmartCard interrupt.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __INTERRUPT__: specifies the SMARTCARD interrupt to enable. This parameter can be one of the following values:
  - SMARTCARD_IT_EOB End of block interrupt
  - SMARTCARD_IT_RTO Receive timeout interrupt
  - SMARTCARD_IT_TXE Transmit data register empty interrupt
  - SMARTCARD_IT_TC Transmission complete interrupt
  - SMARTCARD_IT_TCBGT Transmission complete before guard time interrupt (when interruption available)
  - SMARTCARD_IT_RXNE Receive data register not empty interrupt
  - SMARTCARD_IT_IDLE Idle line detection interrupt
  - SMARTCARD_IT_PE Parity error interrupt
  - SMARTCARD_IT_ERR Error interrupt(frame error, noise error, overrun error)
  - SMARTCARD_IT_TXFNF TX FIFO not full interruption
  - SMARTCARD_IT_RXFNE RXFIFO not empty interruption
  - SMARTCARD_IT_RXFF RXFIFO full interruption
  - SMARTCARD_IT_TXFE TXFIFO empty interruption
  - SMARTCARD_IT_RXFT RXFIFO threshold reached interruption
  - SMARTCARD_IT_TXFT TXFIFO threshold reached interruption

Return value:

- None

__HAL_SMARTCARD_DISABLE_IT

Description:

- Disable the specified SmartCard interrupt.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __INTERRUPT__: specifies the SMARTCARD interrupt to disable. This parameter can be one of the following values:
  - SMARTCARD_IT_EOB End of block interrupt
  - SMARTCARD_IT_RTO Receive timeout interrupt
  - SMARTCARD_IT_TXE Transmit data register empty interrupt
  - SMARTCARD_IT_TC Transmission complete interrupt
  - SMARTCARD_IT_TCBGT Transmission complete before guard time interrupt (when interruption available)
  - SMARTCARD_IT_RXNE Receive data register not empty interrupt
  - SMARTCARD_IT_IDLE Idle line detection interrupt
  - SMARTCARD_IT_PE Parity error interrupt
  - SMARTCARD_IT_ERR Error interrupt(frame error, noise error, overrun error)
  - SMARTCARD_IT_TXFNF TX FIFO not full interruption
  - SMARTCARD_IT_RXFNE RXFIFO not empty interruption
  - SMARTCARD_IT_RXFF RXFIFO full interruption
  - SMARTCARD_IT_TXFE TXFIFO empty interruption
  - SMARTCARD_IT_RXFT RXFIFO threshold reached interruption
  - SMARTCARD_IT_TXFT TXFIFO threshold reached interruption

Return value:

- None

__HAL_SMARTCARD_GET_IT

Description:

- Check whether the specified SmartCard interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __INTERRUPT__: specifies the SMARTCARD interrupt to check. This parameter can be one of the following values:
  - SMARTCARD_IT_EOB End of block interrupt
  - SMARTCARD_IT_RTO Receive timeout interrupt
  - SMARTCARD_IT_TXE Transmit data register empty interrupt
  - SMARTCARD_IT_TC Transmission complete interrupt
  - SMARTCARD_IT_TCBGT Transmission complete before guard time interrupt (when interruption available)
  - SMARTCARD_IT_RXNE Receive data register not empty interrupt
  - SMARTCARD_IT_IDLE Idle line detection interrupt
  - SMARTCARD_IT_PE Parity error interrupt
  - SMARTCARD_IT_ERR Error interrupt(frame error, noise error, overrun error)
  - SMARTCARD_IT_TXFNF TX FIFO not full interruption
  - SMARTCARD_IT_RXFNE RXFIFO not empty interruption
  - SMARTCARD_IT_RXFF RXFIFO full interruption
  - SMARTCARD_IT_TXFE TXFIFO empty interruption
  - SMARTCARD_IT_RXFT RXFIFO threshold reached interruption
  - SMARTCARD_IT_TXFT TXFIFO threshold reached interruption

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_SMARTCARD_GET_IT_SOURCE

Description:

- Check whether the specified SmartCard interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __INTERRUPT__: specifies the SMARTCARD interrupt source to check. This parameter can be one of the following values:
  - SMARTCARD_IT_EOB End of block interrupt
  - SMARTCARD_IT_RTO Receive timeout interrupt
  - SMARTCARD_IT_TXE Transmit data register empty interrupt
  - SMARTCARD_IT_TC Transmission complete interrupt
  - SMARTCARD_IT_TCBGT Transmission complete before guard time interrupt (when interruption available)
  - SMARTCARD_IT_RXNE Receive data register not empty interrupt
  - SMARTCARD_IT_IDLE Idle line detection interrupt
  - SMARTCARD_IT_PE Parity error interrupt
  - SMARTCARD_IT_ERR Error interrupt(frame error, noise error, overrun error)
  - SMARTCARD_IT_TXFNF TX FIFO not full interruption
  - SMARTCARD_IT_RXFNE RXFIFO not empty interruption
  - SMARTCARD_IT_RXFF RXFIFO full interruption
  - SMARTCARD_IT_TXFE TXFIFO empty interruption
  - SMARTCARD_IT_RXFT RXFIFO threshold reached interruption
  - SMARTCARD_IT_TXFT TXFIFO threshold reached interruption

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_SMARTCARD_CLEAR_IT

Description:

- Clear the specified SMARTCARD ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt. This parameter can be one of the following values:
  - SMARTCARD_CLEAR_PEF Parity error clear flag
  - SMARTCARD_CLEAR_FEF Framing error clear flag
  - SMARTCARD_CLEAR_NEF Noise detected clear flag
  - SMARTCARD_CLEAR_OREF OverRun error clear flag
  - SMARTCARD_CLEAR_IDLEF Idle line detection clear flag
  - SMARTCARD_CLEAR_TXFECF TXFIFO empty Clear Flag
  - SMARTCARD_CLEAR_TCF Transmission complete clear flag
  - SMARTCARD_CLEAR_TCBGTF Transmission complete before guard time clear flag (when flag available)
  - SMARTCARD_CLEAR_RTOF Receiver timeout clear flag
  - SMARTCARD_CLEAR_EOBF End of block clear flag

Return value:

- None

__HAL_SMARTCARD_SEND_REQ

Description:

- Set a specific SMARTCARD request flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __REQ__: specifies the request flag to set This parameter can be one of the following values:
  - SMARTCARD_RXDATA_FLUSH_REQUEST Receive data flush Request
  - SMARTCARD_TXDATA_FLUSH_REQUEST Transmit data flush Request

Return value:

- None

__HAL_SMARTCARD_ONE_BIT_SAMPLE_ENABLE

Description:

- Enable the SMARTCARD one bit sample method.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_ONE_BIT_SAMPLE_DISABLE

Description:

- Disable the SMARTCARD one bit sample method.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_ENABLE

Description:

- Enable the USART associated to the SMARTCARD Handle.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_DISABLE

Description:

- Disable the USART associated to the SMARTCARD Handle.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

SMARTCARD interruptions flags mask

SMARTCARD_IT_MASK

SMARTCARD interruptions flags mask

SMARTCARD_CR_MASK

SMARTCARD control register mask

SMARTCARD_CR_POS

SMARTCARD control register position

SMARTCARD_ISR_MASK

SMARTCARD ISR register mask

SMARTCARD_ISR_POS

SMARTCARD ISR register position

SMARTCARD Last Bit

SMARTCARD_LASTBIT_DISABLE

SMARTCARD frame last data bit clock pulse not output to SCLK pin

SMARTCARD_LASTBIT_ENABLE

SMARTCARD frame last data bit clock pulse output to SCLK pin

SMARTCARD Transfer Mode

SMARTCARD_MODE_RX

SMARTCARD RX mode

SMARTCARD_MODE_TX

SMARTCARD TX mode

SMARTCARD_MODE_TX_RX

SMARTCARD RX and TX mode

SMARTCARD advanced feature MSB first

SMARTCARD_ADVFEATURE_MSBFIRST_DISABLE

Most significant bit sent/received first disable

SMARTCARD_ADVFEATURE_MSBFIRST_ENABLE

Most significant bit sent/received first enable

SMARTCARD NACK Enable

SMARTCARD_NACK_DISABLE

SMARTCARD NACK transmission disabled

SMARTCARD_NACK_ENABLE

SMARTCARD NACK transmission enabled

SMARTCARD One Bit Sampling Method

SMARTCARD_ONE_BIT_SAMPLE_DISABLE

SMARTCARD frame one-bit sample disabled

SMARTCARD_ONE_BIT_SAMPLE_ENABLE

SMARTCARD frame one-bit sample enabled

SMARTCARD advanced feature Overrun Disable

SMARTCARD_ADVFEATURE_OVERRUN_ENABLE

RX overrun enable

SMARTCARD_ADVFEATURE_OVERRUN_DISABLE

RX overrun disable

SMARTCARD Parity

SMARTCARD_PARITY_EVEN

SMARTCARD frame even parity

SMARTCARD_PARITY_ODD

SMARTCARD frame odd parity

SMARTCARD Request Parameters

SMARTCARD_RXDATA_FLUSH_REQUEST

Receive data flush request

SMARTCARD_TXDATA_FLUSH_REQUEST

Transmit data flush request

SMARTCARD advanced feature RX pin active level inversion

SMARTCARD_ADVFEATURE_RXINV_DISABLE

RX pin active level inversion disable

SMARTCARD_ADVFEATURE_RXINV_ENABLE

RX pin active level inversion enable

SMARTCARD advanced feature RX TX pins swap

SMARTCARD_ADVFEATURE_SWAP_DISABLE

TX/RX pins swap disable

SMARTCARD_ADVFEATURE_SWAP_ENABLE

TX/RX pins swap enable

SMARTCARD State Code Definition

HAL_SMARTCARD_STATE_RESET

Peripheral is not initialized. Value is allowed for gState and RxState

HAL_SMARTCARD_STATE_READY

Peripheral Initialized and ready for use. Value is allowed for gState and RxState

HAL_SMARTCARD_STATE_BUSY

an internal process is ongoing Value is allowed for gState only

HAL_SMARTCARD_STATE_BUSY_TX

Data Transmission process is ongoing Value is allowed for gState only

HAL_SMARTCARD_STATE_BUSY_RX

Data Reception process is ongoing Value is allowed for RxState only

HAL_SMARTCARD_STATE_BUSY_TX_RX

Data Transmission and Reception process is ongoing Not to be used for neither gState nor RxState. Value is result of combination (Or) between gState and RxState values

HAL_SMARTCARD_STATE_TIMEOUT

Timeout state Value is allowed for gState only

HAL_SMARTCARD_STATE_ERROR

Error Value is allowed for gState only

SMARTCARD Number of Stop Bits

SMARTCARD_STOPBITS_0_5

SMARTCARD frame with 0.5 stop bit

SMARTCARD_STOPBITS_1_5

SMARTCARD frame with 1.5 stop bits

SMARTCARD Timeout Enable

SMARTCARD_TIMEOUT_DISABLE

SMARTCARD receiver timeout disabled

SMARTCARD_TIMEOUT_ENABLE

SMARTCARD receiver timeout enabled

SMARTCARD advanced feature TX pin active level inversion

SMARTCARD_ADVFEATURE_TXINV_DISABLE

TX pin active level inversion disable

SMARTCARD_ADVFEATURE_TXINV_ENABLE

TX pin active level inversion enable

SMARTCARD Word Length

SMARTCARD_WORDLENGTH_9B

SMARTCARD frame length

80 HAL SMARTCARD Extension Driver

80.1 SMARTCARDEx Firmware driver API description

The following section lists the various functions of the SMARTCARDEx library.

80.1.1 SMARTCARD peripheral extended features

The Extended SMARTCARD HAL driver can be used as follows:

1. After having configured the SMARTCARD basic features with HAL_SMARTCARD_Init(), then program SMARTCARD advanced features if required (TX/RX pins swap, TimeOut, auto-retry counter,...) in the hsmartcard AdvancedInit structure.
2. FIFO mode enabling/disabling and RX/TX FIFO threshold programming.

Note: When SMARTCARD operates in FIFO mode, FIFO mode must be enabled prior starting RX/TX transfers. Also RX/TX FIFO thresholds must be configured prior starting RX/TX transfers.

80.1.2 Peripheral Control functions

This subsection provides a set of functions allowing to initialize the SMARTCARD.

- HAL_SMARTCARDEx_BlockLength_Config() API allows to configure the Block Length on the fly
- HAL_SMARTCARDEx_TimeOut_Config() API allows to configure the receiver timeout value on the fly
- HAL_SMARTCARDEx_EnableReceiverTimeOut() API enables the receiver timeout feature
- HAL_SMARTCARDEx_DisableReceiverTimeOut() API disables the receiver timeout feature

This section contains the following APIs:

- ***HAL_SMARTCARDEx_BlockLength_Config()***
- ***HAL_SMARTCARDEx_TimeOut_Config()***
- ***HAL_SMARTCARDEx_EnableReceiverTimeOut()***
- ***HAL_SMARTCARDEx_DisableReceiverTimeOut()***

80.1.3 IO operation functions

This subsection provides a set of FIFO mode related callback functions.

1. TX/RX Fifos Callbacks:
   - HAL_SMARTCARDEx_RxFifoFullCallback()
   - HAL_SMARTCARDEx_TxFifoEmptyCallback()

This section contains the following APIs:

- ***HAL_SMARTCARDEx_RxFifoFullCallback()***
- ***HAL_SMARTCARDEx_TxFifoEmptyCallback()***

80.1.4 Peripheral FIFO Control functions

This subsection provides a set of functions allowing to control the SMARTCARD FIFO feature.

- HAL_SMARTCARDEx_EnableFifoMode() API enables the FIFO mode
- HAL_SMARTCARDEx_DisableFifoMode() API disables the FIFO mode
- HAL_SMARTCARDEx_SetTxFifoThreshold() API sets the TX FIFO threshold
- HAL_SMARTCARDEx_SetRxFifoThreshold() API sets the RX FIFO threshold

This section contains the following APIs:

- ***HAL_SMARTCARDEx_EnableFifoMode()***
- ***HAL_SMARTCARDEx_DisableFifoMode()***
- ***HAL_SMARTCARDEx_SetTxFifoThreshold()***

- *HAL_SMARTCARDEx_SetRxFifoThreshold()*

80.1.5 Detailed description of functions

HAL_SMARTCARDEx_BlockLength_Config

Function name

void HAL_SMARTCARDEx_BlockLength_Config (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t BlockLength)

Function description

Update on the fly the SMARTCARD block length in RTOR register.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **BlockLength:** SMARTCARD block length (8-bit long at most)

Return values

- **None:**

HAL_SMARTCARDEx_TimeOut_Config

Function name

void HAL_SMARTCARDEx_TimeOut_Config (SMARTCARD_HandleTypeDef * hsmartcard, uint32_t TimeOutValue)

Function description

Update on the fly the receiver timeout value in RTOR register.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **TimeOutValue:** receiver timeout value in number of baud blocks. The timeout value must be less or equal to 0x0FFFFFFF.

Return values

- **None:**

HAL_SMARTCARDEx_EnableReceiverTimeOut

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_EnableReceiverTimeOut (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Enable the SMARTCARD receiver timeout feature.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

HAL_SMARTCARDEx_DisableReceiverTimeOut

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_DisableReceiverTimeOut (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Disable the SMARTCARD receiver timeout feature.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

HAL_SMARTCARDEx_RxFifoFullCallback

Function name

void HAL_SMARTCARDEx_RxFifoFullCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

SMARTCARD RX Fifo full callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARDEx_TxFifoEmptyCallback

Function name

void HAL_SMARTCARDEx_TxFifoEmptyCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

SMARTCARD TX Fifo empty callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARDEx_EnableFifoMode

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_EnableFifoMode (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Enable the FIFO mode.

Parameters

- **hsmartcard:** SMARTCARD handle.

Return values

- **HAL:** status

HAL_SMARTCARDEx_DisableFifoMode

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_DisableFifoMode (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Disable the FIFO mode.

Parameters

- **hsmartcard:** SMARTCARD handle.

Return values

- **HAL:** status

HAL_SMARTCARDEx_SetTxFifoThreshold

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_SetTxFifoThreshold (SMARTCARD_HandleTypeDef * hsmartcard, uint32_t Threshold)

Function description

Set the TXFIFO threshold.

Parameters

- **hsmartcard:** SMARTCARD handle.
- **Threshold:** TX FIFO threshold value This parameter can be one of the following values:
  - SMARTCARD_TXFIFO_THRESHOLD_1_8
  - SMARTCARD_TXFIFO_THRESHOLD_1_4
  - SMARTCARD_TXFIFO_THRESHOLD_1_2
  - SMARTCARD_TXFIFO_THRESHOLD_3_4
  - SMARTCARD_TXFIFO_THRESHOLD_7_8
  - SMARTCARD_TXFIFO_THRESHOLD_8_8

Return values

- **HAL:** status

HAL_SMARTCARDEx_SetRxFifoThreshold

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_SetRxFifoThreshold (SMARTCARD_HandleTypeDef * hsmartcard, uint32_t Threshold)

Function description

Set the RXFIFO threshold.

Parameters

- **hsmartcard:** SMARTCARD handle.
- **Threshold:** RX FIFO threshold value This parameter can be one of the following values:
  - SMARTCARD_RXFIFO_THRESHOLD_1_8
  - SMARTCARD_RXFIFO_THRESHOLD_1_4
  - SMARTCARD_RXFIFO_THRESHOLD_1_2
  - SMARTCARD_RXFIFO_THRESHOLD_3_4
  - SMARTCARD_RXFIFO_THRESHOLD_7_8
  - SMARTCARD_RXFIFO_THRESHOLD_8_8

Return values

- **HAL:** status

80.2 SMARTCARDEx Firmware driver defines

The following section lists the various define and macros of the module.

80.2.1 SMARTCARDEx

SMARTCARDEx

SMARTCARD advanced feature initialization type

SMARTCARD_ADVFEATURE_NO_INIT

No advanced feature initialization

SMARTCARD_ADVFEATURE_TXINVERT_INIT

TX pin active level inversion

SMARTCARD_ADVFEATURE_RXINVERT_INIT

RX pin active level inversion

SMARTCARD_ADVFEATURE_DATAINVERT_INIT

Binary data inversion

SMARTCARD_ADVFEATURE_SWAP_INIT

TX/RX pins swap

SMARTCARD_ADVFEATURE_RXOVERRUNDISABLE_INIT

RX overrun disable

SMARTCARD_ADVFEATURE_DMADISABLEONERROR_INIT

DMA disable on Reception Error

SMARTCARD_ADVFEATURE_MSBFIRST_INIT

Most significant bit sent/received first

SMARTCARD_ADVFEATURE_TXCOMPLETION

TX completion indication before of after guard time

SMARTCARD FIFO mode

SMARTCARD_FIFOMODE_DISABLE

FIFO mode disable

SMARTCARD_FIFOMODE_ENABLE

FIFO mode enable

SMARTCARD Flags

SMARTCARD_FLAG_TCBGT

SMARTCARD transmission complete before guard time completion

SMARTCARD_FLAG_REACK

SMARTCARD receive enable acknowledge flag

SMARTCARD_FLAG_TEACK

SMARTCARD transmit enable acknowledge flag

SMARTCARD_FLAG_BUSY

SMARTCARD busy flag

SMARTCARD_FLAG_EOBF

SMARTCARD end of block flag

SMARTCARD_FLAG_RTOF

SMARTCARD receiver timeout flag

SMARTCARD_FLAG_TXE

SMARTCARD transmit data register empty

SMARTCARD_FLAG_TXFNF

SMARTCARD TXFIFO not full

SMARTCARD_FLAG_TC

SMARTCARD transmission complete

SMARTCARD_FLAG_RXNE

SMARTCARD read data register not empty

SMARTCARD_FLAG_RXFNE

SMARTCARD RXFIFO not empty

SMARTCARD_FLAG_IDLE

SMARTCARD idle line detection

SMARTCARD_FLAG_ORE

SMARTCARD overrun error

SMARTCARD_FLAG_NE

SMARTCARD noise error

SMARTCARD_FLAG_FE

SMARTCARD frame error

SMARTCARD_FLAG_PE

SMARTCARD parity error

SMARTCARD_FLAG_TXFE

SMARTCARD TXFIFO Empty flag

SMARTCARD_FLAG_RXFF

SMARTCARD RXFIFO Full flag

SMARTCARD_FLAG_RXFT

SMARTCARD RXFIFO threshold flag

SMARTCARD_FLAG_TXFT

SMARTCARD TXFIFO threshold flag

SMARTCARD Interrupts Definition

SMARTCARD_IT_PE

SMARTCARD parity error interruption

SMARTCARD_IT_TXE

SMARTCARD transmit data register empty interruption

SMARTCARD_IT_TXFNF

SMARTCARD TX FIFO not full interruption

SMARTCARD_IT_TC

SMARTCARD transmission complete interruption

SMARTCARD_IT_RXNE

SMARTCARD read data register not empty interruption

SMARTCARD_IT_RXFNE

SMARTCARD RXFIFO not empty interruption

SMARTCARD_IT_IDLE

SMARTCARD idle line detection interruption

SMARTCARD_IT_ERR

SMARTCARD error interruption

SMARTCARD_IT_ORE

SMARTCARD overrun error interruption

SMARTCARD_IT_NE

SMARTCARD noise error interruption

SMARTCARD_IT_FE

SMARTCARD frame error interruption

SMARTCARD_IT_EOB

SMARTCARD end of block interruption

SMARTCARD_IT_RTO

SMARTCARD receiver timeout interruption

SMARTCARD_IT_TCBGT

SMARTCARD transmission complete before guard time completion interruption

SMARTCARD_IT_RXFF

SMARTCARD RXFIFO full interruption

SMARTCARD_IT_TXFE

SMARTCARD TXFIFO empty interruption

SMARTCARD_IT_RXFT

SMARTCARD RXFIFO threshold reached interruption

SMARTCARD_IT_TXFT

SMARTCARD TXFIFO threshold reached interruption

SMARTCARD Interruption Clear Flags

SMARTCARD_CLEAR_PEF

SMARTCARD parity error clear flag

SMARTCARD_CLEAR_FEF

SMARTCARD framing error clear flag

SMARTCARD_CLEAR_NEF

SMARTCARD noise error detected clear flag

SMARTCARD_CLEAR_OREF

SMARTCARD overrun error clear flag

SMARTCARD_CLEAR_IDLEF

SMARTCARD idle line detected clear flag

SMARTCARD_CLEAR_TXFECF

TXFIFO empty Clear Flag

SMARTCARD_CLEAR_TCF

SMARTCARD transmission complete clear flag

SMARTCARD_CLEAR_TCBGTF

SMARTCARD transmission complete before guard time completion clear flag

SMARTCARD_CLEAR_RTOF

SMARTCARD receiver time out clear flag

SMARTCARD_CLEAR_EOBF

SMARTCARD end of block clear flag

SMARTCARD RXFIFO threshold level

SMARTCARD_RXFIFO_THRESHOLD_1_8

RXFIFO FIFO reaches 1/8 of its depth

SMARTCARD_RXFIFO_THRESHOLD_1_4

RXFIFO FIFO reaches 1/4 of its depth

SMARTCARD_RXFIFO_THRESHOLD_1_2

RXFIFO FIFO reaches 1/2 of its depth

SMARTCARD_RXFIFO_THRESHOLD_3_4

RXFIFO FIFO reaches 3/4 of its depth

SMARTCARD_RXFIFO_THRESHOLD_7_8

RXFIFO FIFO reaches 7/8 of its depth

SMARTCARD_RXFIFO_THRESHOLD_8_8

RXFIFO FIFO becomes full

SMARTCARD Transmission Completion Indication

SMARTCARD_TCBGT

SMARTCARD transmission complete before guard time

SMARTCARD_TC

SMARTCARD transmission complete (flag raised when guard time has elapsed)

SMARTCARD TXFIFO threshold level

SMARTCARD_TXFIFO_THRESHOLD_1_8

TXFIFO reaches 1/8 of its depth

SMARTCARD_TXFIFO_THRESHOLD_1_4

TXFIFO reaches 1/4 of its depth

SMARTCARD_TXFIFO_THRESHOLD_1_2

TXFIFO reaches 1/2 of its depth

SMARTCARD_TXFIFO_THRESHOLD_3_4

TXFIFO reaches 3/4 of its depth

SMARTCARD_TXFIFO_THRESHOLD_7_8

TXFIFO reaches 7/8 of its depth

SMARTCARD_TXFIFO_THRESHOLD_8_8

TXFIFO becomes empty

81 HAL SMBUS Generic Driver

81.1 SMBUS Firmware driver registers structures

81.1.1 SMBUS_InitTypeDef

SMBUS_InitTypeDef is defined in the stm32h7xx_hal_smbus.h

Data Fields

- ***uint32_t Timing***
- ***uint32_t AnalogFilter***
- ***uint32_t OwnAddress1***
- ***uint32_t AddressingMode***
- ***uint32_t DualAddressMode***
- ***uint32_t OwnAddress2***
- ***uint32_t OwnAddress2Masks***
- ***uint32_t GeneralCallMode***
- ***uint32_t NoStretchMode***
- ***uint32_t PacketErrorCheckMode***
- ***uint32_t PeripheralMode***
- ***uint32_t SMBusTimeout***

Field Documentation

- ***uint32_t SMBUS_InitTypeDef::Timing***

  Specifies the SMBUS_TIMINGR_register value. This parameter calculated by referring to SMBUS initialization section in Reference manual
- ***uint32_t SMBUS_InitTypeDef::AnalogFilter***

  Specifies if Analog Filter is enable or not. This parameter can be a value of ***SMBUS_Analog_Filter***
- ***uint32_t SMBUS_InitTypeDef::OwnAddress1***

  Specifies the first device own address. This parameter can be a 7-bit or 10-bit address.
- ***uint32_t SMBUS_InitTypeDef::AddressingMode***

  Specifies if 7-bit or 10-bit addressing mode for master is selected. This parameter can be a value of ***SMBUS_addressing_mode***
- ***uint32_t SMBUS_InitTypeDef::DualAddressMode***

  Specifies if dual addressing mode is selected. This parameter can be a value of ***SMBUS_dual_addressing_mode***
- ***uint32_t SMBUS_InitTypeDef::OwnAddress2***

  Specifies the second device own address if dual addressing mode is selected This parameter can be a 7-bit address.
- ***uint32_t SMBUS_InitTypeDef::OwnAddress2Masks***

  Specifies the acknowledge mask address second device own address if dual addressing mode is selected This parameter can be a value of ***SMBUS_own_address2_masks***.
- ***uint32_t SMBUS_InitTypeDef::GeneralCallMode***

  Specifies if general call mode is selected. This parameter can be a value of ***SMBUS_general_call_addressing_mode***.
- ***uint32_t SMBUS_InitTypeDef::NoStretchMode***

  Specifies if nostretch mode is selected. This parameter can be a value of ***SMBUS_nostretch_mode***
- ***uint32_t SMBUS_InitTypeDef::PacketErrorCheckMode***

  Specifies if Packet Error Check mode is selected. This parameter can be a value of ***SMBUS_packet_error_check_mode***

- ***uint32_t SMBUS_InitTypeDef::PeripheralMode***
  Specifies which mode of Periphal is selected. This parameter can be a value of ***SMBUS_peripheral_mode***
- ***uint32_t SMBUS_InitTypeDef::SMBusTimeout***
  Specifies the content of the 32 Bits SMBUS_TIMEOUT_register value. (Enable bits and different timeout values) This parameter calculated by referring to SMBUS initialization section in Reference manual

81.1.2 __SMBUS_HandleTypeDef

__SMBUS_HandleTypeDef is defined in the stm32h7xx_hal_smbus.h

Data Fields

- ***I2C_TypeDef * Instance***
- ***SMBUS_InitTypeDef Init***
- ***uint8_t * pBuffPtr***
- ***uint16_t XferSize***
- ***__IO uint16_t XferCount***
- ***__IO uint32_t XferOptions***
- ***__IO uint32_t PreviousState***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t State***
- ***__IO uint32_t ErrorCode***
- ***void(* MasterTxCpltCallback***
- ***void(* MasterRxCpltCallback***
- ***void(* SlaveTxCpltCallback***
- ***void(* SlaveRxCpltCallback***
- ***void(* ListenCpltCallback***
- ***void(* ErrorCallback***
- ***void(* AddrCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***I2C_TypeDef* __SMBUS_HandleTypeDef::Instance***
  SMBUS registers base address
- ***SMBUS_InitTypeDef __SMBUS_HandleTypeDef::Init***
  SMBUS communication parameters
- ***uint8_t* __SMBUS_HandleTypeDef::pBuffPtr***
  Pointer to SMBUS transfer buffer
- ***uint16_t __SMBUS_HandleTypeDef::XferSize***
  SMBUS transfer size
- ***__IO uint16_t __SMBUS_HandleTypeDef::XferCount***
  SMBUS transfer counter
- ***__IO uint32_t __SMBUS_HandleTypeDef::XferOptions***
  SMBUS transfer options
- ***__IO uint32_t __SMBUS_HandleTypeDef::PreviousState***
  SMBUS communication Previous state
- ***HAL_LockTypeDef __SMBUS_HandleTypeDef::Lock***
  SMBUS locking object
- ***__IO uint32_t __SMBUS_HandleTypeDef::State***
  SMBUS communication state
- ***__IO uint32_t __SMBUS_HandleTypeDef::ErrorCode***
  SMBUS Error code

- ***void(* __SMBUS_HandleTypeDef::MasterTxCpltCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Master Tx Transfer completed callback
- ***void(* __SMBUS_HandleTypeDef::MasterRxCpltCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Master Rx Transfer completed callback
- ***void(* __SMBUS_HandleTypeDef::SlaveTxCpltCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Slave Tx Transfer completed callback
- ***void(* __SMBUS_HandleTypeDef::SlaveRxCpltCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Slave Rx Transfer completed callback
- ***void(* __SMBUS_HandleTypeDef::ListenCpltCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Listen Complete callback
- ***void(* __SMBUS_HandleTypeDef::ErrorCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Error callback
- ***void(* __SMBUS_HandleTypeDef::AddrCallback)(struct __SMBUS_HandleTypeDef *hsmbus, uint8_t TransferDirection, uint16_t AddrMatchCode)***
  SMBUS Slave Address Match callback
- ***void(* __SMBUS_HandleTypeDef::MspInitCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Msp Init callback
- ***void(* __SMBUS_HandleTypeDef::MspDeInitCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Msp DeInit callback

81.2 SMBUS Firmware driver API description

The following section lists the various functions of the SMBUS library.

81.2.1 How to use this driver

The SMBUS HAL driver can be used as follows:

1. Declare a SMBUS_HandleTypeDef handle structure, for example: SMBUS_HandleTypeDef hsmbus;
2. Initialize the SMBUS low level resources by implementing the HAL_SMBUS_MspInit() API:
   a. Enable the SMBUSx interface clock
   b. SMBUS pins configuration
      - Enable the clock for the SMBUS GPIOs
      - Configure SMBUS pins as alternate function open-drain
   c. NVIC configuration if you need to use interrupt process
      - Configure the SMBUSx interrupt priority
      - Enable the NVIC SMBUS IRQ Channel
3. Configure the Communication Clock Timing, Bus Timeout, Own Address1, Master Addressing mode, Dual Addressing mode, Own Address2, Own Address2 Mask, General call, Nostretch mode, Peripheral mode and Packet Error Check mode in the hsmbus Init structure.
4. Initialize the SMBUS registers by calling the HAL_SMBUS_Init() API:
   – These API's configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_SMBUS_MspInit(&hsmbus) API.
5. To check if target device is ready for communication, use the function HAL_SMBUS_IsDeviceReady()
6. For SMBUS IO operations, only one mode of operations is available within this driver

Interrupt mode IO operation

- Transmit in master/host SMBUS mode an amount of data in non-blocking mode using HAL_SMBUS_Master_Transmit_IT()
  – At transmission end of transfer HAL_SMBUS_MasterTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_SMBUS_MasterTxCpltCallback()

- Receive in master/host SMBUS mode an amount of data in non-blocking mode using HAL_SMBUS_Master_Receive_IT()
  - At reception end of transfer HAL_SMBUS_MasterRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_SMBUS_MasterRxCpltCallback()
- Abort a master/host SMBUS process communication with Interrupt using HAL_SMBUS_Master_Abort_IT()
  - The associated previous transfer callback is called at the end of abort process
  - mean HAL_SMBUS_MasterTxCpltCallback() in case of previous state was master transmit
  - mean HAL_SMBUS_MasterRxCpltCallback() in case of previous state was master receive
- Enable/disable the Address listen mode in slave/device or host/slave SMBUS mode using HAL_SMBUS_EnableListen_IT() HAL_SMBUS_DisableListen_IT()
  - When address slave/device SMBUS match, HAL_SMBUS_AddrCallback() is executed and users can add their own code to check the Address Match Code and the transmission direction request by master/host (Write/Read).
  - At Listen mode end HAL_SMBUS_ListenCpltCallback() is executed and users can add their own code by customization of function pointer HAL_SMBUS_ListenCpltCallback()
- Transmit in slave/device SMBUS mode an amount of data in non-blocking mode using HAL_SMBUS_Slave_Transmit_IT()
  - At transmission end of transfer HAL_SMBUS_SlaveTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_SMBUS_SlaveTxCpltCallback()
- Receive in slave/device SMBUS mode an amount of data in non-blocking mode using HAL_SMBUS_Slave_Receive_IT()
  - At reception end of transfer HAL_SMBUS_SlaveRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_SMBUS_SlaveRxCpltCallback()
- Enable/Disable the SMBUS alert mode using HAL_SMBUS_EnableAlert_IT() or HAL_SMBUS_DisableAlert_IT()
  - When SMBUS Alert is generated HAL_SMBUS_ErrorCallback() is executed and users can add their own code by customization of function pointer HAL_SMBUS_ErrorCallback() to check the Alert Error Code using function HAL_SMBUS_GetError()
- Get HAL state machine or error values using HAL_SMBUS_GetState() or HAL_SMBUS_GetError()
- In case of transfer Error, HAL_SMBUS_ErrorCallback() function is executed and users can add their own code by customization of function pointer HAL_SMBUS_ErrorCallback() to check the Error Code using function HAL_SMBUS_GetError()

SMBUS HAL driver macros list

Below the list of most used macros in SMBUS HAL driver.

- __HAL_SMBUS_ENABLE: Enable the SMBUS peripheral
- __HAL_SMBUS_DISABLE: Disable the SMBUS peripheral
- __HAL_SMBUS_GET_FLAG: Check whether the specified SMBUS flag is set or not
- __HAL_SMBUS_CLEAR_FLAG: Clear the specified SMBUS pending flag
- __HAL_SMBUS_ENABLE_IT: Enable the specified SMBUS interrupt
- __HAL_SMBUS_DISABLE_IT: Disable the specified SMBUS interrupt

Callback registration

The compilation flag USE_HAL_SMBUS_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_SMBUS_RegisterCallback() or HAL_SMBUS_RegisterAddrCallback() to register an interrupt callback.

Function HAL_SMBUS_RegisterCallback() allows to register following callbacks:

- MasterTxCpltCallback : callback for Master transmission end of transfer.
- MasterRxCpltCallback : callback for Master reception end of transfer.
- SlaveTxCpltCallback : callback for Slave transmission end of transfer.
- SlaveRxCpltCallback : callback for Slave reception end of transfer.
- ListenCpltCallback : callback for end of listen mode.
- ErrorCallback : callback for error detection.

- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

For specific callback AddrCallback use dedicated register callbacks : HAL_SMBUS_RegisterAddrCallback.

Use function HAL_SMBUS_UnRegisterCallback to reset a callback to the default weak function. HAL_SMBUS_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- MasterTxCpltCallback : callback for Master transmission end of transfer.
- MasterRxCpltCallback : callback for Master reception end of transfer.
- SlaveTxCpltCallback : callback for Slave transmission end of transfer.
- SlaveRxCpltCallback : callback for Slave reception end of transfer.
- ListenCpltCallback : callback for end of listen mode.
- ErrorCallback : callback for error detection.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit.

For callback AddrCallback use dedicated register callbacks : HAL_SMBUS_UnRegisterAddrCallback.

By default, after the HAL_SMBUS_Init() and when the state is HAL_I2C_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_SMBUS_MasterTxCpltCallback(), HAL_SMBUS_MasterRxCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_SMBUS_Init()/ HAL_SMBUS_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the HAL_SMBUS_Init()/ HAL_SMBUS_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in HAL_I2C_STATE_READY state only. Exception done MspInit/MspDeInit functions that can be registered/unregistered in HAL_I2C_STATE_READY or HAL_I2C_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_SMBUS_RegisterCallback() before calling HAL_SMBUS_DeInit() or HAL_SMBUS_Init() function.

When the compilation flag USE_HAL_SMBUS_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

Note: You can refer to the SMBUS HAL driver header file for more useful macros

81.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and deinitialize the SMBUSx peripheral:

- User must Implement HAL_SMBUS_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, IT and NVIC).
- Call the function HAL_SMBUS_Init() to configure the selected device with the selected configuration:
  - Clock Timing
  - Bus Timeout
  - Analog Filer mode
  - Own Address 1
  - Addressing mode (Master, Slave)
  - Dual Addressing mode
  - Own Address 2
  - Own Address 2 Mask
  - General call mode
  - Nostretch mode
  - Packet Error Check mode
  - Peripheral mode
- Call the function HAL_SMBUS_DeInit() to restore the default configuration of the selected SMBUSx peripheral.
- Enable/Disable Analog/Digital filters with HAL_SMBUS_ConfigAnalogFilter() and HAL_SMBUS_ConfigDigitalFilter().

This section contains the following APIs:

- *HAL_SMBUS_Init()*
- *HAL_SMBUS_DeInit()*
- *HAL_SMBUS_MspInit()*
- *HAL_SMBUS_MspDeInit()*
- *HAL_SMBUS_ConfigAnalogFilter()*
- *HAL_SMBUS_ConfigDigitalFilter()*
- *HAL_SMBUS_RegisterCallback()*
- *HAL_SMBUS_UnRegisterCallback()*
- *HAL_SMBUS_RegisterAddrCallback()*
- *HAL_SMBUS_UnRegisterAddrCallback()*

81.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the SMBUS data transfers.

1. Blocking mode function to check if device is ready for usage is :
   - HAL_SMBUS_IsDeviceReady()
2. There is only one mode of transfer:
   - Non-Blocking mode : The communication is performed using Interrupts. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated SMBUS IRQ when using Interrupt mode.
3. Non-Blocking mode functions with Interrupt are :
   - HAL_SMBUS_Master_Transmit_IT()
   - HAL_SMBUS_Master_Receive_IT()
   - HAL_SMBUS_Slave_Transmit_IT()
   - HAL_SMBUS_Slave_Receive_IT()
   - HAL_SMBUS_EnableListen_IT() or alias HAL_SMBUS_EnableListen_IT()
   - HAL_SMBUS_DisableListen_IT()
   - HAL_SMBUS_EnableAlert_IT()
   - HAL_SMBUS_DisableAlert_IT()
4. A set of Transfer Complete Callbacks are provided in non-Blocking mode:
   - HAL_SMBUS_MasterTxCpltCallback()
   - HAL_SMBUS_MasterRxCpltCallback()
   - HAL_SMBUS_SlaveTxCpltCallback()
   - HAL_SMBUS_SlaveRxCpltCallback()
   - HAL_SMBUS_AddrCallback()
   - HAL_SMBUS_ListenCpltCallback()
   - HAL_SMBUS_ErrorCallback()

This section contains the following APIs:

- *HAL_SMBUS_Master_Transmit_IT()*
- *HAL_SMBUS_Master_Receive_IT()*
- *HAL_SMBUS_Master_Abort_IT()*
- *HAL_SMBUS_Slave_Transmit_IT()*
- *HAL_SMBUS_Slave_Receive_IT()*
- *HAL_SMBUS_EnableListen_IT()*
- *HAL_SMBUS_DisableListen_IT()*
- *HAL_SMBUS_EnableAlert_IT()*
- *HAL_SMBUS_DisableAlert_IT()*
- *HAL_SMBUS_IsDeviceReady()*

81.2.4 Peripheral State and Errors functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- *HAL_SMBUS_GetState()*
- *HAL_SMBUS_GetError()*

81.2.5 Detailed description of functions

HAL_SMBUS_Init

Function name

HAL_StatusTypeDef HAL_SMBUS_Init (SMBUS_HandleTypeDef * hsmbus)

Function description

Initialize the SMBUS according to the specified parameters in the SMBUS_InitTypeDef and initialize the associated handle.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **HAL:** status

HAL_SMBUS_DeInit

Function name

HAL_StatusTypeDef HAL_SMBUS_DeInit (SMBUS_HandleTypeDef * hsmbus)

Function description

DeInitialize the SMBUS peripheral.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **HAL:** status

HAL_SMBUS_MspInit

Function name

void HAL_SMBUS_MspInit (SMBUS_HandleTypeDef * hsmbus)

Function description

Initialize the SMBUS MSP.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_MspDeInit

Function name

void HAL_SMBUS_MspDeInit (SMBUS_HandleTypeDef * hsmbus)

Function description

DeInitialize the SMBUS MSP.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_ConfigAnalogFilter

Function name

HAL_StatusTypeDef HAL_SMBUS_ConfigAnalogFilter (SMBUS_HandleTypeDef * hsmbus, uint32_t AnalogFilter)

Function description

Configure Analog noise filter.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **AnalogFilter:** This parameter can be one of the following values:
  - SMBUS_ANALOGFILTER_ENABLE
  - SMBUS_ANALOGFILTER_DISABLE

Return values

- **HAL:** status

HAL_SMBUS_ConfigDigitalFilter

Function name

HAL_StatusTypeDef HAL_SMBUS_ConfigDigitalFilter (SMBUS_HandleTypeDef * hsmbus, uint32_t DigitalFilter)

Function description

Configure Digital noise filter.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **DigitalFilter:** Coefficient of digital noise filter between Min_Data=0x00 and Max_Data=0x0F.

Return values

- **HAL:** status

HAL_SMBUS_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SMBUS_RegisterCallback (SMBUS_HandleTypeDef * hsmbus, HAL_SMBUS_CallbackIDTypeDef CallbackID, pSMBUS_CallbackTypeDef pCallback)

Function description

Register a User SMBUS Callback To be used instead of the weak predefined callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_SMBUS_MASTER_TX_COMPLETE_CB_ID Master Tx Transfer completed callback ID
  - HAL_SMBUS_MASTER_RX_COMPLETE_CB_ID Master Rx Transfer completed callback ID
  - HAL_SMBUS_SLAVE_TX_COMPLETE_CB_ID Slave Tx Transfer completed callback ID
  - HAL_SMBUS_SLAVE_RX_COMPLETE_CB_ID Slave Rx Transfer completed callback ID
  - HAL_SMBUS_LISTEN_COMPLETE_CB_ID Listen Complete callback ID
  - HAL_SMBUS_ERROR_CB_ID Error callback ID
  - HAL_SMBUS_MSPINIT_CB_ID MspInit callback ID
  - HAL_SMBUS_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

Notes

- The HAL_SMBUS_RegisterCallback() may be called before HAL_SMBUS_Init() in HAL_SMBUS_STATE_RESET to register callbacks for HAL_SMBUS_MSPINIT_CB_ID and HAL_SMBUS_MSPDEINIT_CB_ID.

HAL_SMBUS_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SMBUS_UnRegisterCallback (SMBUS_HandleTypeDef * hsmbus, HAL_SMBUS_CallbackIDTypeDef CallbackID)

Function description

Unregister an SMBUS Callback SMBUS callback is redirected to the weak predefined callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values: This parameter can be one of the following values:
  - HAL_SMBUS_MASTER_TX_COMPLETE_CB_ID Master Tx Transfer completed callback ID
  - HAL_SMBUS_MASTER_RX_COMPLETE_CB_ID Master Rx Transfer completed callback ID
  - HAL_SMBUS_SLAVE_TX_COMPLETE_CB_ID Slave Tx Transfer completed callback ID
  - HAL_SMBUS_SLAVE_RX_COMPLETE_CB_ID Slave Rx Transfer completed callback ID
  - HAL_SMBUS_LISTEN_COMPLETE_CB_ID Listen Complete callback ID
  - HAL_SMBUS_ERROR_CB_ID Error callback ID
  - HAL_SMBUS_MSPINIT_CB_ID MspInit callback ID
  - HAL_SMBUS_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

Notes

- The HAL_SMBUS_UnRegisterCallback() may be called before HAL_SMBUS_Init() in HAL_SMBUS_STATE_RESET to un-register callbacks for HAL_SMBUS_MSPINIT_CB_ID and HAL_SMBUS_MSPDEINIT_CB_ID

HAL_SMBUS_RegisterAddrCallback

Function name

HAL_StatusTypeDef HAL_SMBUS_RegisterAddrCallback (SMBUS_HandleTypeDef * hsmbus, pSMBUS_AddrCallbackTypeDef pCallback)

Function description

Register the Slave Address Match SMBUS Callback To be used instead of the weak HAL_SMBUS_AddrCallback() predefined callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **pCallback:** pointer to the Address Match Callback function

Return values

- **HAL:** status

HAL_SMBUS_UnRegisterAddrCallback

Function name

HAL_StatusTypeDef HAL_SMBUS_UnRegisterAddrCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

UnRegister the Slave Address Match SMBUS Callback Info Ready SMBUS Callback is redirected to the weak HAL_SMBUS_AddrCallback() predefined callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **HAL:** status

HAL_SMBUS_IsDeviceReady

Function name

HAL_StatusTypeDef HAL_SMBUS_IsDeviceReady (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout)

Function description

Check if target device is ready for communication.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **Trials:** Number of trials
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SMBUS_Master_Transmit_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_Master_Transmit_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Transmit in master/host SMBUS mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of SMBUS XferOptions definition

Return values

- **HAL:** status

HAL_SMBUS_Master_Receive_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_Master_Receive_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Receive in master/host SMBUS mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of SMBUS XferOptions definition

Return values

- **HAL:** status

HAL_SMBUS_Master_Abort_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_Master_Abort_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress)

Function description

Abort a master/host SMBUS process communication with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface

Return values

- **HAL:** status

Notes

- This abort can be called only if state is ready

HAL_SMBUS_Slave_Transmit_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_Slave_Transmit_IT (SMBUS_HandleTypeDef * hsmbus, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Transmit in slave/device SMBUS mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of SMBUS XferOptions definition

Return values

- **HAL:** status

HAL_SMBUS_Slave_Receive_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_Slave_Receive_IT (SMBUS_HandleTypeDef * hsmbus, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Receive in slave/device SMBUS mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of SMBUS XferOptions definition

Return values

- **HAL:** status

HAL_SMBUS_EnableAlert_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_EnableAlert_IT (SMBUS_HandleTypeDef * hsmbus)

Function description

Enable the SMBUS alert mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.

Return values

- **HAL:** status

HAL_SMBUS_DisableAlert_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_DisableAlert_IT (SMBUS_HandleTypeDef * hsmbus)

Function description

Disable the SMBUS alert mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.

Return values

- **HAL:** status

HAL_SMBUS_EnableListen_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_EnableListen_IT (SMBUS_HandleTypeDef * hsmbus)

Function description

Enable the Address listen mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **HAL:** status

HAL_SMBUS_DisableListen_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_DisableListen_IT (SMBUS_HandleTypeDef * hsmbus)

Function description

Disable the Address listen mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **HAL:** status

HAL_SMBUS_EV_IRQHandler

Function name

void HAL_SMBUS_EV_IRQHandler (SMBUS_HandleTypeDef * hsmbus)

Function description

Handle SMBUS event interrupt request.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_ER_IRQHandler

Function name

void HAL_SMBUS_ER_IRQHandler (SMBUS_HandleTypeDef * hsmbus)

Function description

Handle SMBUS error interrupt request.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_MasterTxCpltCallback

Function name

void HAL_SMBUS_MasterTxCpltCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

Master Tx Transfer completed callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_MasterRxCpltCallback

Function name

void HAL_SMBUS_MasterRxCpltCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

Master Rx Transfer completed callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_SlaveTxCpltCallback

Function name

void HAL_SMBUS_SlaveTxCpltCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

Slave Tx Transfer completed callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_SlaveRxCpltCallback

Function name

void HAL_SMBUS_SlaveRxCpltCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

Slave Rx Transfer completed callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_AddrCallback

Function name

void HAL_SMBUS_AddrCallback (SMBUS_HandleTypeDef * hsmbus, uint8_t TransferDirection, uint16_t AddrMatchCode)

Function description

Slave Address Match callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **TransferDirection:** Master request Transfer Direction (Write/Read)
- **AddrMatchCode:** Address Match Code

Return values

- **None:**

HAL_SMBUS_ListenCpltCallback

Function name

void HAL_SMBUS_ListenCpltCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

Listen Complete callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_ErrorCallback

Function name

void HAL_SMBUS_ErrorCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

SMBUS error callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_GetState

Function name

uint32_t HAL_SMBUS_GetState (SMBUS_HandleTypeDef * hsmbus)

Function description

Return the SMBUS handle state.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **HAL:** state

HAL_SMBUS_GetError

Function name

uint32_t HAL_SMBUS_GetError (SMBUS_HandleTypeDef * hsmbus)

Function description

Return the SMBUS error code.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **SMBUS:** Error Code

81.3 SMBUS Firmware driver defines

The following section lists the various define and macros of the module.

81.3.1 SMBUS

SMBUS

SMBUS addressing mode

SMBUS_ADDRESSINGMODE_7BIT

SMBUS_ADDRESSINGMODE_10BIT

SMBUS Analog Filter

SMBUS_ANALOGFILTER_ENABLE

SMBUS_ANALOGFILTER_DISABLE

SMBUS dual addressing mode

SMBUS_DUALADDRESS_DISABLE

SMBUS_DUALADDRESS_ENABLE

SMBUS Error Code definition

HAL_SMBUS_ERROR_NONE

No error

HAL_SMBUS_ERROR_BERR

BERR error

HAL_SMBUS_ERROR_ARLO

ARLO error

HAL_SMBUS_ERROR_ACKF

ACKF error

HAL_SMBUS_ERROR_OVR

OVR error

HAL_SMBUS_ERROR_HALTIMEOUT

Timeout error

HAL_SMBUS_ERROR_BUSTIMEOUT

Bus Timeout error

HAL_SMBUS_ERROR_ALERT

Alert error

HAL_SMBUS_ERROR_PECERR

PEC error

HAL_SMBUS_ERROR_INVALID_CALLBACK

Invalid Callback error

HAL_SMBUS_ERROR_INVALID_PARAM

Invalid Parameters error

SMBUS Exported Macros

__HAL_SMBUS_RESET_HANDLE_STATE

Description:

- Reset SMBUS handle state.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.

Return value:

- None

__HAL_SMBUS_ENABLE_IT

Description:

- Enable the specified SMBUS interrupts.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __INTERRUPT__: specifies the interrupt source to enable. This parameter can be one of the following values:
  - SMBUS_IT_ERRI Errors interrupt enable
  - SMBUS_IT_TCI Transfer complete interrupt enable
  - SMBUS_IT_STOPI STOP detection interrupt enable
  - SMBUS_IT_NACKI NACK received interrupt enable
  - SMBUS_IT_ADDRI Address match interrupt enable
  - SMBUS_IT_RXI RX interrupt enable
  - SMBUS_IT_TXI TX interrupt enable

Return value:

- None

__HAL_SMBUS_DISABLE_IT

Description:

- Disable the specified SMBUS interrupts.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __INTERRUPT__: specifies the interrupt source to disable. This parameter can be one of the following values:
  - SMBUS_IT_ERRI Errors interrupt enable
  - SMBUS_IT_TCI Transfer complete interrupt enable
  - SMBUS_IT_STOPI STOP detection interrupt enable
  - SMBUS_IT_NACKI NACK received interrupt enable
  - SMBUS_IT_ADDRI Address match interrupt enable
  - SMBUS_IT_RXI RX interrupt enable
  - SMBUS_IT_TXI TX interrupt enable

Return value:

- None

__HAL_SMBUS_GET_IT_SOURCE

Description:

- Check whether the specified SMBUS interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __INTERRUPT__: specifies the SMBUS interrupt source to check. This parameter can be one of the following values:
  - SMBUS_IT_ERRI Errors interrupt enable
  - SMBUS_IT_TCI Transfer complete interrupt enable
  - SMBUS_IT_STOPI STOP detection interrupt enable
  - SMBUS_IT_NACKI NACK received interrupt enable
  - SMBUS_IT_ADDRI Address match interrupt enable
  - SMBUS_IT_RXI RX interrupt enable
  - SMBUS_IT_TXI TX interrupt enable

Return value:

- The: new state of __IT__ (SET or RESET).

SMBUS_FLAG_MASK

Description:

- Check whether the specified SMBUS flag is set or not.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SMBUS_FLAG_TXE Transmit data register empty
  - SMBUS_FLAG_TXIS Transmit interrupt status
  - SMBUS_FLAG_RXNE Receive data register not empty
  - SMBUS_FLAG_ADDR Address matched (slave mode)
  - SMBUS_FLAG_AF NACK received flag
  - SMBUS_FLAG_STOPF STOP detection flag
  - SMBUS_FLAG_TC Transfer complete (master mode)
  - SMBUS_FLAG_TCR Transfer complete reload
  - SMBUS_FLAG_BERR Bus error
  - SMBUS_FLAG_ARLO Arbitration lost
  - SMBUS_FLAG_OVR Overrun/Underrun
  - SMBUS_FLAG_PECERR PEC error in reception
  - SMBUS_FLAG_TIMEOUT Timeout or Tlow detection flag
  - SMBUS_FLAG_ALERT SMBus alert
  - SMBUS_FLAG_BUSY Bus busy
  - SMBUS_FLAG_DIR Transfer direction (slave mode)

Return value:

- The: new state of __FLAG__ (SET or RESET).

__HAL_SMBUS_GET_FLAG

__HAL_SMBUS_CLEAR_FLAG

Description:

- Clear the SMBUS pending flags which are cleared by writing 1 in a specific bit.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - SMBUS_FLAG_TXE Transmit data register empty
  - SMBUS_FLAG_ADDR Address matched (slave mode)
  - SMBUS_FLAG_AF NACK received flag
  - SMBUS_FLAG_STOPF STOP detection flag
  - SMBUS_FLAG_BERR Bus error
  - SMBUS_FLAG_ARLO Arbitration lost
  - SMBUS_FLAG_OVR Overrun/Underrun
  - SMBUS_FLAG_PECERR PEC error in reception
  - SMBUS_FLAG_TIMEOUT Timeout or Tlow detection flag
  - SMBUS_FLAG_ALERT SMBus alert

Return value:

- None

__HAL_SMBUS_ENABLE

Description:

- Enable the specified SMBUS peripheral.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.

Return value:

- None

__HAL_SMBUS_DISABLE

Description:

- Disable the specified SMBUS peripheral.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.

Return value:

- None

__HAL_SMBUS_GENERATE_NACK

Description:

- Generate a Non-Acknowledge SMBUS peripheral in Slave mode.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.

Return value:

- None

SMBUS Flag definition

SMBUS_FLAG_TXE

SMBUS_FLAG_TXIS

SMBUS_FLAG_RXNE

SMBUS_FLAG_ADDR

SMBUS_FLAG_AF

SMBUS_FLAG_STOPF

SMBUS_FLAG_TC

SMBUS_FLAG_TCR

SMBUS_FLAG_BERR

SMBUS_FLAG_ARLO

SMBUS_FLAG_OVR

SMBUS_FLAG_PECERR

SMBUS_FLAG_TIMEOUT

SMBUS_FLAG_ALERT

SMBUS_FLAG_BUSY

SMBUS_FLAG_DIR

SMBUS general call addressing mode

SMBUS_GENERALCALL_DISABLE

SMBUS_GENERALCALL_ENABLE

SMBUS Interrupt configuration definition

SMBUS_IT_ERRI

SMBUS_IT_TCI

SMBUS_IT_STOPI

SMBUS_IT_NACKI

SMBUS_IT_ADDRI

SMBUS_IT_RXI

SMBUS_IT_TXI

SMBUS_IT_TX

SMBUS_IT_RX

SMBUS_IT_ALERT

SMBUS_IT_ADDR

SMBUS nostretch mode

SMBUS_NOSTRETCH_DISABLE

SMBUS_NOSTRETCH_ENABLE

SMBUS ownaddress2 masks

SMBUS_OA2_NOMASK

SMBUS_OA2_MASK01

SMBUS_OA2_MASK02

SMBUS_OA2_MASK03

SMBUS_OA2_MASK04

SMBUS_OA2_MASK05

SMBUS_OA2_MASK06

SMBUS_OA2_MASK07

SMBUS packet error check mode

SMBUS_PEC_DISABLE

SMBUS_PEC_ENABLE

SMBUS peripheral mode

SMBUS_PERIPHERAL_MODE_SMBUS_HOST

SMBUS_PERIPHERAL_MODE_SMBUS_SLAVE

SMBUS_PERIPHERAL_MODE_SMBUS_SLAVE_ARP

SMBUS ReloadEndMode definition

SMBUS_SOFTEND_MODE

SMBUS_RELOAD_MODE

SMBUS_AUTOEND_MODE

SMBUS_SENDPEC_MODE

SMBUS StartStopMode definition

SMBUS_NO_STARTSTOP

SMBUS_GENERATE_STOP

SMBUS_GENERATE_START_READ

SMBUS_GENERATE_START_WRITE

SMBUS XferOptions definition

SMBUS_FIRST_FRAME

SMBUS_NEXT_FRAME

SMBUS_FIRST_AND_LAST_FRAME_NO_PEC

SMBUS_LAST_FRAME_NO_PEC

SMBUS_FIRST_FRAME_WITH_PEC

SMBUS_FIRST_AND_LAST_FRAME_WITH_PEC

SMBUS_LAST_FRAME_WITH_PEC

SMBUS_OTHER_FRAME_NO_PEC

SMBUS_OTHER_FRAME_WITH_PEC

SMBUS_OTHER_AND_LAST_FRAME_NO_PEC

SMBUS_OTHER_AND_LAST_FRAME_WITH_PEC

82 HAL SMBUS Extension Driver

82.1 SMBUSEx Firmware driver API description

The following section lists the various functions of the SMBUSEx library.

82.1.1 SMBUS peripheral Extended features

Comparing to other previous devices, the SMBUS interface for STM32H7xx devices contains the following additional features

- Disable or enable wakeup from Stop mode(s)
- Disable or enable Fast Mode Plus

82.1.2 How to use this driver

82.1.3 WakeUp Mode Functions

This section provides functions allowing to:

- Configure Wake Up Feature

This section contains the following APIs:

- ***HAL_SMBUSEx_EnableWakeUp()***
- ***HAL_SMBUSEx_DisableWakeUp()***

82.1.4 Fast Mode Plus Functions

This section provides functions allowing to:

- Configure Fast Mode Plus

This section contains the following APIs:

- ***HAL_SMBUSEx_EnableFastModePlus()***
- ***HAL_SMBUSEx_DisableFastModePlus()***

82.1.5 Detailed description of functions

HAL_SMBUSEx_EnableWakeUp

Function name

HAL_StatusTypeDef HAL_SMBUSEx_EnableWakeUp (SMBUS_HandleTypeDef * hsmbus)

Function description

Enable SMBUS wakeup from Stop mode(s).

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.

Return values

- **HAL:** status

HAL_SMBUSEx_DisableWakeUp

Function name

HAL_StatusTypeDef HAL_SMBUSEx_DisableWakeUp (SMBUS_HandleTypeDef * hsmbus)

Function description

Disable SMBUS wakeup from Stop mode(s).

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.

Return values

- **HAL:** status

HAL_SMBUSEx_EnableFastModePlus

Function name

void HAL_SMBUSEx_EnableFastModePlus (uint32_t ConfigFastModePlus)

Function description

Enable the SMBUS fast mode plus driving capability.

Parameters

- **ConfigFastModePlus:** Selects the pin. This parameter can be one of the SMBUS Extended Fast Mode Plus values

Return values

- **None:**

Notes

- For I2C1, fast mode plus driving capability can be enabled on all selected I2C1 pins using SMBUS_FASTMODEPLUS_I2C1 parameter or independently on each one of the following pins PB6, PB7, PB8 and PB9.
- For remaining I2C1 pins (PA14, PA15...) fast mode plus driving capability can be enabled only by using SMBUS_FASTMODEPLUS_I2C1 parameter.
- For all I2C2 pins fast mode plus driving capability can be enabled only by using SMBUS_FASTMODEPLUS_I2C2 parameter.
- For all I2C3 pins fast mode plus driving capability can be enabled only by using SMBUS_FASTMODEPLUS_I2C3 parameter.
- For all I2C4 pins fast mode plus driving capability can be enabled only by using SMBUS_FASTMODEPLUS_I2C4 parameter.
- For all I2C5 pins fast mode plus driving capability can be enabled only by using SMBUS_FASTMODEPLUS_I2C5 parameter.

HAL_SMBUSEx_DisableFastModePlus

Function name

void HAL_SMBUSEx_DisableFastModePlus (uint32_t ConfigFastModePlus)

Function description

Disable the SMBUS fast mode plus driving capability.

Parameters

- **ConfigFastModePlus:** Selects the pin. This parameter can be one of the SMBUS Extended Fast Mode Plus values

Return values

- **None:**

Notes

- For I2C1, fast mode plus driving capability can be disabled on all selected I2C1 pins using SMBUS_FASTMODEPLUS_I2C1 parameter or independently on each one of the following pins PB6, PB7, PB8 and PB9.
- For remaining I2C1 pins (PA14, PA15...) fast mode plus driving capability can be disabled only by using SMBUS_FASTMODEPLUS_I2C1 parameter.
- For all I2C2 pins fast mode plus driving capability can be disabled only by using SMBUS_FASTMODEPLUS_I2C2 parameter.
- For all I2C3 pins fast mode plus driving capability can be disabled only by using SMBUS_FASTMODEPLUS_I2C3 parameter.
- For all I2C4 pins fast mode plus driving capability can be disabled only by using SMBUS_FASTMODEPLUS_I2C4 parameter.
- For all I2C5 pins fast mode plus driving capability can be disabled only by using SMBUS_FASTMODEPLUS_I2C5 parameter.

82.2 SMBUSEx Firmware driver defines

The following section lists the various define and macros of the module.

82.2.1 SMBUSEx

SMBUSEx

SMBUS Extended Fast Mode Plus

SMBUS_FMP_NOT_SUPPORTED

Fast Mode Plus not supported

SMBUS_FASTMODEPLUS_PB6

Enable Fast Mode Plus on PB6

SMBUS_FASTMODEPLUS_PB7

Enable Fast Mode Plus on PB7

SMBUS_FASTMODEPLUS_PB8

Enable Fast Mode Plus on PB8

SMBUS_FASTMODEPLUS_PB9

Enable Fast Mode Plus on PB9

SMBUS_FASTMODEPLUS_I2C1

Enable Fast Mode Plus on I2C1 pins

SMBUS_FASTMODEPLUS_I2C2

Enable Fast Mode Plus on I2C2 pins

SMBUS_FASTMODEPLUS_I2C3

Enable Fast Mode Plus on I2C3 pins

SMBUS_FASTMODEPLUS_I2C4

Enable Fast Mode Plus on I2C4 pins

SMBUS_FASTMODEPLUS_I2C5

Fast Mode Plus I2C5 not supported

83 HAL SPDIFRX Generic Driver

83.1 SPDIFRX Firmware driver registers structures

83.1.1 SPDIFRX_InitTypeDef

SPDIFRX_InitTypeDef is defined in the stm32h7xx_hal_spdifrx.h

Data Fields

- ***uint32_t InputSelection***
- ***uint32_t Retries***
- ***uint32_t WaitForActivity***
- ***uint32_t ChannelSelection***
- ***uint32_t DataFormat***
- ***uint32_t StereoMode***
- ***uint32_t PreambleTypeMask***
- ***uint32_t ChannelStatusMask***
- ***uint32_t ValidityBitMask***
- ***uint32_t ParityErrorMask***
- ***FunctionalState SymbolClockGen***
- ***FunctionalState BackupSymbolClockGen***

Field Documentation

- ***uint32_t SPDIFRX_InitTypeDef::InputSelection***

  Specifies the SPDIF input selection. This parameter can be a value of ***SPDIFRX_Input_Selection***
- ***uint32_t SPDIFRX_InitTypeDef::Retries***

  Specifies the Maximum allowed re-tries during synchronization phase. This parameter can be a value of ***SPDIFRX_Max_Retries***
- ***uint32_t SPDIFRX_InitTypeDef::WaitForActivity***

  Specifies the wait for activity on SPDIF selected input. This parameter can be a value of ***SPDIFRX_Wait_For_Activity***.
- ***uint32_t SPDIFRX_InitTypeDef::ChannelSelection***

  Specifies whether the control flow will take the channel status from channel A or B. This parameter can be a value of ***SPDIFRX_Channel_Selection***
- ***uint32_t SPDIFRX_InitTypeDef::DataFormat***

  Specifies the Data samples format (LSB, MSB, ...). This parameter can be a value of ***SPDIFRX_Data_Format***
- ***uint32_t SPDIFRX_InitTypeDef::StereoMode***

  Specifies whether the peripheral is in stereo or mono mode. This parameter can be a value of ***SPDIFRX_Stereo_Mode***
- ***uint32_t SPDIFRX_InitTypeDef::PreambleTypeMask***

  Specifies whether The preamble type bits are copied or not into the received frame. This parameter can be a value of ***SPDIFRX_PT_Mask***
- ***uint32_t SPDIFRX_InitTypeDef::ChannelStatusMask***

  Specifies whether the channel status and user bits are copied or not into the received frame. This parameter can be a value of ***SPDIFRX_ChannelStatus_Mask***
- ***uint32_t SPDIFRX_InitTypeDef::ValidityBitMask***

  Specifies whether the validity bit is copied or not into the received frame. This parameter can be a value of ***SPDIFRX_V_Mask***

- ***uint32_t SPDIFRX_InitTypeDef::ParityErrorMask***

  Specifies whether the parity error bit is copied or not into the received frame. This parameter can be a value of ***SPDIFRX_PE_Mask***
- ***FunctionalState SPDIFRX_InitTypeDef::SymbolClockGen***

  Enable/Disable the SPDIFRX Symbol Clock generation. This parameter can be set to Enable or Disable
- ***FunctionalState SPDIFRX_InitTypeDef::BackupSymbolClockGen***

  Enable/Disable the SPDIFRX Backup Symbol Clock generation. This parameter can be set to Enable or Disable

83.1.2 SPDIFRX_SetDataFormatTypeDef

SPDIFRX_SetDataFormatTypeDef is defined in the stm32h7xx_hal_spdifrx.h

Data Fields

- ***uint32_t DataFormat***
- ***uint32_t StereoMode***
- ***uint32_t PreambleTypeMask***
- ***uint32_t ChannelStatusMask***
- ***uint32_t ValidityBitMask***
- ***uint32_t ParityErrorMask***

Field Documentation

- ***uint32_t SPDIFRX_SetDataFormatTypeDef::DataFormat***

  Specifies the Data samples format (LSB, MSB, ...). This parameter can be a value of ***SPDIFRX_Data_Format***
- ***uint32_t SPDIFRX_SetDataFormatTypeDef::StereoMode***

  Specifies whether the peripheral is in stereo or mono mode. This parameter can be a value of ***SPDIFRX_Stereo_Mode***
- ***uint32_t SPDIFRX_SetDataFormatTypeDef::PreambleTypeMask***

  Specifies whether The preamble type bits are copied or not into the received frame. This parameter can be a value of ***SPDIFRX_PT_Mask***
- ***uint32_t SPDIFRX_SetDataFormatTypeDef::ChannelStatusMask***

  Specifies whether the channel status and user bits are copied or not into the received frame. This parameter can be a value of ***SPDIFRX_ChannelStatus_Mask***
- ***uint32_t SPDIFRX_SetDataFormatTypeDef::ValidityBitMask***

  Specifies whether the validity bit is copied or not into the received frame. This parameter can be a value of ***SPDIFRX_V_Mask***
- ***uint32_t SPDIFRX_SetDataFormatTypeDef::ParityErrorMask***

  Specifies whether the parity error bit is copied or not into the received frame. This parameter can be a value of ***SPDIFRX_PE_Mask***

83.1.3 __SPDIFRX_HandleTypeDef

__SPDIFRX_HandleTypeDef is defined in the stm32h7xx_hal_spdifrx.h

Data Fields

- ***SPDIFRX_TypeDef * Instance***
- ***SPDIFRX_InitTypeDef Init***
- ***uint32_t * pRxBuffPtr***
- ***uint32_t * pCsBuffPtr***
- ***__IO uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***__IO uint16_t CsXferSize***
- ***__IO uint16_t CsXferCount***
- ***DMA_HandleTypeDef * hdmaCsRx***
- ***DMA_HandleTypeDef * hdmaDrRx***

- *__IO HAL_LockTypeDef Lock*
- *__IO HAL_SPDIFRX_StateTypeDef State*
- *__IO uint32_t ErrorCode*
- *void(* RxHalfCpltCallback*
- *void(* RxCpltCallback*
- *void(* CxHalfCpltCallback*
- *void(* CxCpltCallback*
- *void(* ErrorCallback*
- *void(* MspInitCallback*
- *void(* MspDeInitCallback*

Field Documentation

- *SPDIFRX_TypeDef* __SPDIFRX_HandleTypeDef::Instance*
- *SPDIFRX_InitTypeDef __SPDIFRX_HandleTypeDef::Init*
- *uint32_t* __SPDIFRX_HandleTypeDef::pRxBuffPtr*
- *uint32_t* __SPDIFRX_HandleTypeDef::pCsBuffPtr*
- *__IO uint16_t __SPDIFRX_HandleTypeDef::RxXferSize*
- *__IO uint16_t __SPDIFRX_HandleTypeDef::RxXferCount*
- *__IO uint16_t __SPDIFRX_HandleTypeDef::CsXferSize*
- *__IO uint16_t __SPDIFRX_HandleTypeDef::CsXferCount*
- *DMA_HandleTypeDef* __SPDIFRX_HandleTypeDef::hdmaCsRx*
- *DMA_HandleTypeDef* __SPDIFRX_HandleTypeDef::hdmaDrRx*
- *__IO HAL_LockTypeDef __SPDIFRX_HandleTypeDef::Lock*
- *__IO HAL_SPDIFRX_StateTypeDef __SPDIFRX_HandleTypeDef::State*
- *__IO uint32_t __SPDIFRX_HandleTypeDef::ErrorCode*
- *void(* __SPDIFRX_HandleTypeDef::RxHalfCpltCallback)(struct __SPDIFRX_HandleTypeDef *hspdif)*
  SPDIFRX Data flow half completed callback
- *void(* __SPDIFRX_HandleTypeDef::RxCpltCallback)(struct __SPDIFRX_HandleTypeDef *hspdif)*
  SPDIFRX Data flow completed callback
- *void(* __SPDIFRX_HandleTypeDef::CxHalfCpltCallback)(struct __SPDIFRX_HandleTypeDef *hspdif)*
  SPDIFRX Control flow half completed callback
- *void(* __SPDIFRX_HandleTypeDef::CxCpltCallback)(struct __SPDIFRX_HandleTypeDef *hspdif)*
  SPDIFRX Control flow completed callback
- *void(* __SPDIFRX_HandleTypeDef::ErrorCallback)(struct __SPDIFRX_HandleTypeDef *hspdif)*
  SPDIFRX error callback
- *void(* __SPDIFRX_HandleTypeDef::MspInitCallback)(struct __SPDIFRX_HandleTypeDef *hspdif)*
  SPDIFRX Msp Init callback
- *void(* __SPDIFRX_HandleTypeDef::MspDeInitCallback)(struct __SPDIFRX_HandleTypeDef *hspdif)*
  SPDIFRX Msp DeInit callback

83.2 SPDIFRX Firmware driver API description

The following section lists the various functions of the SPDIFRX library.

83.2.1 How to use this driver

The SPDIFRX HAL driver can be used as follow:

1. Declare SPDIFRX_HandleTypeDef handle structure.

2. Initialize the SPDIFRX low level resources by implement the HAL_SPDIFRX_MspInit() API:
   a. Enable the SPDIFRX interface clock.
   b. SPDIFRX pins configuration:
      - Enable the clock for the SPDIFRX GPIOs.
      - Configure these SPDIFRX pins as alternate function pull-up.
   c. NVIC configuration if you need to use interrupt process (HAL_SPDIFRX_ReceiveCtrlFlow_IT() and HAL_SPDIFRX_ReceiveDataFlow_IT() API's).
      - Configure the SPDIFRX interrupt priority.
      - Enable the NVIC SPDIFRX IRQ handle.
   d. DMA Configuration if you need to use DMA process (HAL_SPDIFRX_ReceiveDataFlow_DMA() and HAL_SPDIFRX_ReceiveCtrlFlow_DMA() API's).
      - Declare a DMA handle structure for the reception of the Data Flow channel.
      - Declare a DMA handle structure for the reception of the Control Flow channel.
      - Enable the DMAx interface clock.
      - Configure the declared DMA handle structure CtrlRx/DataRx with the required parameters.
      - Configure the DMA Channel.
      - Associate the initialized DMA handle to the SPDIFRX DMA CtrlRx/DataRx handle.
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA CtrlRx/DataRx channel.
3. Program the input selection, re-tries number, wait for activity, channel status selection, data format, stereo mode and masking of user bits using HAL_SPDIFRX_Init() function.

Note: The specific SPDIFRX interrupts (RXNE/CSRNE and Error Interrupts) will be managed using the macros __SPDIFRX_ENABLE_IT() and __SPDIFRX_DISABLE_IT() inside the receive process.

Note: Make sure that ck_spdif clock is configured.

4. Three operation modes are available within this driver :

Polling mode for reception operation (for debug purpose)

- Receive data flow in blocking mode using HAL_SPDIFRX_ReceiveDataFlow()
- Receive control flow of data in blocking mode using HAL_SPDIFRX_ReceiveCtrlFlow()

Interrupt mode for reception operation

- Receive an amount of data (Data Flow) in non blocking mode using HAL_SPDIFRX_ReceiveDataFlow_IT()
- Receive an amount of data (Control Flow) in non blocking mode using HAL_SPDIFRX_ReceiveCtrlFlow_IT()
- At reception end of half transfer HAL_SPDIFRX_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_SPDIFRX_RxHalfCpltCallback
- At reception end of transfer HAL_SPDIFRX_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_SPDIFRX_RxCpltCallback
- In case of transfer Error, HAL_SPDIFRX_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SPDIFRX_ErrorCallback

DMA mode for reception operation

- Receive an amount of data (Data Flow) in non blocking mode (DMA) using HAL_SPDIFRX_ReceiveDataFlow_DMA()
- Receive an amount of data (Control Flow) in non blocking mode (DMA) using HAL_SPDIFRX_ReceiveCtrlFlow_DMA()
- At reception end of half transfer HAL_SPDIFRX_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_SPDIFRX_RxHalfCpltCallback
- At reception end of transfer HAL_SPDIFRX_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_SPDIFRX_RxCpltCallback
- In case of transfer Error, HAL_SPDIFRX_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SPDIFRX_ErrorCallback
- Stop the DMA Transfer using HAL_SPDIFRX_DMAStop()

SPDIFRX HAL driver macros list

Below the list of most used macros in SPDIFRX HAL driver.

- __HAL_SPDIFRX_IDLE: Disable the specified SPDIFRX peripheral (IDEL State)
- __HAL_SPDIFRX_SYNC: Enable the synchronization state of the specified SPDIFRX peripheral (SYNC State)
- __HAL_SPDIFRX_RCV: Enable the receive state of the specified SPDIFRX peripheral (RCV State)
- __HAL_SPDIFRX_ENABLE_IT : Enable the specified SPDIFRX interrupts
- __HAL_SPDIFRX_DISABLE_IT : Disable the specified SPDIFRX interrupts
- __HAL_SPDIFRX_GET_FLAG: Check whether the specified SPDIFRX flag is set or not.

Note: You can refer to the SPDIFRX HAL driver header file for more useful macros

Callback registration

83.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the SPDIFRX peripheral:

- User must Implement HAL_SPDIFRX_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_SPDIFRX_Init() to configure the SPDIFRX peripheral with the selected configuration:
  - Input Selection (IN0, IN1,...)
  - Maximum allowed re-tries during synchronization phase
  - Wait for activity on SPDIF selected input
  - Channel status selection (from channel A or B)
  - Data format (LSB, MSB, ...)
  - Stereo mode
  - User bits masking (PT,C,U,V,...)
- Call the function HAL_SPDIFRX_DeInit() to restore the default configuration of the selected SPDIFRXx peripheral.

This section contains the following APIs:

- ***HAL_SPDIFRX_Init()***
- ***HAL_SPDIFRX_DeInit()***
- ***HAL_SPDIFRX_MspInit()***
- ***HAL_SPDIFRX_MspDeInit()***
- ***HAL_SPDIFRX_RegisterCallback()***
- ***HAL_SPDIFRX_UnRegisterCallback()***
- ***HAL_SPDIFRX_SetDataFormat()***

83.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the SPDIFRX data transfers.

1. There is two mode of transfer:
   - Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
   - No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer start-up. The end of the data processing will be indicated through the dedicated SPDIFRX IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
2. Blocking mode functions are :
   - HAL_SPDIFRX_ReceiveDataFlow()
   - HAL_SPDIFRX_ReceiveCtrlFlow() (+@) Do not use blocking mode to receive both control and data flow at the same time.

3. No-Blocking mode functions with Interrupt are :
   – HAL_SPDIFRX_ReceiveCtrlFlow_IT()
   – HAL_SPDIFRX_ReceiveDataFlow_IT()
4. No-Blocking mode functions with DMA are :
   – HAL_SPDIFRX_ReceiveCtrlFlow_DMA()
   – HAL_SPDIFRX_ReceiveDataFlow_DMA()
5. A set of Transfer Complete Callbacks are provided in No_Blocking mode:
   – HAL_SPDIFRX_RxCpltCallback()
   – HAL_SPDIFRX_CxCpltCallback()

This section contains the following APIs:

- ***HAL_SPDIFRX_ReceiveDataFlow()***
- ***HAL_SPDIFRX_ReceiveCtrlFlow()***
- ***HAL_SPDIFRX_ReceiveDataFlow_IT()***
- ***HAL_SPDIFRX_ReceiveCtrlFlow_IT()***
- ***HAL_SPDIFRX_ReceiveDataFlow_DMA()***
- ***HAL_SPDIFRX_ReceiveCtrlFlow_DMA()***
- ***HAL_SPDIFRX_DMAStop()***
- ***HAL_SPDIFRX_IRQHandler()***
- ***HAL_SPDIFRX_RxHalfCpltCallback()***
- ***HAL_SPDIFRX_RxCpltCallback()***
- ***HAL_SPDIFRX_CxHalfCpltCallback()***
- ***HAL_SPDIFRX_CxCpltCallback()***
- ***HAL_SPDIFRX_ErrorCallback()***

83.2.4 Peripheral State and Errors functions

This subsection permit to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- ***HAL_SPDIFRX_GetState()***
- ***HAL_SPDIFRX_GetError()***

83.2.5 Detailed description of functions

HAL_SPDIFRX_Init

Function name

HAL_StatusTypeDef HAL_SPDIFRX_Init (SPDIFRX_HandleTypeDef * hspdif)

Function description

Initializes the SPDIFRX according to the specified parameters in the SPDIFRX_InitTypeDef and create the associated handle.

Parameters

- **hspdif:** SPDIFRX handle

Return values

- **HAL:** status

HAL_SPDIFRX_DeInit

Function name

HAL_StatusTypeDef HAL_SPDIFRX_DeInit (SPDIFRX_HandleTypeDef * hspdif)

Function description

DeInitializes the SPDIFRX peripheral.

Parameters

- **hspdif:** SPDIFRX handle

Return values

- **HAL:** status

HAL_SPDIFRX_MspInit

Function name

void HAL_SPDIFRX_MspInit (SPDIFRX_HandleTypeDef * hspdif)

Function description

SPDIFRX MSP Init.

Parameters

- **hspdif:** SPDIFRX handle

Return values

- **None:**

HAL_SPDIFRX_MspDeInit

Function name

void HAL_SPDIFRX_MspDeInit (SPDIFRX_HandleTypeDef * hspdif)

Function description

SPDIFRX MSP DeInit.

Parameters

- **hspdif:** SPDIFRX handle

Return values

- **None:**

HAL_SPDIFRX_SetDataFormat

Function name

HAL_StatusTypeDef HAL_SPDIFRX_SetDataFormat (SPDIFRX_HandleTypeDef * hspdif, SPDIFRX_SetDataFormatTypeDef sDataFormat)

Function description

Set the SPDIFRX data format according to the specified parameters in the SPDIFRX_InitTypeDef.

Parameters

- **hspdif:** SPDIFRX handle
- **sDataFormat:** SPDIFRX data format

Return values

- **HAL:** status

HAL_SPDIFRX_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SPDIFRX_RegisterCallback (SPDIFRX_HandleTypeDef * hspdif, HAL_SPDIFRX_CallbackIDTypeDef CallbackID, pSPDIFRX_CallbackTypeDef pCallback)

Function description

Register a User SPDIFRX Callback To be used instead of the weak predefined callback.

Parameters

- **hspdif:** SPDIFRX handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_SPDIFRX_RX_HALF_CB_ID SPDIFRX Data flow half completed callback ID
  - HAL_SPDIFRX_RX_CPLT_CB_ID SPDIFRX Data flow completed callback ID
  - HAL_SPDIFRX_CX_HALF_CB_ID SPDIFRX Control flow half completed callback ID
  - HAL_SPDIFRX_CX_CPLT_CB_ID SPDIFRX Control flow completed callback ID
  - HAL_SPDIFRX_ERROR_CB_ID SPDIFRX error callback ID
  - HAL_SPDIFRX_MSPINIT_CB_ID MspInit callback ID
  - HAL_SPDIFRX_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_SPDIFRX_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SPDIFRX_UnRegisterCallback (SPDIFRX_HandleTypeDef * hspdif, HAL_SPDIFRX_CallbackIDTypeDef CallbackID)

Function description

Unregister a SPDIFRX Callback SPDIFRX callback is redirected to the weak predefined callback.

Parameters

- **hspdif:** SPDIFRX handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_SPDIFRX_RX_HALF_CB_ID SPDIFRX Data flow half completed callback ID
  - HAL_SPDIFRX_RX_CPLT_CB_ID SPDIFRX Data flow completed callback ID
  - HAL_SPDIFRX_CX_HALF_CB_ID SPDIFRX Control flow half completed callback ID
  - HAL_SPDIFRX_CX_CPLT_CB_ID SPDIFRX Control flow completed callback ID
  - HAL_SPDIFRX_ERROR_CB_ID SPDIFRX error callback ID
  - HAL_SPDIFRX_MSPINIT_CB_ID MspInit callback ID
  - HAL_SPDIFRX_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_SPDIFRX_ReceiveDataFlow

Function name

HAL_StatusTypeDef HAL_SPDIFRX_ReceiveDataFlow (SPDIFRX_HandleTypeDef * hspdif, uint32_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receives an amount of data (Data Flow) in blocking mode.

Parameters

- **hspdif:** pointer to SPDIFRX_HandleTypeDef structure that contains the configuration information for SPDIFRX module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SPDIFRX_ReceiveCtrlFlow

Function name

HAL_StatusTypeDef HAL_SPDIFRX_ReceiveCtrlFlow (SPDIFRX_HandleTypeDef * hspdif, uint32_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receives an amount of data (Control Flow) in blocking mode.

Parameters

- **hspdif:** pointer to a SPDIFRX_HandleTypeDef structure that contains the configuration information for SPDIFRX module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SPDIFRX_ReceiveCtrlFlow_IT

Function name

HAL_StatusTypeDef HAL_SPDIFRX_ReceiveCtrlFlow_IT (SPDIFRX_HandleTypeDef * hspdif, uint32_t * pData, uint16_t Size)

Function description

Receive an amount of data (Control Flow) with Interrupt.

Parameters

- **hspdif:** SPDIFRX handle
- **pData:** a 32-bit pointer to the Receive data buffer.
- **Size:** number of data sample (Control Flow) to be received

Return values

- **HAL:** status

HAL_SPDIFRX_ReceiveDataFlow_IT

Function name

HAL_StatusTypeDef HAL_SPDIFRX_ReceiveDataFlow_IT (SPDIFRX_HandleTypeDef * hspdif, uint32_t * pData, uint16_t Size)

Function description

Receive an amount of data (Data Flow) in non-blocking mode with Interrupt.

Parameters

- **hspdif:** SPDIFRX handle
- **pData:** a 32-bit pointer to the Receive data buffer.
- **Size:** number of data sample to be received .

Return values

- **HAL:** status

HAL_SPDIFRX_IRQHandler

Function name

void HAL_SPDIFRX_IRQHandler (SPDIFRX_HandleTypeDef * hspdif)

Function description

This function handles SPDIFRX interrupt request.

Parameters

- **hspdif:** SPDIFRX handle

Return values

- **HAL:** status

HAL_SPDIFRX_ReceiveCtrlFlow_DMA

Function name

HAL_StatusTypeDef HAL_SPDIFRX_ReceiveCtrlFlow_DMA (SPDIFRX_HandleTypeDef * hspdif, uint32_t * pData, uint16_t Size)

Function description

Receive an amount of data (Control Flow) with DMA.

Parameters

- **hspdif:** SPDIFRX handle
- **pData:** a 32-bit pointer to the Receive data buffer.
- **Size:** number of data (Control Flow) sample to be received

Return values

- **HAL:** status

HAL_SPDIFRX_ReceiveDataFlow_DMA

Function name

HAL_StatusTypeDef HAL_SPDIFRX_ReceiveDataFlow_DMA (SPDIFRX_HandleTypeDef * hspdif, uint32_t * pData, uint16_t Size)

Function description

Receive an amount of data (Data Flow) mode with DMA.

Parameters

- **hspdif:** SPDIFRX handle
- **pData:** a 32-bit pointer to the Receive data buffer.
- **Size:** number of data sample to be received

Return values

- **HAL:** status

HAL_SPDIFRX_DMAStop

Function name

HAL_StatusTypeDef HAL_SPDIFRX_DMAStop (SPDIFRX_HandleTypeDef * hspdif)

Function description

stop the audio stream receive from the Media.

Parameters

- **hspdif:** SPDIFRX handle

Return values

- **None:**

HAL_SPDIFRX_RxHalfCpltCallback

Function name

void HAL_SPDIFRX_RxHalfCpltCallback (SPDIFRX_HandleTypeDef * hspdif)

Function description

Rx Transfer (Data flow) half completed callbacks.

Parameters

- **hspdif:** SPDIFRX handle

Return values

- **None:**

HAL_SPDIFRX_RxCpltCallback

Function name

void HAL_SPDIFRX_RxCpltCallback (SPDIFRX_HandleTypeDef * hspdif)

Function description

Rx Transfer (Data flow) completed callbacks.

Parameters

- **hspdif:** SPDIFRX handle

Return values

- **None:**

HAL_SPDIFRX_ErrorCallback

Function name

void HAL_SPDIFRX_ErrorCallback (SPDIFRX_HandleTypeDef * hspdif)

Function description

SPDIFRX error callbacks.

Parameters

- **hspdif:** SPDIFRX handle

Return values

- **None:**

HAL_SPDIFRX_CxHalfCpltCallback

Function name

void HAL_SPDIFRX_CxHalfCpltCallback (SPDIFRX_HandleTypeDef * hspdif)

Function description

Rx (Control flow) Transfer half completed callbacks.

Parameters

- **hspdif:** SPDIFRX handle

Return values

- **None:**

HAL_SPDIFRX_CxCpltCallback

Function name

void HAL_SPDIFRX_CxCpltCallback (SPDIFRX_HandleTypeDef * hspdif)

Function description

Rx Transfer (Control flow) completed callbacks.

Parameters

- **hspdif:** SPDIFRX handle

Return values

- **None:**

HAL_SPDIFRX_GetState

Function name

HAL_SPDIFRX_StateTypeDef HAL_SPDIFRX_GetState (SPDIFRX_HandleTypeDef const *const hspdif)

Function description

Return the SPDIFRX state.

Parameters

- **hspdif:** SPDIFRX handle

Return values

- **HAL:** state

HAL_SPDIFRX_GetError

Function name

uint32_t HAL_SPDIFRX_GetError (SPDIFRX_HandleTypeDef const *const hspdif)

Function description

Return the SPDIFRX error code.

Parameters

- **hspdif:** SPDIFRX handle

Return values

- **SPDIFRX:** Error Code

83.3 SPDIFRX Firmware driver defines

The following section lists the various define and macros of the module.

83.3.1 SPDIFRX

SPDIFRX

SPDIFRX Channel Status Mask

SPDIFRX_CHANNELSTATUS_OFF

SPDIFRX_CHANNELSTATUS_ON

SPDIFRX Channel Selection

SPDIFRX_CHANNEL_A

SPDIFRX_CHANNEL_B

SPDIFRX Data Format

SPDIFRX_DATAFORMAT_LSB

SPDIFRX_DATAFORMAT_MSB

SPDIFRX_DATAFORMAT_32BITS

SPDIFRX Error Code

HAL_SPDIFRX_ERROR_NONE

No error

HAL_SPDIFRX_ERROR_TIMEOUT

Timeout error

HAL_SPDIFRX_ERROR_OVR

OVR error

HAL_SPDIFRX_ERROR_PE

Parity error

HAL_SPDIFRX_ERROR_DMA

DMA transfer error

HAL_SPDIFRX_ERROR_UNKNOWN

Unknown Error error

HAL_SPDIFRX_ERROR_INVALID_CALLBACK

Invalid Callback error

SPDIFRX Exported Macros

__HAL_SPDIFRX_RESET_HANDLE_STATE

Description:

- Reset SPDIFRX handle state.

Parameters:

- __HANDLE__: SPDIFRX handle.

Return value:

- None

__HAL_SPDIFRX_IDLE

Description:

- Disable the specified SPDIFRX peripheral (IDLE State).

Parameters:

- __HANDLE__: specifies the SPDIFRX Handle.

Return value:

- None

__HAL_SPDIFRX_SYNC

Description:

- Enable the specified SPDIFRX peripheral (SYNC State).

Parameters:

- __HANDLE__: specifies the SPDIFRX Handle.

Return value:

- None

__HAL_SPDIFRX_RCV

Description:

- Enable the specified SPDIFRX peripheral (RCV State).

Parameters:

- __HANDLE__: specifies the SPDIFRX Handle.

Return value:

- None

__HAL_SPDIFRX_ENABLE_IT

Description:

- Enable or disable the specified SPDIFRX interrupts.

Parameters:

- __HANDLE__: specifies the SPDIFRX Handle.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - SPDIFRX_IT_RXNE
  - SPDIFRX_IT_CSRNE
  - SPDIFRX_IT_PERRIE
  - SPDIFRX_IT_OVRIE
  - SPDIFRX_IT_SBLKIE
  - SPDIFRX_IT_SYNCDIE
  - SPDIFRX_IT_IFEIE

Return value:

- None

__HAL_SPDIFRX_DISABLE_IT

__HAL_SPDIFRX_GET_IT_SOURCE

Description:

- Checks if the specified SPDIFRX interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: specifies the SPDIFRX Handle.
- __INTERRUPT__: specifies the SPDIFRX interrupt source to check. This parameter can be one of the following values:
  - SPDIFRX_IT_RXNE
  - SPDIFRX_IT_CSRNE
  - SPDIFRX_IT_PERRIE
  - SPDIFRX_IT_OVRIE
  - SPDIFRX_IT_SBLKIE
  - SPDIFRX_IT_SYNCDIE
  - SPDIFRX_IT_IFEIE

Return value:

- The: new state of __IT__ (TRUE or FALSE).

__HAL_SPDIFRX_GET_FLAG

Description:

- Checks whether the specified SPDIFRX flag is set or not.

Parameters:

- __HANDLE__: specifies the SPDIFRX Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SPDIFRX_FLAG_RXNE
  - SPDIFRX_FLAG_CSRNE
  - SPDIFRX_FLAG_PERR
  - SPDIFRX_FLAG_OVR
  - SPDIFRX_FLAG_SBD
  - SPDIFRX_FLAG_SYNCD
  - SPDIFRX_FLAG_FERR
  - SPDIFRX_FLAG_SERR
  - SPDIFRX_FLAG_TERR

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_SPDIFRX_CLEAR_IT

Description:

- Clears the specified SPDIFRX SR flag, in setting the proper IFCR register bit.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt This parameter can be one of the following values:
  - SPDIFRX_FLAG_PERR
  - SPDIFRX_FLAG_OVR
  - SPDIFRX_SR_SBD
  - SPDIFRX_SR_SYNCD

Return value:

- None

SPDIFRX Flags Definition

SPDIFRX_FLAG_RXNE

SPDIFRX_FLAG_CSRNE

SPDIFRX_FLAG_PERR

SPDIFRX_FLAG_OVR

SPDIFRX_FLAG_SBD

SPDIFRX_FLAG_SYNCD

SPDIFRX_FLAG_FERR

SPDIFRX_FLAG_SERR

SPDIFRX_FLAG_TERR

SPDIFRX Input Selection

SPDIFRX_INPUT_IN0

SPDIFRX_INPUT_IN1

SPDIFRX_INPUT_IN2

SPDIFRX_INPUT_IN3

SPDIFRX Interrupts Definition

SPDIFRX_IT_RXNE

SPDIFRX_IT_CSRNE

SPDIFRX_IT_PERRIE

SPDIFRX_IT_OVRIE

SPDIFRX_IT_SBLKIE

SPDIFRX_IT_SYNCDIE

SPDIFRX_IT_IFEIE

SPDIFRX Maximum Retries

SPDIFRX_MAXRETRIES_NONE

SPDIFRX_MAXRETRIES_3

SPDIFRX_MAXRETRIES_15

SPDIFRX_MAXRETRIES_63

SPDIFRX Parity Error Mask

SPDIFRX_PARITYERRORMASK_OFF

SPDIFRX_PARITYERRORMASK_ON

SPDIFRX Preamble Type Mask

SPDIFRX_PREAMBLETYPEMASK_OFF

SPDIFRX_PREAMBLETYPEMASK_ON

SPDIFRX State

SPDIFRX_STATE_IDLE

SPDIFRX_STATE_SYNC

SPDIFRX_STATE_RCV

SPDIFRX Stereo Mode

SPDIFRX_STEREOMODE_DISABLE

SPDIFRX_STEREOMODE_ENABLE

SPDIFRX Validity Mask

SPDIFRX_VALIDITYMASK_OFF

SPDIFRX_VALIDITYMASK_ON

SPDIFRX Wait For Activity

SPDIFRX_WAITFORACTIVITY_OFF

SPDIFRX_WAITFORACTIVITY_ON

84 HAL SPI Generic Driver

84.1 SPI Firmware driver registers structures

84.1.1 SPI_InitTypeDef

SPI_InitTypeDef is defined in the stm32h7xx_hal_spi.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t Direction***
- ***uint32_t DataSize***
- ***uint32_t CLKPolarity***
- ***uint32_t CLKPhase***
- ***uint32_t NSS***
- ***uint32_t BaudRatePrescaler***
- ***uint32_t FirstBit***
- ***uint32_t TIMode***
- ***uint32_t CRCCalculation***
- ***uint32_t CRCPolynomial***
- ***uint32_t CRCLength***
- ***uint32_t NSSPMode***
- ***uint32_t NSSPolarity***
- ***uint32_t FifoThreshold***
- ***uint32_t TxCRCInitializationPattern***
- ***uint32_t RxCRCInitializationPattern***
- ***uint32_t MasterSSIdleness***
- ***uint32_t MasterInterDataIdleness***
- ***uint32_t MasterReceiverAutoSusp***
- ***uint32_t MasterKeepIOState***
- ***uint32_t IOSwap***

Field Documentation

- ***uint32_t SPI_InitTypeDef::Mode***
  Specifies the SPI operating mode. This parameter can be a value of ***SPI_Mode***
- ***uint32_t SPI_InitTypeDef::Direction***
  Specifies the SPI bidirectional mode state. This parameter can be a value of ***SPI_Direction***
- ***uint32_t SPI_InitTypeDef::DataSize***
  Specifies the SPI data size. This parameter can be a value of ***SPI_Data_Size***
- ***uint32_t SPI_InitTypeDef::CLKPolarity***
  Specifies the serial clock steady state. This parameter can be a value of ***SPI_Clock_Polarity***
- ***uint32_t SPI_InitTypeDef::CLKPhase***
  Specifies the clock active edge for the bit capture. This parameter can be a value of ***SPI_Clock_Phase***
- ***uint32_t SPI_InitTypeDef::NSS***
  Specifies whether the NSS signal is managed by hardware (NSS pin) or by software using the SSI bit. This parameter can be a value of ***SPI_Slave_Select_Management***

- ***uint32_t SPI_InitTypeDef::BaudRatePrescaler***

  Specifies the Baud Rate prescaler value which will be used to configure the transmit and receive SCK clock. This parameter can be a value of ***SPI_BaudRate_Prescaler***

  Note:

  – The communication clock is derived from the master clock. The slave clock does not need to be set.
- ***uint32_t SPI_InitTypeDef::FirstBit***

  Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of ***SPI_MSB_LSB_Transmission***
- ***uint32_t SPI_InitTypeDef::TIMode***

  Specifies if the TI mode is enabled or not. This parameter can be a value of ***SPI_TI_Mode***
- ***uint32_t SPI_InitTypeDef::CRCCalculation***

  Specifies if the CRC calculation is enabled or not. This parameter can be a value of ***SPI_CRC_Calculation***
- ***uint32_t SPI_InitTypeDef::CRCPolynomial***

  Specifies the polynomial used for the CRC calculation. This parameter must be an odd number between Min_Data = 0 and Max_Data = 65535
- ***uint32_t SPI_InitTypeDef::CRCLength***

  Specifies the CRC Length used for the CRC calculation. This parameter can be a value of ***SPI_CRC_length***
- ***uint32_t SPI_InitTypeDef::NSSPMode***

  Specifies whether the NSSP signal is enabled or not . This parameter can be a value of ***SPI_NSSP_Mode*** This mode is activated by the SSOM bit in the SPIx_CR2 register and it takes effect only if the SPI interface is configured as Motorola SPI master (FRF=0).
- ***uint32_t SPI_InitTypeDef::NSSPolarity***

  Specifies which level of SS input/output external signal (present on SS pin) is considered as active one. This parameter can be a value of ***SPI_NSS_Polarity***
- ***uint32_t SPI_InitTypeDef::FifoThreshold***

  Specifies the FIFO threshold level. This parameter can be a value of ***SPI_Fifo_Threshold***
- ***uint32_t SPI_InitTypeDef::TxCRCInitializationPattern***

  Specifies the transmitter CRC initialization Pattern used for the CRC calculation. This parameter can be a value of ***SPI_CRC_Calculation_Initialization_Pattern***
- ***uint32_t SPI_InitTypeDef::RxCRCInitializationPattern***

  Specifies the receiver CRC initialization Pattern used for the CRC calculation. This parameter can be a value of ***SPI_CRC_Calculation_Initialization_Pattern***
- ***uint32_t SPI_InitTypeDef::MasterSSIdleness***

  Specifies an extra delay, expressed in number of SPI clock cycle periods, inserted additionally between active edge of SS and first data transaction start in master mode. This parameter can be a value of ***SPI_Master_SS_Idleness***
- ***uint32_t SPI_InitTypeDef::MasterInterDataIdleness***

  Specifies minimum time delay (expressed in SPI clock cycles periods) inserted between two consecutive data frames in master mode. This parameter can be a value of ***SPI_Master_InterData_Idleness***
- ***uint32_t SPI_InitTypeDef::MasterReceiverAutoSusp***

  Control continuous SPI transfer in master receiver mode and automatic management in order to avoid overrun condition. This parameter can be a value of ***SPI_Master_RX_AutoSuspend***
- ***uint32_t SPI_InitTypeDef::MasterKeepIOState***

  Control of Alternate function GPIOs state This parameter can be a value of ***SPI_Master_Keep_IO_State***
- ***uint32_t SPI_InitTypeDef::IOSwap***

  Invert MISO/MOSI alternate functions This parameter can be a value of ***SPI_IO_Swap***

84.1.2 __SPI_HandleTypeDef

__SPI_HandleTypeDef is defined in the stm32h7xx_hal_spi.h

Data Fields

- ***SPI_TypeDef * Instance***
- ***SPI_InitTypeDef Init***

- ***const uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***uint32_t CRCSize***
- ***void(* RxISR***
- ***void(* TxISR***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SPI_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* TxCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* TxRxCpltCallback***
- ***void(* TxHalfCpltCallback***
- ***void(* RxHalfCpltCallback***
- ***void(* TxRxHalfCpltCallback***
- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* SuspendCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***SPI_TypeDef* __SPI_HandleTypeDef::Instance***
  SPI registers base address
- ***SPI_InitTypeDef __SPI_HandleTypeDef::Init***
  SPI communication parameters
- ***const uint8_t* __SPI_HandleTypeDef::pTxBuffPtr***
  Pointer to SPI Tx transfer Buffer
- ***uint16_t __SPI_HandleTypeDef::TxXferSize***
  SPI Tx Transfer size
- ***__IO uint16_t __SPI_HandleTypeDef::TxXferCount***
  SPI Tx Transfer Counter
- ***uint8_t* __SPI_HandleTypeDef::pRxBuffPtr***
  Pointer to SPI Rx transfer Buffer
- ***uint16_t __SPI_HandleTypeDef::RxXferSize***
  SPI Rx Transfer size
- ***__IO uint16_t __SPI_HandleTypeDef::RxXferCount***
  SPI Rx Transfer Counter
- ***uint32_t __SPI_HandleTypeDef::CRCSize***
  SPI CRC size used for the transfer
- ***void(* __SPI_HandleTypeDef::RxISR)(struct __SPI_HandleTypeDef *hspi)***
  function pointer on Rx ISR
- ***void(* __SPI_HandleTypeDef::TxISR)(struct __SPI_HandleTypeDef *hspi)***
  function pointer on Tx ISR

- ***DMA_HandleTypeDef* __SPI_HandleTypeDef::hdmatx***
  SPI Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __SPI_HandleTypeDef::hdmarx***
  SPI Rx DMA Handle parameters
- ***HAL_LockTypeDef __SPI_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_SPI_StateTypeDef __SPI_HandleTypeDef::State***
  SPI communication state
- ***__IO uint32_t __SPI_HandleTypeDef::ErrorCode***
  SPI Error code
- ***void(* __SPI_HandleTypeDef::TxCpltCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Tx Completed callback
- ***void(* __SPI_HandleTypeDef::RxCpltCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Rx Completed callback
- ***void(* __SPI_HandleTypeDef::TxRxCpltCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI TxRx Completed callback
- ***void(* __SPI_HandleTypeDef::TxHalfCpltCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Tx Half Completed callback
- ***void(* __SPI_HandleTypeDef::RxHalfCpltCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Rx Half Completed callback
- ***void(* __SPI_HandleTypeDef::TxRxHalfCpltCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI TxRx Half Completed callback
- ***void(* __SPI_HandleTypeDef::ErrorCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Error callback
- ***void(* __SPI_HandleTypeDef::AbortCpltCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Abort callback
- ***void(* __SPI_HandleTypeDef::SuspendCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Suspend callback
- ***void(* __SPI_HandleTypeDef::MspInitCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Msp Init callback
- ***void(* __SPI_HandleTypeDef::MspDeInitCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Msp DeInit callback

84.2 SPI Firmware driver API description

The following section lists the various functions of the SPI library.

84.2.1 How to use this driver

The SPI HAL driver can be used as follows:

1. Declare a SPI_HandleTypeDef handle structure, for example: SPI_HandleTypeDef hspi;

2. Initialize the SPI low level resources by implementing the HAL_SPI_MspInit() API:
   a. Enable the SPIx interface clock
   b. SPI pins configuration
      - Enable the clock for the SPI GPIOs
      - Configure these SPI pins as alternate function push-pull
   c. NVIC configuration if you need to use interrupt process or DMA process
      - Configure the SPIx interrupt priority
      - Enable the NVIC SPI IRQ handle
   d. DMA Configuration if you need to use DMA process
      - Declare a DMA_HandleTypeDef handle structure for the transmit or receive Stream/Channel
      - Enable the DMAx clock
      - Configure the DMA handle parameters
      - Configure the DMA Tx or Rx Stream/Channel
      - Associate the initialized hdma_tx handle to the hspi DMA Tx or Rx handle
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx Stream/Channel
3. Program the Mode, BidirectionalMode , Data size, Baudrate Prescaler, NSS management, Clock polarity and phase, FirstBit and CRC configuration in the hspi Init structure.
4. Initialize the SPI registers by calling the HAL_SPI_Init() API:
   – This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_SPI_MspInit() API.

Callback registration:

1. The compilation flag USE_HAL_SPI_REGISTER_CALLBACKS when set to 1UL allows the user to configure dynamically the driver callbacks. Use Functions HAL_SPI_RegisterCallback() to register an interrupt callback. Function HAL_SPI_RegisterCallback() allows to register following callbacks: (+) TxCpltCallback : SPI Tx Completed callback (+) RxCpltCallback : SPI Rx Completed callback (+) TxRxCpltCallback : SPI TxRx Completed callback (+) TxHalfCpltCallback : SPI Tx Half Completed callback (+) RxHalfCpltCallback : SPI Rx Half Completed callback (+) TxRxHalfCpltCallback : SPI TxRx Half Completed callback (+) ErrorCallback : SPI Error callback (+) AbortCpltCallback : SPI Abort callback (+) SuspendCallback : SPI Suspend callback (+) MspInitCallback : SPI Msp Init callback (+) MspDeInitCallback : SPI Msp DeInit callback This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.
2. Use function HAL_SPI_UnRegisterCallback to reset a callback to the default weak function. HAL_SPI_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks: (+) TxCpltCallback : SPI Tx Completed callback (+) RxCpltCallback : SPI Rx Completed callback (+) TxRxCpltCallback : SPI TxRx Completed callback (+) TxHalfCpltCallback : SPI Tx Half Completed callback (+) RxHalfCpltCallback : SPI Rx Half Completed callback (+) TxRxHalfCpltCallback : SPI TxRx Half Completed callback (+) ErrorCallback : SPI Error callback (+) AbortCpltCallback : SPI Abort callback (+) SuspendCallback : SPI Suspend callback (+) MspInitCallback : SPI Msp Init callback (+) MspDeInitCallback : SPI Msp DeInit callback By default, after the HAL_SPI_Init() and when the state is HAL_SPI_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_SPI_MasterTxCpltCallback(), HAL_SPI_MasterRxCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_SPI_Init()/ HAL_SPI_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the HAL_SPI_Init()/ HAL_SPI_DeInit() keep and use the user MspInit/ MspDeInit callbacks (registered beforehand) whatever the state. Callbacks can be registered/unregistered in HAL_SPI_STATE_READY state only. Exception done MspInit/MspDeInit functions that can be registered/ unregistered in HAL_SPI_STATE_READY or HAL_SPI_STATE_RESET state, thus registered (user) MspInit/ DeInit callbacks can be used during the Init/DeInit. Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_SPI_RegisterCallback() before calling HAL_SPI_DeInit() or HAL_SPI_Init() function. When The compilation define USE_HAL_PPP_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used. SuspendCallback restriction: SuspendCallback is called only when MasterReceiverAutoSusp is enabled and EOT interrupt is activated. SuspendCallback is used in relation with functions HAL_SPI_Transmit_IT, HAL_SPI_Receive_IT and HAL_SPI_TransmitReceive_IT.

Callback registration: (#) The compilation flag USE_HAL_SPI_REGISTER_CALLBACKS when set to 1UL allows the user to configure dynamically the driver callbacks. Use Functions HAL_SPI_RegisterCallback() to register an interrupt callback. Function HAL_SPI_RegisterCallback() allows to register following callbacks:

- TxCpltCallback : SPI Tx Completed callback
- RxCpltCallback : SPI Rx Completed callback
- TxRxCpltCallback : SPI TxRx Completed callback
- TxHalfCpltCallback : SPI Tx Half Completed callback
- RxHalfCpltCallback : SPI Rx Half Completed callback
- TxRxHalfCpltCallback : SPI TxRx Half Completed callback
- ErrorCallback : SPI Error callback
- AbortCpltCallback : SPI Abort callback
- SuspendCallback : SPI Suspend callback
- MspInitCallback : SPI Msp Init callback
- MspDeInitCallback : SPI Msp DeInit callback This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. (#) Use function HAL_SPI_UnRegisterCallback to reset a callback to the default weak function. HAL_SPI_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:
- TxCpltCallback : SPI Tx Completed callback
- RxCpltCallback : SPI Rx Completed callback
- TxRxCpltCallback : SPI TxRx Completed callback
- TxHalfCpltCallback : SPI Tx Half Completed callback
- RxHalfCpltCallback : SPI Rx Half Completed callback
- TxRxHalfCpltCallback : SPI TxRx Half Completed callback
- ErrorCallback : SPI Error callback
- AbortCpltCallback : SPI Abort callback
- SuspendCallback : SPI Suspend callback
- MspInitCallback : SPI Msp Init callback
- MspDeInitCallback : SPI Msp DeInit callback By default, after the HAL_SPI_Init() and when the state is HAL_SPI_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_SPI_MasterTxCpltCallback(), HAL_SPI_MasterRxCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_SPI_Init()/ HAL_SPI_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the HAL_SPI_Init()/ HAL_SPI_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state. Callbacks can be registered/unregistered in HAL_SPI_STATE_READY state only. Exception done MspInit/MspDeInit functions that can be registered/unregistered in HAL_SPI_STATE_READY or HAL_SPI_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_SPI_RegisterCallback() before calling HAL_SPI_DeInit() or HAL_SPI_Init() function. When The compilation define USE_HAL_PPP_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used. SuspendCallback restriction: SuspendCallback is called only when MasterReceiverAutoSusp is enabled and EOT interrupt is activated. SuspendCallback is used in relation with functions HAL_SPI_Transmit_IT, HAL_SPI_Receive_IT and HAL_SPI_TransmitReceive_IT.

Circular mode restriction:

- The DMA circular mode cannot be used when the SPI is configured in these modes:
  - Master 2Lines RxOnly
  - Master 1Line Rx
- The CRC feature is not managed when the DMA circular mode is enabled
- The functions HAL_SPI_DMAPause()/ HAL_SPI_DMAResume() are not supported. Return always HAL_ERROR with ErrorCode set to HAL_SPI_ERROR_NOT_SUPPORTED. Those functions are maintained for backward compatibility reasons.

84.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the SPIx peripheral:

- User must implement HAL_SPI_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_SPI_Init() to configure the selected device with the selected configuration:
  - Mode
  - Direction
  - Data Size
  - Clock Polarity and Phase
  - NSS Management
  - BaudRate Prescaler
  - FirstBit
  - TIMode
  - CRC Calculation
  - CRC Polynomial if CRC enabled
  - CRC Length, used only with Data8 and Data16
  - FIFO reception threshold
  - FIFO transmission threshold
- Call the function HAL_SPI_DeInit() to restore the default configuration of the selected SPIx peripheral.

This section contains the following APIs:

- ***HAL_SPI_Init()***
- ***HAL_SPI_DeInit()***
- ***HAL_SPI_MspInit()***
- ***HAL_SPI_MspDeInit()***
- ***HAL_SPI_RegisterCallback()***
- ***HAL_SPI_UnRegisterCallback()***

84.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the SPI data transfers.

The SPI supports master and slave mode :

1. There are two modes of transfer:
   a. Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
   b. No-Blocking mode: The communication is performed using Interrupts or DMA, These APIs return the HAL status. The end of the data processing will be indicated through the dedicated SPI IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_SPI_TxCpltCallback(), HAL_SPI_RxCpltCallback() and HAL_SPI_TxRxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive process The HAL_SPI_ErrorCallback()user callback will be executed when a communication error is detected
2. APIs provided for these 2 transfer modes (Blocking mode or Non blocking mode using either Interrupt or DMA) exist for 1Line (simplex) and 2Lines (full duplex) modes.

This section contains the following APIs:

- ***HAL_SPI_Transmit()***
- ***HAL_SPI_Receive()***
- ***HAL_SPI_TransmitReceive()***
- ***HAL_SPI_Transmit_IT()***
- ***HAL_SPI_Receive_IT()***
- ***HAL_SPI_TransmitReceive_IT()***
- ***HAL_SPI_Transmit_DMA()***
- ***HAL_SPI_Receive_DMA()***
- ***HAL_SPI_TransmitReceive_DMA()***
- ***HAL_SPI_Abort()***

- *HAL_SPI_Abort_IT()*
- *HAL_SPI_DMAPause()*
- *HAL_SPI_DMAResume()*
- *HAL_SPI_DMAStop()*
- *HAL_SPI_IRQHandler()*
- *HAL_SPI_TxCpltCallback()*
- *HAL_SPI_RxCpltCallback()*
- *HAL_SPI_TxRxCpltCallback()*
- *HAL_SPI_TxHalfCpltCallback()*
- *HAL_SPI_RxHalfCpltCallback()*
- *HAL_SPI_TxRxHalfCpltCallback()*
- *HAL_SPI_ErrorCallback()*
- *HAL_SPI_AbortCpltCallback()*
- *HAL_SPI_SuspendCallback()*

84.2.4 Peripheral State and Errors functions

This subsection provides a set of functions allowing to control the SPI.

- HAL_SPI_GetState() API can be helpful to check in run-time the state of the SPI peripheral
- HAL_SPI_GetError() check in run-time Errors occurring during communication

This section contains the following APIs:

- *HAL_SPI_GetState()*
- *HAL_SPI_GetError()*

84.2.5 Detailed description of functions

HAL_SPI_Init

Function name

HAL_StatusTypeDef HAL_SPI_Init (SPI_HandleTypeDef * hspi)

Function description

Initialize the SPI according to the specified parameters in the SPI_InitTypeDef and initialize the associated handle.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **HAL:** status

HAL_SPI_DeInit

Function name

HAL_StatusTypeDef HAL_SPI_DeInit (SPI_HandleTypeDef * hspi)

Function description

De-Initialize the SPI peripheral.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **HAL:** status

HAL_SPI_MspInit

Function name

void HAL_SPI_MspInit (SPI_HandleTypeDef * hspi)

Function description

Initialize the SPI MSP.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_MspDeInit

Function name

void HAL_SPI_MspDeInit (SPI_HandleTypeDef * hspi)

Function description

De-Initialize the SPI MSP.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SPI_RegisterCallback (SPI_HandleTypeDef * hspi, HAL_SPI_CallbackIDTypeDef CallbackID, pSPI_CallbackTypeDef pCallback)

Function description

Register a User SPI Callback To be used instead of the weak predefined callback.

Parameters

- **hspi:** Pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI.
- **CallbackID:** ID of the callback to be registered
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_SPI_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SPI_UnRegisterCallback (SPI_HandleTypeDef * hspi, HAL_SPI_CallbackIDTypeDef CallbackID)

Function description

Unregister an SPI Callback SPI callback is redirected to the weak predefined callback.

Parameters

- **hspi:** Pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI.
- **CallbackID:** ID of the callback to be unregistered

Return values

- **HAL:** status

HAL_SPI_Transmit

Function name

HAL_StatusTypeDef HAL_SPI_Transmit (SPI_HandleTypeDef * hspi, const uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Transmit an amount of data in blocking mode.

Parameters

- **hspi:** : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** : pointer to data buffer
- **Size:** : amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SPI_Receive

Function name

HAL_StatusTypeDef HAL_SPI_Receive (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **hspi:** : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** : pointer to data buffer
- **Size:** : amount of data to be received
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SPI_TransmitReceive

Function name

HAL_StatusTypeDef HAL_SPI_TransmitReceive (SPI_HandleTypeDef * hspi, const uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)

Function description

Transmit and Receive an amount of data in blocking mode.

Parameters

- **hspi:** : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pTxData:** pointer to transmission data buffer
- **pRxData:** pointer to reception data buffer
- **Size:** : amount of data to be sent and received
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SPI_Transmit_IT

Function name

HAL_StatusTypeDef HAL_SPI_Transmit_IT (SPI_HandleTypeDef * hspi, const uint8_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with Interrupt.

Parameters

- **hspi:** : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** : amount of data to be sent

Return values

- **HAL:** status

HAL_SPI_Receive_IT

Function name

HAL_StatusTypeDef HAL_SPI_Receive_IT (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in non-blocking mode with Interrupt.

Parameters

- **hspi:** : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** : amount of data to be sent

Return values

- **HAL:** status

HAL_SPI_TransmitReceive_IT

Function name

HAL_StatusTypeDef HAL_SPI_TransmitReceive_IT (SPI_HandleTypeDef * hspi, const uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)

Function description

Transmit and Receive an amount of data in non-blocking mode with Interrupt.

Parameters

- **hspi:** : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pTxData:** pointer to transmission data buffer
- **pRxData:** pointer to reception data buffer
- **Size:** : amount of data to be sent and received

Return values

- **HAL:** status

HAL_SPI_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_SPI_Transmit_DMA (SPI_HandleTypeDef * hspi, const uint8_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with DMA.

Parameters

- **hspi:** : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** : amount of data to be sent

Return values

- **HAL:** status

HAL_SPI_Receive_DMA

Function name

HAL_StatusTypeDef HAL_SPI_Receive_DMA (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hspi:** : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** : amount of data to be sent

Return values

- **HAL:** status

Notes

- When the CRC feature is enabled the pData Length must be Size + 1.

HAL_SPI_TransmitReceive_DMA

Function name

HAL_StatusTypeDef HAL_SPI_TransmitReceive_DMA (SPI_HandleTypeDef * hspi, const uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)

Function description

Transmit and Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hspi:** : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pTxData:** pointer to transmission data buffer
- **pRxData:** pointer to reception data buffer
- **Size:** : amount of data to be sent

Return values

- **HAL:** status

Notes

- When the CRC feature is enabled the pRxData Length must be Size + 1

HAL_SPI_DMAPause

Function name

HAL_StatusTypeDef HAL_SPI_DMAPause (SPI_HandleTypeDef * hspi)

Function description

Pause the DMA Transfer.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values

- **HAL_ERROR:**

HAL_SPI_DMAResume

Function name

HAL_StatusTypeDef HAL_SPI_DMAResume (SPI_HandleTypeDef * hspi)

Function description

Resume the DMA Transfer.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values

- **HAL_ERROR:**

HAL_SPI_DMAStop

Function name

HAL_StatusTypeDef HAL_SPI_DMAStop (SPI_HandleTypeDef * hspi)

Function description

Stop the DMA Transfer.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values

- **HAL_ERROR:**

HAL_SPI_Abort

Function name

HAL_StatusTypeDef HAL_SPI_Abort (SPI_HandleTypeDef * hspi)

Function description

Abort ongoing transfer (blocking mode).

Parameters

- **hspi:** SPI handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer (Tx and Rx), started in Interrupt or DMA mode.
- This procedure performs following operations : + Disable SPI Interrupts (depending of transfer direction) + Disable the DMA transfer in the peripheral register (if enabled) + Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode) + Set handle State to READY.
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_SPI_Abort_IT

Function name

HAL_StatusTypeDef HAL_SPI_Abort_IT (SPI_HandleTypeDef * hspi)

Function description

Abort ongoing transfer (Interrupt mode).

Parameters

- **hspi:** SPI handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer (Tx and Rx), started in Interrupt or DMA mode.
- This procedure performs following operations : + Disable SPI Interrupts (depending of transfer direction) + Disable the DMA transfer in the peripheral register (if enabled) + Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode) + Set handle State to READY + At abort completion, call user abort complete callback.
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_SPI_IRQHandler

Function name

void HAL_SPI_IRQHandler (SPI_HandleTypeDef * hspi)

Function description

Handle SPI interrupt request.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values

- **None:**

HAL_SPI_TxCpltCallback

Function name

void HAL_SPI_TxCpltCallback (SPI_HandleTypeDef * hspi)

Function description

Tx Transfer completed callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_RxCpltCallback

Function name

void HAL_SPI_RxCpltCallback (SPI_HandleTypeDef * hspi)

Function description

Rx Transfer completed callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_TxRxCpltCallback

Function name

void HAL_SPI_TxRxCpltCallback (SPI_HandleTypeDef * hspi)

Function description

Tx and Rx Transfer completed callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_TxHalfCpltCallback

Function name

void HAL_SPI_TxHalfCpltCallback (SPI_HandleTypeDef * hspi)

Function description

Tx Half Transfer completed callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_RxHalfCpltCallback

Function name

void HAL_SPI_RxHalfCpltCallback (SPI_HandleTypeDef * hspi)

Function description

Rx Half Transfer completed callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_TxRxHalfCpltCallback

Function name

void HAL_SPI_TxRxHalfCpltCallback (SPI_HandleTypeDef * hspi)

Function description

Tx and Rx Half Transfer callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_ErrorCallback

Function name

void HAL_SPI_ErrorCallback (SPI_HandleTypeDef * hspi)

Function description

SPI error callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_AbortCpltCallback

Function name

void HAL_SPI_AbortCpltCallback (SPI_HandleTypeDef * hspi)

Function description

SPI Abort Complete callback.

Parameters

- **hspi:** SPI handle.

Return values

- **None:**

HAL_SPI_SuspendCallback

Function name

void HAL_SPI_SuspendCallback (SPI_HandleTypeDef * hspi)

Function description

SPI Suspend callback.

Parameters

- **hspi:** SPI handle.

Return values

- **None:**

HAL_SPI_GetState

Function name

HAL_SPI_StateTypeDef HAL_SPI_GetState (const SPI_HandleTypeDef * hspi)

Function description

Return the SPI handle state.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **SPI:** state

HAL_SPI_GetError

Function name

uint32_t HAL_SPI_GetError (const SPI_HandleTypeDef * hspi)

Function description

Return the SPI error code.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **SPI:** error code in bitmap format

84.3 SPI Firmware driver defines

The following section lists the various define and macros of the module.

84.3.1 SPI

SPI

SPI BaudRate Prescaler

SPI_BAUDRATEPRESCALER_2

SPI_BAUDRATEPRESCALER_4

SPI_BAUDRATEPRESCALER_8

SPI_BAUDRATEPRESCALER_16

SPI_BAUDRATEPRESCALER_32

SPI_BAUDRATEPRESCALER_64

SPI_BAUDRATEPRESCALER_128

SPI_BAUDRATEPRESCALER_256

SPI Clock Phase

SPI_PHASE_1EDGE

SPI_PHASE_2EDGE

SPI Clock Polarity

SPI_POLARITY_LOW

SPI_POLARITY_HIGH

SPI CRC Calculation

SPI_CRCCALCULATION_DISABLE

SPI_CRCCALCULATION_ENABLE

SPI CRC Calculation Initialization Pattern

SPI_CRC_INITIALIZATION_ALL_ZERO_PATTERN

SPI_CRC_INITIALIZATION_ALL_ONE_PATTERN

SPI CRC Length

SPI_CRC_LENGTH_DATASIZE

SPI_CRC_LENGTH_4BIT

SPI_CRC_LENGTH_5BIT

SPI_CRC_LENGTH_6BIT

SPI_CRC_LENGTH_7BIT

SPI_CRC_LENGTH_8BIT

SPI_CRC_LENGTH_9BIT

SPI_CRC_LENGTH_10BIT

SPI_CRC_LENGTH_11BIT

SPI_CRC_LENGTH_12BIT

SPI_CRC_LENGTH_13BIT

SPI_CRC_LENGTH_14BIT

SPI_CRC_LENGTH_15BIT

SPI_CRC_LENGTH_16BIT

SPI_CRC_LENGTH_17BIT

SPI_CRC_LENGTH_18BIT

SPI_CRC_LENGTH_19BIT

SPI_CRC_LENGTH_20BIT

SPI_CRC_LENGTH_21BIT

SPI_CRC_LENGTH_22BIT

SPI_CRC_LENGTH_23BIT

SPI_CRC_LENGTH_24BIT

SPI_CRC_LENGTH_25BIT

SPI_CRC_LENGTH_26BIT

SPI_CRC_LENGTH_27BIT

SPI_CRC_LENGTH_28BIT

SPI_CRC_LENGTH_29BIT

SPI_CRC_LENGTH_30BIT

SPI_CRC_LENGTH_31BIT

SPI_CRC_LENGTH_32BIT

SPI Data Size

SPI_DATASIZE_4BIT

SPI_DATASIZE_5BIT

SPI_DATASIZE_6BIT

SPI_DATASIZE_7BIT

SPI_DATASIZE_8BIT

SPI_DATASIZE_9BIT

SPI_DATASIZE_10BIT

SPI_DATASIZE_11BIT

SPI_DATASIZE_12BIT

SPI_DATASIZE_13BIT

SPI_DATASIZE_14BIT

SPI_DATASIZE_15BIT

SPI_DATASIZE_16BIT

SPI_DATASIZE_17BIT

SPI_DATASIZE_18BIT

SPI_DATASIZE_19BIT

SPI_DATASIZE_20BIT

SPI_DATASIZE_21BIT

SPI_DATASIZE_22BIT

SPI_DATASIZE_23BIT

SPI_DATASIZE_24BIT

SPI_DATASIZE_25BIT

SPI_DATASIZE_26BIT

SPI_DATASIZE_27BIT

SPI_DATASIZE_28BIT

SPI_DATASIZE_29BIT

SPI_DATASIZE_30BIT

SPI_DATASIZE_31BIT

SPI_DATASIZE_32BIT

SPI Direction Mode

SPI_DIRECTION_2LINES

SPI_DIRECTION_2LINES_TXONLY

SPI_DIRECTION_2LINES_RXONLY

SPI_DIRECTION_1LINE

SPI Error Codes

HAL_SPI_ERROR_NONE

No error

HAL_SPI_ERROR_MODF

MODF error

HAL_SPI_ERROR_CRC

CRC error

HAL_SPI_ERROR_OVR

OVR error

HAL_SPI_ERROR_FRE

FRE error

HAL_SPI_ERROR_DMA

DMA transfer error

HAL_SPI_ERROR_FLAG

Error on RXP/TXP/DXP/FTLVL/FRLVL Flag

HAL_SPI_ERROR_ABORT

Error during SPI Abort procedure

HAL_SPI_ERROR_UDR

Underrun error

HAL_SPI_ERROR_TIMEOUT

Timeout error

HAL_SPI_ERROR_UNKNOW

Unknown error

HAL_SPI_ERROR_NOT_SUPPORTED

Requested operation not supported

HAL_SPI_ERROR_RELOAD

Reload error

HAL_SPI_ERROR_INVALID_CALLBACK

Invalid Callback error

SPI Exported Macros

__HAL_SPI_RESET_HANDLE_STATE

Description:

- Reset SPI handle state.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, 3, 4, 5 or 6 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_ENABLE_IT

Description:

- Enable the specified SPI interrupts.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, 3, 4, 5 or 6 to select the SPI peripheral.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - SPI_IT_RXP : Rx-Packet available interrupt
  - SPI_IT_TXP : Tx-Packet space available interrupt
  - SPI_IT_DXP : Duplex Packet interrupt
  - SPI_IT_EOT : End of transfer interrupt
  - SPI_IT_TXTF : Transmission Transfer Filled interrupt
  - SPI_IT_UDR : Underrun interrupt
  - SPI_IT_OVR : Overrun interrupt
  - SPI_IT_CRCERR : CRC error interrupt
  - SPI_IT_FRE : TI mode frame format error interrupt
  - SPI_IT_MODF : Mode fault interrupt
  - SPI_IT_TSERF : Additional number of data reloaded interrupt
  - SPI_IT_ERR : Error interrupt

Return value:

- None

__HAL_SPI_DISABLE_IT

Description:

- Disable the specified SPI interrupts.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, 3, 4, 5 or 6 to select the SPI peripheral.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - SPI_IT_RXP : Rx-Packet available interrupt
  - SPI_IT_TXP : Tx-Packet space available interrupt
  - SPI_IT_DXP : Duplex Packet interrupt
  - SPI_IT_EOT : End of transfer interrupt
  - SPI_IT_TXTF : Transmission Transfer Filled interrupt
  - SPI_IT_UDR : Underrun interrupt
  - SPI_IT_OVR : Overrun interrupt
  - SPI_IT_CRCERR : CRC error interrupt
  - SPI_IT_FRE : TI mode frame format error interrupt
  - SPI_IT_MODF : Mode fault interrupt
  - SPI_IT_TSERF : Additional number of data reloaded interrupt
  - SPI_IT_ERR : Error interrupt

Return value:

- None

__HAL_SPI_GET_IT_SOURCE

Description:

- Check whether the specified SPI interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, 3, 4, 5 or 6 to select the SPI peripheral.
- __INTERRUPT__: specifies the SPI interrupt source to check. This parameter can be one of the following values:
  - SPI_IT_RXP : Rx-Packet available interrupt
  - SPI_IT_TXP : Tx-Packet space available interrupt
  - SPI_IT_DXP : Duplex Packet interrupt
  - SPI_IT_EOT : End of transfer interrupt
  - SPI_IT_TXTF : Transmission Transfer Filled interrupt
  - SPI_IT_UDR : Underrun interrupt
  - SPI_IT_OVR : Overrun interrupt
  - SPI_IT_CRCERR : CRC error interrupt
  - SPI_IT_FRE : TI mode frame format error interrupt
  - SPI_IT_MODF : Mode fault interrupt
  - SPI_IT_TSERF : Additional number of data reloaded interrupt
  - SPI_IT_ERR : Error interrupt

Return value:

- The: new state of __IT__ (TRUE or FALSE).

__HAL_SPI_GET_FLAG

Description:

- Check whether the specified SPI flag is set or not.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, 3, 4, 5 or 6 to select the SPI peripheral.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SPI_FLAG_RXP : Rx-Packet available flag
  - SPI_FLAG_TXP : Tx-Packet space available flag
  - SPI_FLAG_DXP : Duplex Packet flag
  - SPI_FLAG_EOT : End of transfer flag
  - SPI_FLAG_TXTF : Transmission Transfer Filled flag
  - SPI_FLAG_UDR : Underrun flag
  - SPI_FLAG_OVR : Overrun flag
  - SPI_FLAG_CRCERR : CRC error flag
  - SPI_FLAG_FRE : TI mode frame format error flag
  - SPI_FLAG_MODF : Mode fault flag
  - SPI_FLAG_TSERF : Additional number of data reloaded flag
  - SPI_FLAG_SUSP : Transfer suspend complete flag
  - SPI_FLAG_TXC : TxFIFO transmission complete flag
  - SPI_FLAG_FRLVL : Fifo reception level flag
  - SPI_FLAG_RXWNE : RxFIFO word not empty flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_SPI_CLEAR_CRCERRFLAG

Description:

- Clear the SPI CRCERR pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_CLEAR_MODFFLAG

Description:

- Clear the SPI MODF pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_CLEAR_OVRFLAG

Description:

- Clear the SPI OVR pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_CLEAR_FREFLAG

Description:

- Clear the SPI FRE pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_CLEAR_UDRFLAG

Description:

- Clear the SPI UDR pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_CLEAR_EOTFLAG

Description:

- Clear the SPI EOT pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_CLEAR_TXTFFLAG

Description:

- Clear the SPI UDR pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_CLEAR_SUSPFLAG

Description:

- Clear the SPI SUSP pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_CLEAR_TSERFFLAG

Description:

- Clear the SPI TSERF pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_ENABLE

Description:

- Enable the SPI peripheral.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_DISABLE

Description:

- Disable the SPI peripheral.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

SPI Fifo Threshold

SPI_FIFO_THRESHOLD_01DATA

SPI_FIFO_THRESHOLD_02DATA

SPI_FIFO_THRESHOLD_03DATA

SPI_FIFO_THRESHOLD_04DATA

SPI_FIFO_THRESHOLD_05DATA

SPI_FIFO_THRESHOLD_06DATA

SPI_FIFO_THRESHOLD_07DATA

SPI_FIFO_THRESHOLD_08DATA

SPI_FIFO_THRESHOLD_09DATA

SPI_FIFO_THRESHOLD_10DATA

SPI_FIFO_THRESHOLD_11DATA

SPI_FIFO_THRESHOLD_12DATA

SPI_FIFO_THRESHOLD_13DATA

SPI_FIFO_THRESHOLD_14DATA

SPI_FIFO_THRESHOLD_15DATA

SPI_FIFO_THRESHOLD_16DATA

SPI FIFO Type

SPI_LOWEND_FIFO_SIZE

SPI_HIGHEND_FIFO_SIZE

SPI Flags Definition

SPI_FLAG_RXP

SPI_FLAG_TXP

SPI_FLAG_DXP

SPI_FLAG_EOT

SPI_FLAG_TXTF

SPI_FLAG_UDR

SPI_FLAG_OVR

SPI_FLAG_CRCERR

SPI_FLAG_FRE

SPI_FLAG_MODF

SPI_FLAG_TSERF

SPI_FLAG_SUSP

SPI_FLAG_TXC

SPI_FLAG_FRLVL

SPI_FLAG_RXWNE

SPI Interrupt Definition

SPI_IT_RXP

SPI_IT_TXP

SPI_IT_DXP

SPI_IT_EOT

SPI_IT_TXTF

SPI_IT_UDR

SPI_IT_OVR

SPI_IT_CRCERR

SPI_IT_FRE

SPI_IT_MODF

SPI_IT_TSERF

SPI_IT_ERR

Control SPI IO Swap

SPI_IO_SWAP_DISABLE

SPI_IO_SWAP_ENABLE

SPI Master Inter-Data Idleness

SPI_MASTER_INTERDATA_IDLENESS_00CYCLE

SPI_MASTER_INTERDATA_IDLENESS_01CYCLE

SPI_MASTER_INTERDATA_IDLENESS_02CYCLE

SPI_MASTER_INTERDATA_IDLENESS_03CYCLE

SPI_MASTER_INTERDATA_IDLENESS_04CYCLE

SPI_MASTER_INTERDATA_IDLENESS_05CYCLE

SPI_MASTER_INTERDATA_IDLENESS_06CYCLE

SPI_MASTER_INTERDATA_IDLENESS_07CYCLE

SPI_MASTER_INTERDATA_IDLENESS_08CYCLE

SPI_MASTER_INTERDATA_IDLENESS_09CYCLE

SPI_MASTER_INTERDATA_IDLENESS_10CYCLE

SPI_MASTER_INTERDATA_IDLENESS_11CYCLE

SPI_MASTER_INTERDATA_IDLENESS_12CYCLE

SPI_MASTER_INTERDATA_IDLENESS_13CYCLE

SPI_MASTER_INTERDATA_IDLENESS_14CYCLE

SPI_MASTER_INTERDATA_IDLENESS_15CYCLE

Keep IO State

SPI_MASTER_KEEP_IO_STATE_DISABLE

SPI_MASTER_KEEP_IO_STATE_ENABLE

SPI Master Receiver AutoSuspend

SPI_MASTER_RX_AUTOSUSP_DISABLE

SPI_MASTER_RX_AUTOSUSP_ENABLE

SPI Master SS Idleness

SPI_MASTER_SS_IDLENESS_00CYCLE

SPI_MASTER_SS_IDLENESS_01CYCLE

SPI_MASTER_SS_IDLENESS_02CYCLE

SPI_MASTER_SS_IDLENESS_03CYCLE

SPI_MASTER_SS_IDLENESS_04CYCLE

SPI_MASTER_SS_IDLENESS_05CYCLE

SPI_MASTER_SS_IDLENESS_06CYCLE

SPI_MASTER_SS_IDLENESS_07CYCLE

SPI_MASTER_SS_IDLENESS_08CYCLE

SPI_MASTER_SS_IDLENESS_09CYCLE

SPI_MASTER_SS_IDLENESS_10CYCLE

SPI_MASTER_SS_IDLENESS_11CYCLE

SPI_MASTER_SS_IDLENESS_12CYCLE

SPI_MASTER_SS_IDLENESS_13CYCLE

SPI_MASTER_SS_IDLENESS_14CYCLE

SPI_MASTER_SS_IDLENESS_15CYCLE

SPI Mode

SPI_MODE_SLAVE

SPI_MODE_MASTER

SPI MSB LSB Transmission

SPI_FIRSTBIT_MSB

SPI_FIRSTBIT_LSB

SPI NSS Pulse Mode

SPI_NSS_PULSE_DISABLE

SPI_NSS_PULSE_ENABLE

SPI NSS Polarity

SPI_NSS_POLARITY_LOW

SPI_NSS_POLARITY_HIGH

SPI Reception FIFO Status Level

SPI_RX_FIFO_0PACKET

SPI_RX_FIFO_1PACKET

SPI_RX_FIFO_2PACKET

SPI_RX_FIFO_3PACKET

SPI Slave Select Management

SPI_NSS_SOFT

SPI_NSS_HARD_INPUT

SPI_NSS_HARD_OUTPUT

SPI TI Mode

SPI_TIMODE_DISABLE

SPI_TIMODE_ENABLE

SPI Underrun Behavior

SPI_UNDERRUN_BEHAV_REGISTER_PATTERN

SPI_UNDERRUN_BEHAV_LAST_RECEIVED

SPI_UNDERRUN_BEHAV_LAST_TRANSMITTED

SPI Underrun Detection

SPI_UNDERRUN_DETECT_BEGIN_DATA_FRAME

SPI_UNDERRUN_DETECT_END_DATA_FRAME

SPI_UNDERRUN_DETECT_BEGIN_ACTIVE_NSS

85 HAL SPI Extension Driver

85.1 SPIEx Firmware driver API description

The following section lists the various functions of the SPIEx library.

85.1.1 IO operation functions

This subsection provides a set of extended functions to manage the SPI data transfers.

1. SPIEx function:
   - HAL_SPIEx_FlushRxFifo()
   - HAL_SPIEx_FlushRxFifo()
   - HAL_SPIEx_EnableLockConfiguration()
   - HAL_SPIEx_ConfigureUnderrun()

This section contains the following APIs:

- ***HAL_SPIEx_FlushRxFifo()***
- ***HAL_SPIEx_EnableLockConfiguration()***
- ***HAL_SPIEx_ConfigureUnderrun()***

85.1.2 Detailed description of functions

HAL_SPIEx_FlushRxFifo

Function name

HAL_StatusTypeDef HAL_SPIEx_FlushRxFifo (const SPI_HandleTypeDef * hspi)

Function description

Flush the RX fifo.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values

- **HAL:** status

HAL_SPIEx_EnableLockConfiguration

Function name

HAL_StatusTypeDef HAL_SPIEx_EnableLockConfiguration (SPI_HandleTypeDef * hspi)

Function description

Enable the Lock for the AF configuration of associated IOs and write protect the Content of Configuration register 2 when SPI is enabled.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPIEx_ConfigureUnderrun

Function name

HAL_StatusTypeDef HAL_SPIEx_ConfigureUnderrun (SPI_HandleTypeDef * hspi, uint32_t UnderrunDetection, uint32_t UnderrunBehaviour)

Function description

Configure the UNDERRUN condition and behavior of slave transmitter.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **UnderrunDetection:** : Detection of underrun condition at slave transmitter This parameter can be a value of SPI Underrun Detection.
- **UnderrunBehaviour:** : Behavior of slave transmitter at underrun condition This parameter can be a value of SPI Underrun Behavior.

Return values

- **None:**

86 HAL SRAM Generic Driver

86.1 SRAM Firmware driver registers structures

86.1.1 __SRAM_HandleTypeDef

__SRAM_HandleTypeDef is defined in the stm32h7xx_hal_sram.h

Data Fields

- ***FMC_NORSRAM_TypeDef * Instance***
- ***FMC_NORSRAM_EXTENDED_TypeDef * Extended***
- ***FMC_NORSRAM_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SRAM_StateTypeDef State***
- ***MDMA_HandleTypeDef * hmdma***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***void(* DmaXferCpltCallback***
- ***void(* DmaXferErrorCallback***

Field Documentation

- ***FMC_NORSRAM_TypeDef* __SRAM_HandleTypeDef::Instance***
  Register base address
- ***FMC_NORSRAM_EXTENDED_TypeDef* __SRAM_HandleTypeDef::Extended***
  Extended mode register base address
- ***FMC_NORSRAM_InitTypeDef __SRAM_HandleTypeDef::Init***
  SRAM device control configuration parameters
- ***HAL_LockTypeDef __SRAM_HandleTypeDef::Lock***
  SRAM locking object
- ***__IO HAL_SRAM_StateTypeDef __SRAM_HandleTypeDef::State***
  SRAM device access state
- ***MDMA_HandleTypeDef* __SRAM_HandleTypeDef::hmdma***
  Pointer DMA handler
- ***void(* __SRAM_HandleTypeDef::MspInitCallback)(struct __SRAM_HandleTypeDef *hsram)***
  SRAM Msp Init callback
- ***void(* __SRAM_HandleTypeDef::MspDeInitCallback)(struct __SRAM_HandleTypeDef *hsram)***
  SRAM Msp DeInit callback
- ***void(* __SRAM_HandleTypeDef::DmaXferCpltCallback)(MDMA_HandleTypeDef *hmdma)***
  SRAM DMA Xfer Complete callback
- ***void(* __SRAM_HandleTypeDef::DmaXferErrorCallback)(MDMA_HandleTypeDef *hmdma)***
  SRAM DMA Xfer Error callback

86.2 SRAM Firmware driver API description

The following section lists the various functions of the SRAM library.

86.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control SRAM memories. It uses the FMC layer functions to interface with SRAM devices. The following sequence should be followed to configure the FMC to interface with SRAM/PSRAM memories:

1. Declare a SRAM_HandleTypeDef handle structure, for example: SRAM_HandleTypeDef hsram; and:
   – Fill the SRAM_HandleTypeDef handle "Init" field with the allowed values of the structure member.
   – Fill the SRAM_HandleTypeDef handle "Instance" field with a predefined base register instance for NOR or SRAM device
   – Fill the SRAM_HandleTypeDef handle "Extended" field with a predefined base register instance for NOR or SRAM extended mode
2. Declare two FMC_NORSRAM_TimingTypeDef structures, for both normal and extended mode timings; for example: FMC_NORSRAM_TimingTypeDef Timing and FMC_NORSRAM_TimingTypeDef ExTiming; and fill its fields with the allowed values of the structure member.
3. Initialize the SRAM Controller by calling the function HAL_SRAM_Init(). This function performs the following sequence:
   a. MSP hardware layer configuration using the function HAL_SRAM_MspInit()
   b. Control register configuration using the FMC NORSRAM interface function FMC_NORSRAM_Init()
   c. Timing register configuration using the FMC NORSRAM interface function FMC_NORSRAM_Timing_Init()
   d. Extended mode Timing register configuration using the FMC NORSRAM interface function FMC_NORSRAM_Extended_Timing_Init()
   e. Enable the SRAM device using the macro __FMC_NORSRAM_ENABLE()
4. At this stage you can perform read/write accesses from/to the memory connected to the NOR/SRAM Bank. You can perform either polling or DMA transfer using the following APIs:
   – HAL_SRAM_Read()/HAL_SRAM_Write() for polling read/write access
   – HAL_SRAM_Read_DMA()/HAL_SRAM_Write_DMA() for DMA read/write transfer
5. You can also control the SRAM device by calling the control APIs HAL_SRAM_WriteOperation_Enable()/ HAL_SRAM_WriteOperation_Disable() to respectively enable/disable the SRAM write operation
6. You can continuously monitor the SRAM device HAL state by calling the function HAL_SRAM_GetState()

Callback registration

The compilation define USE_HAL_SRAM_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_SRAM_RegisterCallback() to register a user callback, it allows to register following callbacks:

- MspInitCallback : SRAM MspInit.
- MspDeInitCallback : SRAM MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. Use function HAL_SRAM_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. It allows to reset following callbacks:
- MspInitCallback : SRAM MspInit.
- MspDeInitCallback : SRAM MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the HAL_SRAM_Init and if the state is HAL_SRAM_STATE_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the HAL_SRAM_Init and HAL_SRAM_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_SRAM_Init and HAL_SRAM_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/ unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_SRAM_RegisterCallback before calling HAL_SRAM_DeInit or HAL_SRAM_Init function. When The compilation define USE_HAL_SRAM_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

86.2.2 SRAM Initialization and de_initialization functions

This section provides functions allowing to initialize/de-initialize the SRAM memory

This section contains the following APIs:

- ***HAL_SRAM_Init()***
- ***HAL_SRAM_DeInit()***

- *HAL_SRAM_MspInit()*
- *HAL_SRAM_MspDeInit()*
- *HAL_SRAM_DMA_XferCpltCallback()*
- *HAL_SRAM_DMA_XferErrorCallback()*

86.2.3 SRAM Input and Output functions

This section provides functions allowing to use and control the SRAM memory

This section contains the following APIs:

- *HAL_SRAM_Read_8b()*
- *HAL_SRAM_Write_8b()*
- *HAL_SRAM_Read_16b()*
- *HAL_SRAM_Write_16b()*
- *HAL_SRAM_Read_32b()*
- *HAL_SRAM_Write_32b()*
- *HAL_SRAM_Read_DMA()*
- *HAL_SRAM_Write_DMA()*
- *HAL_SRAM_RegisterCallback()*
- *HAL_SRAM_UnRegisterCallback()*
- *HAL_SRAM_RegisterDmaCallback()*
- *HAL_SRAM_DMA_XferCpltCallback()*
- *HAL_SRAM_DMA_XferErrorCallback()*

86.2.4 SRAM Control functions

This subsection provides a set of functions allowing to control dynamically the SRAM interface.

This section contains the following APIs:

- *HAL_SRAM_WriteOperation_Enable()*
- *HAL_SRAM_WriteOperation_Disable()*

86.2.5 SRAM State functions

This subsection permits to get in run-time the status of the SRAM controller and the data flow.

This section contains the following APIs:

- *HAL_SRAM_GetState()*

86.2.6 Detailed description of functions

HAL_SRAM_Init

Function name

HAL_StatusTypeDef HAL_SRAM_Init (SRAM_HandleTypeDef * hsram, FMC_NORSRAM_TimingTypeDef * Timing, FMC_NORSRAM_TimingTypeDef * ExtTiming)

Function description

Performs the SRAM device initialization sequence.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **Timing:** Pointer to SRAM control timing structure
- **ExtTiming:** Pointer to SRAM extended mode timing structure

Return values

- **HAL:** status

HAL_SRAM_DeInit

Function name

HAL_StatusTypeDef HAL_SRAM_DeInit (SRAM_HandleTypeDef * hsram)

Function description

Performs the SRAM device De-initialization sequence.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **HAL:** status

HAL_SRAM_MspInit

Function name

void HAL_SRAM_MspInit (SRAM_HandleTypeDef * hsram)

Function description

SRAM MSP Init.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **None:**

HAL_SRAM_MspDeInit

Function name

void HAL_SRAM_MspDeInit (SRAM_HandleTypeDef * hsram)

Function description

SRAM MSP DeInit.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **None:**

HAL_SRAM_Read_8b

Function name

HAL_StatusTypeDef HAL_SRAM_Read_8b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint8_t * pDstBuffer, uint32_t BufferSize)

Function description

Reads 8-bit buffer from SRAM memory.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SRAM_Write_8b

Function name

HAL_StatusTypeDef HAL_SRAM_Write_8b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint8_t * pSrcBuffer, uint32_t BufferSize)

Function description

Writes 8-bit buffer to SRAM memory.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SRAM_Read_16b

Function name

HAL_StatusTypeDef HAL_SRAM_Read_16b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint16_t * pDstBuffer, uint32_t BufferSize)

Function description

Reads 16-bit buffer from SRAM memory.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SRAM_Write_16b

Function name

HAL_StatusTypeDef HAL_SRAM_Write_16b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint16_t * pSrcBuffer, uint32_t BufferSize)

Function description

Writes 16-bit buffer to SRAM memory.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SRAM_Read_32b

Function name

HAL_StatusTypeDef HAL_SRAM_Read_32b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pDstBuffer, uint32_t BufferSize)

Function description

Reads 32-bit buffer from SRAM memory.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SRAM_Write_32b

Function name

HAL_StatusTypeDef HAL_SRAM_Write_32b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)

Function description

Writes 32-bit buffer to SRAM memory.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SRAM_Read_DMA

Function name

HAL_StatusTypeDef HAL_SRAM_Read_DMA (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pDstBuffer, uint32_t BufferSize)

Function description

Reads a Words data from the SRAM memory using DMA transfer.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SRAM_Write_DMA

Function name

HAL_StatusTypeDef HAL_SRAM_Write_DMA (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)

Function description

Writes a Words data buffer to SRAM memory using DMA transfer.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SRAM_DMA_XferCpltCallback

Function name

void HAL_SRAM_DMA_XferCpltCallback (MDMA_HandleTypeDef * hmdma)

Function description

DMA transfer complete callback.

Parameters

- **hmdma:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **None:**

HAL_SRAM_DMA_XferErrorCallback

Function name

void HAL_SRAM_DMA_XferErrorCallback (MDMA_HandleTypeDef * hmdma)

Function description

DMA transfer complete error callback.

Parameters

- **hmdma:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **None:**

HAL_SRAM_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SRAM_RegisterCallback (SRAM_HandleTypeDef * hsram, HAL_SRAM_CallbackIDTypeDef CallbackId, pSRAM_CallbackTypeDef pCallback)

Function description

Register a User SRAM Callback To be used instead of the weak (surcharged) predefined callback.

Parameters

- **hsram:** : SRAM handle
- **CallbackId:** : ID of the callback to be registered This parameter can be one of the following values:
  - HAL_SRAM_MSP_INIT_CB_ID SRAM MspInit callback ID
  - HAL_SRAM_MSP_DEINIT_CB_ID SRAM MspDeInit callback ID
- **pCallback:** : pointer to the Callback function

Return values

- **status:**

HAL_SRAM_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SRAM_UnRegisterCallback (SRAM_HandleTypeDef * hsram, HAL_SRAM_CallbackIDTypeDef CallbackId)

Function description

Unregister a User SRAM Callback SRAM Callback is redirected to the weak (surcharged) predefined callback.

Parameters

- **hsram:** : SRAM handle
- **CallbackId:** : ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_SRAM_MSP_INIT_CB_ID SRAM MspInit callback ID
  - HAL_SRAM_MSP_DEINIT_CB_ID SRAM MspDeInit callback ID
  - HAL_SRAM_DMA_XFER_CPLT_CB_ID SRAM DMA Xfer Complete callback ID
  - HAL_SRAM_DMA_XFER_ERR_CB_ID SRAM DMA Xfer Error callback ID

Return values

- **status:**

HAL_SRAM_RegisterDmaCallback

Function name

HAL_StatusTypeDef HAL_SRAM_RegisterDmaCallback (SRAM_HandleTypeDef * hsram, HAL_SRAM_CallbackIDTypeDef CallbackId, pSRAM_DmaCallbackTypeDef pCallback)

Function description

Register a User SRAM Callback for DMA transfers To be used instead of the weak (surcharged) predefined callback.

Parameters

- **hsram:** : SRAM handle
- **CallbackId:** : ID of the callback to be registered This parameter can be one of the following values:
  - HAL_SRAM_DMA_XFER_CPLT_CB_ID SRAM DMA Xfer Complete callback ID
  - HAL_SRAM_DMA_XFER_ERR_CB_ID SRAM DMA Xfer Error callback ID
- **pCallback:** : pointer to the Callback function

Return values

- **status:**

HAL_SRAM_WriteOperation_Enable

Function name

HAL_StatusTypeDef HAL_SRAM_WriteOperation_Enable (SRAM_HandleTypeDef * hsram)

Function description

Enables dynamically SRAM write operation.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **HAL:** status

HAL_SRAM_WriteOperation_Disable

Function name

HAL_StatusTypeDef HAL_SRAM_WriteOperation_Disable (SRAM_HandleTypeDef * hsram)

Function description

Disables dynamically SRAM write operation.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **HAL:** status

HAL_SRAM_GetState

Function name

HAL_SRAM_StateTypeDef HAL_SRAM_GetState (SRAM_HandleTypeDef * hsram)

Function description

Returns the SRAM controller state.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **HAL:** state

86.3 SRAM Firmware driver defines

The following section lists the various define and macros of the module.

86.3.1 SRAM

SRAM

SRAM Exported Macros

__HAL_SRAM_RESET_HANDLE_STATE

Description:

- Reset SRAM handle state.

Parameters:

- __HANDLE__: SRAM handle

Return value:

- None

87 HAL SWPMI Generic Driver

87.1 SWPMI Firmware driver registers structures

87.1.1 SWPMI_InitTypeDef

SWPMI_InitTypeDef is defined in the stm32h7xx_hal_swpmi.h

Data Fields

- ***uint32_t VoltageClass***
- ***uint32_t BitRate***
- ***uint32_t TxBufferingMode***
- ***uint32_t RxBufferingMode***

Field Documentation

- ***uint32_t SWPMI_InitTypeDef::VoltageClass***
  Specifies the SWP Voltage Class. This parameter can be a value of ***SWPMI_Voltage_Class***
- ***uint32_t SWPMI_InitTypeDef::BitRate***
  Specifies the SWPMI Bitrate. This parameter must be a number between 0 and 255U. The Bitrate is computed using the following formula: SWPMI_freq = SWPMI_clk / (((BitRate) + 1) * 4)
- ***uint32_t SWPMI_InitTypeDef::TxBufferingMode***
  Specifies the transmission buffering mode. This parameter can be a value of ***SWPMI_Tx_Buffering_Mode***
- ***uint32_t SWPMI_InitTypeDef::RxBufferingMode***
  Specifies the reception buffering mode. This parameter can be a value of ***SWPMI_Rx_Buffering_Mode***

87.1.2 __SWPMI_HandleTypeDef

__SWPMI_HandleTypeDef is defined in the stm32h7xx_hal_swpmi.h

Data Fields

- ***SWPMI_TypeDef * Instance***
- ***SWPMI_InitTypeDef Init***
- ***uint32_t * pTxBuffPtr***
- ***uint32_t TxXferSize***
- ***uint32_t TxXferCount***
- ***uint32_t * pRxBuffPtr***
- ***uint32_t RxXferSize***
- ***uint32_t RxXferCount***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SWPMI_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* RxCpltCallback***
- ***void(* RxHalfCpltCallback***
- ***void(* TxCpltCallback***
- ***void(* TxHalfCpltCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***SWPMI_TypeDef* __SWPMI_HandleTypeDef::Instance***
  SWPMI registers base address
- ***SWPMI_InitTypeDef __SWPMI_HandleTypeDef::Init***
  SWPMI communication parameters
- ***uint32_t* __SWPMI_HandleTypeDef::pTxBuffPtr***
  Pointer to SWPMI Tx transfer Buffer
- ***uint32_t __SWPMI_HandleTypeDef::TxXferSize***
  SWPMI Tx Transfer size
- ***uint32_t __SWPMI_HandleTypeDef::TxXferCount***
  SWPMI Tx Transfer Counter
- ***uint32_t* __SWPMI_HandleTypeDef::pRxBuffPtr***
  Pointer to SWPMI Rx transfer Buffer
- ***uint32_t __SWPMI_HandleTypeDef::RxXferSize***
  SWPMI Rx Transfer size
- ***uint32_t __SWPMI_HandleTypeDef::RxXferCount***
  SWPMI Rx Transfer Counter
- ***DMA_HandleTypeDef* __SWPMI_HandleTypeDef::hdmatx***
  SWPMI Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __SWPMI_HandleTypeDef::hdmarx***
  SWPMI Rx DMA Handle parameters
- ***HAL_LockTypeDef __SWPMI_HandleTypeDef::Lock***
  SWPMI object
- ***__IO HAL_SWPMI_StateTypeDef __SWPMI_HandleTypeDef::State***
  SWPMI communication state
- ***__IO uint32_t __SWPMI_HandleTypeDef::ErrorCode***
  SWPMI Error code
- ***void(* __SWPMI_HandleTypeDef::RxCpltCallback)(struct __SWPMI_HandleTypeDef *hswpmi)***
  SWPMI receive complete callback
- ***void(* __SWPMI_HandleTypeDef::RxHalfCpltCallback)(struct __SWPMI_HandleTypeDef *hswpmi)***
  SWPMI receive half complete callback
- ***void(* __SWPMI_HandleTypeDef::TxCpltCallback)(struct __SWPMI_HandleTypeDef *hswpmi)***
  SWPMI transmit complete callback
- ***void(* __SWPMI_HandleTypeDef::TxHalfCpltCallback)(struct __SWPMI_HandleTypeDef *hswpmi)***
  SWPMI transmit half complete callback
- ***void(* __SWPMI_HandleTypeDef::ErrorCallback)(struct __SWPMI_HandleTypeDef *hswpmi)***
  SWPMI error callback
- ***void(* __SWPMI_HandleTypeDef::MspInitCallback)(struct __SWPMI_HandleTypeDef *hswpmi)***
  SWPMI MSP init callback
- ***void(* __SWPMI_HandleTypeDef::MspDeInitCallback)(struct __SWPMI_HandleTypeDef *hswpmi)***
  SWPMI MSP de-init callback

87.2 SWPMI Firmware driver API description

The following section lists the various functions of the SWPMI library.

87.2.1 How to use this driver

The SWPMI HAL driver can be used as follows:

1. Declare a SWPMI_HandleTypeDef handle structure (eg. SWPMI_HandleTypeDef hswpmi).

2. Initialize the SWPMI low level resources by implementing the HAL_SWPMI_MspInit() API:
   a. Enable the SWPMIx interface clock with __HAL_RCC_SWPMIx_CLK_ENABLE().
   b. SWPMI IO configuration:
      - Enable the clock for the SWPMI GPIO.
      - Configure these SWPMI pins as alternate function pull-up.
   c. NVIC configuration if you need to use interrupt process (HAL_SWPMI_Transmit_IT() and HAL_SWPMI_Receive_IT() APIs):
      - Configure the SWPMIx interrupt priority with HAL_NVIC_SetPriority().
      - Enable the NVIC SWPMI IRQ handle with HAL_NVIC_EnableIRQ().
   d. DMA Configuration if you need to use DMA process (HAL_SWPMI_Transmit_DMA() and HAL_SWPMI_Receive_DMA() APIs):
      - Declare a DMA handle structure for the Tx/Rx streams.
      - Enable the DMAx interface clock.
      - Configure the declared DMA handle structure with the required Tx/Rx parameters.
      - Configure the DMA Tx/Rx streams and requests.
      - Associate the initialized DMA handle to the SWPMI DMA Tx/Rx handle.
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx streams.
3. Program the Bite Rate, Tx Buffering mode, Rx Buffering mode in the Init structure.
4. Enable the SWPMI peripheral by calling the HAL_SWPMI_Init() function.

Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_SWPMI_Transmit()
- Receive an amount of data in blocking mode using HAL_SWPMI_Receive()

Interrupt mode IO operation

- Send an amount of data in non-blocking mode using HAL_SWPMI_Transmit_IT()
- At transmission end of transfer HAL_SWPMI_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SWPMI_TxCpltCallback()
- Receive an amount of data in non-blocking mode using HAL_SWPMI_Receive_IT()
- At reception end of transfer HAL_SWPMI_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SWPMI_RxCpltCallback()
- In case of flag error, HAL_SWPMI_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SWPMI_ErrorCallback()

DMA mode IO operation

- Send an amount of data in non-blocking mode (DMA) using HAL_SWPMI_Transmit_DMA()
- At transmission end of transfer HAL_SWPMI_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SWPMI_TxCpltCallback()
- Receive an amount of data in non-blocking mode (DMA) using HAL_SWPMI_Receive_DMA()
- At reception end of transfer HAL_SWPMI_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SWPMI_RxCpltCallback()
- In case of flag error, HAL_SWPMI_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SWPMI_ErrorCallback()
- Stop the DMA Transfer using HAL_SWPMI_DMAStop()

SWPMI HAL driver additional function list

Below the list the others API available SWPMI HAL driver :

- HAL_SWPMI_EnableLoopback(): Enable the loopback mode for test purpose only
- HAL_SWPMI_DisableLoopback(): Disable the loopback mode

SWPMI HAL driver macros list

Below the list of most used macros in SWPMI HAL driver :

- __HAL_SWPMI_ENABLE(): Enable the SWPMI peripheral
- __HAL_SWPMI_DISABLE(): Disable the SWPMI peripheral
- __HAL_SWPMI_TRANSCEIVER_ENABLE(): Enable the SWPMI peripheral transceiver
- __HAL_SWPMI_TRANSCEIVER_DISABLE(): Disable the SWPMI peripheral transceiver
- __HAL_SWPMI_ENABLE_IT(): Enable the specified SWPMI interrupts
- __HAL_SWPMI_DISABLE_IT(): Disable the specified SWPMI interrupts
- __HAL_SWPMI_GET_IT_SOURCE(): Check if the specified SWPMI interrupt source is enabled or disabled
- __HAL_SWPMI_GET_FLAG(): Check whether the specified SWPMI flag is set or not

Callback registration

The compilation define USE_HAL_SWPMI_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use function HAL_SWPMI_RegisterCallback() to register a user callback. It allows to register the following callbacks:

- RxCpltCallback : SWPMI receive complete.
- RxHalfCpltCallback : SWPMI receive half complete.
- TxCpltCallback : SWPMI transmit complete.
- TxHalfCpltCallback : SWPMI transmit half complete.
- ErrorCallback : SWPMI error.
- MspInitCallback : SWPMI MspInit.
- MspDeInitCallback : SWPMI MspDeInit.

This function takes as parameters the HAL peripheral handle, the callback ID and a pointer to the user callback function.

Use function HAL_SWPMI_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. HAL_SWPMI_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the callback ID. This function allows to reset following callbacks:

- RxCpltCallback : SWPMI receive complete.
- RxHalfCpltCallback : SWPMI receive half complete.
- TxCpltCallback : SWPMI transmit complete.
- TxHalfCpltCallback : SWPMI transmit half complete.
- ErrorCallback : SWPMI error.
- MspInitCallback : SWPMI MspInit.
- MspDeInitCallback : SWPMI MspDeInit.

By default, after the HAL_SWPMI_Init and if the state is HAL_SWPMI_STATE_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions: examples HAL_SWPMI_RxCpltCallback(), HAL_SWPMI_ErrorCallback(). Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the HAL_SWPMI_Init and HAL_SWPMI_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_SWPMI_Init and HAL_SWPMI_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_SWPMI_RegisterCallback before calling HAL_SWPMI_DeInit or HAL_SWPMI_Init function.

When the compilation define USE_HAL_SWPMI_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

87.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the SWPMI peripheral.
- De-initialize the SWPMI peripheral.

This section contains the following APIs:

- ***HAL_SWPMI_Init()***
- ***HAL_SWPMI_DeInit()***
- ***HAL_SWPMI_MspInit()***
- ***HAL_SWPMI_MspDeInit()***
- ***HAL_SWPMI_RegisterCallback()***
- ***HAL_SWPMI_UnRegisterCallback()***

87.2.3 IO operation methods

This subsection provides a set of functions allowing to manage the SWPMI data transfers.

1. There are two modes of transfer:
   – Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
   – Non-Blocking mode: The communication is performed using Interrupts or DMA. The end of the data processing will be indicated through the dedicated SWPMI Interrupt handler (HAL_SWPMI_IRQHandler()) when using Interrupt mode or the selected DMA stream interrupt handler when using DMA mode. The HAL_SWPMI_TxCpltCallback(), HAL_SWPMI_RxCpltCallback() user callbacks will be executed respectively at the end of the transmit or receive process. The HAL_SWPMI_ErrorCallback() user callback will be executed when a communication error is detected.
2. Blocking mode API's are:
   – HAL_SWPMI_Transmit()
   – HAL_SWPMI_Receive()
3. Non-Blocking mode API's with Interrupt are:
   – HAL_SWPMI_Transmit_IT()
   – HAL_SWPMI_Receive_IT()
   – HAL_SWPMI_IRQHandler()
4. Non-Blocking mode API's with DMA are:
   – HAL_SWPMI_Transmit_DMA()
   – HAL_SWPMI_Receive_DMA()
   – HAL_SWPMI_DMAPause()
   – HAL_SWPMI_DMAResume()
   – HAL_SWPMI_DMAStop()
5. A set of Transfer Complete Callbacks are provided in Non-Blocking mode:
   – HAL_SWPMI_TxHalfCpltCallback()
   – HAL_SWPMI_TxCpltCallback()
   – HAL_SWPMI_RxHalfCpltCallback()
   – HAL_SWPMI_RxCpltCallback()
   – HAL_SWPMI_ErrorCallback()
6. The capability to launch the above IO operations in loopback mode for user application verification:
   – HAL_SWPMI_EnableLoopback()
   – HAL_SWPMI_DisableLoopback()

This section contains the following APIs:

- ***HAL_SWPMI_Transmit()***
- ***HAL_SWPMI_Receive()***
- ***HAL_SWPMI_Transmit_IT()***
- ***HAL_SWPMI_Receive_IT()***
- ***HAL_SWPMI_Transmit_DMA()***
- ***HAL_SWPMI_Receive_DMA()***

- *HAL_SWPMI_DMAStop()*
- *HAL_SWPMI_EnableLoopback()*
- *HAL_SWPMI_DisableLoopback()*

87.2.4 SWPMI IRQ handler and callbacks

This section provides SWPMI IRQ handler and callback functions called within the IRQ handler.
This section contains the following APIs:

- *HAL_SWPMI_IRQHandler()*
- *HAL_SWPMI_TxCpltCallback()*
- *HAL_SWPMI_TxHalfCpltCallback()*
- *HAL_SWPMI_RxCpltCallback()*
- *HAL_SWPMI_RxHalfCpltCallback()*
- *HAL_SWPMI_ErrorCallback()*

87.2.5 Peripheral Control methods

This subsection provides a set of functions allowing to control the SWPMI.

- HAL_SWPMI_GetState() API is helpful to check in run-time the state of the SWPMI peripheral
- HAL_SWPMI_GetError() API is helpful to check in run-time the error state of the SWPMI peripheral

This section contains the following APIs:

- *HAL_SWPMI_GetState()*
- *HAL_SWPMI_GetError()*

87.2.6 Detailed description of functions

HAL_SWPMI_Init

Function name

HAL_StatusTypeDef HAL_SWPMI_Init (SWPMI_HandleTypeDef * hswpmi)

Function description

Initialize the SWPMI peripheral according to the specified parameters in the SWPMI_InitTypeDef.

Parameters

- **hswpmi:** SWPMI handle

Return values

- **HAL:** status

HAL_SWPMI_DeInit

Function name

HAL_StatusTypeDef HAL_SWPMI_DeInit (SWPMI_HandleTypeDef * hswpmi)

Function description

De-initialize the SWPMI peripheral.

Parameters

- **hswpmi:** SWPMI handle

Return values

- **HAL:** status

HAL_SWPMI_MspInit

Function name

void HAL_SWPMI_MspInit (SWPMI_HandleTypeDef * hswpmi)

Function description

Initialize the SWPMI MSP.

Parameters

- **hswpmi:** SWPMI handle

Return values

- **None:**

HAL_SWPMI_MspDeInit

Function name

void HAL_SWPMI_MspDeInit (SWPMI_HandleTypeDef * hswpmi)

Function description

DeInitialize the SWPMI MSP.

Parameters

- **hswpmi:** SWPMI handle

Return values

- **None:**

HAL_SWPMI_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SWPMI_RegisterCallback (SWPMI_HandleTypeDef * hswpmi, HAL_SWPMI_CallbackIDTypeDef CallbackID, pSWPMI_CallbackTypeDef pCallback)

Function description

Register a user SWPMI callback to be used instead of the weak predefined callback.

Parameters

- **hswpmi:** SWPMI handle.
- **CallbackID:** ID of the callback to be registered. This parameter can be one of the following values:
  - HAL_SWPMI_RX_COMPLETE_CB_ID receive complete callback ID.
  - HAL_SWPMI_RX_HALFCOMPLETE_CB_ID receive half complete callback ID.
  - HAL_SWPMI_TX_COMPLETE_CB_ID transmit complete callback ID.
  - HAL_SWPMI_TX_HALFCOMPLETE_CB_ID transmit half complete callback ID.
  - HAL_SWPMI_ERROR_CB_ID error callback ID.
  - HAL_SWPMI_MSPINIT_CB_ID MSP init callback ID.
  - HAL_SWPMI_MSPDEINIT_CB_ID MSP de-init callback ID.
- **pCallback:** pointer to the callback function.

Return values

- **HAL:** status.

HAL_SWPMI_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SWPMI_UnRegisterCallback (SWPMI_HandleTypeDef * hswpmi, HAL_SWPMI_CallbackIDTypeDef CallbackID)

Function description

Unregister a user SWPMI callback.

Parameters

- **hswpmi:** SWPMI handle.
- **CallbackID:** ID of the callback to be unregistered. This parameter can be one of the following values:
  - HAL_SWPMI_RX_COMPLETE_CB_ID receive complete callback ID.
  - HAL_SWPMI_RX_HALFCOMPLETE_CB_ID receive half complete callback ID.
  - HAL_SWPMI_TX_COMPLETE_CB_ID transmit complete callback ID.
  - HAL_SWPMI_TX_HALFCOMPLETE_CB_ID transmit half complete callback ID.
  - HAL_SWPMI_ERROR_CB_ID error callback ID.
  - HAL_SWPMI_MSPINIT_CB_ID MSP init callback ID.
  - HAL_SWPMI_MSPDEINIT_CB_ID MSP de-init callback ID.

Return values

- **HAL:** status.

HAL_SWPMI_Transmit

Function name

HAL_StatusTypeDef HAL_SWPMI_Transmit (SWPMI_HandleTypeDef * hswpmi, uint32_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Transmit an amount of data in blocking mode.

Parameters

- **hswpmi:** pointer to a SWPMI_HandleTypeDef structure that contains the configuration information for SWPMI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SWPMI_Receive

Function name

HAL_StatusTypeDef HAL_SWPMI_Receive (SWPMI_HandleTypeDef * hswpmi, uint32_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **hswpmi:** pointer to a SWPMI_HandleTypeDef structure that contains the configuration information for SWPMI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SWPMI_Transmit_IT

Function name

HAL_StatusTypeDef HAL_SWPMI_Transmit_IT (SWPMI_HandleTypeDef * hswpmi, uint32_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with interrupt.

Parameters

- **hswpmi:** pointer to a SWPMI_HandleTypeDef structure that contains the configuration information for SWPMI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_SWPMI_Receive_IT

Function name

HAL_StatusTypeDef HAL_SWPMI_Receive_IT (SWPMI_HandleTypeDef * hswpmi, uint32_t * pData, uint16_t Size)

Function description

Receive an amount of data in non-blocking mode with interrupt.

Parameters

- **hswpmi:** SWPMI handle
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received

Return values

- **HAL:** status

HAL_SWPMI_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_SWPMI_Transmit_DMA (SWPMI_HandleTypeDef * hswpmi, uint32_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with DMA interrupt.

Parameters

- **hswpmi:** SWPMI handle
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_SWPMI_Receive_DMA

Function name

HAL_StatusTypeDef HAL_SWPMI_Receive_DMA (SWPMI_HandleTypeDef * hswpmi, uint32_t * pData, uint16_t Size)

Function description

Receive an amount of data in non-blocking mode with DMA interrupt.

Parameters

- **hswpmi:** SWPMI handle
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received

Return values

- **HAL:** status

HAL_SWPMI_DMAStop

Function name

HAL_StatusTypeDef HAL_SWPMI_DMAStop (SWPMI_HandleTypeDef * hswpmi)

Function description

Stop all DMA transfers.

Parameters

- **hswpmi:** SWPMI handle

Return values

- **HAL:** status

HAL_SWPMI_EnableLoopback

Function name

HAL_StatusTypeDef HAL_SWPMI_EnableLoopback (SWPMI_HandleTypeDef * hswpmi)

Function description

Enable the Loopback mode.

Parameters

- **hswpmi:** SWPMI handle

Return values

- **HAL_OK:** / HAL_BUSY

Notes

- Loopback mode is to be used only for test purposes

HAL_SWPMI_DisableLoopback

Function name

HAL_StatusTypeDef HAL_SWPMI_DisableLoopback (SWPMI_HandleTypeDef * hswpmi)

Function description

Disable the Loopback mode.

Parameters

- **hswpmi:** SWPMI handle

Return values

- **HAL_OK:** / HAL_BUSY

Notes

- Loopback mode is to be used only for test purposes

HAL_SWPMI_IRQHandler

Function name

void HAL_SWPMI_IRQHandler (SWPMI_HandleTypeDef * hswpmi)

Function description

Handle SWPMI interrupt request.

Parameters

- **hswpmi:** SWPMI handle

Return values

- **None:**

HAL_SWPMI_TxCpltCallback

Function name

void HAL_SWPMI_TxCpltCallback (SWPMI_HandleTypeDef * hswpmi)

Function description

Tx Transfer completed callback.

Parameters

- **hswpmi:** SWPMI handle

Return values

- **None:**

HAL_SWPMI_TxHalfCpltCallback

Function name

void HAL_SWPMI_TxHalfCpltCallback (SWPMI_HandleTypeDef * hswpmi)

Function description

Tx Half Transfer completed callback.

Parameters

- **hswpmi:** SWPMI handle

Return values

- **None:**

HAL_SWPMI_RxCpltCallback

Function name

void HAL_SWPMI_RxCpltCallback (SWPMI_HandleTypeDef * hswpmi)

Function description

Rx Transfer completed callback.

Parameters

- **hswpmi:** SWPMI handle

Return values

- **None:**

HAL_SWPMI_RxHalfCpltCallback

Function name

void HAL_SWPMI_RxHalfCpltCallback (SWPMI_HandleTypeDef * hswpmi)

Function description

Rx Half Transfer completed callback.

Parameters

- **hswpmi:** SWPMI handle

Return values

- **None:**

HAL_SWPMI_ErrorCallback

Function name

void HAL_SWPMI_ErrorCallback (SWPMI_HandleTypeDef * hswpmi)

Function description

SWPMI error callback.

Parameters

- **hswpmi:** SWPMI handle

Return values

- **None:**

HAL_SWPMI_GetState

Function name

HAL_SWPMI_StateTypeDef HAL_SWPMI_GetState (SWPMI_HandleTypeDef * hswpmi)

Function description

Return the SWPMI handle state.

Parameters

- **hswpmi:** SWPMI handle

Return values

- **HAL:** state

HAL_SWPMI_GetError

Function name

uint32_t HAL_SWPMI_GetError (SWPMI_HandleTypeDef * hswpmi)

Function description

Return the SWPMI error code.

Parameters

- **hswpmi:** : pointer to a SWPMI_HandleTypeDef structure that contains the configuration information for the specified SWPMI.

Return values

- **SWPMI:** Error Code

87.3 SWPMI Firmware driver defines

The following section lists the various define and macros of the module.

87.3.1 SWPMI

SWPMI

SWPMI Error Code Bitmap

HAL_SWPMI_ERROR_NONE

No error

HAL_SWPMI_ERROR_CRC

frame error

HAL_SWPMI_ERROR_OVR

Overrun error

HAL_SWPMI_ERROR_UDR

Underrun error

HAL_SWPMI_ERROR_DMA

DMA transfer error

HAL_SWPMI_ERROR_TIMEOUT

Transfer timeout

HAL_SWPMI_ERROR_TXBEF_TIMEOUT

End Tx buffer timeout

HAL_SWPMI_ERROR_TRANSCEIVER_NOT_READY

Transceiver not ready

HAL_SWPMI_ERROR_INVALID_CALLBACK

Invalid callback error

SWPMI Exported Macros

__HAL_SWPMI_RESET_HANDLE_STATE

Description:

- Reset SWPMI handle state.

Parameters:

- __HANDLE__: specifies the SWPMI Handle.

Return value:

- None

__HAL_SWPMI_ENABLE

Description:

- Enable the SWPMI peripheral.

Parameters:

- __HANDLE__: SWPMI handle

Return value:

- None

__HAL_SWPMI_DISABLE

Description:

- Disable the SWPMI peripheral.

Parameters:

- __HANDLE__: SWPMI handle

Return value:

- None

__HAL_SWPMI_TRANSCEIVER_ENABLE

Description:

- Enable the SWPMI transceiver.

Parameters:

- __HANDLE__: SWPMI handle

Return value:

- None

__HAL_SWPMI_TRANSCEIVER_DISABLE

Description:

- Disable the SWPMI transceiver.

Parameters:

- __HANDLE__: SWPMI handle

Return value:

- None

__HAL_SWPMI_GET_FLAG

Description:

- Check whether the specified SWPMI flag is set or not.

Parameters:

- __HANDLE__: specifies the SWPMI Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SWPMI_FLAG_RXBFF Receive buffer full flag.
  - SWPMI_FLAG_TXBEF Transmit buffer empty flag.
  - SWPMI_FLAG_RXBERF Receive CRC error flag.
  - SWPMI_FLAG_RXOVRF Receive overrun error flag.
  - SWPMI_FLAG_TXUNRF Transmit underrun error flag.
  - SWPMI_FLAG_RXNE Receive data register not empty.
  - SWPMI_FLAG_TXE Transmit data register empty.
  - SWPMI_FLAG_TCF Transfer complete flag.
  - SWPMI_FLAG_SRF Slave resume flag.
  - SWPMI_FLAG_SUSP SUSPEND flag.
  - SWPMI_FLAG_DEACTF DEACTIVATED flag.
  - SWPMI_FLAG_RDYF Transceiver ready flag.

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_SWPMI_CLEAR_FLAG

Description:

- Clear the specified SWPMI ISR flag.

Parameters:

- __HANDLE__: specifies the SWPMI Handle.
- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
  - SWPMI_FLAG_RXBFF Receive buffer full flag.
  - SWPMI_FLAG_TXBEF Transmit buffer empty flag.
  - SWPMI_FLAG_RXBERF Receive CRC error flag.
  - SWPMI_FLAG_RXOVRF Receive overrun error flag.
  - SWPMI_FLAG_TXUNRF Transmit underrun error flag.
  - SWPMI_FLAG_TCF Transfer complete flag.
  - SWPMI_FLAG_SRF Slave resume flag.
  - SWPMI_FLAG_RDYF Transceiver ready flag.

Return value:

- None

__HAL_SWPMI_ENABLE_IT

Description:

- Enable the specified SWPMI interrupt.

Parameters:

- __HANDLE__: specifies the SWPMI Handle.
- __INTERRUPT__: specifies the SWPMI interrupt source to enable. This parameter can be one of the following values:
  - SWPMI_IT_RDYIE Transceiver ready interrupt.
  - SWPMI_IT_SRIE Slave resume interrupt.
  - SWPMI_IT_TCIE Transmit complete interrupt.
  - SWPMI_IT_TIE Transmit interrupt.
  - SWPMI_IT_RIE Receive interrupt.
  - SWPMI_IT_TXUNRIE Transmit underrun error interrupt.
  - SWPMI_IT_RXOVRIE Receive overrun error interrupt.
  - SWPMI_IT_RXBEIE Receive CRC error interrupt.
  - SWPMI_IT_TXBEIE Transmit buffer empty interrupt.
  - SWPMI_IT_RXBFIE Receive buffer full interrupt.

Return value:

- None

__HAL_SWPMI_DISABLE_IT

Description:

- Disable the specified SWPMI interrupt.

Parameters:

- __HANDLE__: specifies the SWPMI Handle.
- __INTERRUPT__: specifies the SWPMI interrupt source to disable. This parameter can be one of the following values:
  - SWPMI_IT_RDYIE Transceiver ready interrupt.
  - SWPMI_IT_SRIE Slave resume interrupt.
  - SWPMI_IT_TCIE Transmit complete interrupt.
  - SWPMI_IT_TIE Transmit interrupt.
  - SWPMI_IT_RIE Receive interrupt.
  - SWPMI_IT_TXUNRIE Transmit underrun error interrupt.
  - SWPMI_IT_RXOVRIE Receive overrun error interrupt.
  - SWPMI_IT_RXBEIE Receive CRC error interrupt.
  - SWPMI_IT_TXBEIE Transmit buffer empty interrupt.
  - SWPMI_IT_RXBFIE Receive buffer full interrupt.

Return value:

- None

__HAL_SWPMI_GET_IT

Description:

- Check whether the specified SWPMI interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the SWPMI Handle.
- __IT__: specifies the SWPMI interrupt to check. This parameter can be one of the following values:
  - SWPMI_IT_RDYIE Transceiver ready interrupt.
  - SWPMI_IT_SRIE Slave resume interrupt.
  - SWPMI_IT_TCIE Transmit complete interrupt.
  - SWPMI_IT_TIE Transmit interrupt.
  - SWPMI_IT_RIE Receive interrupt.
  - SWPMI_IT_TXUNRIE Transmit underrun error interrupt.
  - SWPMI_IT_RXOVRIE Receive overrun error interrupt.
  - SWPMI_IT_RXBERIE Receive CRC error interrupt.
  - SWPMI_IT_TXBEIE Transmit buffer empty interrupt.
  - SWPMI_IT_RXBFIE Receive buffer full interrupt.

Return value:

- The: new state of __IT__ (TRUE or FALSE).

__HAL_SWPMI_GET_IT_SOURCE

Description:

- Check whether the specified SWPMI interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the SWPMI Handle.
- __IT__: specifies the SWPMI interrupt source to check. This parameter can be one of the following values:
  - SWPMI_IT_RDYIE Transceiver ready interrupt.
  - SWPMI_IT_SRIE Slave resume interrupt.
  - SWPMI_IT_TCIE Transmit complete interrupt.
  - SWPMI_IT_TIE Transmit interrupt.
  - SWPMI_IT_RIE Receive interrupt.
  - SWPMI_IT_TXUNRIE Transmit underrun error interrupt.
  - SWPMI_IT_RXOVRIE Receive overrun error interrupt.
  - SWPMI_IT_RXBERIE Receive CRC error interrupt.
  - SWPMI_IT_TXBEIE Transmit buffer empty interrupt.
  - SWPMI_IT_RXBFIE Receive buffer full interrupt.

Return value:

- The: new state of __IT__ (TRUE or FALSE).

SWPMI Status Flags

SWPMI_FLAG_RXBFF

SWPMI_FLAG_TXBEF

SWPMI_FLAG_RXBERF

SWPMI_FLAG_RXOVRF

SWPMI_FLAG_TXUNRF

SWPMI_FLAG_RXNE

SWPMI_FLAG_TXE

SWPMI_FLAG_TCF

SWPMI_FLAG_SRF

SWPMI_FLAG_SUSP

SWPMI_FLAG_DEACTF

SWPMI_FLAG_RDYF

SWPMI Interrupts Definition

SWPMI_IT_RDYIE

SWPMI_IT_SRIE

SWPMI_IT_TCIE

SWPMI_IT_TIE

SWPMI_IT_RIE

SWPMI_IT_TXUNRIE

SWPMI_IT_RXOVRIE

SWPMI_IT_RXBERIE

SWPMI_IT_TXBEIE

SWPMI_IT_RXBFIE

SWPMI Rx Buffering Mode

SWPMI_RX_NO_SOFTWAREBUFFER

SWPMI_RX_SINGLE_SOFTWAREBUFFER

SWPMI_RX_MULTI_SOFTWAREBUFFER

SWPMI Tx Buffering Mode

SWPMI_TX_NO_SOFTWAREBUFFER

SWPMI_TX_SINGLE_SOFTWAREBUFFER

SWPMI_TX_MULTI_SOFTWAREBUFFER

SWPMI Voltage Class

SWPMI_VOLTAGE_CLASS_C

SWPMI_VOLTAGE_CLASS_B

88 HAL TIM Generic Driver

88.1 TIM Firmware driver registers structures

88.1.1 TIM_Base_InitTypeDef

TIM_Base_InitTypeDef is defined in the stm32h7xx_hal_tim.h

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t CounterMode***
- ***uint32_t Period***
- ***uint32_t ClockDivision***
- ***uint32_t RepetitionCounter***
- ***uint32_t AutoReloadPreload***

Field Documentation

- ***uint32_t TIM_Base_InitTypeDef::Prescaler***

  Specifies the prescaler value used to divide the TIM clock. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF
- ***uint32_t TIM_Base_InitTypeDef::CounterMode***

  Specifies the counter mode. This parameter can be a value of ***TIM_Counter_Mode***
- ***uint32_t TIM_Base_InitTypeDef::Period***

  Specifies the period value to be loaded into the active Auto-Reload Register at the next update event. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.
- ***uint32_t TIM_Base_InitTypeDef::ClockDivision***

  Specifies the clock division. This parameter can be a value of ***TIM_ClockDivision***
- ***uint32_t TIM_Base_InitTypeDef::RepetitionCounter***

  Specifies the repetition counter value. Each time the RCR downcounter reaches zero, an update event is generated and counting restarts from the RCR value (N). This means in PWM mode that (N+1) corresponds to:
  - the number of PWM periods in edge-aligned mode
  - the number of half PWM period in center-aligned mode GP timers: this parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF. Advanced timers: this parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.
- ***uint32_t TIM_Base_InitTypeDef::AutoReloadPreload***

  Specifies the auto-reload preload. This parameter can be a value of ***TIM_AutoReloadPreload***

88.1.2 TIM_OC_InitTypeDef

TIM_OC_InitTypeDef is defined in the stm32h7xx_hal_tim.h

Data Fields

- ***uint32_t OCMode***
- ***uint32_t Pulse***
- ***uint32_t OCPolarity***
- ***uint32_t OCNPolarity***
- ***uint32_t OCFastMode***
- ***uint32_t OCIdleState***
- ***uint32_t OCNIdleState***

Field Documentation

- ***uint32_t TIM_OC_InitTypeDef::OCMode***
  Specifies the TIM mode. This parameter can be a value of ***TIM_Output_Compare_and_PWM_modes***
- ***uint32_t TIM_OC_InitTypeDef::Pulse***
  Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF
- ***uint32_t TIM_OC_InitTypeDef::OCPolarity***
  Specifies the output polarity. This parameter can be a value of ***TIM_Output_Compare_Polarity***
- ***uint32_t TIM_OC_InitTypeDef::OCNPolarity***
  Specifies the complementary output polarity. This parameter can be a value of ***TIM_Output_Compare_N_Polarity***
  Note:
  – This parameter is valid only for timer instances supporting break feature.
- ***uint32_t TIM_OC_InitTypeDef::OCFastMode***
  Specifies the Fast mode state. This parameter can be a value of ***TIM_Output_Fast_State***
  Note:
  – This parameter is valid only in PWM1 and PWM2 mode.
- ***uint32_t TIM_OC_InitTypeDef::OCIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_Idle_State***
  Note:
  – This parameter is valid only for timer instances supporting break feature.
- ***uint32_t TIM_OC_InitTypeDef::OCNIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_N_Idle_State***
  Note:
  – This parameter is valid only for timer instances supporting break feature.

88.1.3 TIM_OnePulse_InitTypeDef

TIM_OnePulse_InitTypeDef is defined in the stm32h7xx_hal_tim.h

Data Fields

- ***uint32_t OCMode***
- ***uint32_t Pulse***
- ***uint32_t OCPolarity***
- ***uint32_t OCNPolarity***
- ***uint32_t OCIdleState***
- ***uint32_t OCNIdleState***
- ***uint32_t ICPolarity***
- ***uint32_t ICSelection***
- ***uint32_t ICFilter***

Field Documentation

- ***uint32_t TIM_OnePulse_InitTypeDef::OCMode***
  Specifies the TIM mode. This parameter can be a value of ***TIM_Output_Compare_and_PWM_modes***
- ***uint32_t TIM_OnePulse_InitTypeDef::Pulse***
  Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF
- ***uint32_t TIM_OnePulse_InitTypeDef::OCPolarity***
  Specifies the output polarity. This parameter can be a value of ***TIM_Output_Compare_Polarity***

- ***uint32_t TIM_OnePulse_InitTypeDef::OCNPolarity***
  Specifies the complementary output polarity. This parameter can be a value of ***TIM_Output_Compare_N_Polarity***
  Note:
  – This parameter is valid only for timer instances supporting break feature.
- ***uint32_t TIM_OnePulse_InitTypeDef::OCIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_Idle_State***
  Note:
  – This parameter is valid only for timer instances supporting break feature.
- ***uint32_t TIM_OnePulse_InitTypeDef::OCNIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_N_Idle_State***
  Note:
  – This parameter is valid only for timer instances supporting break feature.
- ***uint32_t TIM_OnePulse_InitTypeDef::ICPolarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Input_Capture_Polarity***
- ***uint32_t TIM_OnePulse_InitTypeDef::ICSelection***
  Specifies the input. This parameter can be a value of ***TIM_Input_Capture_Selection***
- ***uint32_t TIM_OnePulse_InitTypeDef::ICFilter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

88.1.4 TIM_IC_InitTypeDef

TIM_IC_InitTypeDef is defined in the stm32h7xx_hal_tim.h

Data Fields

- ***uint32_t ICPolarity***
- ***uint32_t ICSelection***
- ***uint32_t ICPrescaler***
- ***uint32_t ICFilter***

Field Documentation

- ***uint32_t TIM_IC_InitTypeDef::ICPolarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Input_Capture_Polarity***
- ***uint32_t TIM_IC_InitTypeDef::ICSelection***
  Specifies the input. This parameter can be a value of ***TIM_Input_Capture_Selection***
- ***uint32_t TIM_IC_InitTypeDef::ICPrescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_Input_Capture_Prescaler***
- ***uint32_t TIM_IC_InitTypeDef::ICFilter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

88.1.5 TIM_Encoder_InitTypeDef

TIM_Encoder_InitTypeDef is defined in the stm32h7xx_hal_tim.h

Data Fields

- ***uint32_t EncoderMode***
- ***uint32_t IC1Polarity***
- ***uint32_t IC1Selection***
- ***uint32_t IC1Prescaler***
- ***uint32_t IC1Filter***
- ***uint32_t IC2Polarity***

- ***uint32_t IC2Selection***
- ***uint32_t IC2Prescaler***
- ***uint32_t IC2Filter***

Field Documentation

- ***uint32_t TIM_Encoder_InitTypeDef::EncoderMode***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Encoder_Mode***
- ***uint32_t TIM_Encoder_InitTypeDef::IC1Polarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Encoder_Input_Polarity***
- ***uint32_t TIM_Encoder_InitTypeDef::IC1Selection***
  Specifies the input. This parameter can be a value of ***TIM_Input_Capture_Selection***
- ***uint32_t TIM_Encoder_InitTypeDef::IC1Prescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_Input_Capture_Prescaler***
- ***uint32_t TIM_Encoder_InitTypeDef::IC1Filter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- ***uint32_t TIM_Encoder_InitTypeDef::IC2Polarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Encoder_Input_Polarity***
- ***uint32_t TIM_Encoder_InitTypeDef::IC2Selection***
  Specifies the input. This parameter can be a value of ***TIM_Input_Capture_Selection***
- ***uint32_t TIM_Encoder_InitTypeDef::IC2Prescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_Input_Capture_Prescaler***
- ***uint32_t TIM_Encoder_InitTypeDef::IC2Filter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

88.1.6 TIM_ClockConfigTypeDef

TIM_ClockConfigTypeDef is defined in the stm32h7xx_hal_tim.h

Data Fields

- ***uint32_t ClockSource***
- ***uint32_t ClockPolarity***
- ***uint32_t ClockPrescaler***
- ***uint32_t ClockFilter***

Field Documentation

- ***uint32_t TIM_ClockConfigTypeDef::ClockSource***
  TIM clock sources This parameter can be a value of ***TIM_Clock_Source***
- ***uint32_t TIM_ClockConfigTypeDef::ClockPolarity***
  TIM clock polarity This parameter can be a value of ***TIM_Clock_Polarity***
- ***uint32_t TIM_ClockConfigTypeDef::ClockPrescaler***
  TIM clock prescaler This parameter can be a value of ***TIM_Clock_Prescaler***
- ***uint32_t TIM_ClockConfigTypeDef::ClockFilter***
  TIM clock filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

88.1.7 TIM_ClearInputConfigTypeDef

TIM_ClearInputConfigTypeDef is defined in the stm32h7xx_hal_tim.h

Data Fields

- ***uint32_t ClearInputState***
- ***uint32_t ClearInputSource***
- ***uint32_t ClearInputPolarity***

- ***uint32_t ClearInputPrescaler***
- ***uint32_t ClearInputFilter***

Field Documentation

- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputState***
  TIM clear Input state This parameter can be ENABLE or DISABLE
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputSource***
  TIM clear Input sources This parameter can be a value of ***TIM_ClearInput_Source***
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputPolarity***
  TIM Clear Input polarity This parameter can be a value of ***TIM_ClearInput_Polarity***
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputPrescaler***
  TIM Clear Input prescaler This parameter must be 0: When OCRef clear feature is used with ETR source, ETR prescaler must be off
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputFilter***
  TIM Clear Input filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

88.1.8 TIM_MasterConfigTypeDef

TIM_MasterConfigTypeDef is defined in the stm32h7xx_hal_tim.h

Data Fields

- ***uint32_t MasterOutputTrigger***
- ***uint32_t MasterOutputTrigger2***
- ***uint32_t MasterSlaveMode***

Field Documentation

- ***uint32_t TIM_MasterConfigTypeDef::MasterOutputTrigger***
  Trigger output (TRGO) selection This parameter can be a value of ***TIM_Master_Mode_Selection***
- ***uint32_t TIM_MasterConfigTypeDef::MasterOutputTrigger2***
  Trigger output2 (TRGO2) selection This parameter can be a value of ***TIM_Master_Mode_Selection_2***
- ***uint32_t TIM_MasterConfigTypeDef::MasterSlaveMode***
  Master/slave mode selection This parameter can be a value of ***TIM_Master_Slave_Mode***

  Note:

  – When the Master/slave mode is enabled, the effect of an event on the trigger input (TRGI) is delayed to allow a perfect synchronization between the current timer and its slaves (through TRGO). It is not mandatory in case of timer synchronization mode.

88.1.9 TIM_SlaveConfigTypeDef

TIM_SlaveConfigTypeDef is defined in the stm32h7xx_hal_tim.h

Data Fields

- ***uint32_t SlaveMode***
- ***uint32_t InputTrigger***
- ***uint32_t TriggerPolarity***
- ***uint32_t TriggerPrescaler***
- ***uint32_t TriggerFilter***

Field Documentation

- ***uint32_t TIM_SlaveConfigTypeDef::SlaveMode***
  Slave mode selection This parameter can be a value of ***TIM_Slave_Mode***
- ***uint32_t TIM_SlaveConfigTypeDef::InputTrigger***
  Input Trigger source This parameter can be a value of ***TIM_Trigger_Selection***
- ***uint32_t TIM_SlaveConfigTypeDef::TriggerPolarity***
  Input Trigger polarity This parameter can be a value of ***TIM_Trigger_Polarity***

- ***uint32_t TIM_SlaveConfigTypeDef::TriggerPrescaler***
  Input trigger prescaler This parameter can be a value of ***TIM_Trigger_Prescaler***
- ***uint32_t TIM_SlaveConfigTypeDef::TriggerFilter***
  Input trigger filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

88.1.10 TIM_BreakDeadTimeConfigTypeDef

TIM_BreakDeadTimeConfigTypeDef is defined in the stm32h7xx_hal_tim.h

Data Fields

- ***uint32_t OffStateRunMode***
- ***uint32_t OffStateIDLEMode***
- ***uint32_t LockLevel***
- ***uint32_t DeadTime***
- ***uint32_t BreakState***
- ***uint32_t BreakPolarity***
- ***uint32_t BreakFilter***
- ***uint32_t Break2State***
- ***uint32_t Break2Polarity***
- ***uint32_t Break2Filter***
- ***uint32_t AutomaticOutput***

Field Documentation

- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::OffStateRunMode***
  TIM off state in run mode, This parameter can be a value of ***TIM_OSSR_Off_State_Selection_for_Run_mode_state***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::OffStateIDLEMode***
  TIM off state in IDLE mode, This parameter can be a value of ***TIM_OSSI_Off_State_Selection_for_Idle_mode_state***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::LockLevel***
  TIM Lock level, This parameter can be a value of ***TIM_Lock_level***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::DeadTime***
  TIM dead Time, This parameter can be a number between Min_Data = 0x00 and Max_Data = 0xFF
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakState***
  TIM Break State, This parameter can be a value of ***TIM_Break_Input_enable_disable***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakPolarity***
  TIM Break input polarity, This parameter can be a value of ***TIM_Break_Polarity***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakFilter***
  Specifies the break input filter.This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::Break2State***
  TIM Break2 State, This parameter can be a value of ***TIM_Break2_Input_enable_disable***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::Break2Polarity***
  TIM Break2 input polarity, This parameter can be a value of ***TIM_Break2_Polarity***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::Break2Filter***
  TIM break2 input filter.This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::AutomaticOutput***
  TIM Automatic Output Enable state, This parameter can be a value of ***TIM_AOE_Bit_Set_Reset***

88.1.11 __TIM_HandleTypeDef

__TIM_HandleTypeDef is defined in the stm32h7xx_hal_tim.h

Data Fields

- ***TIM_TypeDef * Instance***

- *TIM_Base_InitTypeDef Init*
- *HAL_TIM_ActiveChannel Channel*
- *DMA_HandleTypeDef * hdma*
- *HAL_LockTypeDef Lock*
- *__IO HAL_TIM_StateTypeDef State*
- *__IO HAL_TIM_ChannelStateTypeDef ChannelState*
- *__IO HAL_TIM_ChannelStateTypeDef ChannelNState*
- *__IO HAL_TIM_DMABurstStateTypeDef DMABurstState*
- *void(* Base_MspInitCallback*
- *void(* Base_MspDeInitCallback*
- *void(* IC_MspInitCallback*
- *void(* IC_MspDeInitCallback*
- *void(* OC_MspInitCallback*
- *void(* OC_MspDeInitCallback*
- *void(* PWM_MspInitCallback*
- *void(* PWM_MspDeInitCallback*
- *void(* OnePulse_MspInitCallback*
- *void(* OnePulse_MspDeInitCallback*
- *void(* Encoder_MspInitCallback*
- *void(* Encoder_MspDeInitCallback*
- *void(* HallSensor_MspInitCallback*
- *void(* HallSensor_MspDeInitCallback*
- *void(* PeriodElapsedCallback*
- *void(* PeriodElapsedHalfCpltCallback*
- *void(* TriggerCallback*
- *void(* TriggerHalfCpltCallback*
- *void(* IC_CaptureCallback*
- *void(* IC_CaptureHalfCpltCallback*
- *void(* OC_DelayElapsedCallback*
- *void(* PWM_PulseFinishedCallback*
- *void(* PWM_PulseFinishedHalfCpltCallback*
- *void(* ErrorCallback*
- *void(* CommutationCallback*
- *void(* CommutationHalfCpltCallback*
- *void(* BreakCallback*
- *void(* Break2Callback*

Field Documentation

- ***TIM_TypeDef* __TIM_HandleTypeDef::Instance***
  Register base address
- ***TIM_Base_InitTypeDef __TIM_HandleTypeDef::Init***
  TIM Time Base required parameters
- ***HAL_TIM_ActiveChannel __TIM_HandleTypeDef::Channel***
  Active channel
- ***DMA_HandleTypeDef* __TIM_HandleTypeDef::hdma[7]***
  DMA Handlers array This array is accessed by a *DMA_Handle_index*
- ***HAL_LockTypeDef __TIM_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_TIM_StateTypeDef __TIM_HandleTypeDef::State***
  TIM operation state

- ***__IO HAL_TIM_ChannelStateTypeDef __TIM_HandleTypeDef::ChannelState[6]***
  TIM channel operation state
- ***__IO HAL_TIM_ChannelStateTypeDef __TIM_HandleTypeDef::ChannelNState[4]***
  TIM complementary channel operation state
- ***__IO HAL_TIM_DMABurstStateTypeDef __TIM_HandleTypeDef::DMABurstState***
  DMA burst operation state
- ***void(* __TIM_HandleTypeDef::Base_MspInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Base Msp Init Callback
- ***void(* __TIM_HandleTypeDef::Base_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Base Msp DeInit Callback
- ***void(* __TIM_HandleTypeDef::IC_MspInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM IC Msp Init Callback
- ***void(* __TIM_HandleTypeDef::IC_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM IC Msp DeInit Callback
- ***void(* __TIM_HandleTypeDef::OC_MspInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM OC Msp Init Callback
- ***void(* __TIM_HandleTypeDef::OC_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM OC Msp DeInit Callback
- ***void(* __TIM_HandleTypeDef::PWM_MspInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM PWM Msp Init Callback
- ***void(* __TIM_HandleTypeDef::PWM_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM PWM Msp DeInit Callback
- ***void(* __TIM_HandleTypeDef::OnePulse_MspInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM One Pulse Msp Init Callback
- ***void(* __TIM_HandleTypeDef::OnePulse_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM One Pulse Msp DeInit Callback
- ***void(* __TIM_HandleTypeDef::Encoder_MspInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Encoder Msp Init Callback
- ***void(* __TIM_HandleTypeDef::Encoder_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Encoder Msp DeInit Callback
- ***void(* __TIM_HandleTypeDef::HallSensor_MspInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Hall Sensor Msp Init Callback
- ***void(* __TIM_HandleTypeDef::HallSensor_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Hall Sensor Msp DeInit Callback
- ***void(* __TIM_HandleTypeDef::PeriodElapsedCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Period Elapsed Callback
- ***void(* __TIM_HandleTypeDef::PeriodElapsedHalfCpltCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Period Elapsed half complete Callback
- ***void(* __TIM_HandleTypeDef::TriggerCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Trigger Callback
- ***void(* __TIM_HandleTypeDef::TriggerHalfCpltCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Trigger half complete Callback
- ***void(* __TIM_HandleTypeDef::IC_CaptureCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Input Capture Callback
- ***void(* __TIM_HandleTypeDef::IC_CaptureHalfCpltCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Input Capture half complete Callback
- ***void(* __TIM_HandleTypeDef::OC_DelayElapsedCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Output Compare Delay Elapsed Callback
- ***void(* __TIM_HandleTypeDef::PWM_PulseFinishedCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM PWM Pulse Finished Callback

- ***void(* __TIM_HandleTypeDef::PWM_PulseFinishedHalfCpltCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM PWM Pulse Finished half complete Callback
- ***void(* __TIM_HandleTypeDef::ErrorCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Error Callback
- ***void(* __TIM_HandleTypeDef::CommutationCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Commutation Callback
- ***void(* __TIM_HandleTypeDef::CommutationHalfCpltCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Commutation half complete Callback
- ***void(* __TIM_HandleTypeDef::BreakCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Break Callback
- ***void(* __TIM_HandleTypeDef::Break2Callback)(struct __TIM_HandleTypeDef *htim)***
  TIM Break2 Callback

88.2 TIM Firmware driver API description

The following section lists the various functions of the TIM library.

88.2.1 TIMER Generic features

The Timer features include:

1. 16-bit up, down, up/down auto-reload counter.
2. 16-bit programmable prescaler allowing dividing (also on the fly) the counter clock frequency either by any factor between 1 and 65536.
3. Up to 4 independent channels for:
   – Input Capture
   – Output Compare
   – PWM generation (Edge and Center-aligned Mode)
   – One-pulse mode output
4. Synchronization circuit to control the timer with external signals and to interconnect several timers together.
5. Supports incremental encoder for positioning purposes

88.2.2 How to use this driver

1. Initialize the TIM low level resources by implementing the following functions depending on the selected feature:
   – Time Base : HAL_TIM_Base_MspInit()
   – Input Capture : HAL_TIM_IC_MspInit()
   – Output Compare : HAL_TIM_OC_MspInit()
   – PWM generation : HAL_TIM_PWM_MspInit()
   – One-pulse mode output : HAL_TIM_OnePulse_MspInit()
   – Encoder mode output : HAL_TIM_Encoder_MspInit()
2. Initialize the TIM low level resources :
   a. Enable the TIM interface clock using __HAL_RCC_TIMx_CLK_ENABLE();
   b. TIM pins configuration
      ◦ Enable the clock for the TIM GPIOs using the following function: __HAL_RCC_GPIOx_CLK_ENABLE();
      ◦ Configure these TIM pins in Alternate function mode using HAL_GPIO_Init();
3. The external Clock can be configured, if needed (the default clock is the internal clock from the APBx), using the following function: HAL_TIM_ConfigClockSource, the clock configuration should be done before any start function.

4. Configure the TIM in the desired functioning mode using one of the Initialization function of this driver:
   – HAL_TIM_Base_Init: to use the Timer to generate a simple time base
   – HAL_TIM_OC_Init and HAL_TIM_OC_ConfigChannel: to use the Timer to generate an Output Compare signal.
   – HAL_TIM_PWM_Init and HAL_TIM_PWM_ConfigChannel: to use the Timer to generate a PWM signal.
   – HAL_TIM_IC_Init and HAL_TIM_IC_ConfigChannel: to use the Timer to measure an external signal.
   – HAL_TIM_OnePulse_Init and HAL_TIM_OnePulse_ConfigChannel: to use the Timer in One Pulse Mode.
   – HAL_TIM_Encoder_Init: to use the Timer Encoder Interface.
5. Activate the TIM peripheral using one of the start functions depending from the feature used:
   – Time Base : HAL_TIM_Base_Start(), HAL_TIM_Base_Start_DMA(), HAL_TIM_Base_Start_IT()
   – Input Capture : HAL_TIM_IC_Start(), HAL_TIM_IC_Start_DMA(), HAL_TIM_IC_Start_IT()
   – Output Compare : HAL_TIM_OC_Start(), HAL_TIM_OC_Start_DMA(), HAL_TIM_OC_Start_IT()
   – PWM generation : HAL_TIM_PWM_Start(), HAL_TIM_PWM_Start_DMA(), HAL_TIM_PWM_Start_IT()
   – One-pulse mode output : HAL_TIM_OnePulse_Start(), HAL_TIM_OnePulse_Start_IT()
   – Encoder mode output : HAL_TIM_Encoder_Start(), HAL_TIM_Encoder_Start_DMA(), HAL_TIM_Encoder_Start_IT().
6. The DMA Burst is managed with the two following functions: HAL_TIM_DMABurst_WriteStart() HAL_TIM_DMABurst_ReadStart()

Callback registration

The compilation define USE_HAL_TIM_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_TIM_RegisterCallback() to register a callback. HAL_TIM_RegisterCallback() takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_TIM_UnRegisterCallback() to reset a callback to the default weak function. HAL_TIM_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID.

These functions allow to register/unregister following callbacks:

- Base_MspInitCallback : TIM Base Msp Init Callback.
- Base_MspDeInitCallback : TIM Base Msp DeInit Callback.
- IC_MspInitCallback : TIM IC Msp Init Callback.
- IC_MspDeInitCallback : TIM IC Msp DeInit Callback.
- OC_MspInitCallback : TIM OC Msp Init Callback.
- OC_MspDeInitCallback : TIM OC Msp DeInit Callback.
- PWM_MspInitCallback : TIM PWM Msp Init Callback.
- PWM_MspDeInitCallback : TIM PWM Msp DeInit Callback.
- OnePulse_MspInitCallback : TIM One Pulse Msp Init Callback.
- OnePulse_MspDeInitCallback : TIM One Pulse Msp DeInit Callback.
- Encoder_MspInitCallback : TIM Encoder Msp Init Callback.
- Encoder_MspDeInitCallback : TIM Encoder Msp DeInit Callback.
- HallSensor_MspInitCallback : TIM Hall Sensor Msp Init Callback.
- HallSensor_MspDeInitCallback : TIM Hall Sensor Msp DeInit Callback.
- PeriodElapsedCallback : TIM Period Elapsed Callback.
- PeriodElapsedHalfCpltCallback : TIM Period Elapsed half complete Callback.
- TriggerCallback : TIM Trigger Callback.
- TriggerHalfCpltCallback : TIM Trigger half complete Callback.
- IC_CaptureCallback : TIM Input Capture Callback.
- IC_CaptureHalfCpltCallback : TIM Input Capture half complete Callback.
- OC_DelayElapsedCallback : TIM Output Compare Delay Elapsed Callback.
- PWM_PulseFinishedCallback : TIM PWM Pulse Finished Callback.
- PWM_PulseFinishedHalfCpltCallback : TIM PWM Pulse Finished half complete Callback.

- ErrorCallback : TIM Error Callback.
- CommutationCallback : TIM Commutation Callback.
- CommutationHalfCpltCallback : TIM Commutation half complete Callback.
- BreakCallback : TIM Break Callback.
- Break2Callback : TIM Break2 Callback.

By default, after the Init and when the state is HAL_TIM_STATE_RESET all interrupt callbacks are set to the corresponding weak functions: examples HAL_TIM_TriggerCallback(), HAL_TIM_ErrorCallback().

Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functionalities in the Init / DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the Init / DeInit keep and use the user MspInit / MspDeInit callbacks(registered beforehand)

Callbacks can be registered / unregistered in HAL_TIM_STATE_READY state only. Exception done MspInit / MspDeInit that can be registered / unregistered in HAL_TIM_STATE_READY or HAL_TIM_STATE_RESET state, thus registered(user) MspInit / DeInit callbacks can be used during the Init / DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_TIM_RegisterCallback() before calling DeInit or Init function.

When The compilation define USE_HAL_TIM_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

88.2.3 Time Base functions

This section provides functions allowing to:

- Initialize and configure the TIM base.
- De-initialize the TIM base.
- Start the Time Base.
- Stop the Time Base.
- Start the Time Base and enable interrupt.
- Stop the Time Base and disable interrupt.
- Start the Time Base and enable DMA transfer.
- Stop the Time Base and disable DMA transfer.

This section contains the following APIs:

- ***HAL_TIM_Base_Init()***
- ***HAL_TIM_Base_DeInit()***
- ***HAL_TIM_Base_MspInit()***
- ***HAL_TIM_Base_MspDeInit()***
- ***HAL_TIM_Base_Start()***
- ***HAL_TIM_Base_Stop()***
- ***HAL_TIM_Base_Start_IT()***
- ***HAL_TIM_Base_Stop_IT()***
- ***HAL_TIM_Base_Start_DMA()***
- ***HAL_TIM_Base_Stop_DMA()***

88.2.4 TIM Output Compare functions

This section provides functions allowing to:

- Initialize and configure the TIM Output Compare.
- De-initialize the TIM Output Compare.
- Start the TIM Output Compare.
- Stop the TIM Output Compare.
- Start the TIM Output Compare and enable interrupt.
- Stop the TIM Output Compare and disable interrupt.
- Start the TIM Output Compare and enable DMA transfer.
- Stop the TIM Output Compare and disable DMA transfer.

This section contains the following APIs:

- *HAL_TIM_OC_Init()*
- *HAL_TIM_OC_DeInit()*
- *HAL_TIM_OC_MspInit()*
- *HAL_TIM_OC_MspDeInit()*
- *HAL_TIM_OC_Start()*
- *HAL_TIM_OC_Stop()*
- *HAL_TIM_OC_Start_IT()*
- *HAL_TIM_OC_Stop_IT()*
- *HAL_TIM_OC_Start_DMA()*
- *HAL_TIM_OC_Stop_DMA()*

88.2.5 TIM PWM functions

This section provides functions allowing to:

- Initialize and configure the TIM PWM.
- De-initialize the TIM PWM.
- Start the TIM PWM.
- Stop the TIM PWM.
- Start the TIM PWM and enable interrupt.
- Stop the TIM PWM and disable interrupt.
- Start the TIM PWM and enable DMA transfer.
- Stop the TIM PWM and disable DMA transfer.

This section contains the following APIs:

- *HAL_TIM_PWM_Init()*
- *HAL_TIM_PWM_DeInit()*
- *HAL_TIM_PWM_MspInit()*
- *HAL_TIM_PWM_MspDeInit()*
- *HAL_TIM_PWM_Start()*
- *HAL_TIM_PWM_Stop()*
- *HAL_TIM_PWM_Start_IT()*
- *HAL_TIM_PWM_Stop_IT()*
- *HAL_TIM_PWM_Start_DMA()*
- *HAL_TIM_PWM_Stop_DMA()*

88.2.6 TIM Input Capture functions

This section provides functions allowing to:

- Initialize and configure the TIM Input Capture.
- De-initialize the TIM Input Capture.
- Start the TIM Input Capture.
- Stop the TIM Input Capture.
- Start the TIM Input Capture and enable interrupt.
- Stop the TIM Input Capture and disable interrupt.
- Start the TIM Input Capture and enable DMA transfer.
- Stop the TIM Input Capture and disable DMA transfer.

This section contains the following APIs:

- *HAL_TIM_IC_Init()*
- *HAL_TIM_IC_DeInit()*
- *HAL_TIM_IC_MspInit()*
- *HAL_TIM_IC_MspDeInit()*
- *HAL_TIM_IC_Start()*

- *HAL_TIM_IC_Stop()*
- *HAL_TIM_IC_Start_IT()*
- *HAL_TIM_IC_Stop_IT()*
- *HAL_TIM_IC_Start_DMA()*
- *HAL_TIM_IC_Stop_DMA()*

88.2.7 TIM One Pulse functions

This section provides functions allowing to:

- Initialize and configure the TIM One Pulse.
- De-initialize the TIM One Pulse.
- Start the TIM One Pulse.
- Stop the TIM One Pulse.
- Start the TIM One Pulse and enable interrupt.
- Stop the TIM One Pulse and disable interrupt.
- Start the TIM One Pulse and enable DMA transfer.
- Stop the TIM One Pulse and disable DMA transfer.

This section contains the following APIs:

- *HAL_TIM_OnePulse_Init()*
- *HAL_TIM_OnePulse_DeInit()*
- *HAL_TIM_OnePulse_MspInit()*
- *HAL_TIM_OnePulse_MspDeInit()*
- *HAL_TIM_OnePulse_Start()*
- *HAL_TIM_OnePulse_Stop()*
- *HAL_TIM_OnePulse_Start_IT()*
- *HAL_TIM_OnePulse_Stop_IT()*

88.2.8 TIM Encoder functions

This section provides functions allowing to:

- Initialize and configure the TIM Encoder.
- De-initialize the TIM Encoder.
- Start the TIM Encoder.
- Stop the TIM Encoder.
- Start the TIM Encoder and enable interrupt.
- Stop the TIM Encoder and disable interrupt.
- Start the TIM Encoder and enable DMA transfer.
- Stop the TIM Encoder and disable DMA transfer.

This section contains the following APIs:

- *HAL_TIM_Encoder_Init()*
- *HAL_TIM_Encoder_DeInit()*
- *HAL_TIM_Encoder_MspInit()*
- *HAL_TIM_Encoder_MspDeInit()*
- *HAL_TIM_Encoder_Start()*
- *HAL_TIM_Encoder_Stop()*
- *HAL_TIM_Encoder_Start_IT()*
- *HAL_TIM_Encoder_Stop_IT()*
- *HAL_TIM_Encoder_Start_DMA()*
- *HAL_TIM_Encoder_Stop_DMA()*

88.2.9 TIM Callbacks functions

This section provides TIM callback functions:

- TIM Period elapsed callback
- TIM Output Compare callback
- TIM Input capture callback
- TIM Trigger callback
- TIM Error callback

This section contains the following APIs:

- *HAL_TIM_PeriodElapsedCallback()*
- *HAL_TIM_PeriodElapsedHalfCpltCallback()*
- *HAL_TIM_OC_DelayElapsedCallback()*
- *HAL_TIM_IC_CaptureCallback()*
- *HAL_TIM_IC_CaptureHalfCpltCallback()*
- *HAL_TIM_PWM_PulseFinishedCallback()*
- *HAL_TIM_PWM_PulseFinishedHalfCpltCallback()*
- *HAL_TIM_TriggerCallback()*
- *HAL_TIM_TriggerHalfCpltCallback()*
- *HAL_TIM_ErrorCallback()*
- *HAL_TIM_RegisterCallback()*
- *HAL_TIM_UnRegisterCallback()*

88.2.10 Detailed description of functions

HAL_TIM_Base_Init

Function name

HAL_StatusTypeDef HAL_TIM_Base_Init (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Time base Unit according to the specified parameters in the TIM_HandleTypeDef and initialize the associated handle.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** status

Notes

- Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode. Ex: call HAL_TIM_Base_DeInit() before HAL_TIM_Base_Init()

HAL_TIM_Base_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_Base_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes the TIM Base peripheral.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** status

HAL_TIM_Base_MspInit

Function name

void HAL_TIM_Base_MspInit (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Base MSP.

Parameters

- **htim:** TIM Base handle

Return values

- **None:**

HAL_TIM_Base_MspDeInit

Function name

void HAL_TIM_Base_MspDeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes TIM Base MSP.

Parameters

- **htim:** TIM Base handle

Return values

- **None:**

HAL_TIM_Base_Start

Function name

HAL_StatusTypeDef HAL_TIM_Base_Start (TIM_HandleTypeDef * htim)

Function description

Starts the TIM Base generation.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** status

HAL_TIM_Base_Stop

Function name

HAL_StatusTypeDef HAL_TIM_Base_Stop (TIM_HandleTypeDef * htim)

Function description

Stops the TIM Base generation.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** status

HAL_TIM_Base_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_Base_Start_IT (TIM_HandleTypeDef * htim)

Function description

Starts the TIM Base generation in interrupt mode.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** status

HAL_TIM_Base_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_Base_Stop_IT (TIM_HandleTypeDef * htim)

Function description

Stops the TIM Base generation in interrupt mode.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** status

HAL_TIM_Base_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIM_Base_Start_DMA (TIM_HandleTypeDef * htim, const uint32_t * pData, uint16_t Length)

Function description

Starts the TIM Base generation in DMA mode.

Parameters

- **htim:** TIM Base handle
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to peripheral.

Return values

- **HAL:** status

HAL_TIM_Base_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIM_Base_Stop_DMA (TIM_HandleTypeDef * htim)

Function description

Stops the TIM Base generation in DMA mode.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** status

HAL_TIM_OC_Init

Function name

HAL_StatusTypeDef HAL_TIM_OC_Init (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Output Compare according to the specified parameters in the TIM_HandleTypeDef and initializes the associated handle.

Parameters

- **htim:** TIM Output Compare handle

Return values

- **HAL:** status

Notes

- Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode. Ex: call HAL_TIM_OC_DeInit() before HAL_TIM_OC_Init()

HAL_TIM_OC_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_OC_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes the TIM peripheral.

Parameters

- **htim:** TIM Output Compare handle

Return values

- **HAL:** status

HAL_TIM_OC_MspInit

Function name

void HAL_TIM_OC_MspInit (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Output Compare MSP.

Parameters

- **htim:** TIM Output Compare handle

Return values

- **None:**

HAL_TIM_OC_MspDeInit

Function name

void HAL_TIM_OC_MspDeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes TIM Output Compare MSP.

Parameters

- **htim:** TIM Output Compare handle

Return values

- **None:**

HAL_TIM_OC_Start

Function name

HAL_StatusTypeDef HAL_TIM_OC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Output Compare signal generation.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return values

- **HAL:** status

HAL_TIM_OC_Stop

Function name

HAL_StatusTypeDef HAL_TIM_OC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Output Compare signal generation.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return values

- **HAL:** status

HAL_TIM_OC_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_OC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Output Compare signal generation in interrupt mode.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_OC_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_OC_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Output Compare signal generation in interrupt mode.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_OC_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIM_OC_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, const uint32_t * pData, uint16_t Length)

Function description

Starts the TIM Output Compare signal generation in DMA mode.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to TIM peripheral

Return values

- **HAL:** status

HAL_TIM_OC_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIM_OC_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Output Compare signal generation in DMA mode.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_PWM_Init

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Init (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM PWM Time Base according to the specified parameters in the TIM_HandleTypeDef and initializes the associated handle.

Parameters

- **htim:** TIM PWM handle

Return values

- **HAL:** status

Notes

- Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode. Ex: call HAL_TIM_PWM_DeInit() before HAL_TIM_PWM_Init()

HAL_TIM_PWM_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_PWM_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes the TIM peripheral.

Parameters

- **htim:** TIM PWM handle

Return values

- **HAL:** status

HAL_TIM_PWM_MspInit

Function name

void HAL_TIM_PWM_MspInit (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM PWM MSP.

Parameters

- **htim:** TIM PWM handle

Return values

- **None:**

HAL_TIM_PWM_MspDeInit

Function name

void HAL_TIM_PWM_MspDeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes TIM PWM MSP.

Parameters

- **htim:** TIM PWM handle

Return values

- **None:**

HAL_TIM_PWM_Start

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the PWM signal generation.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return values

- **HAL:** status

HAL_TIM_PWM_Stop

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the PWM signal generation.

Parameters

- **htim:** TIM PWM handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return values

- **HAL:** status

HAL_TIM_PWM_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the PWM signal generation in interrupt mode.

Parameters

- **htim:** TIM PWM handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_PWM_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the PWM signal generation in interrupt mode.

Parameters

- **htim:** TIM PWM handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_PWM_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, const uint32_t * pData, uint16_t Length)

Function description

Starts the TIM PWM signal generation in DMA mode.

Parameters

- **htim:** TIM PWM handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to TIM peripheral

Return values

- **HAL:** status

HAL_TIM_PWM_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM PWM signal generation in DMA mode.

Parameters

- **htim:** TIM PWM handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_IC_Init

Function name

HAL_StatusTypeDef HAL_TIM_IC_Init (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Input Capture Time base according to the specified parameters in the TIM_HandleTypeDef and initializes the associated handle.

Parameters

- **htim:** TIM Input Capture handle

Return values

- **HAL:** status

Notes

- Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode. Ex: call HAL_TIM_IC_DeInit() before HAL_TIM_IC_Init()

HAL_TIM_IC_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_IC_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes the TIM peripheral.

Parameters

- **htim:** TIM Input Capture handle

Return values

- **HAL:** status

HAL_TIM_IC_MspInit

Function name

void HAL_TIM_IC_MspInit (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Input Capture MSP.

Parameters

- **htim:** TIM Input Capture handle

Return values

- **None:**

HAL_TIM_IC_MspDeInit

Function name

void HAL_TIM_IC_MspDeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes TIM Input Capture MSP.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_IC_Start

Function name

HAL_StatusTypeDef HAL_TIM_IC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Input Capture measurement.

Parameters

- **htim:** TIM Input Capture handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_IC_Stop

Function name

HAL_StatusTypeDef HAL_TIM_IC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Input Capture measurement.

Parameters

- **htim:** TIM Input Capture handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_IC_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_IC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Input Capture measurement in interrupt mode.

Parameters

- **htim:** TIM Input Capture handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_IC_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_IC_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Input Capture measurement in interrupt mode.

Parameters

- **htim:** TIM Input Capture handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_IC_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIM_IC_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)

Function description

Starts the TIM Input Capture measurement in DMA mode.

Parameters

- **htim:** TIM Input Capture handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- **pData:** The destination Buffer address.
- **Length:** The length of data to be transferred from TIM peripheral to memory.

Return values

- **HAL:** status

HAL_TIM_IC_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIM_IC_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Input Capture measurement in DMA mode.

Parameters

- **htim:** TIM Input Capture handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_OnePulse_Init

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_Init (TIM_HandleTypeDef * htim, uint32_t OnePulseMode)

Function description

Initializes the TIM One Pulse Time Base according to the specified parameters in the TIM_HandleTypeDef and initializes the associated handle.

Parameters

- **htim:** TIM One Pulse handle
- **OnePulseMode:** Select the One pulse mode. This parameter can be one of the following values:
  - TIM_OPMODE_SINGLE: Only one pulse will be generated.
  - TIM_OPMODE_REPETITIVE: Repetitive pulses will be generated.

Return values

- **HAL:** status

Notes

- Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode. Ex: call HAL_TIM_OnePulse_DeInit() before HAL_TIM_OnePulse_Init()
- When the timer instance is initialized in One Pulse mode, timer channels 1 and channel 2 are reserved and cannot be used for other purpose.

HAL_TIM_OnePulse_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes the TIM One Pulse.

Parameters

- **htim:** TIM One Pulse handle

Return values

- **HAL:** status

HAL_TIM_OnePulse_MspInit

Function name

void HAL_TIM_OnePulse_MspInit (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM One Pulse MSP.

Parameters

- **htim:** TIM One Pulse handle

Return values

- **None:**

HAL_TIM_OnePulse_MspDeInit

Function name

void HAL_TIM_OnePulse_MspDeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes TIM One Pulse MSP.

Parameters

- **htim:** TIM One Pulse handle

Return values

- **None:**

HAL_TIM_OnePulse_Start

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_Start (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Starts the TIM One Pulse signal generation.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** See note above

Return values

- **HAL:** status

Notes

- Though OutputChannel parameter is deprecated and ignored by the function it has been kept to avoid HAL_TIM API compatibility break.
- The pulse output channel is determined when calling HAL_TIM_OnePulse_ConfigChannel().

HAL_TIM_OnePulse_Stop

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_Stop (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Stops the TIM One Pulse signal generation.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** See note above

Return values

- **HAL:** status

Notes

- Though OutputChannel parameter is deprecated and ignored by the function it has been kept to avoid HAL_TIM API compatibility break.
- The pulse output channel is determined when calling HAL_TIM_OnePulse_ConfigChannel().

HAL_TIM_OnePulse_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_Start_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Starts the TIM One Pulse signal generation in interrupt mode.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** See note above

Return values

- **HAL:** status

Notes

- Though OutputChannel parameter is deprecated and ignored by the function it has been kept to avoid HAL_TIM API compatibility break.
- The pulse output channel is determined when calling HAL_TIM_OnePulse_ConfigChannel().

HAL_TIM_OnePulse_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_Stop_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Stops the TIM One Pulse signal generation in interrupt mode.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** See note above

Return values

- **HAL:** status

Notes

- Though OutputChannel parameter is deprecated and ignored by the function it has been kept to avoid HAL_TIM API compatibility break.
- The pulse output channel is determined when calling HAL_TIM_OnePulse_ConfigChannel().

HAL_TIM_Encoder_Init

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Init (TIM_HandleTypeDef * htim, TIM_Encoder_InitTypeDef * sConfig)

Function description

Initializes the TIM Encoder Interface and initialize the associated handle.

Parameters

- **htim:** TIM Encoder Interface handle
- **sConfig:** TIM Encoder Interface configuration structure

Return values

- **HAL:** status

Notes

- Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode. Ex: call HAL_TIM_Encoder_DeInit() before HAL_TIM_Encoder_Init()
- Encoder mode and External clock mode 2 are not compatible and must not be selected together Ex: A call for HAL_TIM_Encoder_Init will erase the settings of HAL_TIM_ConfigClockSource using TIM_CLOCKSOURCE_ETRMODE2 and vice versa
- When the timer instance is initialized in Encoder mode, timer channels 1 and channel 2 are reserved and cannot be used for other purpose.

HAL_TIM_Encoder_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes the TIM Encoder interface.

Parameters

- **htim:** TIM Encoder Interface handle

Return values

- **HAL:** status

HAL_TIM_Encoder_MspInit

Function name

void HAL_TIM_Encoder_MspInit (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Encoder Interface MSP.

Parameters

- **htim:** TIM Encoder Interface handle

Return values

- **None:**

HAL_TIM_Encoder_MspDeInit

Function name

void HAL_TIM_Encoder_MspDeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes TIM Encoder Interface MSP.

Parameters

- **htim:** TIM Encoder Interface handle

Return values

- **None:**

HAL_TIM_Encoder_Start

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Encoder Interface.

Parameters

- **htim:** TIM Encoder Interface handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values

- **HAL:** status

HAL_TIM_Encoder_Stop

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Encoder Interface.

Parameters

- **htim:** TIM Encoder Interface handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values

- **HAL:** status

HAL_TIM_Encoder_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Encoder Interface in interrupt mode.

Parameters

- **htim:** TIM Encoder Interface handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values

- **HAL:** status

HAL_TIM_Encoder_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Encoder Interface in interrupt mode.

Parameters

- **htim:** TIM Encoder Interface handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values

- **HAL:** status

HAL_TIM_Encoder_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData1, uint32_t * pData2, uint16_t Length)

Function description

Starts the TIM Encoder Interface in DMA mode.

Parameters

- **htim:** TIM Encoder Interface handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected
- **pData1:** The destination Buffer address for IC1.
- **pData2:** The destination Buffer address for IC2.
- **Length:** The length of data to be transferred from TIM peripheral to memory.

Return values

- **HAL:** status

HAL_TIM_Encoder_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Encoder Interface in DMA mode.

Parameters

- **htim:** TIM Encoder Interface handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values

- **HAL:** status

HAL_TIM_IRQHandler

Function name

void HAL_TIM_IRQHandler (TIM_HandleTypeDef * htim)

Function description

This function handles TIM interrupts requests.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_OC_ConfigChannel

Function name

HAL_StatusTypeDef HAL_TIM_OC_ConfigChannel (TIM_HandleTypeDef * htim, const TIM_OC_InitTypeDef * sConfig, uint32_t Channel)

Function description

Initializes the TIM Output Compare Channels according to the specified parameters in the TIM_OC_InitTypeDef.

Parameters

- **htim:** TIM Output Compare handle
- **sConfig:** TIM Output Compare configuration structure
- **Channel:** TIM Channels to configure This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return values

- **HAL:** status

HAL_TIM_PWM_ConfigChannel

Function name

HAL_StatusTypeDef HAL_TIM_PWM_ConfigChannel (TIM_HandleTypeDef * htim, const TIM_OC_InitTypeDef * sConfig, uint32_t Channel)

Function description

Initializes the TIM PWM channels according to the specified parameters in the TIM_OC_InitTypeDef.

Parameters

- **htim:** TIM PWM handle
- **sConfig:** TIM PWM configuration structure
- **Channel:** TIM Channels to be configured This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return values

- **HAL:** status

HAL_TIM_IC_ConfigChannel

Function name

HAL_StatusTypeDef HAL_TIM_IC_ConfigChannel (TIM_HandleTypeDef * htim, const TIM_IC_InitTypeDef * sConfig, uint32_t Channel)

Function description

Initializes the TIM Input Capture Channels according to the specified parameters in the TIM_IC_InitTypeDef.

Parameters

- **htim:** TIM IC handle
- **sConfig:** TIM Input Capture configuration structure
- **Channel:** TIM Channel to configure This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_OnePulse_ConfigChannel

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OnePulse_InitTypeDef * sConfig, uint32_t OutputChannel, uint32_t InputChannel)

Function description

Initializes the TIM One Pulse Channels according to the specified parameters in the TIM_OnePulse_InitTypeDef.

Parameters

- **htim:** TIM One Pulse handle
- **sConfig:** TIM One Pulse configuration structure
- **OutputChannel:** TIM output channel to configure This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
- **InputChannel:** TIM input Channel to configure This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

Notes

- To output a waveform with a minimum delay user can enable the fast mode by calling the __HAL_TIM_ENABLE_OCxFAST macro. Then CCx output is forced in response to the edge detection on TIx input, without taking in account the comparison.

HAL_TIM_ConfigOCrefClear

Function name

HAL_StatusTypeDef HAL_TIM_ConfigOCrefClear (TIM_HandleTypeDef * htim, const TIM_ClearInputConfigTypeDef * sClearInputConfig, uint32_t Channel)

Function description

Configures the OCRef clear feature.

Parameters

- **htim:** TIM handle
- **sClearInputConfig:** pointer to a TIM_ClearInputConfigTypeDef structure that contains the OCREF clear feature and parameters for the TIM peripheral.
- **Channel:** specifies the TIM Channel This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1
  - TIM_CHANNEL_2: TIM Channel 2
  - TIM_CHANNEL_3: TIM Channel 3
  - TIM_CHANNEL_4: TIM Channel 4
  - TIM_CHANNEL_5: TIM Channel 5
  - TIM_CHANNEL_6: TIM Channel 6

Return values

- **HAL:** status

HAL_TIM_ConfigClockSource

Function name

HAL_StatusTypeDef HAL_TIM_ConfigClockSource (TIM_HandleTypeDef * htim, const TIM_ClockConfigTypeDef * sClockSourceConfig)

Function description

Configures the clock source to be used.

Parameters

- **htim:** TIM handle
- **sClockSourceConfig:** pointer to a TIM_ClockConfigTypeDef structure that contains the clock source information for the TIM peripheral.

Return values

- **HAL:** status

HAL_TIM_ConfigTI1Input

Function name

HAL_StatusTypeDef HAL_TIM_ConfigTI1Input (TIM_HandleTypeDef * htim, uint32_t TI1_Selection)

Function description

Selects the signal connected to the TI1 input: direct from CH1_input or a XOR combination between CH1_input, CH2_input & CH3_input.

Parameters

- **htim:** TIM handle.
- **TI1_Selection:** Indicate whether or not channel 1 is connected to the output of a XOR gate. This parameter can be one of the following values:
  - TIM_TI1SELECTION_CH1: The TIMx_CH1 pin is connected to TI1 input
  - TIM_TI1SELECTION_XORCOMBINATION: The TIMx_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)

Return values

- **HAL:** status

HAL_TIM_SlaveConfigSynchro

Function name

HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchro (TIM_HandleTypeDef * htim, const TIM_SlaveConfigTypeDef * sSlaveConfig)

Function description

Configures the TIM in Slave mode.

Parameters

- **htim:** TIM handle.
- **sSlaveConfig:** pointer to a TIM_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1).

Return values

- **HAL:** status

HAL_TIM_SlaveConfigSynchro_IT

Function name

HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchro_IT (TIM_HandleTypeDef * htim, const TIM_SlaveConfigTypeDef * sSlaveConfig)

Function description

Configures the TIM in Slave mode in interrupt mode.

Parameters

- **htim:** TIM handle.
- **sSlaveConfig:** pointer to a TIM_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1).

Return values

- **HAL:** status

HAL_TIM_DMABurst_WriteStart

Function name

HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, const uint32_t * BurstBuffer, uint32_t BurstLength)

Function description

Configure the DMA Burst to transfer Data from the memory to the TIM peripheral.

Parameters

- **htim:** TIM handle
- **BurstBaseAddress:** TIM Base address from where the DMA will start the Data write This parameter can be one of the following values:
  - TIM_DMABASE_CR1
  - TIM_DMABASE_CR2
  - TIM_DMABASE_SMCR
  - TIM_DMABASE_DIER
  - TIM_DMABASE_SR
  - TIM_DMABASE_EGR
  - TIM_DMABASE_CCMR1
  - TIM_DMABASE_CCMR2
  - TIM_DMABASE_CCER
  - TIM_DMABASE_CNT
  - TIM_DMABASE_PSC
  - TIM_DMABASE_ARR
  - TIM_DMABASE_RCR
  - TIM_DMABASE_CCR1
  - TIM_DMABASE_CCR2
  - TIM_DMABASE_CCR3
  - TIM_DMABASE_CCR4
  - TIM_DMABASE_BDTR
  - TIM_DMABASE_CCMR3
  - TIM_DMABASE_CCR5
  - TIM_DMABASE_CCR6
  - TIM_DMABASE_AF1
  - TIM_DMABASE_AF2
  - TIM_DMABASE_TISEL
- **BurstRequestSrc:** TIM DMA Request sources This parameter can be one of the following values:
  - TIM_DMA_UPDATE: TIM update Interrupt source
  - TIM_DMA_CC1: TIM Capture Compare 1 DMA source
  - TIM_DMA_CC2: TIM Capture Compare 2 DMA source
  - TIM_DMA_CC3: TIM Capture Compare 3 DMA source
  - TIM_DMA_CC4: TIM Capture Compare 4 DMA source
  - TIM_DMA_COM: TIM Commutation DMA source
  - TIM_DMA_TRIGGER: TIM Trigger DMA source
- **BurstBuffer:** The Buffer address.
- **BurstLength:** DMA Burst length. This parameter can be one value between: TIM_DMABURSTLENGTH_1TRANSFER and TIM_DMABURSTLENGTH_18TRANSFERS.

Return values

- **HAL:** status

Notes

- This function should be used only when BurstLength is equal to DMA data transfer length.

HAL_TIM_DMABurst_MultiWriteStart

Function name

HAL_StatusTypeDef HAL_TIM_DMABurst_MultiWriteStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, const uint32_t * BurstBuffer, uint32_t BurstLength, uint32_t DataLength)

Function description

Configure the DMA Burst to transfer multiple Data from the memory to the TIM peripheral.

Parameters

- **htim:** TIM handle
- **BurstBaseAddress:** TIM Base address from where the DMA will start the Data write This parameter can be one of the following values:
  - TIM_DMABASE_CR1
  - TIM_DMABASE_CR2
  - TIM_DMABASE_SMCR
  - TIM_DMABASE_DIER
  - TIM_DMABASE_SR
  - TIM_DMABASE_EGR
  - TIM_DMABASE_CCMR1
  - TIM_DMABASE_CCMR2
  - TIM_DMABASE_CCER
  - TIM_DMABASE_CNT
  - TIM_DMABASE_PSC
  - TIM_DMABASE_ARR
  - TIM_DMABASE_RCR
  - TIM_DMABASE_CCR1
  - TIM_DMABASE_CCR2
  - TIM_DMABASE_CCR3
  - TIM_DMABASE_CCR4
  - TIM_DMABASE_BDTR
  - TIM_DMABASE_CCMR3
  - TIM_DMABASE_CCR5
  - TIM_DMABASE_CCR6
  - TIM_DMABASE_AF1
  - TIM_DMABASE_AF2
  - TIM_DMABASE_TISEL
- **BurstRequestSrc:** TIM DMA Request sources This parameter can be one of the following values:
  - TIM_DMA_UPDATE: TIM update Interrupt source
  - TIM_DMA_CC1: TIM Capture Compare 1 DMA source
  - TIM_DMA_CC2: TIM Capture Compare 2 DMA source
  - TIM_DMA_CC3: TIM Capture Compare 3 DMA source
  - TIM_DMA_CC4: TIM Capture Compare 4 DMA source
  - TIM_DMA_COM: TIM Commutation DMA source
  - TIM_DMA_TRIGGER: TIM Trigger DMA source
- **BurstBuffer:** The Buffer address.
- **BurstLength:** DMA Burst length. This parameter can be one value between: TIM_DMABURSTLENGTH_1TRANSFER and TIM_DMABURSTLENGTH_18TRANSFERS.
- **DataLength:** Data length. This parameter can be one value between 1 and 0xFFFF.

Return values

- **HAL:** status

HAL_TIM_DMABurst_WriteStop

Function name

HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStop (TIM_HandleTypeDef * htim, uint32_t BurstRequestSrc)

Function description

Stops the TIM DMA Burst mode.

Parameters

- **htim:** TIM handle
- **BurstRequestSrc:** TIM DMA Request sources to disable

Return values

- **HAL:** status

HAL_TIM_DMABurst_ReadStart

Function name

HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t BurstLength)

Function description

Configure the DMA Burst to transfer Data from the TIM peripheral to the memory.

Parameters

- **htim:** TIM handle
- **BurstBaseAddress:** TIM Base address from where the DMA will start the Data read This parameter can be one of the following values:
  - TIM_DMABASE_CR1
  - TIM_DMABASE_CR2
  - TIM_DMABASE_SMCR
  - TIM_DMABASE_DIER
  - TIM_DMABASE_SR
  - TIM_DMABASE_EGR
  - TIM_DMABASE_CCMR1
  - TIM_DMABASE_CCMR2
  - TIM_DMABASE_CCER
  - TIM_DMABASE_CNT
  - TIM_DMABASE_PSC
  - TIM_DMABASE_ARR
  - TIM_DMABASE_RCR
  - TIM_DMABASE_CCR1
  - TIM_DMABASE_CCR2
  - TIM_DMABASE_CCR3
  - TIM_DMABASE_CCR4
  - TIM_DMABASE_BDTR
  - TIM_DMABASE_CCMR3
  - TIM_DMABASE_CCR5
  - TIM_DMABASE_CCR6
  - TIM_DMABASE_AF1
  - TIM_DMABASE_AF2
  - TIM_DMABASE_TISEL
- **BurstRequestSrc:** TIM DMA Request sources This parameter can be one of the following values:
  - TIM_DMA_UPDATE: TIM update Interrupt source
  - TIM_DMA_CC1: TIM Capture Compare 1 DMA source
  - TIM_DMA_CC2: TIM Capture Compare 2 DMA source
  - TIM_DMA_CC3: TIM Capture Compare 3 DMA source
  - TIM_DMA_CC4: TIM Capture Compare 4 DMA source
  - TIM_DMA_COM: TIM Commutation DMA source
  - TIM_DMA_TRIGGER: TIM Trigger DMA source
- **BurstBuffer:** The Buffer address.
- **BurstLength:** DMA Burst length. This parameter can be one value between: TIM_DMABURSTLENGTH_1TRANSFER and TIM_DMABURSTLENGTH_18TRANSFERS.

Return values

- **HAL:** status

Notes

- This function should be used only when BurstLength is equal to DMA data transfer length.

HAL_TIM_DMABurst_MultiReadStart

Function name

HAL_StatusTypeDef HAL_TIM_DMABurst_MultiReadStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t BurstLength, uint32_t DataLength)

Function description

Configure the DMA Burst to transfer Data from the TIM peripheral to the memory.

Parameters

- **htim:** TIM handle
- **BurstBaseAddress:** TIM Base address from where the DMA will start the Data read This parameter can be one of the following values:
  - TIM_DMABASE_CR1
  - TIM_DMABASE_CR2
  - TIM_DMABASE_SMCR
  - TIM_DMABASE_DIER
  - TIM_DMABASE_SR
  - TIM_DMABASE_EGR
  - TIM_DMABASE_CCMR1
  - TIM_DMABASE_CCMR2
  - TIM_DMABASE_CCER
  - TIM_DMABASE_CNT
  - TIM_DMABASE_PSC
  - TIM_DMABASE_ARR
  - TIM_DMABASE_RCR
  - TIM_DMABASE_CCR1
  - TIM_DMABASE_CCR2
  - TIM_DMABASE_CCR3
  - TIM_DMABASE_CCR4
  - TIM_DMABASE_BDTR
  - TIM_DMABASE_CCMR3
  - TIM_DMABASE_CCR5
  - TIM_DMABASE_CCR6
  - TIM_DMABASE_AF1
  - TIM_DMABASE_AF2
  - TIM_DMABASE_TISEL
- **BurstRequestSrc:** TIM DMA Request sources This parameter can be one of the following values:
  - TIM_DMA_UPDATE: TIM update Interrupt source
  - TIM_DMA_CC1: TIM Capture Compare 1 DMA source
  - TIM_DMA_CC2: TIM Capture Compare 2 DMA source
  - TIM_DMA_CC3: TIM Capture Compare 3 DMA source
  - TIM_DMA_CC4: TIM Capture Compare 4 DMA source
  - TIM_DMA_COM: TIM Commutation DMA source
  - TIM_DMA_TRIGGER: TIM Trigger DMA source
- **BurstBuffer:** The Buffer address.
- **BurstLength:** DMA Burst length. This parameter can be one value between: TIM_DMABURSTLENGTH_1TRANSFER and TIM_DMABURSTLENGTH_18TRANSFERS.
- **DataLength:** Data length. This parameter can be one value between 1 and 0xFFFF.

Return values

- **HAL:** status

HAL_TIM_DMABurst_ReadStop

Function name

HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStop (TIM_HandleTypeDef * htim, uint32_t BurstRequestSrc)

Function description

Stop the DMA burst reading.

Parameters

- **htim:** TIM handle
- **BurstRequestSrc:** TIM DMA Request sources to disable.

Return values

- **HAL:** status

HAL_TIM_GenerateEvent

Function name

HAL_StatusTypeDef HAL_TIM_GenerateEvent (TIM_HandleTypeDef * htim, uint32_t EventSource)

Function description

Generate a software event.

Parameters

- **htim:** TIM handle
- **EventSource:** specifies the event source. This parameter can be one of the following values:
  - TIM_EVENTSOURCE_UPDATE: Timer update Event source
  - TIM_EVENTSOURCE_CC1: Timer Capture Compare 1 Event source
  - TIM_EVENTSOURCE_CC2: Timer Capture Compare 2 Event source
  - TIM_EVENTSOURCE_CC3: Timer Capture Compare 3 Event source
  - TIM_EVENTSOURCE_CC4: Timer Capture Compare 4 Event source
  - TIM_EVENTSOURCE_COM: Timer COM event source
  - TIM_EVENTSOURCE_TRIGGER: Timer Trigger Event source
  - TIM_EVENTSOURCE_BREAK: Timer Break event source
  - TIM_EVENTSOURCE_BREAK2: Timer Break2 event source

Return values

- **HAL:** status

Notes

- Basic timers can only generate an update event.
- TIM_EVENTSOURCE_COM is relevant only with advanced timer instances.
- TIM_EVENTSOURCE_BREAK and TIM_EVENTSOURCE_BREAK2 are relevant only for timer instances supporting break input(s).

HAL_TIM_ReadCapturedValue

Function name

uint32_t HAL_TIM_ReadCapturedValue (const TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Read the captured value from Capture Compare unit.

Parameters

- **htim:** TIM handle.
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **Captured:** value

HAL_TIM_PeriodElapsedCallback

Function name

void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef * htim)

Function description

Period elapsed callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_PeriodElapsedHalfCpltCallback

Function name

void HAL_TIM_PeriodElapsedHalfCpltCallback (TIM_HandleTypeDef * htim)

Function description

Period elapsed half complete callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_OC_DelayElapsedCallback

Function name

void HAL_TIM_OC_DelayElapsedCallback (TIM_HandleTypeDef * htim)

Function description

Output Compare callback in non-blocking mode.

Parameters

- **htim:** TIM OC handle

Return values

- **None:**

HAL_TIM_IC_CaptureCallback

Function name

void HAL_TIM_IC_CaptureCallback (TIM_HandleTypeDef * htim)

Function description

Input Capture callback in non-blocking mode.

Parameters

- **htim:** TIM IC handle

Return values

- **None:**

HAL_TIM_IC_CaptureHalfCpltCallback

Function name

void HAL_TIM_IC_CaptureHalfCpltCallback (TIM_HandleTypeDef * htim)

Function description

Input Capture half complete callback in non-blocking mode.

Parameters

- **htim:** TIM IC handle

Return values

- **None:**

HAL_TIM_PWM_PulseFinishedCallback

Function name

void HAL_TIM_PWM_PulseFinishedCallback (TIM_HandleTypeDef * htim)

Function description

PWM Pulse finished callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_PWM_PulseFinishedHalfCpltCallback

Function name

void HAL_TIM_PWM_PulseFinishedHalfCpltCallback (TIM_HandleTypeDef * htim)

Function description

PWM Pulse finished half complete callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_TriggerCallback

Function name

void HAL_TIM_TriggerCallback (TIM_HandleTypeDef * htim)

Function description

Hall Trigger detection callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_TriggerHalfCpltCallback

Function name

void HAL_TIM_TriggerHalfCpltCallback (TIM_HandleTypeDef * htim)

Function description

Hall Trigger detection half complete callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_ErrorCallback

Function name

void HAL_TIM_ErrorCallback (TIM_HandleTypeDef * htim)

Function description

Timer error callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_RegisterCallback

Function name

HAL_StatusTypeDef HAL_TIM_RegisterCallback (TIM_HandleTypeDef * htim, HAL_TIM_CallbackIDTypeDef CallbackID, pTIM_CallbackTypeDef pCallback)

Function description

Register a User TIM callback to be used instead of the weak predefined callback.

Parameters

- **htim:** tim handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_TIM_BASE_MSPINIT_CB_ID Base MspInit Callback ID
  - HAL_TIM_BASE_MSPDEINIT_CB_ID Base MspDeInit Callback ID
  - HAL_TIM_IC_MSPINIT_CB_ID IC MspInit Callback ID
  - HAL_TIM_IC_MSPDEINIT_CB_ID IC MspDeInit Callback ID
  - HAL_TIM_OC_MSPINIT_CB_ID OC MspInit Callback ID
  - HAL_TIM_OC_MSPDEINIT_CB_ID OC MspDeInit Callback ID
  - HAL_TIM_PWM_MSPINIT_CB_ID PWM MspInit Callback ID
  - HAL_TIM_PWM_MSPDEINIT_CB_ID PWM MspDeInit Callback ID
  - HAL_TIM_ONE_PULSE_MSPINIT_CB_ID One Pulse MspInit Callback ID
  - HAL_TIM_ONE_PULSE_MSPDEINIT_CB_ID One Pulse MspDeInit Callback ID
  - HAL_TIM_ENCODER_MSPINIT_CB_ID Encoder MspInit Callback ID
  - HAL_TIM_ENCODER_MSPDEINIT_CB_ID Encoder MspDeInit Callback ID
  - HAL_TIM_HALL_SENSOR_MSPINIT_CB_ID Hall Sensor MspInit Callback ID
  - HAL_TIM_HALL_SENSOR_MSPDEINIT_CB_ID Hall Sensor MspDeInit Callback ID
  - HAL_TIM_PERIOD_ELAPSED_CB_ID Period Elapsed Callback ID
  - HAL_TIM_PERIOD_ELAPSED_HALF_CB_ID Period Elapsed half complete Callback ID
  - HAL_TIM_TRIGGER_CB_ID Trigger Callback ID
  - HAL_TIM_TRIGGER_HALF_CB_ID Trigger half complete Callback ID
  - HAL_TIM_IC_CAPTURE_CB_ID Input Capture Callback ID
  - HAL_TIM_IC_CAPTURE_HALF_CB_ID Input Capture half complete Callback ID
  - HAL_TIM_OC_DELAY_ELAPSED_CB_ID Output Compare Delay Elapsed Callback ID
  - HAL_TIM_PWM_PULSE_FINISHED_CB_ID PWM Pulse Finished Callback ID
  - HAL_TIM_PWM_PULSE_FINISHED_HALF_CB_ID PWM Pulse Finished half complete Callback ID
  - HAL_TIM_ERROR_CB_ID Error Callback ID
  - HAL_TIM_COMMUTATION_CB_ID Commutation Callback ID
  - HAL_TIM_COMMUTATION_HALF_CB_ID Commutation half complete Callback ID
  - HAL_TIM_BREAK_CB_ID Break Callback ID
  - HAL_TIM_BREAK2_CB_ID Break2 Callback ID
- **pCallback:** pointer to the callback function

Return values

- **status:**

HAL_TIM_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_TIM_UnRegisterCallback (TIM_HandleTypeDef * htim, HAL_TIM_CallbackIDTypeDef CallbackID)

Function description

Unregister a TIM callback TIM callback is redirected to the weak predefined callback.

Parameters

- **htim:** tim handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_TIM_BASE_MSPINIT_CB_ID Base MspInit Callback ID
  - HAL_TIM_BASE_MSPDEINIT_CB_ID Base MspDeInit Callback ID
  - HAL_TIM_IC_MSPINIT_CB_ID IC MspInit Callback ID
  - HAL_TIM_IC_MSPDEINIT_CB_ID IC MspDeInit Callback ID
  - HAL_TIM_OC_MSPINIT_CB_ID OC MspInit Callback ID
  - HAL_TIM_OC_MSPDEINIT_CB_ID OC MspDeInit Callback ID
  - HAL_TIM_PWM_MSPINIT_CB_ID PWM MspInit Callback ID
  - HAL_TIM_PWM_MSPDEINIT_CB_ID PWM MspDeInit Callback ID
  - HAL_TIM_ONE_PULSE_MSPINIT_CB_ID One Pulse MspInit Callback ID
  - HAL_TIM_ONE_PULSE_MSPDEINIT_CB_ID One Pulse MspDeInit Callback ID
  - HAL_TIM_ENCODER_MSPINIT_CB_ID Encoder MspInit Callback ID
  - HAL_TIM_ENCODER_MSPDEINIT_CB_ID Encoder MspDeInit Callback ID
  - HAL_TIM_HALL_SENSOR_MSPINIT_CB_ID Hall Sensor MspInit Callback ID
  - HAL_TIM_HALL_SENSOR_MSPDEINIT_CB_ID Hall Sensor MspDeInit Callback ID
  - HAL_TIM_PERIOD_ELAPSED_CB_ID Period Elapsed Callback ID
  - HAL_TIM_PERIOD_ELAPSED_HALF_CB_ID Period Elapsed half complete Callback ID
  - HAL_TIM_TRIGGER_CB_ID Trigger Callback ID
  - HAL_TIM_TRIGGER_HALF_CB_ID Trigger half complete Callback ID
  - HAL_TIM_IC_CAPTURE_CB_ID Input Capture Callback ID
  - HAL_TIM_IC_CAPTURE_HALF_CB_ID Input Capture half complete Callback ID
  - HAL_TIM_OC_DELAY_ELAPSED_CB_ID Output Compare Delay Elapsed Callback ID
  - HAL_TIM_PWM_PULSE_FINISHED_CB_ID PWM Pulse Finished Callback ID
  - HAL_TIM_PWM_PULSE_FINISHED_HALF_CB_ID PWM Pulse Finished half complete Callback ID
  - HAL_TIM_ERROR_CB_ID Error Callback ID
  - HAL_TIM_COMMUTATION_CB_ID Commutation Callback ID
  - HAL_TIM_COMMUTATION_HALF_CB_ID Commutation half complete Callback ID
  - HAL_TIM_BREAK_CB_ID Break Callback ID
  - HAL_TIM_BREAK2_CB_ID Break2 Callback ID

Return values

- **status:**

HAL_TIM_Base_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_Base_GetState (const TIM_HandleTypeDef * htim)

Function description

Return the TIM Base handle state.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** state

HAL_TIM_OC_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_OC_GetState (const TIM_HandleTypeDef * htim)

Function description

Return the TIM OC handle state.

Parameters

- **htim:** TIM Output Compare handle

Return values

- **HAL:** state

HAL_TIM_PWM_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_PWM_GetState (const TIM_HandleTypeDef * htim)

Function description

Return the TIM PWM handle state.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** state

HAL_TIM_IC_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_IC_GetState (const TIM_HandleTypeDef * htim)

Function description

Return the TIM Input Capture handle state.

Parameters

- **htim:** TIM IC handle

Return values

- **HAL:** state

HAL_TIM_OnePulse_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_OnePulse_GetState (const TIM_HandleTypeDef * htim)

Function description

Return the TIM One Pulse Mode handle state.

Parameters

- **htim:** TIM OPM handle

Return values

- **HAL:** state

HAL_TIM_Encoder_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_Encoder_GetState (const TIM_HandleTypeDef * htim)

Function description

Return the TIM Encoder Mode handle state.

Parameters

- **htim:** TIM Encoder Interface handle

Return values

- **HAL:** state

HAL_TIM_GetActiveChannel

Function name

HAL_TIM_ActiveChannel HAL_TIM_GetActiveChannel (const TIM_HandleTypeDef * htim)

Function description

Return the TIM Encoder Mode handle state.

Parameters

- **htim:** TIM handle

Return values

- **Active:** channel

HAL_TIM_GetChannelState

Function name

HAL_TIM_ChannelStateTypeDef HAL_TIM_GetChannelState (const TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Return actual state of the TIM channel.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1
  - TIM_CHANNEL_2: TIM Channel 2
  - TIM_CHANNEL_3: TIM Channel 3
  - TIM_CHANNEL_4: TIM Channel 4
  - TIM_CHANNEL_5: TIM Channel 5
  - TIM_CHANNEL_6: TIM Channel 6

Return values

- **TIM:** Channel state

HAL_TIM_DMABurstState

Function name

HAL_TIM_DMABurstStateTypeDef HAL_TIM_DMABurstState (const TIM_HandleTypeDef * htim)

Function description

Return actual state of a DMA burst operation.

Parameters

- **htim:** TIM handle

Return values

- **DMA:** burst state

TIM_Base_SetConfig

Function name

void TIM_Base_SetConfig (TIM_TypeDef * TIMx, const TIM_Base_InitTypeDef * Structure)

Function description

Time Base configuration.

Parameters

- **TIMx:** TIM peripheral
- **Structure:** TIM Base configuration structure

Return values

- **None:**

TIM_TI1_SetConfig

Function name

void TIM_TI1_SetConfig (TIM_TypeDef * TIMx, uint32_t TIM_ICPolarity, uint32_t TIM_ICSelection, uint32_t TIM_ICFilter)

Function description

Configure the TI1 as Input.

Parameters

- **TIMx:** to select the TIM peripheral.
- **TIM_ICPolarity:** The Input Polarity. This parameter can be one of the following values:
  - TIM_ICPOLARITY_RISING
  - TIM_ICPOLARITY_FALLING
  - TIM_ICPOLARITY_BOTHEDGE
- **TIM_ICSelection:** specifies the input to be used. This parameter can be one of the following values:
  - TIM_ICSELECTION_DIRECTTI: TIM Input 1 is selected to be connected to IC1.
  - TIM_ICSELECTION_INDIRECTTI: TIM Input 1 is selected to be connected to IC2.
  - TIM_ICSELECTION_TRC: TIM Input 1 is selected to be connected to TRC.
- **TIM_ICFilter:** Specifies the Input Capture Filter. This parameter must be a value between 0x00 and 0x0F.

Return values

- **None:**

Notes

- TIM_ICFilter and TIM_ICPolarity are not used in INDIRECT mode as TI2FP1 (on channel2 path) is used as the input signal. Therefore CCMR1 must be protected against un-initialized filter and polarity values.

TIM_OC2_SetConfig

Function name

void TIM_OC2_SetConfig (TIM_TypeDef * TIMx, const TIM_OC_InitTypeDef * OC_Config)

Function description

Timer Output Compare 2 configuration.

Parameters

- **TIMx:** to select the TIM peripheral
- **OC_Config:** The output configuration structure

Return values

- **None:**

TIM_ETR_SetConfig

Function name

void TIM_ETR_SetConfig (TIM_TypeDef * TIMx, uint32_t TIM_ExtTRGPrescaler, uint32_t TIM_ExtTRGPolarity, uint32_t ExtTRGFilter)

Function description

Configures the TIMx External Trigger (ETR).

Parameters

- **TIMx:** to select the TIM peripheral
- **TIM_ExtTRGPrescaler:** The external Trigger Prescaler. This parameter can be one of the following values:
  - TIM_ETRPRESCALER_DIV1: ETRP Prescaler OFF.
  - TIM_ETRPRESCALER_DIV2: ETRP frequency divided by 2.
  - TIM_ETRPRESCALER_DIV4: ETRP frequency divided by 4.
  - TIM_ETRPRESCALER_DIV8: ETRP frequency divided by 8.
- **TIM_ExtTRGPolarity:** The external Trigger Polarity. This parameter can be one of the following values:
  - TIM_ETRPOLARITY_INVERTED: active low or falling edge active.
  - TIM_ETRPOLARITY_NONINVERTED: active high or rising edge active.
- **ExtTRGFilter:** External Trigger Filter. This parameter must be a value between 0x00 and 0x0F

Return values

- **None:**

TIM_DMADelayPulseHalfCplt

Function name

void TIM_DMADelayPulseHalfCplt (DMA_HandleTypeDef * hdma)

Function description

TIM DMA Delay Pulse half complete callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

TIM_DMAError

Function name

void TIM_DMAError (DMA_HandleTypeDef * hdma)

Function description

TIM DMA error callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

TIM_DMACaptureCplt

Function name

void TIM_DMACaptureCplt (DMA_HandleTypeDef * hdma)

Function description

TIM DMA Capture complete callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

TIM_DMACaptureHalfCplt

Function name

void TIM_DMACaptureHalfCplt (DMA_HandleTypeDef * hdma)

Function description

TIM DMA Capture half complete callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

TIM_CCxChannelCmd

Function name

void TIM_CCxChannelCmd (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ChannelState)

Function description

Enables or disables the TIM Capture Compare Channel x.

Parameters

- **TIMx:** to select the TIM peripheral
- **Channel:** specifies the TIM Channel This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1
  - TIM_CHANNEL_2: TIM Channel 2
  - TIM_CHANNEL_3: TIM Channel 3
  - TIM_CHANNEL_4: TIM Channel 4
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected
- **ChannelState:** specifies the TIM Channel CCxE bit new state. This parameter can be: TIM_CCx_ENABLE or TIM_CCx_DISABLE.

Return values

- **None:**

TIM_ResetCallback

Function name

void TIM_ResetCallback (TIM_HandleTypeDef * htim)

Function description

Reset interrupt callbacks to the legacy weak callbacks.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **None:**

88.3 TIM Firmware driver defines

The following section lists the various define and macros of the module.

88.3.1 TIM

TIM

TIM Automatic Output Enable

TIM_AUTOMATICOUTPUT_DISABLE

MOE can be set only by software

TIM_AUTOMATICOUTPUT_ENABLE

MOE can be set by software or automatically at the next update event (if none of the break inputs BRK and BRK2 is active)

TIM Auto-Reload Preload

TIM_AUTORELOAD_PRELOAD_DISABLE

TIMx_ARR register is not buffered

TIM_AUTORELOAD_PRELOAD_ENABLE

TIMx_ARR register is buffered

TIM Break input 2 Enable

TIM_BREAK2_DISABLE

Break input BRK2 is disabled

TIM_BREAK2_ENABLE

Break input BRK2 is enabled

TIM Break Input 2 Polarity

TIM_BREAK2POLARITY_LOW

Break input BRK2 is active low

TIM_BREAK2POLARITY_HIGH

Break input BRK2 is active high

TIM Break Input Enable

TIM_BREAK_ENABLE

Break input BRK is enabled

TIM_BREAK_DISABLE

Break input BRK is disabled

TIM Break Input Polarity

TIM_BREAKPOLARITY_LOW

Break input BRK is active low

TIM_BREAKPOLARITY_HIGH

Break input BRK is active high

TIM Break System

TIM_BREAK_SYSTEM_ECC

Enables and locks the ECC error signal with Break Input of TIM1/8/15/16/17

TIM_BREAK_SYSTEM_PVD

Enables and locks the PVD connection with TIM1/8/15/16/17 Break Input and also the PVDE and PLS bits of the Power Control Interface

TIM_BREAK_SYSTEM_SRAM_PARITY_ERROR

Enables and locks the SRAM_PARITY error signal with Break Input of TIM1/8/15/16/17

TIM_BREAK_SYSTEM_LOCKUP

Enables and locks the LOCKUP output of CortexM4 with Break Input of TIM1/8/15/16/17

CCx DMA request selection

TIM_CCDMAREQUEST_CC

CCx DMA request sent when capture or compare match event occurs

TIM_CCDMAREQUEST_UPDATE

CCx DMA requests sent when update event occurs

TIM Channel

TIM_CHANNEL_1

Capture/compare channel 1 identifier

TIM_CHANNEL_2

Capture/compare channel 2 identifier

TIM_CHANNEL_3

Capture/compare channel 3 identifier

TIM_CHANNEL_4

Capture/compare channel 4 identifier

TIM_CHANNEL_5

Compare channel 5 identifier

TIM_CHANNEL_6

Compare channel 6 identifier

TIM_CHANNEL_ALL

Global Capture/compare channel identifier

TIM Clear Input Polarity

TIM_CLEARINPUTPOLARITY_INVERTED

Polarity for ETRx pin

TIM_CLEARINPUTPOLARITY_NONINVERTED

Polarity for ETRx pin

TIM Clear Input Prescaler

TIM_CLEARINPUTPRESCALER_DIV1

No prescaler is used

TIM_CLEARINPUTPRESCALER_DIV2

Prescaler for External ETR pin: Capture performed once every 2 events.

TIM_CLEARINPUTPRESCALER_DIV4

Prescaler for External ETR pin: Capture performed once every 4 events.

TIM_CLEARINPUTPRESCALER_DIV8

Prescaler for External ETR pin: Capture performed once every 8 events.

TIM Clear Input Source

TIM_CLEARINPUTSOURCE_NONE

OCREF_CLR is disabled

TIM_CLEARINPUTSOURCE_ETR

OCREF_CLR is connected to ETRF input

TIM Clock Division

TIM_CLOCKDIVISION_DIV1

Clock division: tDTS=tCK_INT

TIM_CLOCKDIVISION_DIV2

Clock division: tDTS=2*tCK_INT

TIM_CLOCKDIVISION_DIV4

Clock division: tDTS=4*tCK_INT

TIM Clock Polarity

TIM_CLOCKPOLARITY_INVERTED

Polarity for ETRx clock sources

TIM_CLOCKPOLARITY_NONINVERTED

Polarity for ETRx clock sources

TIM_CLOCKPOLARITY_RISING

Polarity for TIx clock sources

TIM_CLOCKPOLARITY_FALLING

Polarity for TIx clock sources

TIM_CLOCKPOLARITY_BOTHEDGE

Polarity for TIx clock sources

TIM Clock Prescaler

TIM_CLOCKPRESCALER_DIV1

No prescaler is used

TIM_CLOCKPRESCALER_DIV2

Prescaler for External ETR Clock: Capture performed once every 2 events.

TIM_CLOCKPRESCALER_DIV4

Prescaler for External ETR Clock: Capture performed once every 4 events.

TIM_CLOCKPRESCALER_DIV8

Prescaler for External ETR Clock: Capture performed once every 8 events.

TIM Clock Source

TIM_CLOCKSOURCE_INTERNAL

Internal clock source

TIM_CLOCKSOURCE_ETRMODE1

External clock source mode 1 (ETRF)

TIM_CLOCKSOURCE_ETRMODE2

External clock source mode 2

TIM_CLOCKSOURCE_TI1ED

External clock source mode 1 (TTI1FP1 + edge detect.)

TIM_CLOCKSOURCE_TI1

External clock source mode 1 (TTI1FP1)

TIM_CLOCKSOURCE_TI2

External clock source mode 1 (TTI2FP2)

TIM_CLOCKSOURCE_ITR0

External clock source mode 1 (ITR0)

TIM_CLOCKSOURCE_ITR1

External clock source mode 1 (ITR1)

TIM_CLOCKSOURCE_ITR2

External clock source mode 1 (ITR2)

TIM_CLOCKSOURCE_ITR3

External clock source mode 1 (ITR3)

TIM_CLOCKSOURCE_ITR4

External clock source mode 1 (ITR4)

TIM_CLOCKSOURCE_ITR5

External clock source mode 1 (ITR5)

TIM_CLOCKSOURCE_ITR6

External clock source mode 1 (ITR6)

TIM_CLOCKSOURCE_ITR7

External clock source mode 1 (ITR7)

TIM_CLOCKSOURCE_ITR8

External clock source mode 1 (ITR8)

TIM Commutation Source

TIM_COMMUTATION_TRGI

When Capture/compare control bits are preloaded, they are updated by setting the COMG bit or when an rising edge occurs on trigger input

TIM_COMMUTATION_SOFTWARE

When Capture/compare control bits are preloaded, they are updated by setting the COMG bit

TIM Counter Mode

TIM_COUNTERMODE_UP

Counter used as up-counter

TIM_COUNTERMODE_DOWN

Counter used as down-counter

TIM_COUNTERMODE_CENTERALIGNED1

Center-aligned mode 1

TIM_COUNTERMODE_CENTERALIGNED2

Center-aligned mode 2

TIM_COUNTERMODE_CENTERALIGNED3

Center-aligned mode 3

TIM DMA Base Address

TIM_DMABASE_CR1

TIM_DMABASE_CR2

TIM_DMABASE_SMCR

TIM_DMABASE_DIER

TIM_DMABASE_SR

TIM_DMABASE_EGR

TIM_DMABASE_CCMR1

TIM_DMABASE_CCMR2

TIM_DMABASE_CCER

TIM_DMABASE_CNT

TIM_DMABASE_PSC

TIM_DMABASE_ARR

TIM_DMABASE_RCR

TIM_DMABASE_CCR1

TIM_DMABASE_CCR2

TIM_DMABASE_CCR3

TIM_DMABASE_CCR4

TIM_DMABASE_BDTR

TIM_DMABASE_DCR

TIM_DMABASE_DMAR

TIM_DMABASE_CCMR3

TIM_DMABASE_CCR5

TIM_DMABASE_CCR6

TIM_DMABASE_AF1

TIM_DMABASE_AF2

TIM_DMABASE_TISEL

TIM DMA Burst Length

TIM_DMABURSTLENGTH_1TRANSFER

The transfer is done to 1 register starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_2TRANSFERS

The transfer is done to 2 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_3TRANSFERS

The transfer is done to 3 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_4TRANSFERS

The transfer is done to 4 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_5TRANSFERS

The transfer is done to 5 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_6TRANSFERS

The transfer is done to 6 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_7TRANSFERS

The transfer is done to 7 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_8TRANSFERS

The transfer is done to 8 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_9TRANSFERS

The transfer is done to 9 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_10TRANSFERS

The transfer is done to 10 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_11TRANSFERS

The transfer is done to 11 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_12TRANSFERS

The transfer is done to 12 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_13TRANSFERS

The transfer is done to 13 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_14TRANSFERS

The transfer is done to 14 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_15TRANSFERS

The transfer is done to 15 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_16TRANSFERS

The transfer is done to 16 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_17TRANSFERS

The transfer is done to 17 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_18TRANSFERS

The transfer is done to 18 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM DMA Sources

TIM_DMA_UPDATE

DMA request is triggered by the update event

TIM_DMA_CC1

DMA request is triggered by the capture/compare macth 1 event

TIM_DMA_CC2

DMA request is triggered by the capture/compare macth 2 event event

TIM_DMA_CC3

DMA request is triggered by the capture/compare macth 3 event event

TIM_DMA_CC4

DMA request is triggered by the capture/compare macth 4 event event

TIM_DMA_COM

DMA request is triggered by the commutation event

TIM_DMA_TRIGGER

DMA request is triggered by the trigger event

TIM Encoder Input Polarity

TIM_ENCODERINPUTPOLARITY_RISING

Encoder input with rising edge polarity

TIM_ENCODERINPUTPOLARITY_FALLING

Encoder input with falling edge polarity

TIM Encoder Mode

TIM_ENCODERMODE_TI1

Quadrature encoder mode 1, x2 mode, counts up/down on TI1FP1 edge depending on TI2FP2 level

TIM_ENCODERMODE_TI2

Quadrature encoder mode 2, x2 mode, counts up/down on TI2FP2 edge depending on TI1FP1 level.

TIM_ENCODERMODE_TI12

Quadrature encoder mode 3, x4 mode, counts up/down on both TI1FP1 and TI2FP2 edges depending on the level of the other input.

TIM ETR Polarity

TIM_ETRPOLARITY_INVERTED

Polarity for ETR source

TIM_ETRPOLARITY_NONINVERTED

Polarity for ETR source

TIM ETR Prescaler

TIM_ETRPRESCALER_DIV1

No prescaler is used

TIM_ETRPRESCALER_DIV2

ETR input source is divided by 2

TIM_ETRPRESCALER_DIV4

ETR input source is divided by 4

TIM_ETRPRESCALER_DIV8

ETR input source is divided by 8

TIM Event Source

TIM_EVENTSOURCE_UPDATE

Reinitialize the counter and generates an update of the registers

TIM_EVENTSOURCE_CC1

A capture/compare event is generated on channel 1

TIM_EVENTSOURCE_CC2

A capture/compare event is generated on channel 2

TIM_EVENTSOURCE_CC3

A capture/compare event is generated on channel 3

TIM_EVENTSOURCE_CC4

A capture/compare event is generated on channel 4

TIM_EVENTSOURCE_COM

A commutation event is generated

TIM_EVENTSOURCE_TRIGGER

A trigger event is generated

TIM_EVENTSOURCE_BREAK

A break event is generated

TIM_EVENTSOURCE_BREAK2

A break 2 event is generated

TIM Exported Macros

__HAL_TIM_RESET_HANDLE_STATE

Description:

- Reset TIM handle state.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None

__HAL_TIM_ENABLE

Description:

- Enable the TIM peripheral.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

__HAL_TIM_MOE_ENABLE

Description:

- Enable the TIM main Output.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

__HAL_TIM_DISABLE

Description:

- Disable the TIM peripheral.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

__HAL_TIM_MOE_DISABLE

Description:

- Disable the TIM main Output.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

Notes:

- The Main Output Enable of a timer instance is disabled only if all the CCx and CCxN channels have been disabled

__HAL_TIM_MOE_DISABLE_UNCONDITIONALLY

Description:

- Disable the TIM main Output.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

Notes:

- The Main Output Enable of a timer instance is disabled unconditionally

__HAL_TIM_ENABLE_IT

Description:

- Enable the specified TIM interrupt.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __INTERRUPT__: specifies the TIM interrupt source to enable. This parameter can be one of the following values:
  - TIM_IT_UPDATE: Update interrupt
  - TIM_IT_CC1: Capture/Compare 1 interrupt
  - TIM_IT_CC2: Capture/Compare 2 interrupt
  - TIM_IT_CC3: Capture/Compare 3 interrupt
  - TIM_IT_CC4: Capture/Compare 4 interrupt
  - TIM_IT_COM: Commutation interrupt
  - TIM_IT_TRIGGER: Trigger interrupt
  - TIM_IT_BREAK: Break interrupt

Return value:

- None

__HAL_TIM_DISABLE_IT

Description:

- Disable the specified TIM interrupt.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __INTERRUPT__: specifies the TIM interrupt source to disable. This parameter can be one of the following values:
  - TIM_IT_UPDATE: Update interrupt
  - TIM_IT_CC1: Capture/Compare 1 interrupt
  - TIM_IT_CC2: Capture/Compare 2 interrupt
  - TIM_IT_CC3: Capture/Compare 3 interrupt
  - TIM_IT_CC4: Capture/Compare 4 interrupt
  - TIM_IT_COM: Commutation interrupt
  - TIM_IT_TRIGGER: Trigger interrupt
  - TIM_IT_BREAK: Break interrupt

Return value:

- None

__HAL_TIM_ENABLE_DMA

Description:

- Enable the specified DMA request.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __DMA__: specifies the TIM DMA request to enable. This parameter can be one of the following values:
  - TIM_DMA_UPDATE: Update DMA request
  - TIM_DMA_CC1: Capture/Compare 1 DMA request
  - TIM_DMA_CC2: Capture/Compare 2 DMA request
  - TIM_DMA_CC3: Capture/Compare 3 DMA request
  - TIM_DMA_CC4: Capture/Compare 4 DMA request
  - TIM_DMA_COM: Commutation DMA request
  - TIM_DMA_TRIGGER: Trigger DMA request

Return value:

- None

__HAL_TIM_DISABLE_DMA

Description:

- Disable the specified DMA request.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __DMA__: specifies the TIM DMA request to disable. This parameter can be one of the following values:
  - TIM_DMA_UPDATE: Update DMA request
  - TIM_DMA_CC1: Capture/Compare 1 DMA request
  - TIM_DMA_CC2: Capture/Compare 2 DMA request
  - TIM_DMA_CC3: Capture/Compare 3 DMA request
  - TIM_DMA_CC4: Capture/Compare 4 DMA request
  - TIM_DMA_COM: Commutation DMA request
  - TIM_DMA_TRIGGER: Trigger DMA request

Return value:

- None

__HAL_TIM_GET_FLAG

Description:

- Check whether the specified TIM interrupt flag is set or not.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __FLAG__: specifies the TIM interrupt flag to check. This parameter can be one of the following values:
  - TIM_FLAG_UPDATE: Update interrupt flag
  - TIM_FLAG_CC1: Capture/Compare 1 interrupt flag
  - TIM_FLAG_CC2: Capture/Compare 2 interrupt flag
  - TIM_FLAG_CC3: Capture/Compare 3 interrupt flag
  - TIM_FLAG_CC4: Capture/Compare 4 interrupt flag
  - TIM_FLAG_CC5: Compare 5 interrupt flag
  - TIM_FLAG_CC6: Compare 6 interrupt flag
  - TIM_FLAG_COM: Commutation interrupt flag
  - TIM_FLAG_TRIGGER: Trigger interrupt flag
  - TIM_FLAG_BREAK: Break interrupt flag
  - TIM_FLAG_BREAK2: Break 2 interrupt flag
  - TIM_FLAG_SYSTEM_BREAK: System Break interrupt flag
  - TIM_FLAG_CC1OF: Capture/Compare 1 overcapture flag
  - TIM_FLAG_CC2OF: Capture/Compare 2 overcapture flag
  - TIM_FLAG_CC3OF: Capture/Compare 3 overcapture flag
  - TIM_FLAG_CC4OF: Capture/Compare 4 overcapture flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_TIM_CLEAR_FLAG

Description:

- Clear the specified TIM interrupt flag.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __FLAG__: specifies the TIM interrupt flag to clear. This parameter can be one of the following values:
  - TIM_FLAG_UPDATE: Update interrupt flag
  - TIM_FLAG_CC1: Capture/Compare 1 interrupt flag
  - TIM_FLAG_CC2: Capture/Compare 2 interrupt flag
  - TIM_FLAG_CC3: Capture/Compare 3 interrupt flag
  - TIM_FLAG_CC4: Capture/Compare 4 interrupt flag
  - TIM_FLAG_CC5: Compare 5 interrupt flag
  - TIM_FLAG_CC6: Compare 6 interrupt flag
  - TIM_FLAG_COM: Commutation interrupt flag
  - TIM_FLAG_TRIGGER: Trigger interrupt flag
  - TIM_FLAG_BREAK: Break interrupt flag
  - TIM_FLAG_BREAK2: Break 2 interrupt flag
  - TIM_FLAG_SYSTEM_BREAK: System Break interrupt flag
  - TIM_FLAG_CC1OF: Capture/Compare 1 overcapture flag
  - TIM_FLAG_CC2OF: Capture/Compare 2 overcapture flag
  - TIM_FLAG_CC3OF: Capture/Compare 3 overcapture flag
  - TIM_FLAG_CC4OF: Capture/Compare 4 overcapture flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_TIM_GET_IT_SOURCE

Description:

- Check whether the specified TIM interrupt source is enabled or not.

Parameters:

- __HANDLE__: TIM handle
- __INTERRUPT__: specifies the TIM interrupt source to check. This parameter can be one of the following values:
  - TIM_IT_UPDATE: Update interrupt
  - TIM_IT_CC1: Capture/Compare 1 interrupt
  - TIM_IT_CC2: Capture/Compare 2 interrupt
  - TIM_IT_CC3: Capture/Compare 3 interrupt
  - TIM_IT_CC4: Capture/Compare 4 interrupt
  - TIM_IT_COM: Commutation interrupt
  - TIM_IT_TRIGGER: Trigger interrupt
  - TIM_IT_BREAK: Break interrupt

Return value:

- The: state of TIM_IT (SET or RESET).

__HAL_TIM_CLEAR_IT

Description:

- Clear the TIM interrupt pending bits.

Parameters:

- __HANDLE__: TIM handle
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
  - TIM_IT_UPDATE: Update interrupt
  - TIM_IT_CC1: Capture/Compare 1 interrupt
  - TIM_IT_CC2: Capture/Compare 2 interrupt
  - TIM_IT_CC3: Capture/Compare 3 interrupt
  - TIM_IT_CC4: Capture/Compare 4 interrupt
  - TIM_IT_COM: Commutation interrupt
  - TIM_IT_TRIGGER: Trigger interrupt
  - TIM_IT_BREAK: Break interrupt

Return value:

- None

__HAL_TIM_UIFREMAP_ENABLE

Description:

- Force a continuous copy of the update interrupt flag (UIF) into the timer counter register (bit 31).

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None: mode.

Notes:

- This allows both the counter value and a potential roll-over condition signalled by the UIFCPY flag to be read in an atomic way.

__HAL_TIM_UIFREMAP_DISABLE

Description:

- Disable update interrupt flag (UIF) remapping.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None: mode.

__HAL_TIM_GET_UIFCPY

Description:

- Get update interrupt flag (UIF) copy status.

Parameters:

- __COUNTER__: Counter value.

Return value:

- The: state of UIFCPY (TRUE or FALSE). mode.

__HAL_TIM_IS_TIM_COUNTING_DOWN

Description:

- Indicates whether or not the TIM Counter is used as downcounter.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- False: (Counter used as upcounter) or True (Counter used as downcounter)

Notes:

- This macro is particularly useful to get the counting mode when the timer operates in Center-aligned mode or Encoder mode.

__HAL_TIM_SET_PRESCALER

Description:

- Set the TIM Prescaler on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __PRESC__: specifies the Prescaler new value.

Return value:

- None

__HAL_TIM_SET_COUNTER

Description:

- Set the TIM Counter Register value on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __COUNTER__: specifies the Counter register new value.

Return value:

- None

__HAL_TIM_GET_COUNTER

Description:

- Get the TIM Counter Register value on runtime.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- 16-bit: or 32-bit value of the timer counter register (TIMx_CNT)

__HAL_TIM_SET_AUTORELOAD

Description:

- Set the TIM Autoreload Register value on runtime without calling another time any Init function.

Parameters:

- __HANDLE__: TIM handle.
- __AUTORELOAD__: specifies the Counter register new value.

Return value:

- None

__HAL_TIM_GET_AUTORELOAD

Description:

- Get the TIM Autoreload Register value on runtime.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- 16-bit: or 32-bit value of the timer auto-reload register(TIMx_ARR)

__HAL_TIM_SET_CLOCKDIVISION

Description:

- Set the TIM Clock Division value on runtime without calling another time any Init function.

Parameters:

- __HANDLE__: TIM handle.
- __CKD__: specifies the clock division value. This parameter can be one of the following value:
  - TIM_CLOCKDIVISION_DIV1: tDTS=tCK_INT
  - TIM_CLOCKDIVISION_DIV2: tDTS=2*tCK_INT
  - TIM_CLOCKDIVISION_DIV4: tDTS=4*tCK_INT

Return value:

- None

__HAL_TIM_GET_CLOCKDIVISION

Description:

- Get the TIM Clock Division value on runtime.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- The: clock division can be one of the following values:
  - TIM_CLOCKDIVISION_DIV1: tDTS=tCK_INT
  - TIM_CLOCKDIVISION_DIV2: tDTS=2*tCK_INT
  - TIM_CLOCKDIVISION_DIV4: tDTS=4*tCK_INT

__HAL_TIM_SET_ICPRESCALER

Description:

- Set the TIM Input Capture prescaler on runtime without calling another time

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- __ICPSC__: specifies the Input Capture4 prescaler new value. This parameter can be one of the following values:
  - TIM_ICPSC_DIV1: no prescaler
  - TIM_ICPSC_DIV2: capture is done once every 2 events
  - TIM_ICPSC_DIV4: capture is done once every 4 events
  - TIM_ICPSC_DIV8: capture is done once every 8 events

Return value:

- None

__HAL_TIM_GET_ICPRESCALER

Description:

- Get the TIM Input Capture prescaler on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: get input capture 1 prescaler value
  - TIM_CHANNEL_2: get input capture 2 prescaler value
  - TIM_CHANNEL_3: get input capture 3 prescaler value
  - TIM_CHANNEL_4: get input capture 4 prescaler value

Return value:

- The: input capture prescaler can be one of the following values:
  - TIM_ICPSC_DIV1: no prescaler
  - TIM_ICPSC_DIV2: capture is done once every 2 events
  - TIM_ICPSC_DIV4: capture is done once every 4 events
  - TIM_ICPSC_DIV8: capture is done once every 8 events

__HAL_TIM_SET_COMPARE

Description:

- Set the TIM Capture Compare Register value on runtime without calling another time ConfigChannel function.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected
- __COMPARE__: specifies the Capture Compare register new value.

Return value:

- None

__HAL_TIM_GET_COMPARE

Description:

- Get the TIM Capture Compare Register value on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channel associated with the capture compare register This parameter can be one of the following values:
  - TIM_CHANNEL_1: get capture/compare 1 register value
  - TIM_CHANNEL_2: get capture/compare 2 register value
  - TIM_CHANNEL_3: get capture/compare 3 register value
  - TIM_CHANNEL_4: get capture/compare 4 register value
  - TIM_CHANNEL_5: get capture/compare 5 register value
  - TIM_CHANNEL_6: get capture/compare 6 register value

Return value:

- 16-bit: or 32-bit value of the capture/compare register (TIMx_CCRy)

__HAL_TIM_ENABLE_OCxPRELOAD

Description:

- Set the TIM Output compare preload.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return value:

- None

__HAL_TIM_DISABLE_OCxPRELOAD

Description:

- Reset the TIM Output compare preload.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return value:

- None

__HAL_TIM_ENABLE_OCxFAST

Description:

- Enable fast mode for a given channel.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return value:

- None

Notes:

- When fast mode is enabled an active edge on the trigger input acts like a compare match on CCx output. Delay to sample the trigger input and to activate CCx output is reduced to 3 clock cycles. Fast mode acts only if the channel is configured in PWM1 or PWM2 mode.

__HAL_TIM_DISABLE_OCxFAST

Description:

- Disable fast mode for a given channel.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return value:

- None

Notes:

- When fast mode is disabled CCx output behaves normally depending on counter and CCRx values even when the trigger is ON. The minimum delay to activate CCx output when an active edge occurs on the trigger input is 5 clock cycles.

__HAL_TIM_URS_ENABLE

Description:

- Set the Update Request Source (URS) bit of the TIMx_CR1 register.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None

Notes:

- When the URS bit of the TIMx_CR1 register is set, only counter overflow/underflow generates an update interrupt or DMA request (if enabled)

__HAL_TIM_URS_DISABLE

Description:

- Reset the Update Request Source (URS) bit of the TIMx_CR1 register.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None

Notes:

- When the URS bit of the TIMx_CR1 register is reset, any of the following events generate an update interrupt or DMA request (if enabled): _ Counter overflow underflow _ Setting the UG bit _ Update generation through the slave mode controller

__HAL_TIM_SET_CAPTUREPOLARITY

Description:

- Set the TIM Capture x input polarity on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- __POLARITY__: Polarity for TIx source
  - TIM_INPUTCHANNELPOLARITY_RISING: Rising Edge
  - TIM_INPUTCHANNELPOLARITY_FALLING: Falling Edge
  - TIM_INPUTCHANNELPOLARITY_BOTHEDGE: Rising and Falling Edge

Return value:

- None

__HAL_TIM_SELECT_CCDMAREQUEST

Description:

- Select the Capture/compare DMA request source.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __CCDMA__: specifies Capture/compare DMA request source This parameter can be one of the following values:
  - TIM_CCDMAREQUEST_CC: CCx DMA request generated on Capture/Compare event
  - TIM_CCDMAREQUEST_UPDATE: CCx DMA request generated on Update event

Return value:

- None

TIM Flag Definition

TIM_FLAG_UPDATE

Update interrupt flag

TIM_FLAG_CC1

Capture/Compare 1 interrupt flag

TIM_FLAG_CC2

Capture/Compare 2 interrupt flag

TIM_FLAG_CC3

Capture/Compare 3 interrupt flag

TIM_FLAG_CC4

Capture/Compare 4 interrupt flag

TIM_FLAG_CC5

Capture/Compare 5 interrupt flag

TIM_FLAG_CC6

Capture/Compare 6 interrupt flag

TIM_FLAG_COM

Commutation interrupt flag

TIM_FLAG_TRIGGER

Trigger interrupt flag

TIM_FLAG_BREAK

Break interrupt flag

TIM_FLAG_BREAK2

Break 2 interrupt flag

TIM_FLAG_SYSTEM_BREAK

System Break interrupt flag

TIM_FLAG_CC1OF

Capture 1 overcapture flag

TIM_FLAG_CC2OF

Capture 2 overcapture flag

TIM_FLAG_CC3OF

Capture 3 overcapture flag

TIM_FLAG_CC4OF

Capture 4 overcapture flag

TIM Group Channel 5 and Channel 1, 2 or 3

TIM_GROUPCH5_NONE

No effect of OC5REF on OC1REFC, OC2REFC and OC3REFC

TIM_GROUPCH5_OC1REFC

OC1REFC is the logical AND of OC1REFC and OC5REF

TIM_GROUPCH5_OC2REFC

OC2REFC is the logical AND of OC2REFC and OC5REF

TIM_GROUPCH5_OC3REFC

OC3REFC is the logical AND of OC3REFC and OC5REF

TIM Input Capture Polarity

TIM_ICPOLARITY_RISING

Capture triggered by rising edge on timer input

TIM_ICPOLARITY_FALLING

Capture triggered by falling edge on timer input

TIM_ICPOLARITY_BOTHEDGE

Capture triggered by both rising and falling edges on timer input

TIM Input Capture Prescaler

TIM_ICPSC_DIV1

Capture performed each time an edge is detected on the capture input

TIM_ICPSC_DIV2

Capture performed once every 2 events

TIM_ICPSC_DIV4

Capture performed once every 4 events

TIM_ICPSC_DIV8

Capture performed once every 8 events

TIM Input Capture Selection

TIM_ICSELECTION_DIRECTTI

TIM Input 1, 2, 3 or 4 is selected to be connected to IC1, IC2, IC3 or IC4, respectively

TIM_ICSELECTION_INDIRECTTI

TIM Input 1, 2, 3 or 4 is selected to be connected to IC2, IC1, IC4 or IC3, respectively

TIM_ICSELECTION_TRC

TIM Input 1, 2, 3 or 4 is selected to be connected to TRC

TIM Input Channel polarity

TIM_INPUTCHANNELPOLARITY_RISING

Polarity for TIx source

TIM_INPUTCHANNELPOLARITY_FALLING

Polarity for TIx source

TIM_INPUTCHANNELPOLARITY_BOTHEDGE

Polarity for TIx source

TIM interrupt Definition

TIM_IT_UPDATE

Update interrupt

TIM_IT_CC1

Capture/Compare 1 interrupt

TIM_IT_CC2

Capture/Compare 2 interrupt

TIM_IT_CC3

Capture/Compare 3 interrupt

TIM_IT_CC4

Capture/Compare 4 interrupt

TIM_IT_COM

Commutation interrupt

TIM_IT_TRIGGER

Trigger interrupt

TIM_IT_BREAK

Break interrupt

TIM Lock level

TIM_LOCKLEVEL_OFF

LOCK OFF

TIM_LOCKLEVEL_1

LOCK Level 1

TIM_LOCKLEVEL_2

LOCK Level 2

TIM_LOCKLEVEL_3

LOCK Level 3

TIM Master Mode Selection

TIM_TRGO_RESET

TIMx_EGR.UG bit is used as trigger output (TRGO)

TIM_TRGO_ENABLE

TIMx_CR1.CEN bit is used as trigger output (TRGO)

TIM_TRGO_UPDATE

Update event is used as trigger output (TRGO)

TIM_TRGO_OC1

Capture or a compare match 1 is used as trigger output (TRGO)

TIM_TRGO_OC1REF

OC1REF signal is used as trigger output (TRGO)

TIM_TRGO_OC2REF

OC2REF signal is used as trigger output(TRGO)

TIM_TRGO_OC3REF

OC3REF signal is used as trigger output(TRGO)

TIM_TRGO_OC4REF

OC4REF signal is used as trigger output(TRGO)

TIM Master Mode Selection 2 (TRGO2)

TIM_TRGO2_RESET

TIMx_EGR.UG bit is used as trigger output (TRGO2)

TIM_TRGO2_ENABLE

TIMx_CR1.CEN bit is used as trigger output (TRGO2)

TIM_TRGO2_UPDATE

Update event is used as trigger output (TRGO2)

TIM_TRGO2_OC1

Capture or a compare match 1 is used as trigger output (TRGO2)

TIM_TRGO2_OC1REF

OC1REF signal is used as trigger output (TRGO2)

TIM_TRGO2_OC2REF

OC2REF signal is used as trigger output (TRGO2)

TIM_TRGO2_OC3REF

OC3REF signal is used as trigger output (TRGO2)

TIM_TRGO2_OC4REF

OC4REF signal is used as trigger output (TRGO2)

TIM_TRGO2_OC5REF

OC5REF signal is used as trigger output (TRGO2)

TIM_TRGO2_OC6REF

OC6REF signal is used as trigger output (TRGO2)

TIM_TRGO2_OC4REF_RISINGFALLING

OC4REF rising or falling edges generate pulses on TRGO2

TIM_TRGO2_OC6REF_RISINGFALLING

OC6REF rising or falling edges generate pulses on TRGO2

TIM_TRGO2_OC4REF_RISING_OC6REF_RISING

OC4REF or OC6REF rising edges generate pulses on TRGO2

TIM_TRGO2_OC4REF_RISING_OC6REF_FALLING

OC4REF rising or OC6REF falling edges generate pulses on TRGO2

TIM_TRGO2_OC5REF_RISING_OC6REF_RISING

OC5REF or OC6REF rising edges generate pulses on TRGO2

OC5REF or OC6REF rising edges generate pulses on TRGO2

TIM Master/Slave Mode

TIM_MASTERSLAVEMODE_ENABLE

No action

TIM_MASTERSLAVEMODE_DISABLE

Master/slave mode is selected

TIM One Pulse Mode

TIM_OPMODE_SINGLE

Counter stops counting at the next update event

TIM_OPMODE_REPETITIVE

Counter is not stopped at update event

TIM OSSI OffState Selection for Idle mode state

TIM_OSSI_ENABLE

When inactive, OC/OCN outputs are enabled (still controlled by the timer)

TIM_OSSI_DISABLE

When inactive, OC/OCN outputs are disabled (not controlled any longer by the timer)

TIM OSSR OffState Selection for Run mode state

TIM_OSSR_ENABLE

When inactive, OC/OCN outputs are enabled (still controlled by the timer)

TIM_OSSR_DISABLE

When inactive, OC/OCN outputs are disabled (not controlled any longer by the timer)

TIM Output Compare and PWM Modes

TIM_OCMODE_TIMING

Frozen

TIM_OCMODE_ACTIVE

Set channel to active level on match

TIM_OCMODE_INACTIVE

Set channel to inactive level on match

TIM_OCMODE_TOGGLE

Toggle

TIM_OCMODE_PWM1

PWM mode 1

TIM_OCMODE_PWM2

PWM mode 2

TIM_OCMODE_FORCED_ACTIVE

Force active level

TIM_OCMODE_FORCED_INACTIVE

Force inactive level

TIM_OCMODE_RETRIGERRABLE_OPM1

Retrigerrable OPM mode 1

TIM_OCMODE_RETRIGERRABLE_OPM2

Retrigerrable OPM mode 2

TIM_OCMODE_COMBINED_PWM1

Combined PWM mode 1

TIM_OCMODE_COMBINED_PWM2

Combined PWM mode 2

TIM_OCMODE_ASSYMETRIC_PWM1

Asymmetric PWM mode 1

TIM_OCMODE_ASSYMETRIC_PWM2

Asymmetric PWM mode 2

TIM Output Compare Idle State

TIM_OCIDLESTATE_SET

Output Idle state: OCx=1 when MOE=0

TIM_OCIDLESTATE_RESET

Output Idle state: OCx=0 when MOE=0

TIM Complementary Output Compare Idle State

TIM_OCNIDLESTATE_SET

Complementary output Idle state: OCxN=1 when MOE=0

TIM_OCNIDLESTATE_RESET

Complementary output Idle state: OCxN=0 when MOE=0

TIM Complementary Output Compare Polarity

TIM_OCNPOLARITY_HIGH

Capture/Compare complementary output polarity

TIM_OCNPOLARITY_LOW

Capture/Compare complementary output polarity

TIM Complementary Output Compare State

TIM_OUTPUTNSTATE_DISABLE

OCxN is disabled

TIM_OUTPUTNSTATE_ENABLE

OCxN is enabled

TIM Output Compare Polarity

TIM_OCPOLARITY_HIGH

Capture/Compare output polarity

TIM_OCPOLARITY_LOW

Capture/Compare output polarity

TIM Output Compare State

TIM_OUTPUTSTATE_DISABLE

Capture/Compare 1 output disabled

TIM_OUTPUTSTATE_ENABLE

Capture/Compare 1 output enabled

TIM Output Fast State

TIM_OCFAST_DISABLE

Output Compare fast disable

TIM_OCFAST_ENABLE

Output Compare fast enable

TIM Slave mode

TIM_SLAVEMODE_DISABLE

Slave mode disabled

TIM_SLAVEMODE_RESET

Reset Mode

TIM_SLAVEMODE_GATED

Gated Mode

TIM_SLAVEMODE_TRIGGER

Trigger Mode

TIM_SLAVEMODE_EXTERNAL1

External Clock Mode 1

TIM_SLAVEMODE_COMBINED_RESETTRIGGER

Combined reset + trigger mode

TIM TI1 Input Selection

TIM_TI1SELECTION_CH1

The TIMx_CH1 pin is connected to TI1 input

TIM_TI1SELECTION_XORCOMBINATION

The TIMx_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)

TIM Trigger Polarity

TIM_TRIGGERPOLARITY_INVERTED

Polarity for ETRx trigger sources

TIM_TRIGGERPOLARITY_NONINVERTED

Polarity for ETRx trigger sources

TIM_TRIGGERPOLARITY_RISING

Polarity for TIxFPx or TI1_ED trigger sources

TIM_TRIGGERPOLARITY_FALLING

Polarity for TIxFPx or TI1_ED trigger sources

TIM_TRIGGERPOLARITY_BOTHEDGE

Polarity for TIxFPx or TI1_ED trigger sources

TIM Trigger Prescaler

TIM_TRIGGERPRESCALER_DIV1

No prescaler is used

TIM_TRIGGERPRESCALER_DIV2

Prescaler for External ETR Trigger: Capture performed once every 2 events.

TIM_TRIGGERPRESCALER_DIV4

Prescaler for External ETR Trigger: Capture performed once every 4 events.

TIM_TRIGGERPRESCALER_DIV8

Prescaler for External ETR Trigger: Capture performed once every 8 events.

TIM Trigger Selection

TIM_TS_ITR0

Internal Trigger 0 (ITR0)

TIM_TS_ITR1

Internal Trigger 1 (ITR1)

TIM_TS_ITR2

Internal Trigger 2 (ITR2)

TIM_TS_ITR3

Internal Trigger 3 (ITR3)

TIM_TS_ITR4

Internal Trigger 4 (ITR4)

TIM_TS_ITR5

Internal Trigger 5 (ITR5)

TIM_TS_ITR6

Internal Trigger 6 (ITR6)

TIM_TS_ITR7

Internal Trigger 7 (ITR7)

TIM_TS_ITR8

Internal Trigger 8 (ITR8)

TIM_TS_ITR9

Internal Trigger 9 (ITR9)

TIM_TS_ITR10

Internal Trigger 10 (ITR10)

TIM_TS_ITR11

Internal Trigger 11 (ITR11)

TIM_TS_ITR12

Internal Trigger 12 (ITR12)

TIM_TS_ITR13

Internal Trigger 13 (ITR13)

TIM_TS_TI1F_ED

TI1 Edge Detector (TI1F_ED)

TIM_TS_TI1FP1

Filtered Timer Input 1 (TI1FP1)

TIM_TS_TI2FP2

Filtered Timer Input 2 (TI2FP2)

TIM_TS_ETRF

Filtered External Trigger input (ETRF)

TIM_TS_NONE

No trigger selected

TIM Update Interrupt Flag Remap

TIM_UIFREMAP_DISABLE

Update interrupt flag remap disabled

TIM_UIFREMAP_ENABLE

Update interrupt flag remap enabled

89 HAL TIM Extension Driver

89.1 TIMEx Firmware driver registers structures

89.1.1 TIM_HallSensor_InitTypeDef

TIM_HallSensor_InitTypeDef is defined in the stm32h7xx_hal_tim_ex.h

Data Fields

- ***uint32_t IC1Polarity***
- ***uint32_t IC1Prescaler***
- ***uint32_t IC1Filter***
- ***uint32_t Commutation_Delay***

Field Documentation

- ***uint32_t TIM_HallSensor_InitTypeDef::IC1Polarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Input_Capture_Polarity***
- ***uint32_t TIM_HallSensor_InitTypeDef::IC1Prescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_Input_Capture_Prescaler***
- ***uint32_t TIM_HallSensor_InitTypeDef::IC1Filter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- ***uint32_t TIM_HallSensor_InitTypeDef::Commutation_Delay***
  Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF

89.1.2 TIMEx_BreakInputConfigTypeDef

TIMEx_BreakInputConfigTypeDef is defined in the stm32h7xx_hal_tim_ex.h

Data Fields

- ***uint32_t Source***
- ***uint32_t Enable***
- ***uint32_t Polarity***

Field Documentation

- ***uint32_t TIMEx_BreakInputConfigTypeDef::Source***
  Specifies the source of the timer break input. This parameter can be a value of ***TIMEx_Break_Input_Source***
- ***uint32_t TIMEx_BreakInputConfigTypeDef::Enable***
  Specifies whether or not the break input source is enabled. This parameter can be a value of ***TIMEx_Break_Input_Source_Enable***
- ***uint32_t TIMEx_BreakInputConfigTypeDef::Polarity***
  Specifies the break input source polarity. This parameter can be a value of ***TIMEx_Break_Input_Source_Polarity*** Not relevant when analog watchdog output of the DFSDM1 used as break input source

89.2 TIMEx Firmware driver API description

The following section lists the various functions of the TIMEx library.

89.2.1 TIMER Extended features

The Timer Extended features include:

1. Complementary outputs with programmable dead-time for :
   – Output Compare
   – PWM generation (Edge and Center-aligned Mode)
   – One-pulse mode output
2. Synchronization circuit to control the timer with external signals and to interconnect several timers together.
3. Break input to put the timer output signals in reset state or in a known state.
4. Supports incremental (quadrature) encoder and hall-sensor circuitry for positioning purposes

89.2.2 How to use this driver

1. Initialize the TIM low level resources by implementing the following functions depending on the selected feature:
   – Hall Sensor output : HAL_TIMEx_HallSensor_MspInit()
2. Initialize the TIM low level resources :
   a. Enable the TIM interface clock using __HAL_RCC_TIMx_CLK_ENABLE();
   b. TIM pins configuration
      ◦ Enable the clock for the TIM GPIOs using the following function: __HAL_RCC_GPIOx_CLK_ENABLE();
      ◦ Configure these TIM pins in Alternate function mode using HAL_GPIO_Init();
3. The external Clock can be configured, if needed (the default clock is the internal clock from the APBx), using the following function: HAL_TIM_ConfigClockSource, the clock configuration should be done before any start function.
4. Configure the TIM in the desired functioning mode using one of the initialization function of this driver:
   – HAL_TIMEx_HallSensor_Init() and HAL_TIMEx_ConfigCommutEvent(): to use the Timer Hall Sensor Interface and the commutation event with the corresponding Interrupt and DMA request if needed (Note that One Timer is used to interface with the Hall sensor Interface and another Timer should be used to use the commutation event).
5. Activate the TIM peripheral using one of the start functions:
   – Complementary Output Compare : HAL_TIMEx_OCN_Start(), HAL_TIMEx_OCN_Start_DMA(), HAL_TIMEx_OCN_Start_IT()
   – Complementary PWM generation : HAL_TIMEx_PWMN_Start(), HAL_TIMEx_PWMN_Start_DMA(), HAL_TIMEx_PWMN_Start_IT()
   – Complementary One-pulse mode output : HAL_TIMEx_OnePulseN_Start(), HAL_TIMEx_OnePulseN_Start_IT()
   – Hall Sensor output : HAL_TIMEx_HallSensor_Start(), HAL_TIMEx_HallSensor_Start_DMA(), HAL_TIMEx_HallSensor_Start_IT().

89.2.3 Timer Hall Sensor functions

This section provides functions allowing to:

- Initialize and configure TIM HAL Sensor.
- De-initialize TIM HAL Sensor.
- Start the Hall Sensor Interface.
- Stop the Hall Sensor Interface.
- Start the Hall Sensor Interface and enable interrupts.
- Stop the Hall Sensor Interface and disable interrupts.
- Start the Hall Sensor Interface and enable DMA transfers.
- Stop the Hall Sensor Interface and disable DMA transfers.

This section contains the following APIs:

- ***HAL_TIMEx_HallSensor_Init()***
- ***HAL_TIMEx_HallSensor_DeInit()***
- ***HAL_TIMEx_HallSensor_MspInit()***
- ***HAL_TIMEx_HallSensor_MspDeInit()***

- *HAL_TIMEx_HallSensor_Start()*
- *HAL_TIMEx_HallSensor_Stop()*
- *HAL_TIMEx_HallSensor_Start_IT()*
- *HAL_TIMEx_HallSensor_Stop_IT()*
- *HAL_TIMEx_HallSensor_Start_DMA()*
- *HAL_TIMEx_HallSensor_Stop_DMA()*

89.2.4 Timer Complementary Output Compare functions

This section provides functions allowing to:

- Start the Complementary Output Compare/PWM.
- Stop the Complementary Output Compare/PWM.
- Start the Complementary Output Compare/PWM and enable interrupts.
- Stop the Complementary Output Compare/PWM and disable interrupts.
- Start the Complementary Output Compare/PWM and enable DMA transfers.
- Stop the Complementary Output Compare/PWM and disable DMA transfers.

This section contains the following APIs:

- *HAL_TIMEx_OCN_Start()*
- *HAL_TIMEx_OCN_Stop()*
- *HAL_TIMEx_OCN_Start_IT()*
- *HAL_TIMEx_OCN_Stop_IT()*
- *HAL_TIMEx_OCN_Start_DMA()*
- *HAL_TIMEx_OCN_Stop_DMA()*

89.2.5 Timer Complementary PWM functions

This section provides functions allowing to:

- Start the Complementary PWM.
- Stop the Complementary PWM.
- Start the Complementary PWM and enable interrupts.
- Stop the Complementary PWM and disable interrupts.
- Start the Complementary PWM and enable DMA transfers.
- Stop the Complementary PWM and disable DMA transfers.
- Start the Complementary Input Capture measurement.
- Stop the Complementary Input Capture.
- Start the Complementary Input Capture and enable interrupts.
- Stop the Complementary Input Capture and disable interrupts.
- Start the Complementary Input Capture and enable DMA transfers.
- Stop the Complementary Input Capture and disable DMA transfers.
- Start the Complementary One Pulse generation.
- Stop the Complementary One Pulse.
- Start the Complementary One Pulse and enable interrupts.
- Stop the Complementary One Pulse and disable interrupts.

This section contains the following APIs:

- *HAL_TIMEx_PWMN_Start()*
- *HAL_TIMEx_PWMN_Stop()*
- *HAL_TIMEx_PWMN_Start_IT()*
- *HAL_TIMEx_PWMN_Stop_IT()*
- *HAL_TIMEx_PWMN_Start_DMA()*
- *HAL_TIMEx_PWMN_Stop_DMA()*

89.2.6 Timer Complementary One Pulse functions

This section provides functions allowing to:

- Start the Complementary One Pulse generation.
- Stop the Complementary One Pulse.
- Start the Complementary One Pulse and enable interrupts.
- Stop the Complementary One Pulse and disable interrupts.

This section contains the following APIs:

- ***HAL_TIMEx_OnePulseN_Start()***
- ***HAL_TIMEx_OnePulseN_Stop()***
- ***HAL_TIMEx_OnePulseN_Start_IT()***
- ***HAL_TIMEx_OnePulseN_Stop_IT()***

89.2.7 Peripheral Control functions

This section provides functions allowing to:

- Configure the commutation event in case of use of the Hall sensor interface.
- Configure Output channels for OC and PWM mode.
- Configure Complementary channels, break features and dead time.
- Configure Master synchronization.
- Configure timer remapping capabilities.
- Select timer input source.
- Enable or disable channel grouping.

This section contains the following APIs:

- ***HAL_TIMEx_ConfigCommutEvent()***
- ***HAL_TIMEx_ConfigCommutEvent_IT()***
- ***HAL_TIMEx_ConfigCommutEvent_DMA()***
- ***HAL_TIMEx_MasterConfigSynchronization()***
- ***HAL_TIMEx_ConfigBreakDeadTime()***
- ***HAL_TIMEx_ConfigBreakInput()***
- ***HAL_TIMEx_RemapConfig()***
- ***HAL_TIMEx_TISelection()***
- ***HAL_TIMEx_GroupChannel5()***

89.2.8 Extended Callbacks functions

This section provides Extended TIM callback functions:

- Timer Commutation callback
- Timer Break callback

This section contains the following APIs:

- ***HAL_TIMEx_CommutCallback()***
- ***HAL_TIMEx_CommutHalfCpltCallback()***
- ***HAL_TIMEx_BreakCallback()***
- ***HAL_TIMEx_Break2Callback()***

89.2.9 Extended Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- ***HAL_TIMEx_HallSensor_GetState()***
- ***HAL_TIMEx_GetChannelNState()***

89.2.10 Detailed description of functions

HAL_TIMEx_HallSensor_Init

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Init (TIM_HandleTypeDef * htim, const TIM_HallSensor_InitTypeDef * sConfig)

Function description

Initializes the TIM Hall Sensor Interface and initialize the associated handle.

Parameters

- **htim:** TIM Hall Sensor Interface handle
- **sConfig:** TIM Hall Sensor configuration structure

Return values

- **HAL:** status

Notes

- When the timer instance is initialized in Hall Sensor Interface mode, timer channels 1 and channel 2 are reserved and cannot be used for other purpose.

HAL_TIMEx_HallSensor_DeInit

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes the TIM Hall Sensor interface.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_MspInit

Function name

void HAL_TIMEx_HallSensor_MspInit (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Hall Sensor MSP.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **None:**

HAL_TIMEx_HallSensor_MspDeInit

Function name

void HAL_TIMEx_HallSensor_MspDeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes TIM Hall Sensor MSP.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **None:**

HAL_TIMEx_HallSensor_Start

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start (TIM_HandleTypeDef * htim)

Function description

Starts the TIM Hall Sensor Interface.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_Stop

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop (TIM_HandleTypeDef * htim)

Function description

Stops the TIM Hall sensor Interface.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_Start_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start_IT (TIM_HandleTypeDef * htim)

Function description

Starts the TIM Hall Sensor Interface in interrupt mode.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop_IT (TIM_HandleTypeDef * htim)

Function description

Stops the TIM Hall Sensor Interface in interrupt mode.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start_DMA (TIM_HandleTypeDef * htim, uint32_t * pData, uint16_t Length)

Function description

Starts the TIM Hall Sensor Interface in DMA mode.

Parameters

- **htim:** TIM Hall Sensor Interface handle
- **pData:** The destination Buffer address.
- **Length:** The length of data to be transferred from TIM peripheral to memory.

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop_DMA (TIM_HandleTypeDef * htim)

Function description

Stops the TIM Hall Sensor Interface in DMA mode.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **HAL:** status

HAL_TIMEx_OCN_Start

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Output Compare signal generation on the complementary output.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected

Return values

- **HAL:** status

HAL_TIMEx_OCN_Stop

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Output Compare signal generation on the complementary output.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected

Return values

- **HAL:** status

HAL_TIMEx_OCN_Start_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Output Compare signal generation in interrupt mode on the complementary output.

Parameters

- **htim:** TIM OC handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected

Return values

- **HAL:** status

HAL_TIMEx_OCN_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Output Compare signal generation in interrupt mode on the complementary output.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected

Return values

- **HAL:** status

HAL_TIMEx_OCN_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, const uint32_t * pData, uint16_t Length)

Function description

Starts the TIM Output Compare signal generation in DMA mode on the complementary output.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to TIM peripheral

Return values

- **HAL:** status

HAL_TIMEx_OCN_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Output Compare signal generation in DMA mode on the complementary output.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected

Return values

- **HAL:** status

HAL_TIMEx_PWMN_Start

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the PWM signal generation on the complementary output.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected

Return values

- **HAL:** status

HAL_TIMEx_PWMN_Stop

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the PWM signal generation on the complementary output.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected

Return values

- **HAL:** status

HAL_TIMEx_PWMN_Start_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the PWM signal generation in interrupt mode on the complementary output.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected

Return values

- **HAL:** status

HAL_TIMEx_PWMN_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the PWM signal generation in interrupt mode on the complementary output.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected

Return values

- **HAL:** status

HAL_TIMEx_PWMN_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, const uint32_t * pData, uint16_t Length)

Function description

Starts the TIM PWM signal generation in DMA mode on the complementary output.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to TIM peripheral

Return values

- **HAL:** status

HAL_TIMEx_PWMN_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM PWM signal generation in DMA mode on the complementary output.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected

Return values

- **HAL:** status

HAL_TIMEx_OnePulseN_Start

Function name

HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Start (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Starts the TIM One Pulse signal generation on the complementary output.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** pulse output channel to enable This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

Notes

- OutputChannel must match the pulse output channel chosen when calling HAL_TIM_OnePulse_ConfigChannel().

HAL_TIMEx_OnePulseN_Stop

Function name

HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Stop (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Stops the TIM One Pulse signal generation on the complementary output.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** pulse output channel to disable This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

Notes

- OutputChannel must match the pulse output channel chosen when calling HAL_TIM_OnePulse_ConfigChannel().

HAL_TIMEx_OnePulseN_Start_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Start_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Starts the TIM One Pulse signal generation in interrupt mode on the complementary channel.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** pulse output channel to enable This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

Notes

- OutputChannel must match the pulse output channel chosen when calling HAL_TIM_OnePulse_ConfigChannel().

HAL_TIMEx_OnePulseN_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Stops the TIM One Pulse signal generation in interrupt mode on the complementary channel.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** pulse output channel to disable This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

Notes

- OutputChannel must match the pulse output channel chosen when calling HAL_TIM_OnePulse_ConfigChannel().

HAL_TIMEx_ConfigCommutEvent

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigCommutEvent (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)

Function description

Configure the TIM commutation event sequence.

Parameters

- **htim:** TIM handle
- **InputTrigger:** the Internal trigger corresponding to the Timer Interfacing with the Hall sensor This parameter can be one of the following values:
  - TIM_TS_ITR0: Internal trigger 0 selected
  - TIM_TS_ITR1: Internal trigger 1 selected
  - TIM_TS_ITR2: Internal trigger 2 selected
  - TIM_TS_ITR3: Internal trigger 3 selected
  - TIM_TS_ITR12: Internal trigger 12 selected (*)
  - TIM_TS_ITR13: Internal trigger 13 selected (*)
  - TIM_TS_NONE: No trigger is needed
- **CommutationSource:** the Commutation Event source This parameter can be one of the following values:
  - TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface Timer
  - TIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit

Return values

- **HAL:** status

Notes

- This function is mandatory to use the commutation event in order to update the configuration at each commutation detection on the TRGI input of the Timer, the typical use of this feature is with the use of another Timer(interface Timer) configured in Hall sensor interface, this interface Timer will generate the commutation at its TRGO output (connected to Timer used in this function) each time the TI1 of the Interface Timer detect a commutation at its input TI1.

HAL_TIMEx_ConfigCommutEvent_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigCommutEvent_IT (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)

Function description

Configure the TIM commutation event sequence with interrupt.

Parameters

- **htim:** TIM handle
- **InputTrigger:** the Internal trigger corresponding to the Timer Interfacing with the Hall sensor This parameter can be one of the following values:
  - TIM_TS_ITR0: Internal trigger 0 selected
  - TIM_TS_ITR1: Internal trigger 1 selected
  - TIM_TS_ITR2: Internal trigger 2 selected
  - TIM_TS_ITR3: Internal trigger 3 selected
  - TIM_TS_ITR2: Internal trigger 12 selected (*)
  - TIM_TS_ITR3: Internal trigger 13 selected (*)
  - TIM_TS_NONE: No trigger is needed
- **CommutationSource:** the Commutation Event source This parameter can be one of the following values:
  - TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface Timer
  - TIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit

Return values

- **HAL:** status

Notes

- This function is mandatory to use the commutation event in order to update the configuration at each commutation detection on the TRGI input of the Timer, the typical use of this feature is with the use of another Timer(interface Timer) configured in Hall sensor interface, this interface Timer will generate the commutation at its TRGO output (connected to Timer used in this function) each time the TI1 of the Interface Timer detect a commutation at its input TI1.

HAL_TIMEx_ConfigCommutEvent_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigCommutEvent_DMA (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)

Function description

Configure the TIM commutation event sequence with DMA.

Parameters

- **htim:** TIM handle
- **InputTrigger:** the Internal trigger corresponding to the Timer Interfacing with the Hall sensor This parameter can be one of the following values:
  - TIM_TS_ITR0: Internal trigger 0 selected
  - TIM_TS_ITR1: Internal trigger 1 selected
  - TIM_TS_ITR2: Internal trigger 2 selected
  - TIM_TS_ITR3: Internal trigger 3 selected
  - TIM_TS_ITR2: Internal trigger 12 selected (*)
  - TIM_TS_ITR3: Internal trigger 13 selected (*)
  - TIM_TS_NONE: No trigger is needed

  (*) Value not defined in all devices.
- **CommutationSource:** the Commutation Event source This parameter can be one of the following values:
  - TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface Timer
  - TIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit

Return values

- **HAL:** status

Notes

- This function is mandatory to use the commutation event in order to update the configuration at each commutation detection on the TRGI input of the Timer, the typical use of this feature is with the use of another Timer(interface Timer) configured in Hall sensor interface, this interface Timer will generate the commutation at its TRGO output (connected to Timer used in this function) each time the TI1 of the Interface Timer detect a commutation at its input TI1.
- The user should configure the DMA in his own software, in This function only the COMDE bit is set

HAL_TIMEx_MasterConfigSynchronization

Function name

HAL_StatusTypeDef HAL_TIMEx_MasterConfigSynchronization (TIM_HandleTypeDef * htim, const TIM_MasterConfigTypeDef * sMasterConfig)

Function description

Configures the TIM in master mode.

Parameters

- **htim:** TIM handle.
- **sMasterConfig:** pointer to a TIM_MasterConfigTypeDef structure that contains the selected trigger output (TRGO) and the Master/Slave mode.

Return values

- **HAL:** status

HAL_TIMEx_ConfigBreakDeadTime

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigBreakDeadTime (TIM_HandleTypeDef * htim, const TIM_BreakDeadTimeConfigTypeDef * sBreakDeadTimeConfig)

Function description

Configures the Break feature, dead time, Lock level, OSSI/OSSR State and the AOE(automatic output enable).

Parameters

- **htim:** TIM handle
- **sBreakDeadTimeConfig:** pointer to a TIM_ConfigBreakDeadConfigTypeDef structure that contains the BDTR Register configuration information for the TIM peripheral.

Return values

- **HAL:** status

Notes

- Interrupts can be generated when an active level is detected on the break input, the break 2 input or the system break input. Break interrupt can be enabled by calling the __HAL_TIM_ENABLE_IT macro.

HAL_TIMEx_ConfigBreakInput

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigBreakInput (TIM_HandleTypeDef * htim, uint32_t BreakInput, const TIMEx_BreakInputConfigTypeDef * sBreakInputConfig)

Function description

Configures the break input source.

Parameters

- **htim:** TIM handle.
- **BreakInput:** Break input to configure This parameter can be one of the following values:
  - TIM_BREAKINPUT_BRK: Timer break input
  - TIM_BREAKINPUT_BRK2: Timer break 2 input
- **sBreakInputConfig:** Break input source configuration

Return values

- **HAL:** status

HAL_TIMEx_GroupChannel5

Function name

HAL_StatusTypeDef HAL_TIMEx_GroupChannel5 (TIM_HandleTypeDef * htim, uint32_t Channels)

Function description

Group channel 5 and channel 1, 2 or 3.

Parameters

- **htim:** TIM handle.
- **Channels:** specifies the reference signal(s) the OC5REF is combined with. This parameter can be any combination of the following values: TIM_GROUPCH5_NONE: No effect of OC5REF on OC1REFC, OC2REFC and OC3REFC TIM_GROUPCH5_OC1REFC: OC1REFC is the logical AND of OC1REFC and OC5REF TIM_GROUPCH5_OC2REFC: OC2REFC is the logical AND of OC2REFC and OC5REF TIM_GROUPCH5_OC3REFC: OC3REFC is the logical AND of OC3REFC and OC5REF

Return values

- **HAL:** status

HAL_TIMEx_RemapConfig

Function name

HAL_StatusTypeDef HAL_TIMEx_RemapConfig (TIM_HandleTypeDef * htim, uint32_t Remap)

Function description

Configures the TIMx Remapping input capabilities.

Parameters

- **htim:** TIM handle.

- **Remap:** specifies the TIM remapping source. For TIM1, the parameter is one of the following values:
  - TIM_TIM1_ETR_GPIO: TIM1_ETR is connected to GPIO
  - TIM_TIM1_ETR_COMP1: TIM1_ETR is connected to COMP1 output
  - TIM_TIM1_ETR_COMP2: TIM1_ETR is connected to COMP2 output
  - TIM_TIM1_ETR_ADC1_AWD1: TIM1_ETR is connected to ADC1 AWD1
  - TIM_TIM1_ETR_ADC1_AWD2: TIM1_ETR is connected to ADC1 AWD2
  - TIM_TIM1_ETR_ADC1_AWD3: TIM1_ETR is connected to ADC1 AWD3
  - TIM_TIM1_ETR_ADC3_AWD1: TIM1_ETR is connected to ADC3 AWD1
  - TIM_TIM1_ETR_ADC3_AWD2: TIM1_ETR is connected to ADC3 AWD2
  - TIM_TIM1_ETR_ADC3_AWD3: TIM1_ETR is connected to ADC3 AWD3

  For TIM2, the parameter is one of the following values:
  - TIM_TIM2_ETR_GPIO: TIM2_ETR is connected to GPIO
  - TIM_TIM2_ETR_COMP1: TIM2_ETR is connected to COMP1 output
  - TIM_TIM2_ETR_COMP2: TIM2_ETR is connected to COMP2 output
  - TIM_TIM2_ETR_LSE: TIM2_ETR is connected to LSE
  - TIM_TIM2_ETR_SAI1_FSA: TIM2_ETR is connected to SAI1 FS_A
  - TIM_TIM2_ETR_SAI1_FSB: TIM2_ETR is connected to SAI1 FS_B

  For TIM3, the parameter is one of the following values:
  - TIM_TIM3_ETR_GPIO: TIM3_ETR is connected to GPIO
  - TIM_TIM3_ETR_COMP1: TIM3_ETR is connected to COMP1 output

  For TIM5, the parameter is one of the following values:
  - TIM_TIM5_ETR_GPIO: TIM5_ETR is connected to GPIO
  - TIM_TIM5_ETR_SAI2_FSA: TIM5_ETR is connected to SAI2 FS_A (*)
  - TIM_TIM5_ETR_SAI2_FSB: TIM5_ETR is connected to SAI2 FS_B (*)
  - TIM_TIM5_ETR_SAI4_FSA: TIM5_ETR is connected to SAI2 FS_A (*)
  - TIM_TIM5_ETR_SAI4_FSB: TIM5_ETR is connected to SAI2 FS_B (*)

  For TIM8, the parameter is one of the following values:
  - TIM_TIM8_ETR_GPIO: TIM8_ETR is connected to GPIO
  - TIM_TIM8_ETR_COMP1: TIM8_ETR is connected to COMP1 output
  - TIM_TIM8_ETR_COMP2: TIM8_ETR is connected to COMP2 output
  - TIM_TIM8_ETR_ADC2_AWD1: TIM8_ETR is connected to ADC2 AWD1
  - TIM_TIM8_ETR_ADC2_AWD2: TIM8_ETR is connected to ADC2 AWD2
  - TIM_TIM8_ETR_ADC2_AWD3: TIM8_ETR is connected to ADC2 AWD3
  - TIM_TIM8_ETR_ADC3_AWD1: TIM8_ETR is connected to ADC3 AWD1
  - TIM_TIM8_ETR_ADC3_AWD2: TIM8_ETR is connected to ADC3 AWD2
  - TIM_TIM8_ETR_ADC3_AWD3: TIM8_ETR is connected to ADC3 AWD3

  For TIM23, the parameter is one of the following values: (*)
  - TIM_TIM23_ETR_GPIO TIM23_ETR is connected to GPIO
  - TIM_TIM23_ETR_COMP1 TIM23_ETR is connected to COMP1 output
  - TIM_TIM23_ETR_COMP2 TIM23_ETR is connected to COMP2 output

  For TIM24, the parameter is one of the following values: (*)
  - TIM_TIM24_ETR_GPIO TIM24_ETR is connected to GPIO
  - TIM_TIM24_ETR_SAI4_FSA TIM24_ETR is connected to SAI4 FS_A
  - TIM_TIM24_ETR_SAI4_FSB TIM24_ETR is connected to SAI4 FS_B
  - TIM_TIM24_ETR_SAI1_FSA TIM24_ETR is connected to SAI1 FS_A
  - TIM_TIM24_ETR_SAI1_FSB TIM24_ETR is connected to SAI1 FS_B

  (*) Value not defined in all devices.

Return values

- **HAL:** status

HAL_TIMEx_TISelection

Function name

HAL_StatusTypeDef HAL_TIMEx_TISelection (TIM_HandleTypeDef * htim, uint32_t TISelection, uint32_t Channel)

Function description

Select the timer input source.

Parameters

- **htim:** TIM handle.
- **Channel:** specifies the TIM Channel This parameter can be one of the following values:
  - TIM_CHANNEL_1: TI1 input channel
  - TIM_CHANNEL_2: TI2 input channel
  - TIM_CHANNEL_3: TIM Channel 3
  - TIM_CHANNEL_4: TIM Channel 4

- **TISelection:** parameter of the TIM_TISelectionStruct structure is detailed as follows: For TIM1, the parameter is one of the following values:
  - TIM_TIM1_TI1_GPIO: TIM1 TI1 is connected to GPIO
  - TIM_TIM1_TI1_COMP1: TIM1 TI1 is connected to COMP1 output

  For TIM2, the parameter is one of the following values:
  - TIM_TIM2_TI4_GPIO: TIM2 TI4 is connected to GPIO
  - TIM_TIM2_TI4_COMP1: TIM2 TI4 is connected to COMP1 output
  - TIM_TIM2_TI4_COMP2: TIM2 TI4 is connected to COMP2 output
  - TIM_TIM2_TI4_COMP1_COMP2: TIM2 TI4 is connected to logical OR between COMP1 and COMP2 output

  For TIM3, the parameter is one of the following values:
  - TIM_TIM3_TI1_GPIO: TIM3 TI1 is connected to GPIO
  - TIM_TIM3_TI1_COMP1: TIM3 TI1 is connected to COMP1 output
  - TIM_TIM3_TI1_COMP2: TIM3 TI1 is connected to COMP2 output
  - TIM_TIM3_TI1_COMP1_COMP2: TIM3 TI1 is connected to logical OR between COMP1 and COMP2 output

  For TIM5, the parameter is one of the following values:
  - TIM_TIM5_TI1_GPIO: TIM5 TI1 is connected to GPIO
  - TIM_TIM5_TI1_CAN_TMP: TIM5 TI1 is connected to CAN TMP
  - TIM_TIM5_TI1_CAN_RTP: TIM5 TI1 is connected to CAN RTP

  For TIM8, the parameter is one of the following values:
  - TIM_TIM8_TI1_GPIO: TIM8 TI1 is connected to GPIO
  - TIM_TIM8_TI1_COMP2: TIM8 TI1 is connected to COMP2 output

  For TIM12, the parameter can have the following values: (*)
  - TIM_TIM12_TI1_GPIO: TIM12 TI1 is connected to GPIO
  - TIM_TIM12_TI1_SPDIF_FS: TIM12 TI1 is connected to SPDIF FS

  For TIM15, the parameter is one of the following values:
  - TIM_TIM15_TI1_GPIO: TIM15 TI1 is connected to GPIO
  - TIM_TIM15_TI1_TIM2: TIM15 TI1 is connected to TIM2 CH1
  - TIM_TIM15_TI1_TIM3: TIM15 TI1 is connected to TIM3 CH1
  - TIM_TIM15_TI1_TIM4: TIM15 TI1 is connected to TIM4 CH1
  - TIM_TIM15_TI1_LSE: TIM15 TI1 is connected to LSE
  - TIM_TIM15_TI1_CSI: TIM15 TI1 is connected to CSI
  - TIM_TIM15_TI1_MCO2: TIM15 TI1 is connected to MCO2
  - TIM_TIM15_TI2_GPIO: TIM15 TI2 is connected to GPIO
  - TIM_TIM15_TI2_TIM2: TIM15 TI2 is connected to TIM2 CH2
  - TIM_TIM15_TI2_TIM3: TIM15 TI2 is connected to TIM3 CH2
  - TIM_TIM15_TI2_TIM4: TIM15 TI2 is connected to TIM4 CH2

  For TIM16, the parameter can have the following values:
  - TIM_TIM16_TI1_GPIO: TIM16 TI1 is connected to GPIO
  - TIM_TIM16_TI1_LSI: TIM16 TI1 is connected to LSI
  - TIM_TIM16_TI1_LSE: TIM16 TI1 is connected to LSE
  - TIM_TIM16_TI1_RTC: TIM16 TI1 is connected to RTC wakeup interrupt

  For TIM17, the parameter can have the following values:
  - TIM_TIM17_TI1_GPIO: TIM17 TI1 is connected to GPIO
  - TIM_TIM17_TI1_SPDIF_FS: TIM17 TI1 is connected to SPDIF FS (*)
  - TIM_TIM17_TI1_HSE_1MHZ: TIM17 TI1 is connected to HSE 1MHz
  - TIM_TIM17_TI1_MCO1: TIM17 TI1 is connected to MCO1

  For TIM23, the parameter can have the following values: (*)
  - TIM_TIM23_TI4_GPIO TIM23_TI4 is connected to GPIO
  - TIM_TIM23_TI4_COMP1 TIM23_TI4 is connected to COMP1 output

-

Return values

- **HAL:** status

HAL_TIMEx_CommutCallback

Function name

void HAL_TIMEx_CommutCallback (TIM_HandleTypeDef * htim)

Function description

Hall commutation changed callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_CommutHalfCpltCallback

Function name

void HAL_TIMEx_CommutHalfCpltCallback (TIM_HandleTypeDef * htim)

Function description

Hall commutation changed half complete callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_BreakCallback

Function name

void HAL_TIMEx_BreakCallback (TIM_HandleTypeDef * htim)

Function description

Hall Break detection callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_Break2Callback

Function name

void HAL_TIMEx_Break2Callback (TIM_HandleTypeDef * htim)

Function description

Hall Break2 detection callback in non blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_HallSensor_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIMEx_HallSensor_GetState (const TIM_HandleTypeDef * htim)

Function description

Return the TIM Hall Sensor interface handle state.

Parameters

- **htim:** TIM Hall Sensor handle

Return values

- **HAL:** state

HAL_TIMEx_GetChannelNState

Function name

HAL_TIM_ChannelStateTypeDef HAL_TIMEx_GetChannelNState (const TIM_HandleTypeDef * htim, uint32_t ChannelN)

Function description

Return actual state of the TIM complementary channel.

Parameters

- **htim:** TIM handle
- **ChannelN:** TIM Complementary channel This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1
  - TIM_CHANNEL_2: TIM Channel 2
  - TIM_CHANNEL_3: TIM Channel 3

Return values

- **TIM:** Complementary channel state

TIMEx_DMACommutationCplt

Function name

void TIMEx_DMACommutationCplt (DMA_HandleTypeDef * hdma)

Function description

TIM DMA Commutation callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

TIMEx_DMACommutationHalfCplt

Function name

void TIMEx_DMACommutationHalfCplt (DMA_HandleTypeDef * hdma)

Function description

TIM DMA Commutation half complete callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

89.3 TIMEx Firmware driver defines

The following section lists the various define and macros of the module.

89.3.1 TIMEx

TIMEx

TIM Extended Break input

TIM_BREAKINPUT_BRK

Timer break input

TIM_BREAKINPUT_BRK2

Timer break2 input

TIM Extended Break input source

TIM_BREAKINPUTSOURCE_BKIN

An external source (GPIO) is connected to the BKIN pin

TIM_BREAKINPUTSOURCE_COMP1

The COMP1 output is connected to the break input

TIM_BREAKINPUTSOURCE_COMP2

The COMP2 output is connected to the break input

TIM_BREAKINPUTSOURCE_DFSDM1

The analog watchdog output of the DFSDM1 peripheral is connected to the break input

TIM Extended Break input source enabling

TIM_BREAKINPUTSOURCE_DISABLE

Break input source is disabled

TIM_BREAKINPUTSOURCE_ENABLE

Break input source is enabled

TIM Extended Break input polarity

TIM_BREAKINPUTSOURCE_POLARITY_LOW

Break input source is active low

TIM_BREAKINPUTSOURCE_POLARITY_HIGH

Break input source is active_high

TIM Extended Remapping

TIM_TIM1_ETR_GPIO

TIM1_ETR is connected to GPIO

TIM_TIM1_ETR_COMP1

TIM1_ETR is connected to COMP1 OUT

TIM_TIM1_ETR_COMP2

TIM1_ETR is connected to COMP2 OUT

TIM_TIM1_ETR_ADC1_AWD1

TIM1_ETR is connected to ADC1 AWD1

TIM_TIM1_ETR_ADC1_AWD2

TIM1_ETR is connected to ADC1 AWD2

TIM_TIM1_ETR_ADC1_AWD3

TIM1_ETR is connected to ADC1 AWD3

TIM_TIM1_ETR_ADC3_AWD1

TIM1_ETR is connected to ADC3 AWD1

TIM_TIM1_ETR_ADC3_AWD2

TIM1_ETR is connected to ADC3 AWD2

TIM_TIM1_ETR_ADC3_AWD3

TIM1_ETR is connected to ADC3 AWD3

TIM_TIM8_ETR_GPIO

TIM8_ETR is connected to GPIO

TIM_TIM8_ETR_COMP1

TIM8_ETR is connected to COMP1 OUT

TIM_TIM8_ETR_COMP2

TIM8_ETR is connected to COMP2 OUT

TIM_TIM8_ETR_ADC2_AWD1

TIM8_ETR is connected to ADC2 AWD1

TIM_TIM8_ETR_ADC2_AWD2

TIM8_ETR is connected to ADC2 AWD2

TIM_TIM8_ETR_ADC2_AWD3

TIM8_ETR is connected to ADC2 AWD3

TIM_TIM8_ETR_ADC3_AWD1

TIM8_ETR is connected to ADC3 AWD1

TIM_TIM8_ETR_ADC3_AWD2

TIM8_ETR is connected to ADC3 AWD2

TIM_TIM8_ETR_ADC3_AWD3

TIM8_ETR is connected to ADC3 AWD3

TIM_TIM2_ETR_GPIO

TIM2_ETR is connected to GPIO

TIM_TIM2_ETR_COMP1

TIM2_ETR is connected to COMP1 OUT

TIM_TIM2_ETR_COMP2

TIM2_ETR is connected to COMP2 OUT

TIM_TIM2_ETR_RCC_LSE

TIM2_ETR is connected to RCC LSE

TIM_TIM2_ETR_SAI1_FSA

TIM2_ETR is connected to SAI1 FS_A

TIM_TIM2_ETR_SAI1_FSB

TIM2_ETR is connected to SAI1 FS_B

TIM_TIM3_ETR_GPIO

TIM3_ETR is connected to GPIO

TIM_TIM3_ETR_COMP1

TIM3_ETR is connected to COMP1 OUT

TIM_TIM5_ETR_GPIO

TIM5_ETR is connected to GPIO

TIM_TIM5_ETR_SAI2_FSA

TIM5_ETR is connected to SAI2 FS_A

TIM_TIM5_ETR_SAI2_FSB

TIM5_ETR is connected to SAI2 FS_B

TIM_TIM5_ETR_SAI4_FSA

TIM5_ETR is connected to SAI4 FS_A

TIM_TIM5_ETR_SAI4_FSB

TIM5_ETR is connected to SAI4 FS_B

TIM_TIM23_ETR_GPIO

TIM23_ETR is connected to GPIO

TIM_TIM23_ETR_COMP1

TIM23_ETR is connected to COMP1 OUT

TIM_TIM23_ETR_COMP2

TIM23_ETR is connected to COMP2 OUT

TIM_TIM24_ETR_GPIO

TIM24_ETR is connected to GPIO

TIM_TIM24_ETR_SAI4_FSA

TIM24_ETR is connected to SAI4 FS_A

TIM_TIM24_ETR_SAI4_FSB

TIM24_ETR is connected to SAI4 FS_B

TIM_TIM24_ETR_SAI1_FSA

TIM24_ETR is connected to SAI1 FS_A

TIM_TIM24_ETR_SAI1_FSB

TIM24_ETR is connected to SAI1 FS_B

TIM Extended Timer input selection

TIM_TIM1_TI1_GPIO

TIM1_TI1 is connected to GPIO

TIM_TIM1_TI1_COMP1

TIM1_TI1 is connected to COMP1 OUT

TIM_TIM8_TI1_GPIO

TIM8_TI1 is connected to GPIO

TIM_TIM8_TI1_COMP2

TIM8_TI1 is connected to COMP2 OUT

TIM_TIM2_TI4_GPIO

TIM2_TI4 is connected to GPIO

TIM_TIM2_TI4_COMP1

TIM2_TI4 is connected to COMP1 OUT

TIM_TIM2_TI4_COMP2

TIM2_TI4 is connected to COMP2 OUT

TIM_TIM2_TI4_COMP1_COMP2

TIM2_TI4 is connected to COMP2 OUT OR COMP2 OUT

TIM_TIM3_TI1_GPIO

TIM3_TI1 is connected to GPIO

TIM_TIM3_TI1_COMP1

TIM3_TI1 is connected to COMP1 OUT

TIM_TIM3_TI1_COMP2

TIM3_TI1 is connected to COMP2 OUT

TIM_TIM3_TI1_COMP1_COMP2

TIM3_TI1 is connected to COMP1 OUT or COMP2 OUT

TIM_TIM5_TI1_GPIO

TIM5_TI1 is connected to GPIO

TIM_TIM5_TI1_CAN_TMP

TIM5_TI1 is connected to CAN TMP

TIM_TIM5_TI1_CAN_RTP

TIM5_TI1 is connected to CAN RTP

TIM_TIM12_TI1_GPIO

TIM12 TI1 is connected to GPIO

TIM_TIM12_TI1_SPDIF_FS

TIM12 TI1 is connected to SPDIF FS

TIM_TIM15_TI1_GPIO

TIM15_TI1 is connected to GPIO

TIM_TIM15_TI1_TIM2_CH1

TIM15_TI1 is connected to TIM2 CH1

TIM_TIM15_TI1_TIM3_CH1

TIM15_TI1 is connected to TIM3 CH1

TIM_TIM15_TI1_TIM4_CH1

TIM15_TI1 is connected to TIM4 CH1

TIM_TIM15_TI1_RCC_LSE

TIM15_TI1 is connected to RCC LSE

TIM_TIM15_TI1_RCC_CSI

TIM15_TI1 is connected to RCC CSI

TIM_TIM15_TI1_RCC_MCO2

TIM15_TI1 is connected to RCC MCO2

TIM_TIM15_TI2_GPIO

TIM15_TI2 is connected to GPIO

TIM_TIM15_TI2_TIM2_CH2

TIM15_TI2 is connected to TIM2 CH2

TIM_TIM15_TI2_TIM3_CH2

TIM15_TI2 is connected to TIM3 CH2

TIM_TIM15_TI2_TIM4_CH2

TIM15_TI2 is connected to TIM4 CH2

TIM_TIM16_TI1_GPIO

TIM16 TI1 is connected to GPIO

TIM_TIM16_TI1_RCC_LSI

TIM16 TI1 is connected to RCC LSI

TIM_TIM16_TI1_RCC_LSE

TIM16 TI1 is connected to RCC LSE

TIM_TIM16_TI1_WKUP_IT

TIM16 TI1 is connected to WKUP_IT

TIM_TIM17_TI1_GPIO

TIM17 TI1 is connected to GPIO

TIM_TIM17_TI1_SPDIF_FS

TIM17 TI1 is connected to SPDIF FS

TIM_TIM17_TI1_RCC_HSE1MHZ

TIM17 TI1 is connected to RCC HSE 1Mhz

TIM_TIM17_TI1_RCC_MCO1

TIM17 TI1 is connected to RCC MCO1

TIM_TIM23_TI4_GPIO

TIM23_TI4 is connected to GPIO

TIM_TIM23_TI4_COMP1

TIM23_TI4 is connected to COMP1 OUT

TIM_TIM23_TI4_COMP2

TIM23_TI4 is connected to COMP2 OUT

TIM_TIM23_TI4_COMP1_COMP2

TIM23_TI4 is connected to COMP1 OUT or COMP2 OUT

TIM_TIM24_TI1_GPIO

TIM24_TI1 is connected to GPIO

TIM_TIM24_TI1_CAN_TMP

TIM24_TI1 is connected to CAN TMP

TIM_TIM24_TI1_CAN_RTP

TIM24_TI1 is connected to CAN RTP

TIM_TIM24_TI1_CAN_SOC

TIM24_TI1 is connected to CAN SOC

90 HAL UART Generic Driver

90.1 UART Firmware driver registers structures

90.1.1 UART_InitTypeDef

UART_InitTypeDef is defined in the stm32h7xx_hal_uart.h

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t Mode***
- ***uint32_t HwFlowCtl***
- ***uint32_t OverSampling***
- ***uint32_t OneBitSampling***
- ***uint32_t ClockPrescaler***

Field Documentation

- ***uint32_t UART_InitTypeDef::BaudRate***

  This member configures the UART communication baud rate. The baud rate register is computed using the following formula: LPUART: ======= Baud Rate Register = ((256 * lpuart_ker_ckpres) / ((huart->Init.BaudRate))) where lpuart_ker_ck_pres is the UART input clock divided by a prescaler UART: =====

  – If oversampling is 16 or in LIN mode, Baud Rate Register = ((uart_ker_ckpres) / ((huart->Init.BaudRate)))

  – If oversampling is 8, Baud Rate Register[15:4] = ((2 * uart_ker_ckpres) / ((huart->Init.BaudRate)))[15:4] Baud Rate Register[3] = 0 Baud Rate Register[2:0] = (((2 * uart_ker_ckpres) / ((huart->Init.BaudRate)))[3:0]) >> 1 where uart_ker_ck_pres is the UART input clock divided by a prescaler

- ***uint32_t UART_InitTypeDef::WordLength***

  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***UARTEx_Word_Length***.

- ***uint32_t UART_InitTypeDef::StopBits***

  Specifies the number of stop bits transmitted. This parameter can be a value of ***UART_Stop_Bits***.

- ***uint32_t UART_InitTypeDef::Parity***

  Specifies the parity mode. This parameter can be a value of ***UART_Parity***

  Note:

  – When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).

- ***uint32_t UART_InitTypeDef::Mode***

  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***UART_Mode***.

- ***uint32_t UART_InitTypeDef::HwFlowCtl***

  Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of ***UART_Hardware_Flow_Control***.

- ***uint32_t UART_InitTypeDef::OverSampling***

  Specifies whether the Over sampling 8 is enabled or disabled, to achieve higher speed (up to f_PCLK/8). This parameter can be a value of ***UART_Over_Sampling***.

- ***uint32_t UART_InitTypeDef::OneBitSampling***

  Specifies whether a single sample or three samples' majority vote is selected. Selecting the single sample method increases the receiver tolerance to clock deviations. This parameter can be a value of ***UART_OneBit_Sampling***.

- ***uint32_t UART_InitTypeDef::ClockPrescaler***

  Specifies the prescaler value used to divide the UART clock source. This parameter can be a value of ***UART_ClockPrescaler***.

90.1.2 UART_AdvFeatureInitTypeDef

UART_AdvFeatureInitTypeDef is defined in the stm32h7xx_hal_uart.h

Data Fields

- ***uint32_t AdvFeatureInit***
- ***uint32_t TxPinLevelInvert***
- ***uint32_t RxPinLevelInvert***
- ***uint32_t DataInvert***
- ***uint32_t Swap***
- ***uint32_t OverrunDisable***
- ***uint32_t DMADisableonRxError***
- ***uint32_t AutoBaudRateEnable***
- ***uint32_t AutoBaudRateMode***
- ***uint32_t MSBFirst***

Field Documentation

- ***uint32_t UART_AdvFeatureInitTypeDef::AdvFeatureInit***

  Specifies which advanced UART features is initialized. Several Advanced Features may be initialized at the same time . This parameter can be a value of ***UART_Advanced_Features_Initialization_Type***.
- ***uint32_t UART_AdvFeatureInitTypeDef::TxPinLevelInvert***

  Specifies whether the TX pin active level is inverted. This parameter can be a value of ***UART_Tx_Inv***.
- ***uint32_t UART_AdvFeatureInitTypeDef::RxPinLevelInvert***

  Specifies whether the RX pin active level is inverted. This parameter can be a value of ***UART_Rx_Inv***.
- ***uint32_t UART_AdvFeatureInitTypeDef::DataInvert***

  Specifies whether data are inverted (positive/direct logic vs negative/inverted logic). This parameter can be a value of ***UART_Data_Inv***.
- ***uint32_t UART_AdvFeatureInitTypeDef::Swap***

  Specifies whether TX and RX pins are swapped. This parameter can be a value of ***UART_Rx_Tx_Swap***.
- ***uint32_t UART_AdvFeatureInitTypeDef::OverrunDisable***

  Specifies whether the reception overrun detection is disabled. This parameter can be a value of ***UART_Overrun_Disable***.
- ***uint32_t UART_AdvFeatureInitTypeDef::DMADisableonRxError***

  Specifies whether the DMA is disabled in case of reception error. This parameter can be a value of ***UART_DMA_Disable_on_Rx_Error***.
- ***uint32_t UART_AdvFeatureInitTypeDef::AutoBaudRateEnable***

  Specifies whether auto Baud rate detection is enabled. This parameter can be a value of ***UART_AutoBaudRate_Enable***.
- ***uint32_t UART_AdvFeatureInitTypeDef::AutoBaudRateMode***

  If auto Baud rate detection is enabled, specifies how the rate detection is carried out. This parameter can be a value of ***UART_AutoBaud_Rate_Mode***.
- ***uint32_t UART_AdvFeatureInitTypeDef::MSBFirst***

  Specifies whether MSB is sent first on UART line. This parameter can be a value of ***UART_MSB_First***.

90.1.3 __UART_HandleTypeDef

__UART_HandleTypeDef is defined in the stm32h7xx_hal_uart.h

Data Fields

- ***USART_TypeDef * Instance***
- ***UART_InitTypeDef Init***
- ***UART_AdvFeatureInitTypeDef AdvancedInit***

- ***const uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***uint16_t Mask***
- ***uint32_t FifoMode***
- ***uint16_t NbRxDataToProcess***
- ***uint16_t NbTxDataToProcess***
- ***__IO HAL_UART_RxTypeTypeDef ReceptionType***
- ***__IO HAL_UART_RxEventTypeTypeDef RxEventType***
- ***void(* RxISR***
- ***void(* TxISR***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_UART_StateTypeDef gState***
- ***__IO HAL_UART_StateTypeDef RxState***
- ***__IO uint32_t ErrorCode***
- ***void(* TxHalfCpltCallback***
- ***void(* TxCpltCallback***
- ***void(* RxHalfCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* AbortTransmitCpltCallback***
- ***void(* AbortReceiveCpltCallback***
- ***void(* WakeupCallback***
- ***void(* RxFifoFullCallback***
- ***void(* TxFifoEmptyCallback***
- ***void(* RxEventCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***USART_TypeDef* __UART_HandleTypeDef::Instance***
  UART registers base address
- ***UART_InitTypeDef __UART_HandleTypeDef::Init***
  UART communication parameters
- ***UART_AdvFeatureInitTypeDef __UART_HandleTypeDef::AdvancedInit***
  UART Advanced Features initialization parameters
- ***const uint8_t* __UART_HandleTypeDef::pTxBuffPtr***
  Pointer to UART Tx transfer Buffer
- ***uint16_t __UART_HandleTypeDef::TxXferSize***
  UART Tx Transfer size
- ***__IO uint16_t __UART_HandleTypeDef::TxXferCount***
  UART Tx Transfer Counter
- ***uint8_t* __UART_HandleTypeDef::pRxBuffPtr***
  Pointer to UART Rx transfer Buffer

- ***uint16_t __UART_HandleTypeDef::RxXferSize***
  UART Rx Transfer size
- ***__IO uint16_t __UART_HandleTypeDef::RxXferCount***
  UART Rx Transfer Counter
- ***uint16_t __UART_HandleTypeDef::Mask***
  UART Rx RDR register mask
- ***uint32_t __UART_HandleTypeDef::FifoMode***
  Specifies if the FIFO mode is being used. This parameter can be a value of ***UARTEx_FIFO_mode***.
- ***uint16_t __UART_HandleTypeDef::NbRxDataToProcess***
  Number of data to process during RX ISR execution
- ***uint16_t __UART_HandleTypeDef::NbTxDataToProcess***
  Number of data to process during TX ISR execution
- ***__IO HAL_UART_RxTypeTypeDef __UART_HandleTypeDef::ReceptionType***
  Type of ongoing reception
- ***__IO HAL_UART_RxEventTypeTypeDef __UART_HandleTypeDef::RxEventType***
  Type of Rx Event
- ***void(* __UART_HandleTypeDef::RxISR)(struct __UART_HandleTypeDef *huart)***
  Function pointer on Rx IRQ handler
- ***void(* __UART_HandleTypeDef::TxISR)(struct __UART_HandleTypeDef *huart)***
  Function pointer on Tx IRQ handler
- ***DMA_HandleTypeDef* __UART_HandleTypeDef::hdmatx***
  UART Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __UART_HandleTypeDef::hdmarx***
  UART Rx DMA Handle parameters
- ***HAL_LockTypeDef __UART_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_UART_StateTypeDef __UART_HandleTypeDef::gState***
  UART state information related to global Handle management and also related to Tx operations. This parameter can be a value of **HAL_UART_StateTypeDef**
- ***__IO HAL_UART_StateTypeDef __UART_HandleTypeDef::RxState***
  UART state information related to Rx operations. This parameter can be a value of **HAL_UART_StateTypeDef**
- ***__IO uint32_t __UART_HandleTypeDef::ErrorCode***
  UART Error code
- ***void(* __UART_HandleTypeDef::TxHalfCpltCallback)(struct __UART_HandleTypeDef *huart)***
  UART Tx Half Complete Callback
- ***void(* __UART_HandleTypeDef::TxCpltCallback)(struct __UART_HandleTypeDef *huart)***
  UART Tx Complete Callback
- ***void(* __UART_HandleTypeDef::RxHalfCpltCallback)(struct __UART_HandleTypeDef *huart)***
  UART Rx Half Complete Callback
- ***void(* __UART_HandleTypeDef::RxCpltCallback)(struct __UART_HandleTypeDef *huart)***
  UART Rx Complete Callback
- ***void(* __UART_HandleTypeDef::ErrorCallback)(struct __UART_HandleTypeDef *huart)***
  UART Error Callback
- ***void(* __UART_HandleTypeDef::AbortCpltCallback)(struct __UART_HandleTypeDef *huart)***
  UART Abort Complete Callback
- ***void(* __UART_HandleTypeDef::AbortTransmitCpltCallback)(struct __UART_HandleTypeDef *huart)***
  UART Abort Transmit Complete Callback
- ***void(* __UART_HandleTypeDef::AbortReceiveCpltCallback)(struct __UART_HandleTypeDef *huart)***
  UART Abort Receive Complete Callback

- ***void(* __UART_HandleTypeDef::WakeupCallback)(struct __UART_HandleTypeDef *huart)***
  UART Wakeup Callback
- ***void(* __UART_HandleTypeDef::RxFifoFullCallback)(struct __UART_HandleTypeDef *huart)***
  UART Rx Fifo Full Callback
- ***void(* __UART_HandleTypeDef::TxFifoEmptyCallback)(struct __UART_HandleTypeDef *huart)***
  UART Tx Fifo Empty Callback
- ***void(* __UART_HandleTypeDef::RxEventCallback)(struct __UART_HandleTypeDef *huart, uint16_t Pos)***
  UART Reception Event Callback
- ***void(* __UART_HandleTypeDef::MspInitCallback)(struct __UART_HandleTypeDef *huart)***
  UART Msp Init callback
- ***void(* __UART_HandleTypeDef::MspDeInitCallback)(struct __UART_HandleTypeDef *huart)***
  UART Msp DeInit callback

90.2 UART Firmware driver API description

The following section lists the various functions of the UART library.

90.2.1 How to use this driver

The UART HAL driver can be used as follows:

1. Declare a UART_HandleTypeDef handle structure (eg. UART_HandleTypeDef huart).
2. Initialize the UART low level resources by implementing the HAL_UART_MspInit() API:
   – Enable the USARTx interface clock.
   – UART pins configuration:
     ◦ Enable the clock for the UART GPIOs.
     ◦ Configure these UART pins as alternate function pull-up.
   – NVIC configuration if you need to use interrupt process (HAL_UART_Transmit_IT() and HAL_UART_Receive_IT() APIs):
     ◦ Configure the USARTx interrupt priority.
     ◦ Enable the NVIC USART IRQ handle.
   – UART interrupts handling:

Note: The specific UART interrupts (Transmission complete interrupt, RXNE interrupt, RX/TX FIFOs related interrupts and Error Interrupts) are managed using the macros __HAL_UART_ENABLE_IT() and __HAL_UART_DISABLE_IT() inside the transmit and receive processes.

   – DMA Configuration if you need to use DMA process (HAL_UART_Transmit_DMA() and HAL_UART_Receive_DMA() APIs):
     ◦ Declare a DMA handle structure for the Tx/Rx channel.
     ◦ Enable the DMAx interface clock.
     ◦ Configure the declared DMA handle structure with the required Tx/Rx parameters.
     ◦ Configure the DMA Tx/Rx channel.
     ◦ Associate the initialized DMA handle to the UART DMA Tx/Rx handle.
     ◦ Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
3. Program the Baud Rate, Word Length, Stop Bit, Parity, Prescaler value , Hardware flow control and Mode (Receiver/Transmitter) in the huart handle Init structure.
4. If required, program UART advanced features (TX/RX pins swap, auto Baud rate detection,...) in the huart handle AdvancedInit structure.
5. For the UART asynchronous mode, initialize the UART registers by calling the HAL_UART_Init() API.
6. For the UART Half duplex mode, initialize the UART registers by calling the HAL_HalfDuplex_Init() API.
7. For the UART LIN (Local Interconnection Network) mode, initialize the UART registers by calling the HAL_LIN_Init() API.

8. For the UART Multiprocessor mode, initialize the UART registers by calling the HAL_MultiProcessor_Init() API.
9. For the UART RS485 Driver Enabled mode, initialize the UART registers by calling the HAL_RS485Ex_Init() API.

Note: These API's (HAL_UART_Init(), HAL_HalfDuplex_Init(), HAL_LIN_Init(), HAL_MultiProcessor_Init(), also configure the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_UART_MspInit() API.

90.2.2 Callback registration

The compilation define USE_HAL_UART_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_UART_RegisterCallback() to register a user callback. Function HAL_UART_RegisterCallback() allows to register following callbacks:

- TxHalfCpltCallback : Tx Half Complete Callback.
- TxCpltCallback : Tx Complete Callback.
- RxHalfCpltCallback : Rx Half Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- AbortTransmitCpltCallback : Abort Transmit Complete Callback.
- AbortReceiveCpltCallback : Abort Receive Complete Callback.
- WakeupCallback : Wakeup Callback.
- RxFifoFullCallback : Rx Fifo Full Callback.
- TxFifoEmptyCallback : Tx Fifo Empty Callback.
- MspInitCallback : UART MspInit.
- MspDeInitCallback : UART MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_UART_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. HAL_UART_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- TxHalfCpltCallback : Tx Half Complete Callback.
- TxCpltCallback : Tx Complete Callback.
- RxHalfCpltCallback : Rx Half Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- AbortTransmitCpltCallback : Abort Transmit Complete Callback.
- AbortReceiveCpltCallback : Abort Receive Complete Callback.
- WakeupCallback : Wakeup Callback.
- RxFifoFullCallback : Rx Fifo Full Callback.
- TxFifoEmptyCallback : Tx Fifo Empty Callback.
- MspInitCallback : UART MspInit.
- MspDeInitCallback : UART MspDeInit.

For specific callback RxEventCallback, use dedicated registration/reset functions: respectively HAL_UART_RegisterRxEventCallback() , HAL_UART_UnRegisterRxEventCallback().

By default, after the HAL_UART_Init() and when the state is HAL_UART_STATE_RESET all callbacks are set to the corresponding weak (surcharged) functions: examples HAL_UART_TxCpltCallback(), HAL_UART_RxHalfCpltCallback(). Exception done for MspInit and MspDeInit functions that are respectively reset to the legacy weak (surcharged) functions in the HAL_UART_Init() and HAL_UART_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_UART_Init() and HAL_UART_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_UART_STATE_READY state only. Exception done MspInit/ MspDeInit that can be registered/unregistered in HAL_UART_STATE_READY or HAL_UART_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_UART_RegisterCallback() before calling HAL_UART_DeInit() or HAL_UART_Init() function.

When The compilation define USE_HAL_UART_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and weak (surcharged) callbacks are used.

90.2.3 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx or the UARTy in asynchronous mode.

- For the asynchronous mode the parameters below can be configured:
  - Baud Rate
  - Word Length
  - Stop Bit
  - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit.
  - Hardware flow control
  - Receiver/transmitter modes
  - Over Sampling Method
  - One-Bit Sampling Method
- For the asynchronous mode, the following advanced features can be configured as well:
  - TX and/or RX pin level inversion
  - data logical level inversion
  - RX and TX pins swap
  - RX overrun detection disabling
  - DMA disabling on RX error
  - MSB first on communication line
  - auto Baud rate detection

The HAL_UART_Init(), HAL_HalfDuplex_Init(), HAL_LIN_Init()and HAL_MultiProcessor_Init()API follow respectively the UART asynchronous, UART Half duplex, UART LIN mode and UART multiprocessor mode configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- ***HAL_UART_Init()***
- ***HAL_HalfDuplex_Init()***
- ***HAL_LIN_Init()***
- ***HAL_MultiProcessor_Init()***
- ***HAL_UART_DeInit()***
- ***HAL_UART_MspInit()***
- ***HAL_UART_MspDeInit()***
- ***HAL_UART_RegisterCallback()***
- ***HAL_UART_UnRegisterCallback()***
- ***HAL_UART_RegisterRxEventCallback()***
- ***HAL_UART_UnRegisterRxEventCallback()***

90.2.4 IO operation functions

This section contains the following APIs:

- ***HAL_UART_Transmit()***
- ***HAL_UART_Receive()***
- ***HAL_UART_Transmit_IT()***
- ***HAL_UART_Receive_IT()***

- *HAL_UART_Transmit_DMA()*
- *HAL_UART_Receive_DMA()*
- *HAL_UART_DMAPause()*
- *HAL_UART_DMAResume()*
- *HAL_UART_DMAStop()*
- *HAL_UART_Abort()*
- *HAL_UART_AbortTransmit()*
- *HAL_UART_AbortReceive()*
- *HAL_UART_Abort_IT()*
- *HAL_UART_AbortTransmit_IT()*
- *HAL_UART_AbortReceive_IT()*
- *HAL_UART_IRQHandler()*
- *HAL_UART_TxCpltCallback()*
- *HAL_UART_TxHalfCpltCallback()*
- *HAL_UART_RxCpltCallback()*
- *HAL_UART_RxHalfCpltCallback()*
- *HAL_UART_ErrorCallback()*
- *HAL_UART_AbortCpltCallback()*
- *HAL_UART_AbortTransmitCpltCallback()*
- *HAL_UART_AbortReceiveCpltCallback()*
- *HAL_UARTEx_RxEventCallback()*

90.2.5 Peripheral Control functions

This subsection provides a set of functions allowing to control the UART.

- HAL_UART_ReceiverTimeout_Config() API allows to configure the receiver timeout value on the fly
- HAL_UART_EnableReceiverTimeout() API enables the receiver timeout feature
- HAL_UART_DisableReceiverTimeout() API disables the receiver timeout feature
- HAL_MultiProcessor_EnableMuteMode() API enables mute mode
- HAL_MultiProcessor_DisableMuteMode() API disables mute mode
- HAL_MultiProcessor_EnterMuteMode() API enters mute mode
- UART_SetConfig() API configures the UART peripheral
- UART_AdvFeatureConfig() API optionally configures the UART advanced features
- UART_CheckIdleState() API ensures that TEACK and/or REACK are set after initialization
- HAL_HalfDuplex_EnableTransmitter() API disables receiver and enables transmitter
- HAL_HalfDuplex_EnableReceiver() API disables transmitter and enables receiver
- HAL_LIN_SendBreak() API transmits the break characters

This section contains the following APIs:

- *HAL_UART_ReceiverTimeout_Config()*
- *HAL_UART_EnableReceiverTimeout()*
- *HAL_UART_DisableReceiverTimeout()*
- *HAL_MultiProcessor_EnableMuteMode()*
- *HAL_MultiProcessor_DisableMuteMode()*
- *HAL_MultiProcessor_EnterMuteMode()*
- *HAL_HalfDuplex_EnableTransmitter()*
- *HAL_HalfDuplex_EnableReceiver()*
- *HAL_LIN_SendBreak()*

90.2.6 Peripheral State and Error functions

This subsection provides functions allowing to :

- Return the UART handle state.
- Return the UART handle error code

This section contains the following APIs:

- *HAL_UART_GetState()*
- *HAL_UART_GetError()*

90.2.7 Detailed description of functions

HAL_UART_Init

Function name

HAL_StatusTypeDef HAL_UART_Init (UART_HandleTypeDef * huart)

Function description

Initialize the UART mode according to the specified parameters in the UART_InitTypeDef and initialize the associated handle.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_HalfDuplex_Init

Function name

HAL_StatusTypeDef HAL_HalfDuplex_Init (UART_HandleTypeDef * huart)

Function description

Initialize the half-duplex mode according to the specified parameters in the UART_InitTypeDef and creates the associated handle.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_LIN_Init

Function name

HAL_StatusTypeDef HAL_LIN_Init (UART_HandleTypeDef * huart, uint32_t BreakDetectLength)

Function description

Initialize the LIN mode according to the specified parameters in the UART_InitTypeDef and creates the associated handle.

Parameters

- **huart:** UART handle.
- **BreakDetectLength:** Specifies the LIN break detection length. This parameter can be one of the following values:
  - UART_LINBREAKDETECTLENGTH_10B 10-bit break detection
  - UART_LINBREAKDETECTLENGTH_11B 11-bit break detection

Return values

- **HAL:** status

HAL_MultiProcessor_Init

Function name

HAL_StatusTypeDef HAL_MultiProcessor_Init (UART_HandleTypeDef * huart, uint8_t Address, uint32_t WakeUpMethod)

Function description

Initialize the multiprocessor mode according to the specified parameters in the UART_InitTypeDef and initialize the associated handle.

Parameters

- **huart:** UART handle.
- **Address:** UART node address (4-, 6-, 7- or 8-bit long).
- **WakeUpMethod:** Specifies the UART wakeup method. This parameter can be one of the following values:
  - UART_WAKEUPMETHOD_IDLELINE WakeUp by an idle line detection
  - UART_WAKEUPMETHOD_ADDRESSMARK WakeUp by an address mark

Return values

- **HAL:** status

Notes

- If the user resorts to idle line detection wake up, the Address parameter is useless and ignored by the initialization function.
- If the user resorts to address mark wake up, the address length detection is configured by default to 4 bits only. For the UART to be able to manage 6-, 7- or 8-bit long addresses detection, the API HAL_MultiProcessorEx_AddressLength_Set() must be called after HAL_MultiProcessor_Init().

HAL_UART_DeInit

Function name

HAL_StatusTypeDef HAL_UART_DeInit (UART_HandleTypeDef * huart)

Function description

DeInitialize the UART peripheral.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_UART_MspInit

Function name

void HAL_UART_MspInit (UART_HandleTypeDef * huart)

Function description

Initialize the UART MSP.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_MspDeInit

Function name

void HAL_UART_MspDeInit (UART_HandleTypeDef * huart)

Function description

DeInitialize the UART MSP.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_RegisterCallback

Function name

HAL_StatusTypeDef HAL_UART_RegisterCallback (UART_HandleTypeDef * huart, HAL_UART_CallbackIDTypeDef CallbackID, pUART_CallbackTypeDef pCallback)

Function description

Register a User UART Callback To be used instead of the weak predefined callback.

Parameters

- **huart:** uart handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_UART_TX_HALFCOMPLETE_CB_ID Tx Half Complete Callback ID
  - HAL_UART_TX_COMPLETE_CB_ID Tx Complete Callback ID
  - HAL_UART_RX_HALFCOMPLETE_CB_ID Rx Half Complete Callback ID
  - HAL_UART_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_UART_ERROR_CB_ID Error Callback ID
  - HAL_UART_ABORT_COMPLETE_CB_ID Abort Complete Callback ID
  - HAL_UART_ABORT_TRANSMIT_COMPLETE_CB_ID Abort Transmit Complete Callback ID
  - HAL_UART_ABORT_RECEIVE_COMPLETE_CB_ID Abort Receive Complete Callback ID
  - HAL_UART_WAKEUP_CB_ID Wakeup Callback ID
  - HAL_UART_RX_FIFO_FULL_CB_ID Rx Fifo Full Callback ID
  - HAL_UART_TX_FIFO_EMPTY_CB_ID Tx Fifo Empty Callback ID
  - HAL_UART_MSPINIT_CB_ID MspInit Callback ID
  - HAL_UART_MSPDEINIT_CB_ID MspDeInit Callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

Notes

- The HAL_UART_RegisterCallback() may be called before HAL_UART_Init(), HAL_HalfDuplex_Init(), HAL_LIN_Init(), HAL_MultiProcessor_Init() or HAL_RS485Ex_Init() in HAL_UART_STATE_RESET to register callbacks for HAL_UART_MSPINIT_CB_ID and HAL_UART_MSPDEINIT_CB_ID

HAL_UART_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_UART_UnRegisterCallback (UART_HandleTypeDef * huart, HAL_UART_CallbackIDTypeDef CallbackID)

Function description

Unregister an UART Callback UART callaback is redirected to the weak predefined callback.

Parameters

- **huart:** uart handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_UART_TX_HALFCOMPLETE_CB_ID Tx Half Complete Callback ID
  - HAL_UART_TX_COMPLETE_CB_ID Tx Complete Callback ID
  - HAL_UART_RX_HALFCOMPLETE_CB_ID Rx Half Complete Callback ID
  - HAL_UART_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_UART_ERROR_CB_ID Error Callback ID
  - HAL_UART_ABORT_COMPLETE_CB_ID Abort Complete Callback ID
  - HAL_UART_ABORT_TRANSMIT_COMPLETE_CB_ID Abort Transmit Complete Callback ID
  - HAL_UART_ABORT_RECEIVE_COMPLETE_CB_ID Abort Receive Complete Callback ID
  - HAL_UART_WAKEUP_CB_ID Wakeup Callback ID
  - HAL_UART_RX_FIFO_FULL_CB_ID Rx Fifo Full Callback ID
  - HAL_UART_TX_FIFO_EMPTY_CB_ID Tx Fifo Empty Callback ID
  - HAL_UART_MSPINIT_CB_ID MspInit Callback ID
  - HAL_UART_MSPDEINIT_CB_ID MspDeInit Callback ID

Return values

- **HAL:** status

Notes

- The HAL_UART_UnRegisterCallback() may be called before HAL_UART_Init(), HAL_HalfDuplex_Init(), HAL_LIN_Init(), HAL_MultiProcessor_Init() or HAL_RS485Ex_Init() in HAL_UART_STATE_RESET to un-register callbacks for HAL_UART_MSPINIT_CB_ID and HAL_UART_MSPDEINIT_CB_ID

HAL_UART_RegisterRxEventCallback

Function name

HAL_StatusTypeDef HAL_UART_RegisterRxEventCallback (UART_HandleTypeDef * huart, pUART_RxEventCallbackTypeDef pCallback)

Function description

Register a User UART Rx Event Callback To be used instead of the weak predefined callback.

Parameters

- **huart:** Uart handle
- **pCallback:** Pointer to the Rx Event Callback function

Return values

- **HAL:** status

HAL_UART_UnRegisterRxEventCallback

Function name

HAL_StatusTypeDef HAL_UART_UnRegisterRxEventCallback (UART_HandleTypeDef * huart)

Function description

UnRegister the UART Rx Event Callback UART Rx Event Callback is redirected to the weak HAL_UARTEx_RxEventCallback() predefined callback.

Parameters

- **huart:** Uart handle

Return values

- **HAL:** status

HAL_UART_Transmit

Function name

HAL_StatusTypeDef HAL_UART_Transmit (UART_HandleTypeDef * huart, const uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Send an amount of data in blocking mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be sent.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must indicate the number of u16 provided through pData.
- When FIFO mode is enabled, writing a data in the TDR register adds one data to the TXFIFO. Write operations to the TDR register are performed when TXFNF flag is set. From hardware perspective, TXFNF flag and TXE are mapped on the same bit-field.

HAL_UART_Receive

Function name

HAL_StatusTypeDef HAL_UART_Receive (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must indicate the number of u16 available through pData.
- When FIFO mode is enabled, the RXFNE flag is set as long as the RXFIFO is not empty. Read operations from the RDR register are performed when RXFNE flag is set. From hardware perspective, RXFNE flag and RXNE are mapped on the same bit-field.

HAL_UART_Transmit_IT

Function name

HAL_StatusTypeDef HAL_UART_Transmit_IT (UART_HandleTypeDef * huart, const uint8_t * pData, uint16_t Size)

Function description

Send an amount of data in interrupt mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be sent.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must indicate the number of u16 provided through pData.

HAL_UART_Receive_IT

Function name

HAL_StatusTypeDef HAL_UART_Receive_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in interrupt mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must indicate the number of u16 available through pData.

HAL_UART_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_UART_Transmit_DMA (UART_HandleTypeDef * huart, const uint8_t * pData, uint16_t Size)

Function description

Send an amount of data in DMA mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be sent.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must indicate the number of u16 provided through pData.

HAL_UART_Receive_DMA

Function name

HAL_StatusTypeDef HAL_UART_Receive_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in DMA mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- When the UART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position).
- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must indicate the number of u16 available through pData.

HAL_UART_DMAPause

Function name

HAL_StatusTypeDef HAL_UART_DMAPause (UART_HandleTypeDef * huart)

Function description

Pause the DMA Transfer.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_UART_DMAResume

Function name

HAL_StatusTypeDef HAL_UART_DMAResume (UART_HandleTypeDef * huart)

Function description

Resume the DMA Transfer.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_UART_DMAStop

Function name

HAL_StatusTypeDef HAL_UART_DMAStop (UART_HandleTypeDef * huart)

Function description

Stop the DMA Transfer.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_UART_Abort

Function name

HAL_StatusTypeDef HAL_UART_Abort (UART_HandleTypeDef * huart)

Function description

Abort ongoing transfers (blocking mode).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_UART_AbortTransmit

Function name

HAL_StatusTypeDef HAL_UART_AbortTransmit (UART_HandleTypeDef * huart)

Function description

Abort ongoing Transmit transfer (blocking mode).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_UART_AbortReceive

Function name

HAL_StatusTypeDef HAL_UART_AbortReceive (UART_HandleTypeDef * huart)

Function description

Abort ongoing Receive transfer (blocking mode).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_UART_Abort_IT

Function name

HAL_StatusTypeDef HAL_UART_Abort_IT (UART_HandleTypeDef * huart)

Function description

Abort ongoing transfers (Interrupt mode).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_UART_AbortTransmit_IT

Function name

HAL_StatusTypeDef HAL_UART_AbortTransmit_IT (UART_HandleTypeDef * huart)

Function description

Abort ongoing Transmit transfer (Interrupt mode).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_UART_AbortReceive_IT

Function name

HAL_StatusTypeDef HAL_UART_AbortReceive_IT (UART_HandleTypeDef * huart)

Function description

Abort ongoing Receive transfer (Interrupt mode).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_UART_IRQHandler

Function name

void HAL_UART_IRQHandler (UART_HandleTypeDef * huart)

Function description

Handle UART interrupt request.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_TxHalfCpltCallback

Function name

void HAL_UART_TxHalfCpltCallback (UART_HandleTypeDef * huart)

Function description

Tx Half Transfer completed callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_TxCpltCallback

Function name

void HAL_UART_TxCpltCallback (UART_HandleTypeDef * huart)

Function description

Tx Transfer completed callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_RxHalfCpltCallback

Function name

void HAL_UART_RxHalfCpltCallback (UART_HandleTypeDef * huart)

Function description

Rx Half Transfer completed callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_RxCpltCallback

Function name

void HAL_UART_RxCpltCallback (UART_HandleTypeDef * huart)

Function description

Rx Transfer completed callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_ErrorCallback

Function name

void HAL_UART_ErrorCallback (UART_HandleTypeDef * huart)

Function description

UART error callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_AbortCpltCallback

Function name

void HAL_UART_AbortCpltCallback (UART_HandleTypeDef * huart)

Function description

UART Abort Complete callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_AbortTransmitCpltCallback

Function name

void HAL_UART_AbortTransmitCpltCallback (UART_HandleTypeDef * huart)

Function description

UART Abort Complete callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_AbortReceiveCpltCallback

Function name

void HAL_UART_AbortReceiveCpltCallback (UART_HandleTypeDef * huart)

Function description

UART Abort Receive Complete callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UARTEx_RxEventCallback

Function name

void HAL_UARTEx_RxEventCallback (UART_HandleTypeDef * huart, uint16_t Size)

Function description

Reception Event Callback (Rx event notification called after use of advanced reception service).

Parameters

- **huart:** UART handle
- **Size:** Number of data available in application reception buffer (indicates a position in reception buffer until which, data are available)

Return values

- **None:**

HAL_UART_ReceiverTimeout_Config

Function name

void HAL_UART_ReceiverTimeout_Config (UART_HandleTypeDef * huart, uint32_t TimeoutValue)

Function description

Update on the fly the receiver timeout value in RTOR register.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
- **TimeoutValue:** receiver timeout value in number of baud blocks. The timeout value must be less or equal to 0x0FFFFFFF.

Return values

- **None:**

HAL_UART_EnableReceiverTimeout

Function name

HAL_StatusTypeDef HAL_UART_EnableReceiverTimeout (UART_HandleTypeDef * huart)

Function description

Enable the UART receiver timeout feature.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

HAL_UART_DisableReceiverTimeout

Function name

HAL_StatusTypeDef HAL_UART_DisableReceiverTimeout (UART_HandleTypeDef * huart)

Function description

Disable the UART receiver timeout feature.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

HAL_LIN_SendBreak

Function name

HAL_StatusTypeDef HAL_LIN_SendBreak (UART_HandleTypeDef * huart)

Function description

Transmit break characters.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_MultiProcessor_EnableMuteMode

Function name

HAL_StatusTypeDef HAL_MultiProcessor_EnableMuteMode (UART_HandleTypeDef * huart)

Function description

Enable UART in mute mode (does not mean UART enters mute mode; to enter mute mode, HAL_MultiProcessor_EnterMuteMode() API must be called).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_MultiProcessor_DisableMuteMode

Function name

HAL_StatusTypeDef HAL_MultiProcessor_DisableMuteMode (UART_HandleTypeDef * huart)

Function description

Disable UART mute mode (does not mean the UART actually exits mute mode as it may not have been in mute mode at this very moment).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_MultiProcessor_EnterMuteMode

Function name

void HAL_MultiProcessor_EnterMuteMode (UART_HandleTypeDef * huart)

Function description

Enter UART mute mode (means UART actually enters mute mode).

Parameters

- **huart:** UART handle.

Return values

- **None:**

Notes

- To exit from mute mode, HAL_MultiProcessor_DisableMuteMode() API must be called.

HAL_HalfDuplex_EnableTransmitter

Function name

HAL_StatusTypeDef HAL_HalfDuplex_EnableTransmitter (UART_HandleTypeDef * huart)

Function description

Enable the UART transmitter and disable the UART receiver.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_HalfDuplex_EnableReceiver

Function name

HAL_StatusTypeDef HAL_HalfDuplex_EnableReceiver (UART_HandleTypeDef * huart)

Function description

Enable the UART receiver and disable the UART transmitter.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status.

HAL_UART_GetState

Function name

HAL_UART_StateTypeDef HAL_UART_GetState (const UART_HandleTypeDef * huart)

Function description

Return the UART handle state.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART.

Return values

- **HAL:** state

HAL_UART_GetError

Function name

uint32_t HAL_UART_GetError (const UART_HandleTypeDef * huart)

Function description

Return the UART handle error code.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART.

Return values

- **UART:** Error Code

UART_InitCallbacksToDefault

Function name

void UART_InitCallbacksToDefault (UART_HandleTypeDef * huart)

Function description

Initialize the callbacks to their default values.

Parameters

- **huart:** UART handle.

Return values

- **none:**

UART_SetConfig

Function name

HAL_StatusTypeDef UART_SetConfig (UART_HandleTypeDef * huart)

Function description

Configure the UART peripheral.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

UART_CheckIdleState

Function name

HAL_StatusTypeDef UART_CheckIdleState (UART_HandleTypeDef * huart)

Function description

Check the UART Idle State.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

UART_WaitOnFlagUntilTimeout

Function name

HAL_StatusTypeDef UART_WaitOnFlagUntilTimeout (UART_HandleTypeDef * huart, uint32_t Flag, FlagStatus Status, uint32_t Tickstart, uint32_t Timeout)

Function description

This function handles UART Communication Timeout.

Parameters

- **huart:** UART handle.
- **Flag:** Specifies the UART flag to check
- **Status:** The actual Flag status (SET or RESET)
- **Tickstart:** Tick start value
- **Timeout:** Timeout duration

Return values

- **HAL:** status

UART_AdvFeatureConfig

Function name

void UART_AdvFeatureConfig (UART_HandleTypeDef * huart)

Function description

Configure the UART peripheral advanced features.

Parameters

- **huart:** UART handle.

Return values

- **None:**

UART_Start_Receive_IT

Function name

HAL_StatusTypeDef UART_Start_Receive_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

Start Receive operation in interrupt mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- This function could be called by all HAL UART API providing reception in Interrupt mode.
- When calling this function, parameters validity is considered as already checked, i.e. Rx State, buffer address, ... UART Handle is assumed as Locked.

UART_Start_Receive_DMA

Function name

HAL_StatusTypeDef UART_Start_Receive_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

Start Receive operation in DMA mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- This function could be called by all HAL UART API providing reception in DMA mode.
- When calling this function, parameters validity is considered as already checked, i.e. Rx State, buffer address, ... UART Handle is assumed as Locked.

90.3 UART Firmware driver defines

The following section lists the various define and macros of the module.

90.3.1 UART

UART

UART Advanced Feature Initialization Type

UART_ADVFEATURE_NO_INIT

No advanced feature initialization

UART_ADVFEATURE_TXINVERT_INIT

TX pin active level inversion

UART_ADVFEATURE_RXINVERT_INIT

RX pin active level inversion

UART_ADVFEATURE_DATAINVERT_INIT

Binary data inversion

UART_ADVFEATURE_SWAP_INIT

TX/RX pins swap

UART_ADVFEATURE_RXOVERRUNDISABLE_INIT

RX overrun disable

UART_ADVFEATURE_DMADISABLEONERROR_INIT

DMA disable on Reception Error

UART_ADVFEATURE_AUTOBAUDRATE_INIT

Auto Baud rate detection initialization

UART_ADVFEATURE_MSBFIRST_INIT

Most significant bit sent/received first

UART Advanced Feature Auto BaudRate Enable

UART_ADVFEATURE_AUTOBAUDRATE_DISABLE

RX Auto Baud rate detection enable

UART_ADVFEATURE_AUTOBAUDRATE_ENABLE

RX Auto Baud rate detection disable

UART Advanced Feature AutoBaud Rate Mode

UART_ADVFEATURE_AUTOBAUDRATE_ONSTARTBIT

Auto Baud rate detection on start bit

UART_ADVFEATURE_AUTOBAUDRATE_ONFALLINGEDGE

Auto Baud rate detection on falling edge

UART_ADVFEATURE_AUTOBAUDRATE_ON0X7FFRAME

Auto Baud rate detection on 0x7F frame detection

UART_ADVFEATURE_AUTOBAUDRATE_ON0X55FRAME

Auto Baud rate detection on 0x55 frame detection

UART Clock Prescaler

UART_PRESCALER_DIV1

fclk_pres = fclk

UART_PRESCALER_DIV2

fclk_pres = fclk/2

UART_PRESCALER_DIV4

fclk_pres = fclk/4

UART_PRESCALER_DIV6

fclk_pres = fclk/6

UART_PRESCALER_DIV8

fclk_pres = fclk/8

UART_PRESCALER_DIV10

fclk_pres = fclk/10

UART_PRESCALER_DIV12

fclk_pres = fclk/12

UART_PRESCALER_DIV16

fclk_pres = fclk/16

UART_PRESCALER_DIV32

fclk_pres = fclk/32

UART_PRESCALER_DIV64

fclk_pres = fclk/64

UART_PRESCALER_DIV128

fclk_pres = fclk/128

UART_PRESCALER_DIV256

fclk_pres = fclk/256

UART Driver Enable Assertion Time LSB Position In CR1 Register

UART_CR1_DEAT_ADDRESS_LSB_POS

UART Driver Enable assertion time LSB position in CR1 register

UART Driver Enable DeAssertion Time LSB Position In CR1 Register

UART_CR1_DEDT_ADDRESS_LSB_POS

UART Driver Enable de-assertion time LSB position in CR1 register

UART Address-matching LSB Position In CR2 Register

UART_CR2_ADDRESS_LSB_POS

UART address-matching LSB position in CR2 register

UART Advanced Feature Binary Data Inversion

UART_ADVFEATURE_DATAINV_DISABLE

Binary data inversion disable

UART_ADVFEATURE_DATAINV_ENABLE

Binary data inversion enable

UART Advanced Feature DMA Disable On Rx Error

UART_ADVFEATURE_DMA_ENABLEONRXERROR

DMA enable on Reception Error

UART_ADVFEATURE_DMA_DISABLEONRXERROR

DMA disable on Reception Error

UART DMA Rx

UART_DMA_RX_DISABLE

UART DMA RX disabled

UART_DMA_RX_ENABLE

UART DMA RX enabled

UART DMA Tx

UART_DMA_TX_DISABLE

UART DMA TX disabled

UART_DMA_TX_ENABLE

UART DMA TX enabled

UART DriverEnable Polarity

UART_DE_POLARITY_HIGH

Driver enable signal is active high

UART_DE_POLARITY_LOW

Driver enable signal is active low

UART Error Definition

HAL_UART_ERROR_NONE

No error

HAL_UART_ERROR_PE

Parity error

HAL_UART_ERROR_NE

Noise error

HAL_UART_ERROR_FE

Frame error

HAL_UART_ERROR_ORE

Overrun error

HAL_UART_ERROR_DMA

DMA transfer error

HAL_UART_ERROR_RTO

Receiver Timeout error

HAL_UART_ERROR_INVALID_CALLBACK

Invalid Callback error

UART Exported Macros

__HAL_UART_RESET_HANDLE_STATE

Description:

- Reset UART handle states.

Parameters:

- __HANDLE__: UART handle.

Return value:

- None

__HAL_UART_FLUSH_DRREGISTER

Description:

- Flush the UART Data registers.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_FLAG

Description:

- Clear the specified UART pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - UART_CLEAR_PEF Parity Error Clear Flag
  - UART_CLEAR_FEF Framing Error Clear Flag
  - UART_CLEAR_NEF Noise detected Clear Flag
  - UART_CLEAR_OREF Overrun Error Clear Flag
  - UART_CLEAR_IDLEF IDLE line detected Clear Flag
  - UART_CLEAR_TXFECF TXFIFO empty clear Flag
  - UART_CLEAR_TCF Transmission Complete Clear Flag
  - UART_CLEAR_RTOF Receiver Timeout clear flag
  - UART_CLEAR_LBDF LIN Break Detection Clear Flag
  - UART_CLEAR_CTSF CTS Interrupt Clear Flag
  - UART_CLEAR_CMF Character Match Clear Flag
  - UART_CLEAR_WUF Wake Up from stop mode Clear Flag

Return value:

- None

__HAL_UART_CLEAR_PEFLAG

Description:

- Clear the UART PE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_FEFLAG

Description:

- Clear the UART FE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_NEFLAG

Description:

- Clear the UART NE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_OREFLAG

Description:

- Clear the UART ORE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_IDLEFLAG

Description:

- Clear the UART IDLE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_TXFECF

Description:

- Clear the UART TX FIFO empty clear flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_GET_FLAG

Description:

- Check whether the specified UART flag is set or not.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - UART_FLAG_TXFT TXFIFO threshold flag
  - UART_FLAG_RXFT RXFIFO threshold flag
  - UART_FLAG_RXFF RXFIFO Full flag
  - UART_FLAG_TXFE TXFIFO Empty flag
  - UART_FLAG_REACK Receive enable acknowledge flag
  - UART_FLAG_TEACK Transmit enable acknowledge flag
  - UART_FLAG_WUF Wake up from stop mode flag
  - UART_FLAG_RWU Receiver wake up flag (if the UART in mute mode)
  - UART_FLAG_SBKF Send Break flag
  - UART_FLAG_CMF Character match flag
  - UART_FLAG_BUSY Busy flag
  - UART_FLAG_ABRF Auto Baud rate detection flag
  - UART_FLAG_ABRE Auto Baud rate detection error flag
  - UART_FLAG_CTS CTS Change flag
  - UART_FLAG_LBDF LIN Break detection flag
  - UART_FLAG_TXE Transmit data register empty flag
  - UART_FLAG_TXFNF UART TXFIFO not full flag
  - UART_FLAG_TC Transmission Complete flag
  - UART_FLAG_RXNE Receive data register not empty flag
  - UART_FLAG_RXFNE UART RXFIFO not empty flag
  - UART_FLAG_RTOF Receiver Timeout flag
  - UART_FLAG_IDLE Idle Line detection flag
  - UART_FLAG_ORE Overrun Error flag
  - UART_FLAG_NE Noise Error flag
  - UART_FLAG_FE Framing Error flag
  - UART_FLAG_PE Parity Error flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_UART_ENABLE_IT

Description:

- Enable the specified UART interrupt.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __INTERRUPT__: specifies the UART interrupt source to enable. This parameter can be one of the following values:
  - UART_IT_RXFF RXFIFO Full interrupt
  - UART_IT_TXFE TXFIFO Empty interrupt
  - UART_IT_RXFT RXFIFO threshold interrupt
  - UART_IT_TXFT TXFIFO threshold interrupt
  - UART_IT_WUF Wakeup from stop mode interrupt
  - UART_IT_CM Character match interrupt
  - UART_IT_CTS CTS change interrupt
  - UART_IT_LBD LIN Break detection interrupt
  - UART_IT_TXE Transmit Data Register empty interrupt
  - UART_IT_TXFNF TX FIFO not full interrupt
  - UART_IT_TC Transmission complete interrupt
  - UART_IT_RXNE Receive Data register not empty interrupt
  - UART_IT_RXFNE RXFIFO not empty interrupt
  - UART_IT_RTO Receive Timeout interrupt
  - UART_IT_IDLE Idle line detection interrupt
  - UART_IT_PE Parity Error interrupt
  - UART_IT_ERR Error interrupt (frame error, noise error, overrun error)

Return value:

- None

__HAL_UART_DISABLE_IT

Description:

- Disable the specified UART interrupt.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __INTERRUPT__: specifies the UART interrupt source to disable. This parameter can be one of the following values:
  - UART_IT_RXFF RXFIFO Full interrupt
  - UART_IT_TXFE TXFIFO Empty interrupt
  - UART_IT_RXFT RXFIFO threshold interrupt
  - UART_IT_TXFT TXFIFO threshold interrupt
  - UART_IT_WUF Wakeup from stop mode interrupt
  - UART_IT_CM Character match interrupt
  - UART_IT_CTS CTS change interrupt
  - UART_IT_LBD LIN Break detection interrupt
  - UART_IT_TXE Transmit Data Register empty interrupt
  - UART_IT_TXFNF TX FIFO not full interrupt
  - UART_IT_TC Transmission complete interrupt
  - UART_IT_RXNE Receive Data register not empty interrupt
  - UART_IT_RXFNE RXFIFO not empty interrupt
  - UART_IT_RTO Receive Timeout interrupt
  - UART_IT_IDLE Idle line detection interrupt
  - UART_IT_PE Parity Error interrupt
  - UART_IT_ERR Error interrupt (Frame error, noise error, overrun error)

Return value:

- None

__HAL_UART_GET_IT

Description:

- Check whether the specified UART interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __INTERRUPT__: specifies the UART interrupt to check. This parameter can be one of the following values:
  - UART_IT_RXFF RXFIFO Full interrupt
  - UART_IT_TXFE TXFIFO Empty interrupt
  - UART_IT_RXFT RXFIFO threshold interrupt
  - UART_IT_TXFT TXFIFO threshold interrupt
  - UART_IT_WUF Wakeup from stop mode interrupt
  - UART_IT_CM Character match interrupt
  - UART_IT_CTS CTS change interrupt
  - UART_IT_LBD LIN Break detection interrupt
  - UART_IT_TXE Transmit Data Register empty interrupt
  - UART_IT_TXFNF TX FIFO not full interrupt
  - UART_IT_TC Transmission complete interrupt
  - UART_IT_RXNE Receive Data register not empty interrupt
  - UART_IT_RXFNE RXFIFO not empty interrupt
  - UART_IT_RTO Receive Timeout interrupt
  - UART_IT_IDLE Idle line detection interrupt
  - UART_IT_PE Parity Error interrupt
  - UART_IT_ERR Error interrupt (Frame error, noise error, overrun error)

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_UART_GET_IT_SOURCE

Description:

- Check whether the specified UART interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __INTERRUPT__: specifies the UART interrupt source to check. This parameter can be one of the following values:
  - UART_IT_RXFF RXFIFO Full interrupt
  - UART_IT_TXFE TXFIFO Empty interrupt
  - UART_IT_RXFT RXFIFO threshold interrupt
  - UART_IT_TXFT TXFIFO threshold interrupt
  - UART_IT_WUF Wakeup from stop mode interrupt
  - UART_IT_CM Character match interrupt
  - UART_IT_CTS CTS change interrupt
  - UART_IT_LBD LIN Break detection interrupt
  - UART_IT_TXE Transmit Data Register empty interrupt
  - UART_IT_TXFNF TX FIFO not full interrupt
  - UART_IT_TC Transmission complete interrupt
  - UART_IT_RXNE Receive Data register not empty interrupt
  - UART_IT_RXFNE RXFIFO not empty interrupt
  - UART_IT_RTO Receive Timeout interrupt
  - UART_IT_IDLE Idle line detection interrupt
  - UART_IT_PE Parity Error interrupt
  - UART_IT_ERR Error interrupt (Frame error, noise error, overrun error)

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_UART_CLEAR_IT

Description:

- Clear the specified UART ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt This parameter can be one of the following values:
  - UART_CLEAR_PEF Parity Error Clear Flag
  - UART_CLEAR_FEF Framing Error Clear Flag
  - UART_CLEAR_NEF Noise detected Clear Flag
  - UART_CLEAR_OREF Overrun Error Clear Flag
  - UART_CLEAR_IDLEF IDLE line detected Clear Flag
  - UART_CLEAR_RTOF Receiver timeout clear flag
  - UART_CLEAR_TXFECF TXFIFO empty Clear Flag
  - UART_CLEAR_TCF Transmission Complete Clear Flag
  - UART_CLEAR_LBDF LIN Break Detection Clear Flag
  - UART_CLEAR_CTSF CTS Interrupt Clear Flag
  - UART_CLEAR_CMF Character Match Clear Flag
  - UART_CLEAR_WUF Wake Up from stop mode Clear Flag

Return value:

- None

__HAL_UART_SEND_REQ

Description:

- Set a specific UART request flag.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __REQ__: specifies the request flag to set This parameter can be one of the following values:
  - UART_AUTOBAUD_REQUEST Auto-Baud Rate Request
  - UART_SENDBREAK_REQUEST Send Break Request
  - UART_MUTE_MODE_REQUEST Mute Mode Request
  - UART_RXDATA_FLUSH_REQUEST Receive Data flush Request
  - UART_TXDATA_FLUSH_REQUEST Transmit data flush Request

Return value:

- None

__HAL_UART_ONE_BIT_SAMPLE_ENABLE

Description:

- Enable the UART one bit sample method.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_ONE_BIT_SAMPLE_DISABLE

Description:

- Disable the UART one bit sample method.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_ENABLE

Description:

- Enable UART.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_DISABLE

Description:

- Disable UART.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_HWCONTROL_CTS_ENABLE

Description:

- Enable CTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to enable CTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying CTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

__HAL_UART_HWCONTROL_CTS_DISABLE

Description:

- Disable CTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to disable CTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying CTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

__HAL_UART_HWCONTROL_RTS_ENABLE

Description:

- Enable RTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to enable RTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying RTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

__HAL_UART_HWCONTROL_RTS_DISABLE

Description:

- Disable RTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to disable RTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying RTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

UART Status Flags

UART_FLAG_TXFT

UART TXFIFO threshold flag

UART_FLAG_RXFT

UART RXFIFO threshold flag

UART_FLAG_RXFF

UART RXFIFO Full flag

UART_FLAG_TXFE

UART TXFIFO Empty flag

UART_FLAG_REACK

UART receive enable acknowledge flag

UART_FLAG_TEACK

UART transmit enable acknowledge flag

UART_FLAG_WUF

UART wake-up from stop mode flag

UART_FLAG_RWU

UART receiver wake-up from mute mode flag

UART_FLAG_SBKF

UART send break flag

UART_FLAG_CMF

UART character match flag

UART_FLAG_BUSY

UART busy flag

UART_FLAG_ABRF

UART auto Baud rate flag

UART_FLAG_ABRE

UART auto Baud rate error

UART_FLAG_RTOF

UART receiver timeout flag

UART_FLAG_CTS

UART clear to send flag

UART_FLAG_CTSIF

UART clear to send interrupt flag

UART_FLAG_LBDF

UART LIN break detection flag

UART_FLAG_TXE

UART transmit data register empty

UART_FLAG_TXFNF

UART TXFIFO not full

UART_FLAG_TC

UART transmission complete

UART_FLAG_RXNE

UART read data register not empty

UART_FLAG_RXFNE

UART RXFIFO not empty

UART_FLAG_IDLE

UART idle flag

UART_FLAG_ORE

UART overrun error

UART_FLAG_NE

UART noise error

UART_FLAG_FE

UART frame error

UART_FLAG_PE

UART parity error

UART Half Duplex Selection

UART_HALF_DUPLEX_DISABLE

UART half-duplex disabled

UART_HALF_DUPLEX_ENABLE

UART half-duplex enabled

UART Hardware Flow Control

UART_HWCONTROL_NONE

No hardware control

UART_HWCONTROL_RTS

Request To Send

UART_HWCONTROL_CTS

Clear To Send

UART_HWCONTROL_RTS_CTS

Request and Clear To Send

UART Interruptions Flag Mask

UART_IT_MASK

UART interruptions flags mask

UART Interrupts Definition

UART_IT_PE

UART parity error interruption

UART_IT_TXE

UART transmit data register empty interruption

UART_IT_TXFNF

UART TX FIFO not full interruption

UART_IT_TC

UART transmission complete interruption

UART_IT_RXNE

UART read data register not empty interruption

UART_IT_RXFNE

UART RXFIFO not empty interruption

UART_IT_IDLE

UART idle interruption

UART_IT_LBD

UART LIN break detection interruption

UART_IT_CTS

UART CTS interruption

UART_IT_CM

UART character match interruption

UART_IT_WUF

UART wake-up from stop mode interruption

UART_IT_RXFF

UART RXFIFO full interruption

UART_IT_TXFE

UART TXFIFO empty interruption

UART_IT_RXFT

UART RXFIFO threshold reached interruption

UART_IT_TXFT

UART TXFIFO threshold reached interruption

UART_IT_RTO

UART receiver timeout interruption

UART_IT_ERR

UART error interruption

UART_IT_ORE

UART overrun error interruption

UART_IT_NE

UART noise error interruption

UART_IT_FE

UART frame error interruption

UART Interruption Clear Flags

UART_CLEAR_PEF

Parity Error Clear Flag

UART_CLEAR_FEF

Framing Error Clear Flag

UART_CLEAR_NEF

Noise Error detected Clear Flag

UART_CLEAR_OREF

Overrun Error Clear Flag

UART_CLEAR_IDLEF

IDLE line detected Clear Flag

UART_CLEAR_TXFECF

TXFIFO empty clear flag

UART_CLEAR_TCF

Transmission Complete Clear Flag

UART_CLEAR_LBDF

LIN Break Detection Clear Flag

UART_CLEAR_CTSF

CTS Interrupt Clear Flag

UART_CLEAR_CMF

Character Match Clear Flag

UART_CLEAR_WUF

Wake Up from stop mode Clear Flag

UART_CLEAR_RTOF

UART receiver timeout clear flag

UART Local Interconnection Network mode

UART_LIN_DISABLE

Local Interconnect Network disable

UART_LIN_ENABLE

Local Interconnect Network enable

UART LIN Break Detection

UART_LINBREAKDETECTLENGTH_10B

LIN 10-bit break detection length

UART_LINBREAKDETECTLENGTH_11B

LIN 11-bit break detection length

UART Transfer Mode

UART_MODE_RX

RX mode

UART_MODE_TX

TX mode

UART_MODE_TX_RX

RX and TX mode

UART Advanced Feature MSB First

UART_ADVFEATURE_MSBFIRST_DISABLE

Most significant bit sent/received first disable

UART_ADVFEATURE_MSBFIRST_ENABLE

Most significant bit sent/received first enable

UART Advanced Feature Mute Mode Enable

UART_ADVFEATURE_MUTEMODE_DISABLE

UART mute mode disable

UART_ADVFEATURE_MUTEMODE_ENABLE

UART mute mode enable

UART One Bit Sampling Method

UART_ONE_BIT_SAMPLE_DISABLE

One-bit sampling disable

UART_ONE_BIT_SAMPLE_ENABLE

One-bit sampling enable

UART Advanced Feature Overrun Disable

UART_ADVFEATURE_OVERRUN_ENABLE

RX overrun enable

UART_ADVFEATURE_OVERRUN_DISABLE

RX overrun disable

UART Over Sampling

UART_OVERSAMPLING_16

Oversampling by 16

UART_OVERSAMPLING_8

Oversampling by 8

UART Parity

UART_PARITY_NONE

No parity

UART_PARITY_EVEN

Even parity

UART_PARITY_ODD

Odd parity

UART Receiver Timeout

UART_RECEIVER_TIMEOUT_DISABLE

UART Receiver Timeout disable

UART_RECEIVER_TIMEOUT_ENABLE

UART Receiver Timeout enable

UART Reception type values

HAL_UART_RECEPTION_STANDARD

Standard reception

HAL_UART_RECEPTION_TOIDLE

Reception till completion or IDLE event

HAL_UART_RECEPTION_TORTO

Reception till completion or RTO event

HAL_UART_RECEPTION_TOCHARMATCH

Reception till completion or CM event

UART Request Parameters

UART_AUTOBAUD_REQUEST

Auto-Baud Rate Request

UART_SENDBREAK_REQUEST

Send Break Request

UART_MUTE_MODE_REQUEST

Mute Mode Request

UART_RXDATA_FLUSH_REQUEST

Receive Data flush Request

UART_TXDATA_FLUSH_REQUEST

Transmit data flush Request

UART RxEvent type values

HAL_UART_RXEVENT_TC

RxEvent linked to Transfer Complete event

HAL_UART_RXEVENT_HT

RxEvent linked to Half Transfer event

HAL_UART_RXEVENT_IDLE

RxEvent linked to IDLE event

UART Advanced Feature RX Pin Active Level Inversion

UART_ADVFEATURE_RXINV_DISABLE

RX pin active level inversion disable

UART_ADVFEATURE_RXINV_ENABLE

RX pin active level inversion enable

UART Advanced Feature RX TX Pins Swap

UART_ADVFEATURE_SWAP_DISABLE

TX/RX pins swap disable

UART_ADVFEATURE_SWAP_ENABLE

TX/RX pins swap enable

UART State

UART_STATE_DISABLE

UART disabled

UART_STATE_ENABLE

UART enabled

UART State Code Definition

HAL_UART_STATE_RESET

Peripheral is not initialized Value is allowed for gState and RxState

HAL_UART_STATE_READY

Peripheral Initialized and ready for use Value is allowed for gState and RxState

HAL_UART_STATE_BUSY

an internal process is ongoing Value is allowed for gState only

HAL_UART_STATE_BUSY_TX

Data Transmission process is ongoing Value is allowed for gState only

HAL_UART_STATE_BUSY_RX

Data Reception process is ongoing Value is allowed for RxState only

HAL_UART_STATE_BUSY_TX_RX

Data Transmission and Reception process is ongoing Not to be used for neither gState nor RxState.Value is result of combination (Or) between gState and RxState values

HAL_UART_STATE_TIMEOUT

Timeout state Value is allowed for gState only

HAL_UART_STATE_ERROR

Error Value is allowed for gState only

UART Number of Stop Bits

UART_STOPBITS_0_5

UART frame with 0.5 stop bit

UART_STOPBITS_1

UART frame with 1 stop bit

UART_STOPBITS_1_5

UART frame with 1.5 stop bits

UART_STOPBITS_2

UART frame with 2 stop bits

UART Advanced Feature Stop Mode Enable

UART_ADVFEATURE_STOPMODE_DISABLE

UART stop mode disable

UART_ADVFEATURE_STOPMODE_ENABLE

UART stop mode enable

UART polling-based communications time-out value

HAL_UART_TIMEOUT_VALUE

UART polling-based communications time-out value

UART Advanced Feature TX Pin Active Level Inversion

UART_ADVFEATURE_TXINV_DISABLE

TX pin active level inversion disable

UART_ADVFEATURE_TXINV_ENABLE

TX pin active level inversion enable

UART WakeUp From Stop Selection

UART_WAKEUP_ON_ADDRESS

UART wake-up on address

UART_WAKEUP_ON_STARTBIT

UART wake-up on start bit

UART_WAKEUP_ON_READDATA_NONEMPTY

UART wake-up on receive data register not empty or RXFIFO is not empty

UART WakeUp Methods

UART_WAKEUPMETHOD_IDLELINE

UART wake-up on idle line

UART_WAKEUPMETHOD_ADDRESSMARK

UART wake-up on address mark

91 HAL UART Extension Driver

91.1 UARTEx Firmware driver registers structures

91.1.1 UART_WakeUpTypeDef

UART_WakeUpTypeDef is defined in the stm32h7xx_hal_uart_ex.h

Data Fields

- ***uint32_t WakeUpEvent***
- ***uint16_t AddressLength***
- ***uint8_t Address***

Field Documentation

- ***uint32_t UART_WakeUpTypeDef::WakeUpEvent***
  Specifies which event will activate the Wakeup from Stop mode flag (WUF). This parameter can be a value of ***UART_WakeUp_from_Stop_Selection***. If set to UART_WAKEUP_ON_ADDRESS, the two other fields below must be filled up.
- ***uint16_t UART_WakeUpTypeDef::AddressLength***
  Specifies whether the address is 4 or 7-bit long. This parameter can be a value of ***UARTEx_WakeUp_Address_Length***.
- ***uint8_t UART_WakeUpTypeDef::Address***
  UART/USART node address (7-bit long max).

91.2 UARTEx Firmware driver API description

The following section lists the various functions of the UARTEx library.

91.2.1 UART peripheral extended features

91.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx or the UARTy in asynchronous mode.

- For the asynchronous mode the parameters below can be configured:
  - Baud Rate
  - Word Length
  - Stop Bit
  - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit.
  - Hardware flow control
  - Receiver/transmitter modes
  - Over Sampling Method
  - One-Bit Sampling Method
- For the asynchronous mode, the following advanced features can be configured as well:
  - TX and/or RX pin level inversion
  - data logical level inversion
  - RX and TX pins swap
  - RX overrun detection disabling
  - DMA disabling on RX error
  - MSB first on communication line
  - auto Baud rate detection

The HAL_RS485Ex_Init() API follows the UART RS485 mode configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- ***HAL_RS485Ex_Init()***

91.2.3 IO operation functions

This section contains the following APIs:

- ***HAL_UARTEx_WakeupCallback()***
- ***HAL_UARTEx_RxFifoFullCallback()***
- ***HAL_UARTEx_TxFifoEmptyCallback()***

91.2.4 Peripheral Control functions

This section provides the following functions:

- HAL_MultiProcessorEx_AddressLength_Set() API optionally sets the UART node address detection length to more than 4 bits for multiprocessor address mark wake up.
- HAL_UARTEx_StopModeWakeUpSourceConfig() API defines the wake-up from stop mode trigger: address match, Start Bit detection or RXNE bit status.
- HAL_UARTEx_EnableStopMode() API enables the UART to wake up the MCU from stop mode
- HAL_UARTEx_DisableStopMode() API disables the above functionality
- HAL_UARTEx_EnableFifoMode() API enables the FIFO mode
- HAL_UARTEx_DisableFifoMode() API disables the FIFO mode
- HAL_UARTEx_SetTxFifoThreshold() API sets the TX FIFO threshold
- HAL_UARTEx_SetRxFifoThreshold() API sets the RX FIFO threshold

This subsection also provides a set of additional functions providing enhanced reception services to user. (For example, these functions allow application to handle use cases where number of data to be received is unknown).

1. Compared to standard reception services which only consider number of received data elements as reception completion criteria, these functions also consider additional events as triggers for updating reception status to caller : (+) Detection of inactivity period (RX line has not been active for a given period).
   - RX inactivity detected by IDLE event, i.e. RX line has been in idle state (normally high state) for 1 frame time, after last received byte.
   - RX inactivity detected by RTO, i.e. line has been in idle state for a programmable time, after last received byte. (+) Detection that a specific character has been received.
2. There are two mode of transfer: (+) Blocking mode: The reception is performed in polling mode, until either expected number of data is received, or till IDLE event occurs. Reception is handled only during function execution. When function exits, no data reception could occur. HAL status and number of actually received data elements, are returned by function after finishing transfer. (+) Non-Blocking mode: The reception is performed using Interrupts or DMA. These API's return the HAL status. The end of the data processing will be indicated through the dedicated UART IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_UARTEx_RxEventCallback() user callback will be executed during Receive process The HAL_UART_ErrorCallback()user callback will be executed when a reception error is detected.
3. Blocking mode API: (+) HAL_UARTEx_ReceiveToIdle()
4. Non-Blocking mode API with Interrupt: (+) HAL_UARTEx_ReceiveToIdle_IT()
5. Non-Blocking mode API with DMA: (+) HAL_UARTEx_ReceiveToIdle_DMA()

This subsection also provides a set of additional functions providing enhanced reception services to user. (For example, these functions allow application to handle use cases where number of data to be received is unknown). (#) Compared to standard reception services which only consider number of received data elements as reception completion criteria, these functions also consider additional events as triggers for updating reception status to caller :

- Detection of inactivity period (RX line has not been active for a given period).
  - RX inactivity detected by IDLE event, i.e. RX line has been in idle state (normally high state) for 1 frame time, after last received byte.
  - RX inactivity detected by RTO, i.e. line has been in idle state for a programmable time, after last received byte.

- Detection that a specific character has been received. (#) There are two mode of transfer:
- Blocking mode: The reception is performed in polling mode, until either expected number of data is received, or till IDLE event occurs. Reception is handled only during function execution. When function exits, no data reception could occur. HAL status and number of actually received data elements, are returned by function after finishing transfer.
- Non-Blocking mode: The reception is performed using Interrupts or DMA. These API's return the HAL status. The end of the data processing will be indicated through the dedicated UART IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_UARTEx_RxEventCallback() user callback will be executed during Receive process The HAL_UART_ErrorCallback()user callback will be executed when a reception error is detected. (#) Blocking mode API:
- HAL_UARTEx_ReceiveToIdle() (#) Non-Blocking mode API with Interrupt:
- HAL_UARTEx_ReceiveToIdle_IT() (#) Non-Blocking mode API with DMA:
- HAL_UARTEx_ReceiveToIdle_DMA()

This section contains the following APIs:

- ***HAL_MultiProcessorEx_AddressLength_Set()***
- ***HAL_UARTEx_StopModeWakeUpSourceConfig()***
- ***HAL_UARTEx_EnableStopMode()***
- ***HAL_UARTEx_DisableStopMode()***
- ***HAL_UARTEx_EnableFifoMode()***
- ***HAL_UARTEx_DisableFifoMode()***
- ***HAL_UARTEx_SetTxFifoThreshold()***
- ***HAL_UARTEx_SetRxFifoThreshold()***
- ***HAL_UARTEx_ReceiveToIdle()***
- ***HAL_UARTEx_ReceiveToIdle_IT()***
- ***HAL_UARTEx_ReceiveToIdle_DMA()***
- ***HAL_UARTEx_GetRxEventType()***

91.2.5 Detailed description of functions

HAL_RS485Ex_Init

Function name

HAL_StatusTypeDef HAL_RS485Ex_Init (UART_HandleTypeDef * huart, uint32_t Polarity, uint32_t AssertionTime, uint32_t DeassertionTime)

Function description

Initialize the RS485 Driver enable feature according to the specified parameters in the UART_InitTypeDef and creates the associated handle.

Parameters

- **huart:** UART handle.
- **Polarity:** Select the driver enable polarity. This parameter can be one of the following values:
  - UART_DE_POLARITY_HIGH DE signal is active high
  - UART_DE_POLARITY_LOW DE signal is active low
- **AssertionTime:** Driver Enable assertion time: 5-bit value defining the time between the activation of the DE (Driver Enable) signal and the beginning of the start bit. It is expressed in sample time units (1/8 or 1/16 bit time, depending on the oversampling rate)
- **DeassertionTime:** Driver Enable deassertion time: 5-bit value defining the time between the end of the last stop bit, in a transmitted message, and the de-activation of the DE (Driver Enable) signal. It is expressed in sample time units (1/8 or 1/16 bit time, depending on the oversampling rate).

Return values

- **HAL:** status

HAL_UARTEx_WakeupCallback

Function name

void HAL_UARTEx_WakeupCallback (UART_HandleTypeDef * huart)

Function description

UART wakeup from Stop mode callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UARTEx_RxFifoFullCallback

Function name

void HAL_UARTEx_RxFifoFullCallback (UART_HandleTypeDef * huart)

Function description

UART RX Fifo full callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UARTEx_TxFifoEmptyCallback

Function name

void HAL_UARTEx_TxFifoEmptyCallback (UART_HandleTypeDef * huart)

Function description

UART TX Fifo empty callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UARTEx_StopModeWakeUpSourceConfig

Function name

HAL_StatusTypeDef HAL_UARTEx_StopModeWakeUpSourceConfig (UART_HandleTypeDef * huart, UART_WakeUpTypeDef WakeUpSelection)

Function description

Set Wakeup from Stop mode interrupt flag selection.

Parameters

- **huart:** UART handle.
- **WakeUpSelection:** Address match, Start Bit detection or RXNE/RXFNE bit status. This parameter can be one of the following values:
  - UART_WAKEUP_ON_ADDRESS
  - UART_WAKEUP_ON_STARTBIT
  - UART_WAKEUP_ON_READDATA_NONEMPTY

Return values

- **HAL:** status

Notes

- It is the application responsibility to enable the interrupt used as usart_wkup interrupt source before entering low-power mode.

HAL_UARTEx_EnableStopMode

Function name

HAL_StatusTypeDef HAL_UARTEx_EnableStopMode (UART_HandleTypeDef * huart)

Function description

Enable UART Stop Mode.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- The UART is able to wake up the MCU from Stop 1 mode as long as UART clock is HSI or LSE.

HAL_UARTEx_DisableStopMode

Function name

HAL_StatusTypeDef HAL_UARTEx_DisableStopMode (UART_HandleTypeDef * huart)

Function description

Disable UART Stop Mode.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_MultiProcessorEx_AddressLength_Set

Function name

HAL_StatusTypeDef HAL_MultiProcessorEx_AddressLength_Set (UART_HandleTypeDef * huart, uint32_t AddressLength)

Function description

By default in multiprocessor mode, when the wake up method is set to address mark, the UART handles only 4-bit long addresses detection; this API allows to enable longer addresses detection (6-, 7- or 8-bit long).

Parameters

- **huart:** UART handle.
- **AddressLength:** This parameter can be one of the following values:
  - UART_ADDRESS_DETECT_4B 4-bit long address
  - UART_ADDRESS_DETECT_7B 6-, 7- or 8-bit long address

Return values

- **HAL:** status

Notes

- Addresses detection lengths are: 6-bit address detection in 7-bit data mode, 7-bit address detection in 8-bit data mode, 8-bit address detection in 9-bit data mode.

HAL_UARTEx_EnableFifoMode

Function name

HAL_StatusTypeDef HAL_UARTEx_EnableFifoMode (UART_HandleTypeDef * huart)

Function description

Enable the FIFO mode.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_UARTEx_DisableFifoMode

Function name

HAL_StatusTypeDef HAL_UARTEx_DisableFifoMode (UART_HandleTypeDef * huart)

Function description

Disable the FIFO mode.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_UARTEx_SetTxFifoThreshold

Function name

HAL_StatusTypeDef HAL_UARTEx_SetTxFifoThreshold (UART_HandleTypeDef * huart, uint32_t Threshold)

Function description

Set the TXFIFO threshold.

Parameters

- **huart:** UART handle.
- **Threshold:** TX FIFO threshold value This parameter can be one of the following values:
  - UART_TXFIFO_THRESHOLD_1_8
  - UART_TXFIFO_THRESHOLD_1_4
  - UART_TXFIFO_THRESHOLD_1_2
  - UART_TXFIFO_THRESHOLD_3_4
  - UART_TXFIFO_THRESHOLD_7_8
  - UART_TXFIFO_THRESHOLD_8_8

Return values

- **HAL:** status

HAL_UARTEx_SetRxFifoThreshold

Function name

HAL_StatusTypeDef HAL_UARTEx_SetRxFifoThreshold (UART_HandleTypeDef * huart, uint32_t Threshold)

Function description

Set the RXFIFO threshold.

Parameters

- **huart:** UART handle.
- **Threshold:** RX FIFO threshold value This parameter can be one of the following values:
  - UART_RXFIFO_THRESHOLD_1_8
  - UART_RXFIFO_THRESHOLD_1_4
  - UART_RXFIFO_THRESHOLD_1_2
  - UART_RXFIFO_THRESHOLD_3_4
  - UART_RXFIFO_THRESHOLD_7_8
  - UART_RXFIFO_THRESHOLD_8_8

Return values

- **HAL:** status

HAL_UARTEx_ReceiveToIdle

Function name

HAL_StatusTypeDef HAL_UARTEx_ReceiveToIdle (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint16_t * RxLen, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode till either the expected number of data is received or an IDLE event occurs.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (uint8_t or uint16_t data elements).
- **Size:** Amount of data elements (uint8_t or uint16_t) to be received.
- **RxLen:** Number of data elements finally received (could be lower than Size, in case reception ends on IDLE event)
- **Timeout:** Timeout duration expressed in ms (covers the whole reception sequence).

Return values

- **HAL:** status

Notes

- HAL_OK is returned if reception is completed (expected number of data has been received) or if reception is stopped after IDLE event (less than the expected number of data has been received) In this case, RxLen output parameter indicates number of data available in reception buffer.
- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of uint16_t. In this case, Size must indicate the number of uint16_t available through pData.
- When FIFO mode is enabled, the RXFNE flag is set as long as the RXFIFO is not empty. Read operations from the RDR register are performed when RXFNE flag is set. From hardware perspective, RXFNE flag and RXNE are mapped on the same bit-field.

HAL_UARTEx_ReceiveToIdle_IT

Function name

HAL_StatusTypeDef HAL_UARTEx_ReceiveToIdle_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in interrupt mode till either the expected number of data is received or an IDLE event occurs.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (uint8_t or uint16_t data elements).
- **Size:** Amount of data elements (uint8_t or uint16_t) to be received.

Return values

- **HAL:** status

Notes

- Reception is initiated by this function call. Further progress of reception is achieved thanks to UART interrupts raised by RXNE and IDLE events. Callback is called at end of reception indicating number of received data elements.
- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of uint16_t. In this case, Size must indicate the number of uint16_t available through pData.

HAL_UARTEx_ReceiveToIdle_DMA

Function name

HAL_StatusTypeDef HAL_UARTEx_ReceiveToIdle_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in DMA mode till either the expected number of data is received or an IDLE event occurs.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (uint8_t or uint16_t data elements).
- **Size:** Amount of data elements (uint8_t or uint16_t) to be received.

Return values

- **HAL:** status

Notes

- Reception is initiated by this function call. Further progress of reception is achieved thanks to DMA services, transferring automatically received data elements in user reception buffer and calling registered callbacks at half/end of reception. UART IDLE events are also used to consider reception phase as ended. In all cases, callback execution will indicate number of received data elements.
- When the UART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position).
- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of uint16_t. In this case, Size must indicate the number of uint16_t available through pData.

HAL_UARTEx_GetRxEventType

Function name

HAL_UART_RxEventTypeTypeDef HAL_UARTEx_GetRxEventType (UART_HandleTypeDef * huart)

Function description

Provide Rx Event type that has lead to RxEvent callback execution.

Parameters

- **huart:** UART handle.

Return values

- **Rx:** Event Type (return vale will be a value of UART RxEvent type values)

Notes

- When HAL_UARTEx_ReceiveToIdle_IT() or HAL_UARTEx_ReceiveToIdle_DMA() API are called, progress of reception process is provided to application through calls of Rx Event callback (either default one HAL_UARTEx_RxEventCallback() or user registered one). As several types of events could occur (IDLE event, Half Transfer, or Transfer Complete), this function allows to retrieve the Rx Event type that has lead to Rx Event callback execution.
- This function is expected to be called within the user implementation of Rx Event Callback, in order to provide the accurate value : In Interrupt Mode : HAL_UART_RXEVENT_TC : when Reception has been completed (expected nb of data has been received)HAL_UART_RXEVENT_IDLE : when Idle event occurred prior reception has been completed (nb of received data is lower than expected one) In DMA Mode :HAL_UART_RXEVENT_TC : when Reception has been completed (expected nb of data has been received)HAL_UART_RXEVENT_HT : when half of expected nb of data has been receivedHAL_UART_RXEVENT_IDLE : when Idle event occurred prior reception has been completed (nb of received data is lower than expected one). In DMA mode, RxEvent callback could be called several times; When DMA is configured in Normal Mode, HT event does not stop Reception process; When DMA is configured in Circular Mode, HT, TC or IDLE events don't stop Reception process;

91.3 UARTEx Firmware driver defines

The following section lists the various define and macros of the module.

91.3.1 UARTEx

UARTEx

UARTEx FIFO mode

UART_FIFOMODE_DISABLE

FIFO mode disable

UART_FIFOMODE_ENABLE

FIFO mode enable

UARTEx RXFIFO threshold level

UART_RXFIFO_THRESHOLD_1_8

RX FIFO reaches 1/8 of its depth

UART_RXFIFO_THRESHOLD_1_4

RX FIFO reaches 1/4 of its depth

UART_RXFIFO_THRESHOLD_1_2

RX FIFO reaches 1/2 of its depth

UART_RXFIFO_THRESHOLD_3_4

RX FIFO reaches 3/4 of its depth

UART_RXFIFO_THRESHOLD_7_8

RX FIFO reaches 7/8 of its depth

UART_RXFIFO_THRESHOLD_8_8

RX FIFO becomes full

UARTEx TXFIFO threshold level

UART_TXFIFO_THRESHOLD_1_8

TX FIFO reaches 1/8 of its depth

UART_TXFIFO_THRESHOLD_1_4

TX FIFO reaches 1/4 of its depth

UART_TXFIFO_THRESHOLD_1_2

TX FIFO reaches 1/2 of its depth

UART_TXFIFO_THRESHOLD_3_4

TX FIFO reaches 3/4 of its depth

UART_TXFIFO_THRESHOLD_7_8

TX FIFO reaches 7/8 of its depth

UART_TXFIFO_THRESHOLD_8_8

TX FIFO becomes empty

UARTEx WakeUp Address Length

UART_ADDRESS_DETECT_4B

4-bit long wake-up address

UART_ADDRESS_DETECT_7B

7-bit long wake-up address

UARTEx Word Length

UART_WORDLENGTH_7B

7-bit long UART frame

UART_WORDLENGTH_8B

8-bit long UART frame

UART_WORDLENGTH_9B

9-bit long UART frame

92 HAL USART Generic Driver

92.1 USART Firmware driver registers structures

92.1.1 USART_InitTypeDef

USART_InitTypeDef is defined in the stm32h7xx_hal_usart.h

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t Mode***
- ***uint32_t CLKPolarity***
- ***uint32_t CLKPhase***
- ***uint32_t CLKLastBit***
- ***uint32_t ClockPrescaler***

Field Documentation

- ***uint32_t USART_InitTypeDef::BaudRate***

  This member configures the Usart communication baud rate. The baud rate is computed using the following formula: Baud Rate Register[15:4] = ((2 * fclk_pres) / ((huart->Init.BaudRate)))[15:4] Baud Rate Register[3] = 0 Baud Rate Register[2:0] = (((2 * fclk_pres) / ((huart->Init.BaudRate)))[3:0]) >> 1 where fclk_pres is the USART input clock frequency (fclk) divided by a prescaler.

  Note:

  – Oversampling by 8 is systematically applied to achieve high baud rates.

- ***uint32_t USART_InitTypeDef::WordLength***

  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***USARTEx_Word_Length***.

- ***uint32_t USART_InitTypeDef::StopBits***

  Specifies the number of stop bits transmitted. This parameter can be a value of ***USART_Stop_Bits***.

- ***uint32_t USART_InitTypeDef::Parity***

  Specifies the parity mode. This parameter can be a value of ***USART_Parity***

  Note:

  – When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).

- ***uint32_t USART_InitTypeDef::Mode***

  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***USART_Mode***.

- ***uint32_t USART_InitTypeDef::CLKPolarity***

  Specifies the steady state of the serial clock. This parameter can be a value of ***USART_Clock_Polarity***.

- ***uint32_t USART_InitTypeDef::CLKPhase***

  Specifies the clock transition on which the bit capture is made. This parameter can be a value of ***USART_Clock_Phase***.

- ***uint32_t USART_InitTypeDef::CLKLastBit***

  Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of ***USART_Last_Bit***.

- ***uint32_t USART_InitTypeDef::ClockPrescaler***

  Specifies the prescaler value used to divide the USART clock source. This parameter can be a value of ***USART_ClockPrescaler***.

92.1.2 __USART_HandleTypeDef

__USART_HandleTypeDef is defined in the stm32h7xx_hal_usart.h

Data Fields

- ***USART_TypeDef * Instance***
- ***USART_InitTypeDef Init***
- ***const uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***uint16_t Mask***
- ***uint16_t NbRxDataToProcess***
- ***uint16_t NbTxDataToProcess***
- ***uint32_t SlaveMode***
- ***uint32_t FifoMode***
- ***void(* RxISR***
- ***void(* TxISR***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_USART_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* TxHalfCpltCallback***
- ***void(* TxCpltCallback***
- ***void(* RxHalfCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* TxRxCpltCallback***
- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* RxFifoFullCallback***
- ***void(* TxFifoEmptyCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***USART_TypeDef* __USART_HandleTypeDef::Instance***
  USART registers base address
- ***USART_InitTypeDef __USART_HandleTypeDef::Init***
  USART communication parameters
- ***const uint8_t* __USART_HandleTypeDef::pTxBuffPtr***
  Pointer to USART Tx transfer Buffer
- ***uint16_t __USART_HandleTypeDef::TxXferSize***
  USART Tx Transfer size
- ***__IO uint16_t __USART_HandleTypeDef::TxXferCount***
  USART Tx Transfer Counter
- ***uint8_t* __USART_HandleTypeDef::pRxBuffPtr***
  Pointer to USART Rx transfer Buffer
- ***uint16_t __USART_HandleTypeDef::RxXferSize***
  USART Rx Transfer size

- ***__IO uint16_t __USART_HandleTypeDef::RxXferCount***
  USART Rx Transfer Counter
- ***uint16_t __USART_HandleTypeDef::Mask***
  USART Rx RDR register mask
- ***uint16_t __USART_HandleTypeDef::NbRxDataToProcess***
  Number of data to process during RX ISR execution
- ***uint16_t __USART_HandleTypeDef::NbTxDataToProcess***
  Number of data to process during TX ISR execution
- ***uint32_t __USART_HandleTypeDef::SlaveMode***
  Enable/Disable UART SPI Slave Mode. This parameter can be a value of ***USARTEx_Slave_Mode***
- ***uint32_t __USART_HandleTypeDef::FifoMode***
  Specifies if the FIFO mode will be used. This parameter can be a value of ***USARTEx_FIFO_mode***.
- ***void(* __USART_HandleTypeDef::RxISR)(struct __USART_HandleTypeDef *husart)***
  Function pointer on Rx IRQ handler
- ***void(* __USART_HandleTypeDef::TxISR)(struct __USART_HandleTypeDef *husart)***
  Function pointer on Tx IRQ handler
- ***DMA_HandleTypeDef* __USART_HandleTypeDef::hdmatx***
  USART Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __USART_HandleTypeDef::hdmarx***
  USART Rx DMA Handle parameters
- ***HAL_LockTypeDef __USART_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_USART_StateTypeDef __USART_HandleTypeDef::State***
  USART communication state
- ***__IO uint32_t __USART_HandleTypeDef::ErrorCode***
  USART Error code
- ***void(* __USART_HandleTypeDef::TxHalfCpltCallback)(struct __USART_HandleTypeDef *husart)***
  USART Tx Half Complete Callback
- ***void(* __USART_HandleTypeDef::TxCpltCallback)(struct __USART_HandleTypeDef *husart)***
  USART Tx Complete Callback
- ***void(* __USART_HandleTypeDef::RxHalfCpltCallback)(struct __USART_HandleTypeDef *husart)***
  USART Rx Half Complete Callback
- ***void(* __USART_HandleTypeDef::RxCpltCallback)(struct __USART_HandleTypeDef *husart)***
  USART Rx Complete Callback
- ***void(* __USART_HandleTypeDef::TxRxCpltCallback)(struct __USART_HandleTypeDef *husart)***
  USART Tx Rx Complete Callback
- ***void(* __USART_HandleTypeDef::ErrorCallback)(struct __USART_HandleTypeDef *husart)***
  USART Error Callback
- ***void(* __USART_HandleTypeDef::AbortCpltCallback)(struct __USART_HandleTypeDef *husart)***
  USART Abort Complete Callback
- ***void(* __USART_HandleTypeDef::RxFifoFullCallback)(struct __USART_HandleTypeDef *husart)***
  USART Rx Fifo Full Callback
- ***void(* __USART_HandleTypeDef::TxFifoEmptyCallback)(struct __USART_HandleTypeDef *husart)***
  USART Tx Fifo Empty Callback
- ***void(* __USART_HandleTypeDef::MspInitCallback)(struct __USART_HandleTypeDef *husart)***
  USART Msp Init callback
- ***void(* __USART_HandleTypeDef::MspDeInitCallback)(struct __USART_HandleTypeDef *husart)***
  USART Msp DeInit callback

92.2 USART Firmware driver API description

The following section lists the various functions of the USART library.

92.2.1 How to use this driver

The USART HAL driver can be used as follows:

1. Declare a USART_HandleTypeDef handle structure (eg. USART_HandleTypeDef husart).
2. Initialize the USART low level resources by implementing the HAL_USART_MspInit() API:
   - Enable the USARTx interface clock.
   - USART pins configuration:
     - Enable the clock for the USART GPIOs.
     - Configure these USART pins as alternate function pull-up.
   - NVIC configuration if you need to use interrupt process (HAL_USART_Transmit_IT(), HAL_USART_Receive_IT() and HAL_USART_TransmitReceive_IT() APIs):
     - Configure the USARTx interrupt priority.
     - Enable the NVIC USART IRQ handle.
   - USART interrupts handling:

Note: The specific USART interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_USART_ENABLE_IT() and __HAL_USART_DISABLE_IT() inside the transmit and receive process.

   - DMA Configuration if you need to use DMA process (HAL_USART_Transmit_DMA() HAL_USART_Receive_DMA() and HAL_USART_TransmitReceive_DMA() APIs):
     - Declare a DMA handle structure for the Tx/Rx channel.
     - Enable the DMAx interface clock.
     - Configure the declared DMA handle structure with the required Tx/Rx parameters.
     - Configure the DMA Tx/Rx channel.
     - Associate the initialized DMA handle to the USART DMA Tx/Rx handle.
     - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
3. Program the Baud Rate, Word Length, Stop Bit, Parity, and Mode (Receiver/Transmitter) in the husart handle Init structure.
4. Initialize the USART registers by calling the HAL_USART_Init() API:
   - This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_USART_MspInit(&husart) API.

Note: To configure and enable/disable the USART to wake up the MCU from stop mode, resort to UART API's HAL_UARTEx_StopModeWakeUpSourceConfig(), HAL_UARTEx_EnableStopMode() and HAL_UARTEx_DisableStopMode() in casting the USART handle to UART type UART_HandleTypeDef.

92.2.2 Callback registration

The compilation define USE_HAL_USART_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_USART_RegisterCallback() to register a user callback. Function HAL_USART_RegisterCallback() allows to register following callbacks:

- TxHalfCpltCallback : Tx Half Complete Callback.
- TxCpltCallback : Tx Complete Callback.
- RxHalfCpltCallback : Rx Half Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- TxRxCpltCallback : Tx Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- RxFifoFullCallback : Rx Fifo Full Callback.
- TxFifoEmptyCallback : Tx Fifo Empty Callback.

- MspInitCallback : USART MspInit.
- MspDeInitCallback : USART MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_USART_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. HAL_USART_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- TxHalfCpltCallback : Tx Half Complete Callback.
- TxCpltCallback : Tx Complete Callback.
- RxHalfCpltCallback : Rx Half Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- TxRxCpltCallback : Tx Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- RxFifoFullCallback : Rx Fifo Full Callback.
- TxFifoEmptyCallback : Tx Fifo Empty Callback.
- MspInitCallback : USART MspInit.
- MspDeInitCallback : USART MspDeInit.

By default, after the HAL_USART_Init() and when the state is HAL_USART_STATE_RESET all callbacks are set to the corresponding weak (surcharged) functions: examples HAL_USART_TxCpltCallback(), HAL_USART_RxHalfCpltCallback(). Exception done for MspInit and MspDeInit functions that are respectively reset to the legacy weak (surcharged) functions in the HAL_USART_Init() and HAL_USART_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_USART_Init() and HAL_USART_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_USART_STATE_READY state only. Exception done MspInit/ MspDeInit that can be registered/unregistered in HAL_USART_STATE_READY or HAL_USART_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_USART_RegisterCallback() before calling HAL_USART_DeInit() or HAL_USART_Init() function.

When The compilation define USE_HAL_USART_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and weak (surcharged) callbacks are used.

92.2.3 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USART in asynchronous and in synchronous modes.

- For the asynchronous mode only these parameters can be configured:
  - Baud Rate
  - Word Length
  - Stop Bit
  - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit.
  - USART polarity
  - USART phase
  - USART LastBit
  - Receiver/transmitter modes

The HAL_USART_Init() function follows the USART synchronous configuration procedure (details for the procedure are available in reference manual).

This section contains the following APIs:

- ***HAL_USART_Init()***
- ***HAL_USART_DeInit()***
- ***HAL_USART_MspInit()***
- ***HAL_USART_MspDeInit()***

- *HAL_USART_RegisterCallback()*
- *HAL_USART_UnRegisterCallback()*

92.2.4 IO operation functions

This subsection provides a set of functions allowing to manage the USART synchronous data transfers.

The USART supports master mode only: it cannot receive or send data related to an input clock (SCLK is always an output).

1. There are two modes of transfer:
   – Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
   – No-Blocking mode: The communication is performed using Interrupts or DMA, These API's return the HAL status. The end of the data processing will be indicated through the dedicated USART IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_USART_TxCpltCallback(), HAL_USART_RxCpltCallback() and HAL_USART_TxRxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive process The HAL_USART_ErrorCallback()user callback will be executed when a communication error is detected
2. Blocking mode API's are :
   – HAL_USART_Transmit() in simplex mode
   – HAL_USART_Receive() in full duplex receive only
   – HAL_USART_TransmitReceive() in full duplex mode
3. Non-Blocking mode API's with Interrupt are :
   – HAL_USART_Transmit_IT() in simplex mode
   – HAL_USART_Receive_IT() in full duplex receive only
   – HAL_USART_TransmitReceive_IT() in full duplex mode
   – HAL_USART_IRQHandler()
4. No-Blocking mode API's with DMA are :
   – HAL_USART_Transmit_DMA() in simplex mode
   – HAL_USART_Receive_DMA() in full duplex receive only
   – HAL_USART_TransmitReceive_DMA() in full duplex mode
   – HAL_USART_DMAPause()
   – HAL_USART_DMAResume()
   – HAL_USART_DMAStop()
5. A set of Transfer Complete Callbacks are provided in Non_Blocking mode:
   – HAL_USART_TxCpltCallback()
   – HAL_USART_RxCpltCallback()
   – HAL_USART_TxHalfCpltCallback()
   – HAL_USART_RxHalfCpltCallback()
   – HAL_USART_ErrorCallback()
   – HAL_USART_TxRxCpltCallback()
6. Non-Blocking mode transfers could be aborted using Abort API's :
   – HAL_USART_Abort()
   – HAL_USART_Abort_IT()
7. For Abort services based on interrupts (HAL_USART_Abort_IT), a Abort Complete Callbacks is provided:
   – HAL_USART_AbortCpltCallback()

8. In Non-Blocking mode transfers, possible errors are split into 2 categories. Errors are handled as follows :
   – Error is considered as Recoverable and non blocking : Transfer could go till end, but error severity is to be evaluated by user : this concerns Frame Error, Parity Error or Noise Error in Interrupt mode reception . Received character is then retrieved and stored in Rx buffer, Error code is set to allow user to identify error type, and HAL_USART_ErrorCallback() user callback is executed. Transfer is kept ongoing on USART side. If user wants to abort it, Abort services should be called by user.
   – Error is considered as Blocking : Transfer could not be completed properly and is aborted. This concerns Overrun Error In Interrupt mode reception and all errors in DMA mode. Error code is set to allow user to identify error type, and HAL_USART_ErrorCallback() user callback is executed.

This section contains the following APIs:

- ***HAL_USART_Transmit()***
- ***HAL_USART_Receive()***
- ***HAL_USART_TransmitReceive()***
- ***HAL_USART_Transmit_IT()***
- ***HAL_USART_Receive_IT()***
- ***HAL_USART_TransmitReceive_IT()***
- ***HAL_USART_Transmit_DMA()***
- ***HAL_USART_Receive_DMA()***
- ***HAL_USART_TransmitReceive_DMA()***
- ***HAL_USART_DMAPause()***
- ***HAL_USART_DMAResume()***
- ***HAL_USART_DMAStop()***
- ***HAL_USART_Abort()***
- ***HAL_USART_Abort_IT()***
- ***HAL_USART_IRQHandler()***
- ***HAL_USART_TxCpltCallback()***
- ***HAL_USART_TxHalfCpltCallback()***
- ***HAL_USART_RxCpltCallback()***
- ***HAL_USART_RxHalfCpltCallback()***
- ***HAL_USART_TxRxCpltCallback()***
- ***HAL_USART_ErrorCallback()***
- ***HAL_USART_AbortCpltCallback()***

92.2.5 Peripheral State and Error functions

This subsection provides functions allowing to :

- Return the USART handle state
- Return the USART handle error code

This section contains the following APIs:

- ***HAL_USART_GetState()***
- ***HAL_USART_GetError()***

92.2.6 Detailed description of functions

HAL_USART_Init

Function name

HAL_StatusTypeDef HAL_USART_Init (USART_HandleTypeDef * husart)

Function description

Initialize the USART mode according to the specified parameters in the USART_InitTypeDef and initialize the associated handle.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USART_DeInit

Function name

HAL_StatusTypeDef HAL_USART_DeInit (USART_HandleTypeDef * husart)

Function description

DeInitialize the USART peripheral.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USART_MspInit

Function name

void HAL_USART_MspInit (USART_HandleTypeDef * husart)

Function description

Initialize the USART MSP.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_MspDeInit

Function name

void HAL_USART_MspDeInit (USART_HandleTypeDef * husart)

Function description

DeInitialize the USART MSP.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_RegisterCallback

Function name

HAL_StatusTypeDef HAL_USART_RegisterCallback (USART_HandleTypeDef * husart, HAL_USART_CallbackIDTypeDef CallbackID, pUSART_CallbackTypeDef pCallback)

Function description

Register a User USART Callback To be used instead of the weak predefined callback.

Parameters

- **husart:** usart handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_USART_TX_HALFCOMPLETE_CB_ID Tx Half Complete Callback ID
  - HAL_USART_TX_COMPLETE_CB_ID Tx Complete Callback ID
  - HAL_USART_RX_HALFCOMPLETE_CB_ID Rx Half Complete Callback ID
  - HAL_USART_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_USART_TX_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_USART_ERROR_CB_ID Error Callback ID
  - HAL_USART_ABORT_COMPLETE_CB_ID Abort Complete Callback ID
  - HAL_USART_RX_FIFO_FULL_CB_ID Rx Fifo Full Callback ID
  - HAL_USART_TX_FIFO_EMPTY_CB_ID Tx Fifo Empty Callback ID
  - HAL_USART_MSPINIT_CB_ID MspInit Callback ID
  - HAL_USART_MSPDEINIT_CB_ID MspDeInit Callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status +

Notes

- The HAL_USART_RegisterCallback() may be called before HAL_USART_Init() in HAL_USART_STATE_RESET to register callbacks for HAL_USART_MSPINIT_CB_ID and HAL_USART_MSPDEINIT_CB_ID

HAL_USART_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_USART_UnRegisterCallback (USART_HandleTypeDef * husart, HAL_USART_CallbackIDTypeDef CallbackID)

Function description

Unregister an USART Callback USART callaback is redirected to the weak predefined callback.

Parameters

- **husart:** usart handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_USART_TX_HALFCOMPLETE_CB_ID Tx Half Complete Callback ID
  - HAL_USART_TX_COMPLETE_CB_ID Tx Complete Callback ID
  - HAL_USART_RX_HALFCOMPLETE_CB_ID Rx Half Complete Callback ID
  - HAL_USART_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_USART_TX_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_USART_ERROR_CB_ID Error Callback ID
  - HAL_USART_ABORT_COMPLETE_CB_ID Abort Complete Callback ID
  - HAL_USART_RX_FIFO_FULL_CB_ID Rx Fifo Full Callback ID
  - HAL_USART_TX_FIFO_EMPTY_CB_ID Tx Fifo Empty Callback ID
  - HAL_USART_MSPINIT_CB_ID MspInit Callback ID
  - HAL_USART_MSPDEINIT_CB_ID MspDeInit Callback ID

Return values

- **HAL:** status

Notes

- The HAL_USART_UnRegisterCallback() may be called before HAL_USART_Init() in HAL_USART_STATE_RESET to un-register callbacks for HAL_USART_MSPINIT_CB_ID and HAL_USART_MSPDEINIT_CB_ID

HAL_USART_Transmit

Function name

HAL_StatusTypeDef HAL_USART_Transmit (USART_HandleTypeDef * husart, const uint8_t * pTxData, uint16_t Size, uint32_t Timeout)

Function description

Simplex send an amount of data in blocking mode.

Parameters

- **husart:** USART handle.
- **pTxData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be sent.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must indicate the number of u16 provided through pTxData.

HAL_USART_Receive

Function name

HAL_StatusTypeDef HAL_USART_Receive (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **husart:** USART handle.
- **pRxData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- To receive synchronous data, dummy data are simultaneously transmitted.
- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must indicate the number of u16 available through pRxData.

HAL_USART_TransmitReceive

Function name

HAL_StatusTypeDef HAL_USART_TransmitReceive (USART_HandleTypeDef * husart, const uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)

Function description

Full-Duplex Send and Receive an amount of data in blocking mode.

Parameters

- **husart:** USART handle.
- **pTxData:** pointer to TX data buffer (u8 or u16 data elements).
- **pRxData:** pointer to RX data buffer (u8 or u16 data elements).
- **Size:** amount of data elements (u8 or u16) to be sent (same amount to be received).
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data and the received data are handled as sets of u16. In this case, Size must indicate the number of u16 available through pTxData and through pRxData.

HAL_USART_Transmit_IT

Function name

HAL_StatusTypeDef HAL_USART_Transmit_IT (USART_HandleTypeDef * husart, const uint8_t * pTxData, uint16_t Size)

Function description

Send an amount of data in interrupt mode.

Parameters

- **husart:** USART handle.
- **pTxData:** pointer to data buffer (u8 or u16 data elements).
- **Size:** amount of data elements (u8 or u16) to be sent.

Return values

- **HAL:** status

Notes

- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must indicate the number of u16 provided through pTxData.

HAL_USART_Receive_IT

Function name

HAL_StatusTypeDef HAL_USART_Receive_IT (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)

Function description

Receive an amount of data in interrupt mode.

Parameters

- **husart:** USART handle.
- **pRxData:** pointer to data buffer (u8 or u16 data elements).
- **Size:** amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- To receive synchronous data, dummy data are simultaneously transmitted.
- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must indicate the number of u16 available through pRxData.

HAL_USART_TransmitReceive_IT

Function name

HAL_StatusTypeDef HAL_USART_TransmitReceive_IT (USART_HandleTypeDef * husart, const uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)

Function description

Full-Duplex Send and Receive an amount of data in interrupt mode.

Parameters

- **husart:** USART handle.
- **pTxData:** pointer to TX data buffer (u8 or u16 data elements).
- **pRxData:** pointer to RX data buffer (u8 or u16 data elements).
- **Size:** amount of data elements (u8 or u16) to be sent (same amount to be received).

Return values

- **HAL:** status

Notes

- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data and the received data are handled as sets of u16. In this case, Size must indicate the number of u16 available through pTxData and through pRxData.

HAL_USART_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_USART_Transmit_DMA (USART_HandleTypeDef * husart, const uint8_t * pTxData, uint16_t Size)

Function description

Send an amount of data in DMA mode.

Parameters

- **husart:** USART handle.
- **pTxData:** pointer to data buffer (u8 or u16 data elements).
- **Size:** amount of data elements (u8 or u16) to be sent.

Return values

- **HAL:** status

Notes

- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must indicate the number of u16 provided through pTxData.

HAL_USART_Receive_DMA

Function name

HAL_StatusTypeDef HAL_USART_Receive_DMA (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)

Function description

Receive an amount of data in DMA mode.

Parameters

- **husart:** USART handle.
- **pRxData:** pointer to data buffer (u8 or u16 data elements).
- **Size:** amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- When the USART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position).
- The USART DMA transmit channel must be configured in order to generate the clock for the slave.
- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must indicate the number of u16 available through pRxData.

HAL_USART_TransmitReceive_DMA

Function name

HAL_StatusTypeDef HAL_USART_TransmitReceive_DMA (USART_HandleTypeDef * husart, const uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)

Function description

Full-Duplex Transmit Receive an amount of data in non-blocking mode.

Parameters

- **husart:** USART handle.
- **pTxData:** pointer to TX data buffer (u8 or u16 data elements).
- **pRxData:** pointer to RX data buffer (u8 or u16 data elements).
- **Size:** amount of data elements (u8 or u16) to be received/sent.

Return values

- **HAL:** status

Notes

- When the USART parity is enabled (PCE = 1) the data received contain the parity bit.
- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data and the received data are handled as sets of u16. In this case, Size must indicate the number of u16 available through pTxData and through pRxData.

HAL_USART_DMAPause

Function name

HAL_StatusTypeDef HAL_USART_DMAPause (USART_HandleTypeDef * husart)

Function description

Pause the DMA Transfer.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USART_DMAResume

Function name

HAL_StatusTypeDef HAL_USART_DMAResume (USART_HandleTypeDef * husart)

Function description

Resume the DMA Transfer.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USART_DMAStop

Function name

HAL_StatusTypeDef HAL_USART_DMAStop (USART_HandleTypeDef * husart)

Function description

Stop the DMA Transfer.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USART_Abort

Function name

HAL_StatusTypeDef HAL_USART_Abort (USART_HandleTypeDef * husart)

Function description

Abort ongoing transfers (blocking mode).

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable USART Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_USART_Abort_IT

Function name

HAL_StatusTypeDef HAL_USART_Abort_IT (USART_HandleTypeDef * husart)

Function description

Abort ongoing transfers (Interrupt mode).

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable USART Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_USART_IRQHandler

Function name

void HAL_USART_IRQHandler (USART_HandleTypeDef * husart)

Function description

Handle USART interrupt request.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_TxHalfCpltCallback

Function name

void HAL_USART_TxHalfCpltCallback (USART_HandleTypeDef * husart)

Function description

Tx Half Transfer completed callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_TxCpltCallback

Function name

void HAL_USART_TxCpltCallback (USART_HandleTypeDef * husart)

Function description

Tx Transfer completed callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_RxCpltCallback

Function name

void HAL_USART_RxCpltCallback (USART_HandleTypeDef * husart)

Function description

Rx Transfer completed callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_RxHalfCpltCallback

Function name

void HAL_USART_RxHalfCpltCallback (USART_HandleTypeDef * husart)

Function description

Rx Half Transfer completed callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_TxRxCpltCallback

Function name

void HAL_USART_TxRxCpltCallback (USART_HandleTypeDef * husart)

Function description

Tx/Rx Transfers completed callback for the non-blocking process.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_ErrorCallback

Function name

void HAL_USART_ErrorCallback (USART_HandleTypeDef * husart)

Function description

USART error callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_AbortCpltCallback

Function name

void HAL_USART_AbortCpltCallback (USART_HandleTypeDef * husart)

Function description

USART Abort Complete callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_GetState

Function name

HAL_USART_StateTypeDef HAL_USART_GetState (const USART_HandleTypeDef * husart)

Function description

Return the USART handle state.

Parameters

- **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART.

Return values

- **USART:** handle state

HAL_USART_GetError

Function name

uint32_t HAL_USART_GetError (const USART_HandleTypeDef * husart)

Function description

Return the USART error code.

Parameters

- **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART.

Return values

- **USART:** handle Error Code

92.3 USART Firmware driver defines

The following section lists the various define and macros of the module.

92.3.1 USART

USART

USART Clock

USART_CLOCK_DISABLE

USART clock disable

USART_CLOCK_ENABLE

USART clock enable

USART Clock Prescaler

USART_PRESCALER_DIV1

fclk_pres = fclk

USART_PRESCALER_DIV2

fclk_pres = fclk/2

USART_PRESCALER_DIV4

fclk_pres = fclk/4

USART_PRESCALER_DIV6

fclk_pres = fclk/6

USART_PRESCALER_DIV8

fclk_pres = fclk/8

USART_PRESCALER_DIV10

fclk_pres = fclk/10

USART_PRESCALER_DIV12

fclk_pres = fclk/12

USART_PRESCALER_DIV16

fclk_pres = fclk/16

USART_PRESCALER_DIV32

fclk_pres = fclk/32

USART_PRESCALER_DIV64

fclk_pres = fclk/64

USART_PRESCALER_DIV128

fclk_pres = fclk/128

USART_PRESCALER_DIV256

fclk_pres = fclk/256

USART Clock Phase

USART_PHASE_1EDGE

USART frame phase on first clock transition

USART_PHASE_2EDGE

USART frame phase on second clock transition

USART Clock Polarity

USART_POLARITY_LOW

Driver enable signal is active high

USART_POLARITY_HIGH

Driver enable signal is active low

USART Error Definition

HAL_USART_ERROR_NONE

No error

HAL_USART_ERROR_PE

Parity error

HAL_USART_ERROR_NE

Noise error

HAL_USART_ERROR_FE

Frame error

HAL_USART_ERROR_ORE

Overrun error

HAL_USART_ERROR_DMA

DMA transfer error

HAL_USART_ERROR_UDR

SPI slave underrun error

HAL_USART_ERROR_INVALID_CALLBACK

Invalid Callback error

HAL_USART_ERROR_RTO

Receiver Timeout error

USART Exported Macros

__HAL_USART_RESET_HANDLE_STATE

Description:

- Reset USART handle state.

Parameters:

- __HANDLE__: USART handle.

Return value:

- None

__HAL_USART_GET_FLAG

Description:

- Check whether the specified USART flag is set or not.

Parameters:

- __HANDLE__: specifies the USART Handle
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - USART_FLAG_TXFT TXFIFO threshold flag
  - USART_FLAG_RXFT RXFIFO threshold flag
  - USART_FLAG_RXFF RXFIFO Full flag
  - USART_FLAG_TXFE TXFIFO Empty flag
  - USART_FLAG_REACK Receive enable acknowledge flag
  - USART_FLAG_TEACK Transmit enable acknowledge flag
  - USART_FLAG_BUSY Busy flag
  - USART_FLAG_UDR SPI slave underrun error flag
  - USART_FLAG_TXE Transmit data register empty flag
  - USART_FLAG_TXFNF TXFIFO not full flag
  - USART_FLAG_TC Transmission Complete flag
  - USART_FLAG_RXNE Receive data register not empty flag
  - USART_FLAG_RXFNE RXFIFO not empty flag
  - USART_FLAG_RTOF Receiver Timeout flag
  - USART_FLAG_IDLE Idle Line detection flag
  - USART_FLAG_ORE OverRun Error flag
  - USART_FLAG_NE Noise Error flag
  - USART_FLAG_FE Framing Error flag
  - USART_FLAG_PE Parity Error flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_USART_CLEAR_FLAG

Description:

- Clear the specified USART pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - USART_CLEAR_PEF Parity Error Clear Flag
  - USART_CLEAR_FEF Framing Error Clear Flag
  - USART_CLEAR_NEF Noise detected Clear Flag
  - USART_CLEAR_OREF Overrun Error Clear Flag
  - USART_CLEAR_IDLEF IDLE line detected Clear Flag
  - USART_CLEAR_TXFECF TXFIFO empty clear Flag
  - USART_CLEAR_TCF Transmission Complete Clear Flag
  - USART_CLEAR_RTOF Receiver Timeout clear flag
  - USART_CLEAR_UDRF SPI slave underrun error Clear Flag

Return value:

- None

__HAL_USART_CLEAR_PEFLAG

Description:

- Clear the USART PE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_FEFLAG

Description:

- Clear the USART FE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_NEFLAG

Description:

- Clear the USART NE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_OREFLAG

Description:

- Clear the USART ORE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_IDLEFLAG

Description:

- Clear the USART IDLE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_TXFECF

Description:

- Clear the USART TX FIFO empty clear flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_UDRFLAG

Description:

- Clear SPI slave underrun error flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_ENABLE_IT

Description:

- Enable the specified USART interrupt.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __INTERRUPT__: specifies the USART interrupt source to enable. This parameter can be one of the following values:
  - USART_IT_RXFF RXFIFO Full interrupt
  - USART_IT_TXFE TXFIFO Empty interrupt
  - USART_IT_RXFT RXFIFO threshold interrupt
  - USART_IT_TXFT TXFIFO threshold interrupt
  - USART_IT_TXE Transmit Data Register empty interrupt
  - USART_IT_TXFNF TX FIFO not full interrupt
  - USART_IT_TC Transmission complete interrupt
  - USART_IT_RXNE Receive Data register not empty interrupt
  - USART_IT_RXFNE RXFIFO not empty interrupt
  - USART_IT_IDLE Idle line detection interrupt
  - USART_IT_PE Parity Error interrupt
  - USART_IT_ERR Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

__HAL_USART_DISABLE_IT

Description:

- Disable the specified USART interrupt.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __INTERRUPT__: specifies the USART interrupt source to disable. This parameter can be one of the following values:
  - USART_IT_RXFF RXFIFO Full interrupt
  - USART_IT_TXFE TXFIFO Empty interrupt
  - USART_IT_RXFT RXFIFO threshold interrupt
  - USART_IT_TXFT TXFIFO threshold interrupt
  - USART_IT_TXE Transmit Data Register empty interrupt
  - USART_IT_TXFNF TX FIFO not full interrupt
  - USART_IT_TC Transmission complete interrupt
  - USART_IT_RXNE Receive Data register not empty interrupt
  - USART_IT_RXFNE RXFIFO not empty interrupt
  - USART_IT_IDLE Idle line detection interrupt
  - USART_IT_PE Parity Error interrupt
  - USART_IT_ERR Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

__HAL_USART_GET_IT

Description:

- Check whether the specified USART interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __INTERRUPT__: specifies the USART interrupt source to check. This parameter can be one of the following values:
  - USART_IT_RXFF RXFIFO Full interrupt
  - USART_IT_TXFE TXFIFO Empty interrupt
  - USART_IT_RXFT RXFIFO threshold interrupt
  - USART_IT_TXFT TXFIFO threshold interrupt
  - USART_IT_TXE Transmit Data Register empty interrupt
  - USART_IT_TXFNF TX FIFO not full interrupt
  - USART_IT_TC Transmission complete interrupt
  - USART_IT_RXNE Receive Data register not empty interrupt
  - USART_IT_RXFNE RXFIFO not empty interrupt
  - USART_IT_IDLE Idle line detection interrupt
  - USART_IT_ORE OverRun Error interrupt
  - USART_IT_NE Noise Error interrupt
  - USART_IT_FE Framing Error interrupt
  - USART_IT_PE Parity Error interrupt

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_USART_GET_IT_SOURCE

Description:

- Check whether the specified USART interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __INTERRUPT__: specifies the USART interrupt source to check. This parameter can be one of the following values:
  - USART_IT_RXFF RXFIFO Full interrupt
  - USART_IT_TXFE TXFIFO Empty interrupt
  - USART_IT_RXFT RXFIFO threshold interrupt
  - USART_IT_TXFT TXFIFO threshold interrupt
  - USART_IT_TXE Transmit Data Register empty interrupt
  - USART_IT_TXFNF TX FIFO not full interrupt
  - USART_IT_TC Transmission complete interrupt
  - USART_IT_RXNE Receive Data register not empty interrupt
  - USART_IT_RXFNE RXFIFO not empty interrupt
  - USART_IT_IDLE Idle line detection interrupt
  - USART_IT_ORE OverRun Error interrupt
  - USART_IT_NE Noise Error interrupt
  - USART_IT_FE Framing Error interrupt
  - USART_IT_PE Parity Error interrupt

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_USART_CLEAR_IT

Description:

- Clear the specified USART ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt. This parameter can be one of the following values:
  - USART_CLEAR_PEF Parity Error Clear Flag
  - USART_CLEAR_FEF Framing Error Clear Flag
  - USART_CLEAR_NEF Noise detected Clear Flag
  - USART_CLEAR_OREF Overrun Error Clear Flag
  - USART_CLEAR_IDLEF IDLE line detected Clear Flag
  - USART_CLEAR_RTOF Receiver timeout clear flag
  - USART_CLEAR_TXFECF TXFIFO empty clear Flag
  - USART_CLEAR_TCF Transmission Complete Clear Flag

Return value:

- None

__HAL_USART_SEND_REQ

Description:

- Set a specific USART request flag.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __REQ__: specifies the request flag to set. This parameter can be one of the following values:
  - USART_RXDATA_FLUSH_REQUEST Receive Data flush Request
  - USART_TXDATA_FLUSH_REQUEST Transmit data flush Request

Return value:

- None

__HAL_USART_ONE_BIT_SAMPLE_ENABLE

Description:

- Enable the USART one bit sample method.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_ONE_BIT_SAMPLE_DISABLE

Description:

- Disable the USART one bit sample method.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_ENABLE

Description:

- Enable USART.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_DISABLE

Description:

- Disable USART.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

USART Flags

USART_FLAG_TXFT

USART TXFIFO threshold flag

USART_FLAG_RXFT

USART RXFIFO threshold flag

USART_FLAG_RXFF

USART RXFIFO Full flag

USART_FLAG_TXFE

USART TXFIFO Empty flag

USART_FLAG_REACK

USART receive enable acknowledge flag

USART_FLAG_TEACK

USART transmit enable acknowledge flag

USART_FLAG_BUSY

USART busy flag

USART_FLAG_UDR

SPI slave underrun error flag

USART_FLAG_TXE

USART transmit data register empty

USART_FLAG_TXFNF

USART TXFIFO not full

USART_FLAG_RTOF

USART receiver timeout flag

USART_FLAG_TC

USART transmission complete

USART_FLAG_RXNE

USART read data register not empty

USART_FLAG_RXFNE

USART RXFIFO not empty

USART_FLAG_IDLE

USART idle flag

USART_FLAG_ORE

USART overrun error

USART_FLAG_NE

USART noise error

USART_FLAG_FE

USART frame error

USART_FLAG_PE

USART parity error

USART Interruption Flags Mask

USART_IT_MASK

USART interruptions flags mask

USART_CR_MASK

USART control register mask

USART_CR_POS

USART control register position

USART_ISR_MASK

USART ISR register mask

USART_ISR_POS

USART ISR register position

USART Interrupts Definition

USART_IT_PE

USART parity error interruption

USART_IT_TXE

USART transmit data register empty interruption

USART_IT_TXFNF

USART TX FIFO not full interruption

USART_IT_TC

USART transmission complete interruption

USART_IT_RXNE

USART read data register not empty interruption

USART_IT_RXFNE

USART RXFIFO not empty interruption

USART_IT_IDLE

USART idle interruption

USART_IT_ERR

USART error interruption

USART_IT_ORE

USART overrun error interruption

USART_IT_NE

USART noise error interruption

USART_IT_FE

USART frame error interruption

USART_IT_RXFF

USART RXFIFO full interruption

USART_IT_TXFE

USART TXFIFO empty interruption

USART_IT_RXFT

USART RXFIFO threshold reached interruption

USART_IT_TXFT

USART TXFIFO threshold reached interruption

USART Interruption Clear Flags

USART_CLEAR_PEF

Parity Error Clear Flag

USART_CLEAR_FEF

Framing Error Clear Flag

USART_CLEAR_NEF

Noise Error detected Clear Flag

USART_CLEAR_OREF

OverRun Error Clear Flag

USART_CLEAR_IDLEF

IDLE line detected Clear Flag

USART_CLEAR_TCF

Transmission Complete Clear Flag

USART_CLEAR_UDRF

SPI slave underrun error Clear Flag

USART_CLEAR_TXFECF

TXFIFO Empty Clear Flag

USART_CLEAR_RTOF

USART receiver timeout clear flag

USART Last Bit

USART_LASTBIT_DISABLE

USART frame last data bit clock pulse not output to SCLK pin

USART_LASTBIT_ENABLE

USART frame last data bit clock pulse output to SCLK pin

USART Mode

USART_MODE_RX

RX mode

USART_MODE_TX

TX mode

USART_MODE_TX_RX

RX and TX mode

USART Parity

USART_PARITY_NONE

No parity

USART_PARITY_EVEN

Even parity

USART_PARITY_ODD

Odd parity

USART Request Parameters

USART_RXDATA_FLUSH_REQUEST

Receive Data flush Request

USART_TXDATA_FLUSH_REQUEST

Transmit data flush Request

USART Number of Stop Bits

USART_STOPBITS_0_5

USART frame with 0.5 stop bit

USART_STOPBITS_1

USART frame with 1 stop bit

USART_STOPBITS_1_5

USART frame with 1.5 stop bits

USART_STOPBITS_2

USART frame with 2 stop bits

93 HAL USART Extension Driver

93.1 USARTEx Firmware driver API description

The following section lists the various functions of the USARTEx library.

93.1.1 USART peripheral extended features

93.1.2 IO operation functions

This section contains the following APIs:

- ***HAL_USARTEx_RxFifoFullCallback()***
- ***HAL_USARTEx_TxFifoEmptyCallback()***

93.1.3 Peripheral Control functions

This section provides the following functions:

- HAL_USARTEx_EnableSPISlaveMode() API enables the SPI slave mode
- HAL_USARTEx_DisableSPISlaveMode() API disables the SPI slave mode
- HAL_USARTEx_ConfigNSS API configures the Slave Select input pin (NSS)
- HAL_USARTEx_EnableFifoMode() API enables the FIFO mode
- HAL_USARTEx_DisableFifoMode() API disables the FIFO mode
- HAL_USARTEx_SetTxFifoThreshold() API sets the TX FIFO threshold
- HAL_USARTEx_SetRxFifoThreshold() API sets the RX FIFO threshold

This section contains the following APIs:

- ***HAL_USARTEx_EnableSlaveMode()***
- ***HAL_USARTEx_DisableSlaveMode()***
- ***HAL_USARTEx_ConfigNSS()***
- ***HAL_USARTEx_EnableFifoMode()***
- ***HAL_USARTEx_DisableFifoMode()***
- ***HAL_USARTEx_SetTxFifoThreshold()***
- ***HAL_USARTEx_SetRxFifoThreshold()***

93.1.4 Detailed description of functions

HAL_USARTEx_RxFifoFullCallback

Function name

void HAL_USARTEx_RxFifoFullCallback (USART_HandleTypeDef * husart)

Function description

USART RX Fifo full callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USARTEx_TxFifoEmptyCallback

Function name

void HAL_USARTEx_TxFifoEmptyCallback (USART_HandleTypeDef * husart)

Function description

USART TX Fifo empty callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USARTEx_EnableSlaveMode

Function name

HAL_StatusTypeDef HAL_USARTEx_EnableSlaveMode (USART_HandleTypeDef * husart)

Function description

Enable the SPI slave mode.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

Notes

- When the USART operates in SPI slave mode, it handles data flow using the serial interface clock derived from the external SCLK signal provided by the external master SPI device.
- In SPI slave mode, the USART must be enabled before starting the master communications (or between frames while the clock is stable). Otherwise, if the USART slave is enabled while the master is in the middle of a frame, it will become desynchronized with the master.
- The data register of the slave needs to be ready before the first edge of the communication clock or before the end of the ongoing communication, otherwise the SPI slave will transmit zeros.

HAL_USARTEx_DisableSlaveMode

Function name

HAL_StatusTypeDef HAL_USARTEx_DisableSlaveMode (USART_HandleTypeDef * husart)

Function description

Disable the SPI slave mode.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USARTEx_ConfigNSS

Function name

HAL_StatusTypeDef HAL_USARTEx_ConfigNSS (USART_HandleTypeDef * husart, uint32_t NSSConfig)

Function description

Configure the Slave Select input pin (NSS).

Parameters

- **husart:** USART handle.
- **NSSConfig:** NSS configuration. This parameter can be one of the following values:
  - USART_NSS_HARD
  - USART_NSS_SOFT

Return values

- **HAL:** status

Notes

- Software NSS management: SPI slave will always be selected and NSS input pin will be ignored.
- Hardware NSS management: the SPI slave selection depends on NSS input pin. The slave is selected when NSS is low and deselected when NSS is high.

HAL_USARTEx_EnableFifoMode

Function name

HAL_StatusTypeDef HAL_USARTEx_EnableFifoMode (USART_HandleTypeDef * husart)

Function description

Enable the FIFO mode.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USARTEx_DisableFifoMode

Function name

HAL_StatusTypeDef HAL_USARTEx_DisableFifoMode (USART_HandleTypeDef * husart)

Function description

Disable the FIFO mode.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USARTEx_SetTxFifoThreshold

Function name

HAL_StatusTypeDef HAL_USARTEx_SetTxFifoThreshold (USART_HandleTypeDef * husart, uint32_t Threshold)

Function description

Set the TXFIFO threshold.

Parameters

- **husart:** USART handle.
- **Threshold:** TX FIFO threshold value This parameter can be one of the following values:
  - USART_TXFIFO_THRESHOLD_1_8
  - USART_TXFIFO_THRESHOLD_1_4
  - USART_TXFIFO_THRESHOLD_1_2
  - USART_TXFIFO_THRESHOLD_3_4
  - USART_TXFIFO_THRESHOLD_7_8
  - USART_TXFIFO_THRESHOLD_8_8

Return values

- **HAL:** status

HAL_USARTEx_SetRxFifoThreshold

Function name

HAL_StatusTypeDef HAL_USARTEx_SetRxFifoThreshold (USART_HandleTypeDef * husart, uint32_t Threshold)

Function description

Set the RXFIFO threshold.

Parameters

- **husart:** USART handle.
- **Threshold:** RX FIFO threshold value This parameter can be one of the following values:
  - USART_RXFIFO_THRESHOLD_1_8
  - USART_RXFIFO_THRESHOLD_1_4
  - USART_RXFIFO_THRESHOLD_1_2
  - USART_RXFIFO_THRESHOLD_3_4
  - USART_RXFIFO_THRESHOLD_7_8
  - USART_RXFIFO_THRESHOLD_8_8

Return values

- **HAL:** status

93.2 USARTEx Firmware driver defines

The following section lists the various define and macros of the module.

93.2.1 USARTEx

USARTEx

USARTEx FIFO mode

USART_FIFOMODE_DISABLE

FIFO mode disable

USART_FIFOMODE_ENABLE

FIFO mode enable

USARTEx RXFIFO threshold level

USART_RXFIFO_THRESHOLD_1_8

RXFIFO FIFO reaches 1/8 of its depth

USART_RXFIFO_THRESHOLD_1_4

RXFIFO FIFO reaches 1/4 of its depth

USART_RXFIFO_THRESHOLD_1_2

RXFIFO FIFO reaches 1/2 of its depth

USART_RXFIFO_THRESHOLD_3_4

RXFIFO FIFO reaches 3/4 of its depth

USART_RXFIFO_THRESHOLD_7_8

RXFIFO FIFO reaches 7/8 of its depth

USART_RXFIFO_THRESHOLD_8_8

RXFIFO FIFO becomes full

USARTEx Synchronous Slave mode enable

USART_SLAVEMODE_DISABLE

USART SPI Slave Mode Enable

USART_SLAVEMODE_ENABLE

USART SPI Slave Mode Disable

USARTEx Slave Select Management

USART_NSS_HARD

SPI slave selection depends on NSS input pin

USART_NSS_SOFT

SPI slave is always selected and NSS input pin is ignored

USARTEx TXFIFO threshold level

USART_TXFIFO_THRESHOLD_1_8

TXFIFO reaches 1/8 of its depth

USART_TXFIFO_THRESHOLD_1_4

TXFIFO reaches 1/4 of its depth

USART_TXFIFO_THRESHOLD_1_2

TXFIFO reaches 1/2 of its depth

USART_TXFIFO_THRESHOLD_3_4

TXFIFO reaches 3/4 of its depth

USART_TXFIFO_THRESHOLD_7_8

TXFIFO reaches 7/8 of its depth

USART_TXFIFO_THRESHOLD_8_8

TXFIFO becomes empty

USARTEx Word Length

USART_WORDLENGTH_7B

7-bit long USART frame

USART_WORDLENGTH_8B

8-bit long USART frame

USART_WORDLENGTH_9B

9-bit long USART frame

94 HAL WWDG Generic Driver

94.1 WWDG Firmware driver registers structures

94.1.1 WWDG_InitTypeDef

WWDG_InitTypeDef is defined in the stm32h7xx_hal_wwdg.h

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t Window***
- ***uint32_t Counter***
- ***uint32_t EWIMode***

Field Documentation

- ***uint32_t WWDG_InitTypeDef::Prescaler***
  Specifies the prescaler value of the WWDG. This parameter can be a value of ***WWDG_Prescaler***
- ***uint32_t WWDG_InitTypeDef::Window***
  Specifies the WWDG window value to be compared to the downcounter. This parameter must be a number Min_Data = 0x40 and Max_Data = 0x7F
- ***uint32_t WWDG_InitTypeDef::Counter***
  Specifies the WWDG free-running downcounter value. This parameter must be a number between Min_Data = 0x40 and Max_Data = 0x7F
- ***uint32_t WWDG_InitTypeDef::EWIMode***
  Specifies if WWDG Early Wakeup Interrupt is enable or not. This parameter can be a value of ***WWDG_EWI_Mode***

94.1.2 __WWDG_HandleTypeDef

__WWDG_HandleTypeDef is defined in the stm32h7xx_hal_wwdg.h

Data Fields

- ***WWDG_TypeDef * Instance***
- ***WWDG_InitTypeDef Init***
- ***void(* EwiCallback***
- ***void(* MspInitCallback***

Field Documentation

- ***WWDG_TypeDef* __WWDG_HandleTypeDef::Instance***
  Register base address
- ***WWDG_InitTypeDef __WWDG_HandleTypeDef::Init***
  WWDG required parameters
- ***void(* __WWDG_HandleTypeDef::EwiCallback)(struct __WWDG_HandleTypeDef *hwwdg)***
  WWDG Early WakeUp Interrupt callback
- ***void(* __WWDG_HandleTypeDef::MspInitCallback)(struct __WWDG_HandleTypeDef *hwwdg)***
  WWDG Msp Init callback

94.2 WWDG Firmware driver API description

The following section lists the various functions of the WWDG library.

94.2.1 WWDG Specific features

Once enabled the WWDG generates a system reset on expiry of a programmed time period, unless the program refreshes the counter (T[6;0] downcounter) before reaching 0x3F value (i.e. a reset is generated when the counter value rolls down from 0x40 to 0x3F).

- An MCU reset is also generated if the counter value is refreshed before the counter has reached the refresh window value. This implies that the counter must be refreshed in a limited window.
- Once enabled the WWDG cannot be disabled except by a system reset.
- If required by application, an Early Wakeup Interrupt can be triggered in order to be warned before WWDG expiration. The Early Wakeup Interrupt (EWI) can be used if specific safety operations or data logging must be performed before the actual reset is generated. When the downcounter reaches 0x40, interrupt occurs. This mechanism requires WWDG interrupt line to be enabled in NVIC. Once enabled, EWI interrupt cannot be disabled except by a system reset.
- WWDGRST flag in RCC CSR register can be used to inform when a WWDG reset occurs.
- The WWDG counter input clock is derived from the APB clock divided by a programmable prescaler.
- WWDG clock (Hz) = PCLK1 / (4096 * Prescaler)
- WWDG timeout (mS) = 1000 * (T[5;0] + 1) / WWDG clock (Hz) where T[5;0] are the lowest 6 bits of Counter.
- WWDG Counter refresh is allowed between the following limits :
  - min time (mS) = 1000 * (Counter - Window) / WWDG clock
  - max time (mS) = 1000 * (Counter - 0x40) / WWDG clock
- Typical values (case of STM32H74x/5x devices):
  - Counter min (T[5;0] = 0x00) @100MHz (PCLK1) with zero prescaler: max timeout before reset: approximately 40.96us
  - Counter max (T[5;0] = 0x3F) @100MHz (PCLK1) with prescaler dividing by 128: max timeout before reset: approximately 335.54ms
- Typical values (case of STM32H7Ax/Bx devices):
  - Counter min (T[5;0] = 0x00) @140MHz (PCLK1) with zero prescaler: max timeout before reset: approximately 29.25us
  - Counter max (T[5;0] = 0x3F) @140MHz (PCLK1) with prescaler dividing by 128: max timeout before reset: approximately 239.67ms
- Typical values (case of STM32H72x/3x devices):
  - Counter min (T[5;0] = 0x00) @125MHz (PCLK1) with zero prescaler: max timeout before reset: approximately 32.76us
  - Counter max (T[5;0] = 0x3F) @125MHz (PCLK1) with prescaler dividing by 128: max timeout before reset: approximately 268.43ms

94.2.2 How to use this driver

Common driver usage

- Enable WWDG APB1 clock using __HAL_RCC_WWDG_CLK_ENABLE().
- Configure the WWDG prescaler, refresh window value, counter value and early interrupt status using HAL_WWDG_Init() function. This will automatically enable WWDG and start its downcounter. Time reference can be taken from function exit. Care must be taken to provide a counter value greater than 0x40 to prevent generation of immediate reset.
- If the Early Wakeup Interrupt (EWI) feature is enabled, an interrupt is generated when the counter reaches 0x40. When HAL_WWDG_IRQHandler is triggered by the interrupt service routine, flag will be automatically cleared and HAL_WWDG_WakeupCallback user callback will be executed. User can add his own code by customization of callback HAL_WWDG_WakeupCallback.
- Then the application program must refresh the WWDG counter at regular intervals during normal operation to prevent an MCU reset, using HAL_WWDG_Refresh() function. This operation must occur only when the counter is lower than the refresh window value already programmed.

Callback registration

The compilation define USE_HAL_WWDG_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_WWDG_RegisterCallback() to register a user callback.

- Function HAL_WWDG_RegisterCallback() allows to register following callbacks:
  - EwiCallback : callback for Early WakeUp Interrupt.
  - MspInitCallback : WWDG MspInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.
- Use function HAL_WWDG_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. HAL_WWDG_UnRegisterCallback() takes as parameters the HAL peripheral handle and the Callback ID. This function allows to reset following callbacks:
  - EwiCallback : callback for Early WakeUp Interrupt.
  - MspInitCallback : WWDG MspInit.

When calling HAL_WWDG_Init function, callbacks are reset to the corresponding legacy weak (surcharged) functions: HAL_WWDG_EarlyWakeupCallback() and HAL_WWDG_MspInit() only if they have not been registered before.

When compilation define USE_HAL_WWDG_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

WWDG HAL driver macros list

Below the list of available macros in WWDG HAL driver.

- __HAL_WWDG_ENABLE: Enable the WWDG peripheral
- __HAL_WWDG_GET_FLAG: Get the selected WWDG's flag status
- __HAL_WWDG_CLEAR_FLAG: Clear the WWDG's pending flags
- __HAL_WWDG_ENABLE_IT: Enable the WWDG early wakeup interrupt

94.2.3 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and start the WWDG according to the specified parameters in the WWDG_InitTypeDef of associated handle.
- Initialize the WWDG MSP.

This section contains the following APIs:

- ***HAL_WWDG_Init()***
- ***HAL_WWDG_MspInit()***
- ***HAL_WWDG_RegisterCallback()***
- ***HAL_WWDG_UnRegisterCallback()***

94.2.4 IO operation functions

This section provides functions allowing to:

- Refresh the WWDG.
- Handle WWDG interrupt request and associated function callback.

This section contains the following APIs:

- ***HAL_WWDG_Refresh()***
- ***HAL_WWDG_IRQHandler()***
- ***HAL_WWDG_EarlyWakeupCallback()***

94.2.5 Detailed description of functions

HAL_WWDG_Init

Function name

HAL_StatusTypeDef HAL_WWDG_Init (WWDG_HandleTypeDef * hwwdg)

Function description

Initialize the WWDG according to the specified.

Parameters

- **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- **HAL:** status

HAL_WWDG_MspInit

Function name

void HAL_WWDG_MspInit (WWDG_HandleTypeDef * hwwdg)

Function description

Initialize the WWDG MSP.

Parameters

- **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- **None:**

Notes

- When rewriting this function in user file, mechanism may be added to avoid multiple initialize when HAL_WWDG_Init function is called again to change parameters.

HAL_WWDG_RegisterCallback

Function name

HAL_StatusTypeDef HAL_WWDG_RegisterCallback (WWDG_HandleTypeDef * hwwdg, HAL_WWDG_CallbackIDTypeDef CallbackID, pWWDG_CallbackTypeDef pCallback)

Function description

Register a User WWDG Callback To be used instead of the weak (surcharged) predefined callback.

Parameters

- **hwwdg:** WWDG handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_WWDG_EWI_CB_ID Early WakeUp Interrupt Callback ID
  - HAL_WWDG_MSPINIT_CB_ID MspInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **status:**

HAL_WWDG_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_WWDG_UnRegisterCallback (WWDG_HandleTypeDef * hwwdg, HAL_WWDG_CallbackIDTypeDef CallbackID)

Function description

Unregister a WWDG Callback WWDG Callback is redirected to the weak (surcharged) predefined callback.

Parameters

- **hwwdg:** WWDG handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_WWDG_EWI_CB_ID Early WakeUp Interrupt Callback ID
  - HAL_WWDG_MSPINIT_CB_ID MspInit callback ID

Return values

- **status:**

HAL_WWDG_Refresh

Function name

HAL_StatusTypeDef HAL_WWDG_Refresh (WWDG_HandleTypeDef * hwwdg)

Function description

Refresh the WWDG.

Parameters

- **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- **HAL:** status

HAL_WWDG_IRQHandler

Function name

void HAL_WWDG_IRQHandler (WWDG_HandleTypeDef * hwwdg)

Function description

Handle WWDG interrupt request.

Parameters

- **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- **None:**

Notes

- The Early Wakeup Interrupt (EWI) can be used if specific safety operations or data logging must be performed before the actual reset is generated. The EWI interrupt is enabled by calling HAL_WWDG_Init function with EWIMode set to WWDG_EWI_ENABLE. When the downcounter reaches the value 0x40, and EWI interrupt is generated and the corresponding Interrupt Service Routine (ISR) can be used to trigger specific actions (such as communications or data logging), before resetting the device.

HAL_WWDG_EarlyWakeupCallback

Function name

void HAL_WWDG_EarlyWakeupCallback (WWDG_HandleTypeDef * hwwdg)

Function description

WWDG Early Wakeup callback.

Parameters

- **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- **None:**

94.3 WWDG Firmware driver defines

The following section lists the various define and macros of the module.

94.3.1 WWDG

WWDG

WWDG Early Wakeup Interrupt Mode

WWDG_EWI_DISABLE

EWI Disable

WWDG_EWI_ENABLE

EWI Enable

WWDG Exported Macros

__HAL_WWDG_ENABLE

Description:

- Enable the WWDG peripheral.

Parameters:

- __HANDLE__: WWDG handle

Return value:

- None

__HAL_WWDG_ENABLE_IT

Description:

- Enable the WWDG early wakeup interrupt.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the interrupt to enable. This parameter can be one of the following values:
  - WWDG_IT_EWI: Early wakeup interrupt

Return value:

- None

Notes:

- Once enabled this interrupt cannot be disabled except by a system reset.

__HAL_WWDG_GET_IT

Description:

- Check whether the selected WWDG interrupt has occurred or not.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the it to check. This parameter can be one of the following values:
  - WWDG_FLAG_EWIF: Early wakeup interrupt IT

Return value:

- The: new state of WWDG_FLAG (SET or RESET).

__HAL_WWDG_CLEAR_IT

Description:

- Clear the WWDG interrupt pending bits.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
  - WWDG_FLAG_EWIF: Early wakeup interrupt flag

__HAL_WWDG_GET_FLAG

Description:

- Check whether the specified WWDG flag is set or not.

Parameters:

- __HANDLE__: WWDG handle
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - WWDG_FLAG_EWIF: Early wakeup interrupt flag

Return value:

- The: new state of WWDG_FLAG (SET or RESET).

__HAL_WWDG_CLEAR_FLAG

Description:

- Clear the WWDG's pending flags.

Parameters:

- __HANDLE__: WWDG handle
- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
  - WWDG_FLAG_EWIF: Early wakeup interrupt flag

Return value:

- None

__HAL_WWDG_GET_IT_SOURCE

Description:

- Check whether the specified WWDG interrupt source is enabled or not.

Parameters:

- __HANDLE__: WWDG Handle.
- __INTERRUPT__: specifies the WWDG interrupt source to check. This parameter can be one of the following values:
  - WWDG_IT_EWI: Early Wakeup Interrupt

Return value:

- state: of __INTERRUPT__ (TRUE or FALSE).

WWDG Flag definition

WWDG_FLAG_EWIF

Early wakeup interrupt flag

WWDG Interrupt definition

WWDG_IT_EWI

Early wakeup interrupt

WWDG Prescaler

WWDG_PRESCALER_1

WWDG counter clock = (PCLK1/4096)/1

WWDG_PRESCALER_2

WWDG counter clock = (PCLK1/4096)/2

WWDG_PRESCALER_4

WWDG counter clock = (PCLK1/4096)/4

WWDG_PRESCALER_8

WWDG counter clock = (PCLK1/4096)/8

WWDG_PRESCALER_16

WWDG counter clock = (PCLK1/4096)/16

WWDG_PRESCALER_32

WWDG counter clock = (PCLK1/4096)/32

WWDG_PRESCALER_64

WWDG counter clock = (PCLK1/4096)/64

WWDG_PRESCALER_128

WWDG counter clock = (PCLK1/4096)/128

95 LL ADC Generic Driver

95.1 ADC Firmware driver registers structures

95.1.1 LL_ADC_CommonInitTypeDef

LL_ADC_CommonInitTypeDef is defined in the stm32h7xx_ll_adc.h

Data Fields

- ***uint32_t CommonClock***
- ***uint32_t Multimode***
- ***uint32_t MultiDMATransfer***
- ***uint32_t MultiTwoSamplingDelay***

Field Documentation

- ***uint32_t LL_ADC_CommonInitTypeDef::CommonClock***

  Set parameter common to several ADC: Clock source and prescaler. This parameter can be a value of ***ADC_LL_EC_COMMON_CLOCK_SOURCE***

  Note:

  – On this STM32 series, if ADC group injected is used, some clock ratio constraints between ADC clock and AHB clock must be respected. Refer to reference manual.

  This feature can be modified afterwards using unitary function **LL_ADC_SetCommonClock()**.
- ***uint32_t LL_ADC_CommonInitTypeDef::Multimode***

  Set ADC multimode configuration to operate in independent mode or multimode (for devices with several ADC instances). This parameter can be a value of ***ADC_LL_EC_MULTI_MODE***This feature can be modified afterwards using unitary function **LL_ADC_SetMultimode()**.
- ***uint32_t LL_ADC_CommonInitTypeDef::MultiDMATransfer***

  Set ADC dual ADC mode DMA transfer data format: Each DMA, 32 down to 10-bits or 8-bits resolution. This parameter can be a value of ***ADC_LL_EC_MULTI_DMA_TRANSFER***This feature can be modified afterwards using unitary function **LL_ADC_SetMultiDMATransfer()**.
- ***uint32_t LL_ADC_CommonInitTypeDef::MultiTwoSamplingDelay***

  Set ADC multimode delay between 2 sampling phases. This parameter can be a value of ***ADC_LL_EC_MULTI_TWOSMP_DELAY***This feature can be modified afterwards using unitary function **LL_ADC_SetMultiTwoSamplingDelay()**.

95.1.2 LL_ADC_InitTypeDef

LL_ADC_InitTypeDef is defined in the stm32h7xx_ll_adc.h

Data Fields

- ***uint32_t Resolution***
- ***uint32_t LeftBitShift***
- ***uint32_t LowPowerMode***

Field Documentation

- ***uint32_t LL_ADC_InitTypeDef::Resolution***

  Set ADC resolution. This parameter can be a value of ***ADC_LL_EC_RESOLUTION***This feature can be modified afterwards using unitary function **LL_ADC_SetResolution()**.
- ***uint32_t LL_ADC_InitTypeDef::LeftBitShift***

  Configures the left shifting applied to the final result with or without oversampling. This parameter can be a value of ***ADC_LL_EC_LEFT_BIT_SHIFT***.
- ***uint32_t LL_ADC_InitTypeDef::LowPowerMode***

  Set ADC low power mode. This parameter can be a value of ***ADC_LL_EC_LP_MODE***This feature can be modified afterwards using unitary function **LL_ADC_SetLowPowerMode()**.

95.1.3 LL_ADC_REG_InitTypeDef

LL_ADC_REG_InitTypeDef is defined in the stm32h7xx_ll_adc.h

Data Fields

- ***uint32_t TriggerSource***
- ***uint32_t SequencerLength***
- ***uint32_t SequencerDiscont***
- ***uint32_t ContinuousMode***
- ***uint32_t DataTransferMode***
- ***uint32_t Overrun***

Field Documentation

- ***uint32_t LL_ADC_REG_InitTypeDef::TriggerSource***

  Set ADC group regular conversion trigger source: internal (SW start) or from external peripheral (timer event, external interrupt line). This parameter can be a value of ***ADC_LL_EC_REG_TRIGGER_SOURCE***

  Note:

  – On this STM32 series, setting trigger source to external trigger also set trigger polarity to rising edge (default setting for compatibility with some ADC on other STM32 families having this setting set by HW default value). In case of need to modify trigger edge, use function **LL_ADC_REG_SetTriggerEdge()**.

  This feature can be modified afterwards using unitary function **LL_ADC_REG_SetTriggerSource()**.

- ***uint32_t LL_ADC_REG_InitTypeDef::SequencerLength***

  Set ADC group regular sequencer length. This parameter can be a value of ***ADC_LL_EC_REG_SEQ_SCAN_LENGTH***This feature can be modified afterwards using unitary function **LL_ADC_REG_SetSequencerLength()**.

- ***uint32_t LL_ADC_REG_InitTypeDef::SequencerDiscont***

  Set ADC group regular sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks. This parameter can be a value of ***ADC_LL_EC_REG_SEQ_DISCONT_MODE***

  Note:

  – This parameter has an effect only if group regular sequencer is enabled (scan length of 2 ranks or more).

  This feature can be modified afterwards using unitary function **LL_ADC_REG_SetSequencerDiscont()**.

- ***uint32_t LL_ADC_REG_InitTypeDef::ContinuousMode***

  Set ADC continuous conversion mode on ADC group regular, whether ADC conversions are performed in single mode (one conversion per trigger) or in continuous mode (after the first trigger, following conversions launched successively automatically). This parameter can be a value of ***ADC_LL_EC_REG_CONTINUOUS_MODE*** Note: It is not possible to enable both ADC group regular continuous mode and discontinuous mode.This feature can be modified afterwards using unitary function **LL_ADC_REG_SetContinuousMode()**.

- ***uint32_t LL_ADC_REG_InitTypeDef::DataTransferMode***

  Set ADC group regular conversion data transfer mode: no transfer, transfer by DMA (Limited/Unlimited) or DFSDM. This parameter can be a value of ***ADC_LL_EC_REG_DATA_TRANSFER_MODE***This feature can be modified afterwards using unitary function **LL_ADC_REG_SetDataTransferMode()**.

- ***uint32_t LL_ADC_REG_InitTypeDef::Overrun***

  Set ADC group regular behavior in case of overrun: data preserved or overwritten. This parameter can be a value of ***ADC_LL_EC_REG_OVR_DATA_BEHAVIOR***This feature can be modified afterwards using unitary function **LL_ADC_REG_SetOverrun()**.

95.1.4 LL_ADC_INJ_InitTypeDef

LL_ADC_INJ_InitTypeDef is defined in the stm32h7xx_ll_adc.h

Data Fields

- ***uint32_t TriggerSource***
- ***uint32_t SequencerLength***
- ***uint32_t SequencerDiscont***

- ***uint32_t TrigAuto***

Field Documentation

- ***uint32_t LL_ADC_INJ_InitTypeDef::TriggerSource***

  Set ADC group injected conversion trigger source: internal (SW start) or from external peripheral (timer event, external interrupt line). This parameter can be a value of ***ADC_LL_EC_INJ_TRIGGER_SOURCE***

  Note:

  – On this STM32 series, setting trigger source to external trigger also set trigger polarity to rising edge (default setting for compatibility with some ADC on other STM32 families having this setting set by HW default value). In case of need to modify trigger edge, use function **LL_ADC_INJ_SetTriggerEdge()**.

  This feature can be modified afterwards using unitary function **LL_ADC_INJ_SetTriggerSource()**.

- ***uint32_t LL_ADC_INJ_InitTypeDef::SequencerLength***

  Set ADC group injected sequencer length. This parameter can be a value of ***ADC_LL_EC_INJ_SEQ_SCAN_LENGTH***This feature can be modified afterwards using unitary function **LL_ADC_INJ_SetSequencerLength()**.

- ***uint32_t LL_ADC_INJ_InitTypeDef::SequencerDiscont***

  Set ADC group injected sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks. This parameter can be a value of ***ADC_LL_EC_INJ_SEQ_DISCONT_MODE***

  Note:

  – This parameter has an effect only if group injected sequencer is enabled (scan length of 2 ranks or more).

  This feature can be modified afterwards using unitary function **LL_ADC_INJ_SetSequencerDiscont()**.

- ***uint32_t LL_ADC_INJ_InitTypeDef::TrigAuto***

  Set ADC group injected conversion trigger: independent or from ADC group regular. This parameter can be a value of ***ADC_LL_EC_INJ_TRIG_AUTO*** Note: This parameter must be set to set to independent trigger if injected trigger source is set to an external trigger.This feature can be modified afterwards using unitary function **LL_ADC_INJ_SetTrigAuto()**.

95.2 ADC Firmware driver API description

The following section lists the various functions of the ADC library.

95.2.1 Detailed description of functions

LL_ADC_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_ADC_DMA_GetRegAddr (ADC_TypeDef * ADCx, uint32_t Register)

Function description

Function to help to configure DMA transfer from ADC: retrieve the ADC register address from ADC instance and a list of ADC registers intended to be used (most commonly) with DMA transfer.

Parameters

- **ADCx:** ADC instance
- **Register:** This parameter can be one of the following values:
  – LL_ADC_DMA_REG_REGULAR_DATA
  – LL_ADC_DMA_REG_REGULAR_DATA_MULTI (1)

  (1) Available on devices with several ADC instances.

Return values

- **ADC:** register address

Notes

- These ADC registers are data registers: when ADC conversion data is available in ADC data registers, ADC generates a DMA transfer request.
- This macro is intended to be used with LL DMA driver, refer to function "LL_DMA_ConfigAddresses()". Example: LL_DMA_ConfigAddresses(DMA1, LL_DMA_CHANNEL_1, LL_ADC_DMA_GetRegAddr(ADC1, LL_ADC_DMA_REG_REGULAR_DATA), (uint32_t)&< array or variable >, LL_DMA_DIRECTION_PERIPH_TO_MEMORY);
- For devices with several ADC: in multimode, some devices use a different data register outside of ADC instance scope (common data register). This macro manages this register difference, only ADC instance has to be set as parameter.

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_DMA_GetRegAddr
- CDR RDATA_MST LL_ADC_DMA_GetRegAddr
- CDR RDATA_SLV LL_ADC_DMA_GetRegAddr

LL_ADC_SetCommonClock

Function name

__STATIC_INLINE void LL_ADC_SetCommonClock (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t CommonClock)

Function description

Set parameter common to several ADC: Clock source and prescaler.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **CommonClock:** This parameter can be one of the following values:
  - LL_ADC_CLOCK_SYNC_PCLK_DIV1
  - LL_ADC_CLOCK_SYNC_PCLK_DIV2
  - LL_ADC_CLOCK_SYNC_PCLK_DIV4
  - LL_ADC_CLOCK_ASYNC_DIV1
  - LL_ADC_CLOCK_ASYNC_DIV2
  - LL_ADC_CLOCK_ASYNC_DIV4
  - LL_ADC_CLOCK_ASYNC_DIV6
  - LL_ADC_CLOCK_ASYNC_DIV8
  - LL_ADC_CLOCK_ASYNC_DIV10
  - LL_ADC_CLOCK_ASYNC_DIV12
  - LL_ADC_CLOCK_ASYNC_DIV16
  - LL_ADC_CLOCK_ASYNC_DIV32
  - LL_ADC_CLOCK_ASYNC_DIV64
  - LL_ADC_CLOCK_ASYNC_DIV128
  - LL_ADC_CLOCK_ASYNC_DIV256

Return values

- **None:**

Notes

- On this STM32 series, if ADC group injected is used, some clock ratio constraints between ADC clock and AHB clock must be respected. Refer to reference manual.
- On this STM32 series, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled. This check can be done with function LL_ADC_IsEnabled() for each ADC instance or by using helper macro helper macro __LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE().

Reference Manual to LL API cross reference:

- CCR CKMODE LL_ADC_SetCommonClock
- CCR PRESC LL_ADC_SetCommonClock

LL_ADC_GetCommonClock

Function name

__STATIC_INLINE uint32_t LL_ADC_GetCommonClock (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get parameter common to several ADC: Clock source and prescaler.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_CLOCK_SYNC_PCLK_DIV1
  - LL_ADC_CLOCK_SYNC_PCLK_DIV2
  - LL_ADC_CLOCK_SYNC_PCLK_DIV4
  - LL_ADC_CLOCK_ASYNC_DIV1
  - LL_ADC_CLOCK_ASYNC_DIV2
  - LL_ADC_CLOCK_ASYNC_DIV4
  - LL_ADC_CLOCK_ASYNC_DIV6
  - LL_ADC_CLOCK_ASYNC_DIV8
  - LL_ADC_CLOCK_ASYNC_DIV10
  - LL_ADC_CLOCK_ASYNC_DIV12
  - LL_ADC_CLOCK_ASYNC_DIV16
  - LL_ADC_CLOCK_ASYNC_DIV32
  - LL_ADC_CLOCK_ASYNC_DIV64
  - LL_ADC_CLOCK_ASYNC_DIV128
  - LL_ADC_CLOCK_ASYNC_DIV256

Reference Manual to LL API cross reference:

- CCR CKMODE LL_ADC_GetCommonClock
- CCR PRESC LL_ADC_GetCommonClock

LL_ADC_SetCommonPathInternalCh

Function name

__STATIC_INLINE void LL_ADC_SetCommonPathInternalCh (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t PathInternal)

Function description

Set parameter common to several ADC: measurement path to internal channels (VrefInt, temperature sensor, ...).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **PathInternal:** This parameter can be a combination of the following values:
  - LL_ADC_PATH_INTERNAL_NONE
  - LL_ADC_PATH_INTERNAL_VREFINT
  - LL_ADC_PATH_INTERNAL_TEMPSENSOR
  - LL_ADC_PATH_INTERNAL_VBAT

Return values

- **None:**

Notes

- One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VREFINT | LL_ADC_PATH_INTERNAL_TEMPSENSOR)
- Stabilization time of measurement path to internal channel: After enabling internal paths, before starting ADC conversion, a delay is required for internal voltage reference and temperature sensor stabilization time. Refer to device datasheet. Refer to literal LL_ADC_DELAY_VREFINT_STAB_US. Refer to literal LL_ADC_DELAY_TEMPSENSOR_STAB_US.
- ADC internal channel sampling time constraint: For ADC conversion of internal channels, a sampling time minimum value is required. Refer to device datasheet.
- On this STM32 series, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled. This check can be done with function LL_ADC_IsEnabled() for each ADC instance or by using helper macro helper macro __LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE().

Reference Manual to LL API cross reference:

- CCR VREFEN LL_ADC_SetCommonPathInternalCh
- CCR TSEN LL_ADC_SetCommonPathInternalCh
- CCR VBATEN LL_ADC_SetCommonPathInternalCh

LL_ADC_GetCommonPathInternalCh

Function name

__STATIC_INLINE uint32_t LL_ADC_GetCommonPathInternalCh (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get parameter common to several ADC: measurement path to internal channels (VrefInt, temperature sensor, ...).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **Returned:** value can be a combination of the following values:
  - LL_ADC_PATH_INTERNAL_NONE
  - LL_ADC_PATH_INTERNAL_VREFINT
  - LL_ADC_PATH_INTERNAL_TEMPSENSOR
  - LL_ADC_PATH_INTERNAL_VBAT

Notes

- One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VREFINT | LL_ADC_PATH_INTERNAL_TEMPSENSOR)

Reference Manual to LL API cross reference:

- CCR VREFEN LL_ADC_GetCommonPathInternalCh
- CCR TSEN LL_ADC_GetCommonPathInternalCh
- CCR VBATEN LL_ADC_GetCommonPathInternalCh

LL_ADC_SetCommonPathInternalChAdd

Function name

__STATIC_INLINE void LL_ADC_SetCommonPathInternalChAdd (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t PathInternal)

Function description

Set parameter common to several ADC: measurement path to internal channels (VrefInt, temperature sensor, ...).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **PathInternal:** This parameter can be a combination of the following values:
  - LL_ADC_PATH_INTERNAL_NONE
  - LL_ADC_PATH_INTERNAL_VREFINT
  - LL_ADC_PATH_INTERNAL_TEMPSENSOR
  - LL_ADC_PATH_INTERNAL_VBAT

Return values

- **None:**

Notes

- One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VREFINT | LL_ADC_PATH_INTERNAL_TEMPSENSOR)
- Stabilization time of measurement path to internal channel: After enabling internal paths, before starting ADC conversion, a delay is required for internal voltage reference and temperature sensor stabilization time. Refer to device datasheet. Refer to literal LL_ADC_DELAY_VREFINT_STAB_US. Refer to literal LL_ADC_DELAY_TEMPSENSOR_STAB_US.
- ADC internal channel sampling time constraint: For ADC conversion of internal channels, a sampling time minimum value is required. Refer to device datasheet.

Reference Manual to LL API cross reference:

- CCR VREFEN LL_ADC_SetCommonPathInternalChAdd
- CCR TSEN LL_ADC_SetCommonPathInternalChAdd
- CCR VBATEN LL_ADC_SetCommonPathInternalChAdd

LL_ADC_SetCommonPathInternalChRem

Function name

__STATIC_INLINE void LL_ADC_SetCommonPathInternalChRem (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t PathInternal)

Function description

Set parameter common to several ADC: measurement path to internal channels (VrefInt, temperature sensor, ...).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **PathInternal:** This parameter can be a combination of the following values:
  - LL_ADC_PATH_INTERNAL_NONE
  - LL_ADC_PATH_INTERNAL_VREFINT
  - LL_ADC_PATH_INTERNAL_TEMPSENSOR
  - LL_ADC_PATH_INTERNAL_VBAT

Return values

- **None:**

Notes

- One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VREFINT | LL_ADC_PATH_INTERNAL_TEMPSENSOR)

Reference Manual to LL API cross reference:

- CCR VREFEN LL_ADC_SetCommonPathInternalChRem
- CCR TSEN LL_ADC_SetCommonPathInternalChRem
- CCR VBATEN LL_ADC_SetCommonPathInternalChRem

LL_ADC_SetCalibrationOffsetFactor

Function name

__STATIC_INLINE void LL_ADC_SetCalibrationOffsetFactor (ADC_TypeDef * ADCx, uint32_t SingleDiff, uint32_t CalibrationFactor)

Function description

Set ADC calibration factor in the mode single-ended or differential (for devices with differential mode available).

Parameters

- **ADCx:** ADC instance
- **SingleDiff:** This parameter can be one of the following values:
  - LL_ADC_SINGLE_ENDED
  - LL_ADC_DIFFERENTIAL_ENDED
  - LL_ADC_BOTH_SINGLE_DIFF_ENDED
- **CalibrationFactor:** Value between Min_Data=0x00 and Max_Data=0x7F

Return values

- **None:**

Notes

- This function is intended to set calibration parameters without having to perform a new calibration using LL_ADC_StartCalibration().
- For devices with differential mode available: Calibration of offset is specific to each of single-ended and differential modes (calibration factor must be specified for each of these differential modes, if used afterwards and if the application requires their calibration). Calibration of linearity is common to both single-ended and differential modes (calibration factor can be specified only once).
- In case of setting calibration factors of both modes single ended and differential (parameter LL_ADC_BOTH_SINGLE_DIFF_ENDED): both calibration factors must be concatenated. To perform this processing, use helper macro __LL_ADC_CALIB_FACTOR_SINGLE_DIFF().
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be enabled, without calibration on going, without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CALFACT CALFACT_S LL_ADC_SetCalibrationOffsetFactor
- CALFACT CALFACT_D LL_ADC_SetCalibrationOffsetFactor

LL_ADC_GetCalibrationOffsetFactor

Function name

__STATIC_INLINE uint32_t LL_ADC_GetCalibrationOffsetFactor (ADC_TypeDef * ADCx, uint32_t SingleDiff)

Function description

Get ADC calibration factor in the mode single-ended or differential (for devices with differential mode available).

Parameters

- **ADCx:** ADC instance
- **SingleDiff:** This parameter can be one of the following values:
  - LL_ADC_SINGLE_ENDED
  - LL_ADC_DIFFERENTIAL_ENDED

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x7F

Notes

- Calibration factors are set by hardware after performing a calibration run using function LL_ADC_StartCalibration().
- For devices with differential mode available: Calibration of offset is specific to each of single-ended and differential modes Calibration of linearity is common to both single-ended and differential modes

Reference Manual to LL API cross reference:

- CALFACT CALFACT_S LL_ADC_GetCalibrationOffsetFactor
- CALFACT CALFACT_D LL_ADC_GetCalibrationOffsetFactor

LL_ADC_SetCalibrationLinearFactor

Function name

__STATIC_INLINE void LL_ADC_SetCalibrationLinearFactor (ADC_TypeDef * ADCx, uint32_t LinearityWord, uint32_t CalibrationFactor)

Function description

Set ADC Linear calibration factor in the mode single-ended.

Parameters

- **ADCx:** ADC instance
- **LinearityWord:** This parameter can be one of the following values:
  - LL_ADC_CALIB_LINEARITY_WORD1
  - LL_ADC_CALIB_LINEARITY_WORD2
  - LL_ADC_CALIB_LINEARITY_WORD3
  - LL_ADC_CALIB_LINEARITY_WORD4
  - LL_ADC_CALIB_LINEARITY_WORD5
  - LL_ADC_CALIB_LINEARITY_WORD6
- **CalibrationFactor:** Value between Min_Data=0x00 and Max_Data=0x3FFFFFFF

Return values

- **None:**

Notes

- This function is intended to set linear calibration parameters without having to perform a new calibration using LL_ADC_StartCalibration().
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be enabled, without calibration on going, without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CALFACT2 LINCALFACT LL_ADC_SetCalibrationLinearFactor
- CALFACT2 LINCALFACT LL_ADC_SetCalibrationLinearFactor

LL_ADC_GetCalibrationLinearFactor

Function name

__STATIC_INLINE uint32_t LL_ADC_GetCalibrationLinearFactor (ADC_TypeDef * ADCx, uint32_t LinearityWord)

Function description

Get ADC Linear calibration factor in the mode single-ended.

Parameters

- **ADCx:** ADC instance
- **LinearityWord:** This parameter can be one of the following values:
  - LL_ADC_CALIB_LINEARITY_WORD1
  - LL_ADC_CALIB_LINEARITY_WORD2
  - LL_ADC_CALIB_LINEARITY_WORD3
  - LL_ADC_CALIB_LINEARITY_WORD4
  - LL_ADC_CALIB_LINEARITY_WORD5
  - LL_ADC_CALIB_LINEARITY_WORD6

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x3FFFFFFF

Notes

- Calibration factors are set by hardware after performing a calibration run using function LL_ADC_StartCalibration().

Reference Manual to LL API cross reference:

- CALFACT2 LINCALFACT LL_ADC_GetCalibrationLinearFactor
- CALFACT2 LINCALFACT LL_ADC_GetCalibrationLinearFactor

LL_ADC_SetResolution

Function name

__STATIC_INLINE void LL_ADC_SetResolution (ADC_TypeDef * ADCx, uint32_t Resolution)

Function description

Set ADC resolution.

Parameters

- **ADCx:** ADC instance
- **Resolution:** This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_16B
  - LL_ADC_RESOLUTION_14B
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR RES LL_ADC_SetResolution

LL_ADC_GetResolution

Function name

__STATIC_INLINE uint32_t LL_ADC_GetResolution (ADC_TypeDef * ADCx)

Function description

Get ADC resolution.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_RESOLUTION_16B (1)
  - LL_ADC_RESOLUTION_14B
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B (2) (1): Specific to ADC instance: ADC1, ADC2 (2): Specific to ADC instance: ADC3

Reference Manual to LL API cross reference:

- CFGR RES LL_ADC_GetResolution

LL_ADC_SetLowPowerMode

Function name

__STATIC_INLINE void LL_ADC_SetLowPowerMode (ADC_TypeDef * ADCx, uint32_t LowPowerMode)

Function description

Set ADC low power mode.

Parameters

- **ADCx:** ADC instance
- **LowPowerMode:** This parameter can be one of the following values:
  - LL_ADC_LP_MODE_NONE
  - LL_ADC_LP_AUTOWAIT

Return values

- **None:**

Notes

- Description of ADC low power modes: ADC low power mode "auto wait": Dynamic low power mode, ADC conversions occurrences are limited to the minimum necessary in order to reduce power consumption. New ADC conversion starts only when the previous unitary conversion data (for ADC group regular) or previous sequence conversions data (for ADC group injected) has been retrieved by user software. In the meantime, ADC remains idle: does not performs any other conversion. This mode allows to automatically adapt the ADC conversions triggers to the speed of the software that reads the data. Moreover, this avoids risk of overrun for low frequency applications. How to use this low power mode: It is not recommended to use with interruption or DMA since these modes have to clear immediately the EOC flag (by CPU to free the IRQ pending event or by DMA). Auto wait will work but fort a very short time, discarding its intended benefit (except specific case of high load of CPU or DMA transfers which can justify usage of auto wait).Do use with polling: 1. Start conversion, 2. Later on, when conversion data is needed: poll for end of conversion to ensure that conversion is completed and retrieve ADC conversion data. This will trig another ADC conversion start. ADC low power mode "auto power-off" (feature available on this device if parameter LL_ADC_LP_AUTOPOWEROFF is available): the ADC automatically powers-off after a conversion and automatically wakes up when a new conversion is triggered (with startup time between trigger and start of sampling). This feature can be combined with low power mode "auto wait".
- With ADC low power mode "auto wait", the ADC conversion data read is corresponding to previous ADC conversion start, independently of delay during which ADC was idle. Therefore, the ADC conversion data may be outdated: does not correspond to the current voltage level on the selected ADC channel.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR AUTDLY LL_ADC_SetLowPowerMode

LL_ADC_GetLowPowerMode

Function name

__STATIC_INLINE uint32_t LL_ADC_GetLowPowerMode (ADC_TypeDef * ADCx)

Function description

Get ADC low power mode:

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_LP_MODE_NONE
  - LL_ADC_LP_AUTOWAIT

Notes

- Description of ADC low power modes: ADC low power mode "auto wait": Dynamic low power mode, ADC conversions occurrences are limited to the minimum necessary in order to reduce power consumption. New ADC conversion starts only when the previous unitary conversion data (for ADC group regular) or previous sequence conversions data (for ADC group injected) has been retrieved by user software. In the meantime, ADC remains idle: does not performs any other conversion. This mode allows to automatically adapt the ADC conversions triggers to the speed of the software that reads the data. Moreover, this avoids risk of overrun for low frequency applications. How to use this low power mode: It is not recommended to use with interruption or DMA since these modes have to clear immediately the EOC flag (by CPU to free the IRQ pending event or by DMA). Auto wait will work but fort a very short time, discarding its intended benefit (except specific case of high load of CPU or DMA transfers which can justify usage of auto wait).Do use with polling: 1. Start conversion, 2. Later on, when conversion data is needed: poll for end of conversion to ensure that conversion is completed and retrieve ADC conversion data. This will trig another ADC conversion start. ADC low power mode "auto power-off" (feature available on this device if parameter LL_ADC_LP_AUTOPOWEROFF is available): the ADC automatically powers-off after a conversion and automatically wakes up when a new conversion is triggered (with startup time between trigger and start of sampling). This feature can be combined with low power mode "auto wait".
- With ADC low power mode "auto wait", the ADC conversion data read is corresponding to previous ADC conversion start, independently of delay during which ADC was idle. Therefore, the ADC conversion data may be outdated: does not correspond to the current voltage level on the selected ADC channel.

Reference Manual to LL API cross reference:

- CFGR AUTDLY LL_ADC_GetLowPowerMode

LL_ADC_SetChannelPreselection

Function name

__STATIC_INLINE void LL_ADC_SetChannelPreselection (ADC_TypeDef * ADCx, uint32_t Channel)

Function description

Set ADC selected Channel.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19

Return values

- **None:**

Notes

- This function set the pre-selection of channel configuration.
- Caution: Channel selections is dependent to ADC instance and IP version: For STM32H72x/3x This is applicable only for ADC1/ADC2 For Rest of STM32H7xxx This is applicable only all the ADCs instances.

LL_ADC_GetChannelPreselection

Function name

__STATIC_INLINE uint32_t LL_ADC_GetChannelPreselection (ADC_TypeDef * ADCx, uint32_t Channel)

Function description

Gets ADC pre-selected Channel.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19

Return values

- **the:** preselection state of Channel (!= 0 : pre-selected, == 0 : not pre-selected)

Notes

- This function gets the pre-selected ADC channel.
- Caution: Channel selections is dependent to ADC instance and IP version: For STM32H72x/3x This is applicable only for ADC1/ADC2 For Rest of STM32H7xxx This is applicable on all the ADCs instances.

LL_ADC_SetOffset

Function name

__STATIC_INLINE void LL_ADC_SetOffset (ADC_TypeDef * ADCx, uint32_t Offsety, uint32_t Channel, uint32_t OffsetLevel)

Function description

Set ADC selected offset number 1, 2, 3 or 4.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)

  (1) On STM32H7, parameter available only on ADC instance: ADC3.
- (2) On STM32H7, parameter available only on ADC instance: ADC2.
- (3) On STM32H7, fast channel (0.125 us for 14-bit resolution (ADC conversion rate up to 8 Ms/s)). Other channels are slow channels (conversion rate: refer to reference manual).
- **OffsetLevel:** Value between Min_Data=0x000 and Max_Data=0x3FFFFFF

Return values

- **None:**

Notes

- This function set the 2 items of offset configuration: ADC channel to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)Offset level (offset to be subtracted from the raw converted data).
- Caution: Offset format is dependent to ADC resolution: offset has to be left-aligned on bit 15 (handling maximum ADC resolution 16 bit), the LSB (right bits) are set to 0.
- This function enables the offset, by default. It can be forced to disable state using function LL_ADC_SetOffsetState().
- If a channel is mapped on several offsets numbers, only the offset with the lowest value is considered for the subtraction.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.
- On STM32H7, some fast channels are available: fast analog inputs coming from GPIO pads (ADC_IN0..5).

Reference Manual to LL API cross reference:

- OFR1 OFFSET1_CH LL_ADC_SetOffset
- OFR1 OFFSET1 LL_ADC_SetOffset
- OFR1 OFFSET1_EN LL_ADC_SetOffset
- OFR2 OFFSET2_CH LL_ADC_SetOffset
- OFR2 OFFSET2 LL_ADC_SetOffset
- OFR2 OFFSET2_EN LL_ADC_SetOffset
- OFR3 OFFSET3_CH LL_ADC_SetOffset
- OFR3 OFFSET3 LL_ADC_SetOffset
- OFR3 OFFSET3_EN LL_ADC_SetOffset
- OFR4 OFFSET4_CH LL_ADC_SetOffset
- OFR4 OFFSET4 LL_ADC_SetOffset
- OFR4 OFFSET4_EN LL_ADC_SetOffset

LL_ADC_GetOffsetChannel

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOffsetChannel (ADC_TypeDef * ADCx, uint32_t Offsety)

Function description

Get for the ADC selected offset number 1, 2, 3 or 4: Channel to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)

  (1) On STM32H7, parameter available only on ADC instance: ADC3.
- (2) On STM32H7, parameter available only on ADC instance: ADC2.
- (3) On STM32H7, fast channel (0.125 us for 14-bit resolution (ADC conversion rate up to 8 Ms/s)). Other channels are slow channels (conversion rate: refer to reference manual).
- (1, 2) For ADC channel read back from ADC register, comparison with internal channel parameter to be done using helper macro __LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL().

Notes

- Usage of the returned channel number: To reinject this channel into another function LL_ADC_xxx: the returned channel number is only partly formatted on definition of literals LL_ADC_CHANNEL_x. Therefore, it has to be compared with parts of literals LL_ADC_CHANNEL_x or using helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Then the selected literal LL_ADC_CHANNEL_x can be used as parameter for another function.To get the channel number in decimal format: process the returned value with the helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB().
- On STM32H7, some fast channels are available: fast analog inputs coming from GPIO pads (ADC_IN0..5).

Reference Manual to LL API cross reference:

- OFR1 OFFSET1_CH LL_ADC_GetOffsetChannel
- OFR2 OFFSET2_CH LL_ADC_GetOffsetChannel
- OFR3 OFFSET3_CH LL_ADC_GetOffsetChannel
- OFR4 OFFSET4_CH LL_ADC_GetOffsetChannel

LL_ADC_GetOffsetLevel

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOffsetLevel (ADC_TypeDef * ADCx, uint32_t Offsety)

Function description

Get for the ADC selected offset number 1, 2, 3 or 4: Offset level (offset to be subtracted from the raw converted data).

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0x3FFFFFF

Notes

- Caution: Offset format is dependent to ADC resolution: offset has to be left-aligned on bit 15 (handling maximum ADC resolution 16 bit), the LSB (right bits) are set to 0.

Reference Manual to LL API cross reference:

- OFR1 OFFSET1 LL_ADC_GetOffsetLevel
- OFR2 OFFSET2 LL_ADC_GetOffsetLevel
- OFR3 OFFSET3 LL_ADC_GetOffsetLevel
- OFR4 OFFSET4 LL_ADC_GetOffsetLevel

LL_ADC_SetDataRightShift

Function name

__STATIC_INLINE void LL_ADC_SetDataRightShift (ADC_TypeDef * ADCx, uint32_t Offsety, uint32_t RigthShift)

Function description

Set data right shift for the ADC selected offset number 1, 2, 3 or 4: signed offset saturation if enabled or disabled.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4
- **RigthShift:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_RSHIFT_ENABLE
  - LL_ADC_OFFSET_RSHIFT_DISABLE

Return values

- **Returned:** None

Reference Manual to LL API cross reference:

- CFGR2 RSHIFT LL_ADC_SetDataRightShift
-

LL_ADC_GetDataRightShift

Function name

__STATIC_INLINE uint32_t LL_ADC_GetDataRightShift (ADC_TypeDef * ADCx, uint32_t Offsety)

Function description

Get data right shift for the ADC selected offset number 1, 2, 3 or 4: signed offset saturation if enabled or disabled.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_OFFSET_RSHIFT_ENABLE
  - LL_ADC_OFFSET_RSHIFT_DISABLE

Reference Manual to LL API cross reference:

- CFGR2 RSHIFT LL_ADC_GetDataRightShift
-

LL_ADC_SetOffsetSignedSaturation

Function name

__STATIC_INLINE void LL_ADC_SetOffsetSignedSaturation (ADC_TypeDef * ADCx, uint32_t Offsety, uint32_t OffsetSignedSaturation)

Function description

Set signed saturation for the ADC selected offset number 1, 2, 3 or 4: signed offset saturation if enabled or disabled.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4
- **OffsetSignedSaturation:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_SIGNED_SATURATION_ENABLE
  - LL_ADC_OFFSET_SIGNED_SATURATION_DISABLE

Return values

- **Returned:** None

Reference Manual to LL API cross reference:

- OFR1 SSATE LL_ADC_SetOffsetSignedSaturation
- OFR2 SSATE LL_ADC_SetOffsetSignedSaturation
- OFR3 SSATE LL_ADC_SetOffsetSignedSaturation
- OFR4 SSATE LL_ADC_SetOffsetSignedSaturation

LL_ADC_GetOffsetSignedSaturation

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOffsetSignedSaturation (ADC_TypeDef * ADCx, uint32_t Offsety)

Function description

Get signed saturation for the ADC selected offset number 1, 2, 3 or 4: signed offset saturation if enabled or disabled.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_OFFSET_SIGNED_SATURATION_ENABLE
  - LL_ADC_OFFSET_SIGNED_SATURATION_DISABLE

Reference Manual to LL API cross reference:

- OFR1 SSATE LL_ADC_GetOffsetSignedSaturation
- OFR2 SSATE LL_ADC_GetOffsetSignedSaturation
- OFR3 SSATE LL_ADC_GetOffsetSignedSaturation
- OFR4 SSATE LL_ADC_GetOffsetSignedSaturation

LL_ADC_REG_SetTriggerSource

Function name

__STATIC_INLINE void LL_ADC_REG_SetTriggerSource (ADC_TypeDef * ADCx, uint32_t TriggerSource)

Function description

Set ADC group regular conversion trigger source: internal (SW start) or from external peripheral (timer event, external interrupt line).

Parameters

- **ADCx:** ADC instance
- **TriggerSource:** This parameter can be one of the following values:
  - LL_ADC_REG_TRIG_SOFTWARE
  - LL_ADC_REG_TRIG_EXT_TIM1_CH1
  - LL_ADC_REG_TRIG_EXT_TIM1_CH2
  - LL_ADC_REG_TRIG_EXT_TIM1_CH3
  - LL_ADC_REG_TRIG_EXT_TIM2_CH2
  - LL_ADC_REG_TRIG_EXT_TIM3_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM4_CH4
  - LL_ADC_REG_TRIG_EXT_EXTI_LINE11
  - LL_ADC_REG_TRIG_EXT_TIM8_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM8_TRGO2
  - LL_ADC_REG_TRIG_EXT_TIM1_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM1_TRGO2
  - LL_ADC_REG_TRIG_EXT_TIM2_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM4_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM6_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM15_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM3_CH4
  - LL_ADC_REG_TRIG_EXT_HRTIM_TRG1
  - LL_ADC_REG_TRIG_EXT_HRTIM_TRG3
  - LL_ADC_REG_TRIG_EXT_LPTIM1_OUT
  - LL_ADC_REG_TRIG_EXT_LPTIM2_OUT
  - LL_ADC_REG_TRIG_EXT_LPTIM3_OUT

Return values

- **None:**

Notes

- On this STM32 series, setting trigger source to external trigger also set trigger polarity to rising edge (default setting for compatibility with some ADC on other STM32 families having this setting set by HW default value). In case of need to modify trigger edge, use function LL_ADC_REG_SetTriggerEdge().
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR EXTSEL LL_ADC_REG_SetTriggerSource
- CFGR EXTEN LL_ADC_REG_SetTriggerSource

LL_ADC_REG_GetTriggerSource

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetTriggerSource (ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion trigger source: internal (SW start) or from external peripheral (timer event, external interrupt line).

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_TRIG_SOFTWARE
  - LL_ADC_REG_TRIG_EXT_TIM1_CH1
  - LL_ADC_REG_TRIG_EXT_TIM1_CH2
  - LL_ADC_REG_TRIG_EXT_TIM1_CH3
  - LL_ADC_REG_TRIG_EXT_TIM2_CH2
  - LL_ADC_REG_TRIG_EXT_TIM3_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM4_CH4
  - LL_ADC_REG_TRIG_EXT_EXTI_LINE11
  - LL_ADC_REG_TRIG_EXT_TIM8_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM8_TRGO2
  - LL_ADC_REG_TRIG_EXT_TIM1_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM1_TRGO2
  - LL_ADC_REG_TRIG_EXT_TIM2_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM4_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM6_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM15_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM3_CH4
  - LL_ADC_REG_TRIG_EXT_HRTIM_TRG1
  - LL_ADC_REG_TRIG_EXT_HRTIM_TRG3
  - LL_ADC_REG_TRIG_EXT_LPTIM1_OUT
  - LL_ADC_REG_TRIG_EXT_LPTIM2_OUT
  - LL_ADC_REG_TRIG_EXT_LPTIM3_OUT

Notes

- To determine whether group regular trigger source is internal (SW start) or external, without detail of which peripheral is selected as external trigger, (equivalent to "if(LL_ADC_REG_GetTriggerSource(ADC1) == LL_ADC_REG_TRIG_SOFTWARE)") use function LL_ADC_REG_IsTriggerSourceSWStart.
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.

Reference Manual to LL API cross reference:

- CFGR EXTSEL LL_ADC_REG_GetTriggerSource
- CFGR EXTEN LL_ADC_REG_GetTriggerSource

LL_ADC_REG_IsTriggerSourceSWStart

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_IsTriggerSourceSWStart (ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion trigger source internal (SW start) or external.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** "0" if trigger source external trigger Value "1" if trigger source SW start.

Notes

- In case of group regular trigger source set to external trigger, to determine which peripheral is selected as external trigger, use function LL_ADC_REG_GetTriggerSource().

Reference Manual to LL API cross reference:

- CFGR EXTEN LL_ADC_REG_IsTriggerSourceSWStart

LL_ADC_REG_SetTriggerEdge

Function name

__STATIC_INLINE void LL_ADC_REG_SetTriggerEdge (ADC_TypeDef * ADCx, uint32_t ExternalTriggerEdge)

Function description

Set ADC group regular conversion trigger polarity.

Parameters

- **ADCx:** ADC instance
- **ExternalTriggerEdge:** This parameter can be one of the following values:
  - LL_ADC_REG_TRIG_EXT_RISING
  - LL_ADC_REG_TRIG_EXT_FALLING
  - LL_ADC_REG_TRIG_EXT_RISINGFALLING

Return values

- **None:**

Notes

- Applicable only for trigger source set to external trigger.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR EXTEN LL_ADC_REG_SetTriggerEdge

LL_ADC_REG_GetTriggerEdge

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetTriggerEdge (ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion trigger polarity.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_TRIG_EXT_RISING
  - LL_ADC_REG_TRIG_EXT_FALLING
  - LL_ADC_REG_TRIG_EXT_RISINGFALLING

Notes

- Applicable only for trigger source set to external trigger.

Reference Manual to LL API cross reference:

- CFGR EXTEN LL_ADC_REG_GetTriggerEdge

LL_ADC_REG_SetSequencerLength

Function name

__STATIC_INLINE void LL_ADC_REG_SetSequencerLength (ADC_TypeDef * ADCx, uint32_t SequencerNbRanks)

Function description

Set ADC group regular sequencer length and scan direction.

Parameters

- **ADCx:** ADC instance
- **SequencerNbRanks:** This parameter can be one of the following values:
  - LL_ADC_REG_SEQ_SCAN_DISABLE
  - LL_ADC_REG_SEQ_SCAN_ENABLE_2RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_3RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_4RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_5RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_6RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_7RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_8RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_9RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_10RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_11RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_12RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_13RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_14RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_15RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_16RANKS

Return values

- **None:**

Notes

- Description of ADC group regular sequencer features: For devices with sequencer fully configurable (function "LL_ADC_REG_SetSequencerRanks()" available): sequencer length and each rank affectation to a channel are configurable. This function performs configuration of: Sequence length: Number of ranks in the scan sequence.Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from rank 1 to rank n). Sequencer ranks are selected using function "LL_ADC_REG_SetSequencerRanks()". For devices with sequencer not fully configurable (function "LL_ADC_REG_SetSequencerChannels()" available): sequencer length and each rank affectation to a channel are defined by channel number. This function performs configuration of: Sequence length: Number of ranks in the scan sequence is defined by number of channels set in the sequence, rank of each channel is fixed by channel HW number. (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...).Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from lowest channel number to highest channel number). Sequencer ranks are selected using function "LL_ADC_REG_SetSequencerChannels()".
- Sequencer disabled is equivalent to sequencer of 1 rank: ADC conversion on only 1 channel.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- SQR1 L LL_ADC_REG_SetSequencerLength

LL_ADC_REG_GetSequencerLength

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerLength (ADC_TypeDef * ADCx)

Function description

Get ADC group regular sequencer length and scan direction.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_SEQ_SCAN_DISABLE
  - LL_ADC_REG_SEQ_SCAN_ENABLE_2RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_3RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_4RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_5RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_6RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_7RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_8RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_9RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_10RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_11RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_12RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_13RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_14RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_15RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_16RANKS

Notes

- Description of ADC group regular sequencer features: For devices with sequencer fully configurable (function "LL_ADC_REG_SetSequencerRanks()" available): sequencer length and each rank affectation to a channel are configurable. This function retrieves: Sequence length: Number of ranks in the scan sequence.Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from rank 1 to rank n). Sequencer ranks are selected using function "LL_ADC_REG_SetSequencerRanks()". For devices with sequencer not fully configurable (function "LL_ADC_REG_SetSequencerChannels()" available): sequencer length and each rank affectation to a channel are defined by channel number. This function retrieves: Sequence length: Number of ranks in the scan sequence is defined by number of channels set in the sequence, rank of each channel is fixed by channel HW number. (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...).Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from lowest channel number to highest channel number). Sequencer ranks are selected using function "LL_ADC_REG_SetSequencerChannels()".
- Sequencer disabled is equivalent to sequencer of 1 rank: ADC conversion on only 1 channel.

Reference Manual to LL API cross reference:

- SQR1 L LL_ADC_REG_GetSequencerLength

LL_ADC_REG_SetSequencerDiscont

Function name

__STATIC_INLINE void LL_ADC_REG_SetSequencerDiscont (ADC_TypeDef * ADCx, uint32_t SeqDiscont)

Function description

Set ADC group regular sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.

Parameters

- **ADCx:** ADC instance
- **SeqDiscont:** This parameter can be one of the following values:
  - LL_ADC_REG_SEQ_DISCONT_DISABLE
  - LL_ADC_REG_SEQ_DISCONT_1RANK
  - LL_ADC_REG_SEQ_DISCONT_2RANKS
  - LL_ADC_REG_SEQ_DISCONT_3RANKS
  - LL_ADC_REG_SEQ_DISCONT_4RANKS
  - LL_ADC_REG_SEQ_DISCONT_5RANKS
  - LL_ADC_REG_SEQ_DISCONT_6RANKS
  - LL_ADC_REG_SEQ_DISCONT_7RANKS
  - LL_ADC_REG_SEQ_DISCONT_8RANKS

Return values

- **None:**

Notes

- It is not possible to enable both ADC group regular continuous mode and sequencer discontinuous mode.
- It is not possible to enable both ADC auto-injected mode and ADC group regular sequencer discontinuous mode.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR DISCEN LL_ADC_REG_SetSequencerDiscont
- CFGR DISCNUM LL_ADC_REG_SetSequencerDiscont

LL_ADC_REG_GetSequencerDiscont

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerDiscont (ADC_TypeDef * ADCx)

Function description

Get ADC group regular sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_SEQ_DISCONT_DISABLE
  - LL_ADC_REG_SEQ_DISCONT_1RANK
  - LL_ADC_REG_SEQ_DISCONT_2RANKS
  - LL_ADC_REG_SEQ_DISCONT_3RANKS
  - LL_ADC_REG_SEQ_DISCONT_4RANKS
  - LL_ADC_REG_SEQ_DISCONT_5RANKS
  - LL_ADC_REG_SEQ_DISCONT_6RANKS
  - LL_ADC_REG_SEQ_DISCONT_7RANKS
  - LL_ADC_REG_SEQ_DISCONT_8RANKS

Reference Manual to LL API cross reference:

- CFGR DISCEN LL_ADC_REG_GetSequencerDiscont
- CFGR DISCNUM LL_ADC_REG_GetSequencerDiscont

LL_ADC_REG_SetSequencerRanks

Function name

__STATIC_INLINE void LL_ADC_REG_SetSequencerRanks (ADC_TypeDef * ADCx, uint32_t Rank, uint32_t Channel)

Function description

Set ADC group regular sequence: channel on the selected scan sequence rank.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_REG_RANK_1
  - LL_ADC_REG_RANK_2
  - LL_ADC_REG_RANK_3
  - LL_ADC_REG_RANK_4
  - LL_ADC_REG_RANK_5
  - LL_ADC_REG_RANK_6
  - LL_ADC_REG_RANK_7
  - LL_ADC_REG_RANK_8
  - LL_ADC_REG_RANK_9
  - LL_ADC_REG_RANK_10
  - LL_ADC_REG_RANK_11
  - LL_ADC_REG_RANK_12
  - LL_ADC_REG_RANK_13
  - LL_ADC_REG_RANK_14
  - LL_ADC_REG_RANK_15
  - LL_ADC_REG_RANK_16
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)

  (1) On STM32H7, parameter available only on ADC instance: ADC3.
- (2) On STM32H7, parameter available only on ADC instance: ADC2.
- (3) On STM32H7, fast channel (0.125 us for 14-bit resolution (ADC conversion rate up to 8 Ms/s)). Other channels are slow channels (conversion rate: refer to reference manual).

Return values

- **None:**

Notes

- This function performs configuration of: Channels ordering into each rank of scan sequence: whatever channel can be placed into whatever rank.
- On this STM32 series, ADC group regular sequencer is fully configurable: sequencer length and each rank affectation to a channel are configurable. Refer to description of function LL_ADC_REG_SetSequencerLength().
- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- On this STM32 series, to measure internal channels (VrefInt, TempSensor, ...), measurement paths to internal channels must be enabled separately. This can be done using function LL_ADC_SetCommonPathInternalCh().
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- SQR1 SQ1 LL_ADC_REG_SetSequencerRanks
- SQR1 SQ2 LL_ADC_REG_SetSequencerRanks
- SQR1 SQ3 LL_ADC_REG_SetSequencerRanks
- SQR1 SQ4 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ5 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ6 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ7 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ8 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ9 LL_ADC_REG_SetSequencerRanks
- SQR3 SQ10 LL_ADC_REG_SetSequencerRanks
- SQR3 SQ11 LL_ADC_REG_SetSequencerRanks
- SQR3 SQ12 LL_ADC_REG_SetSequencerRanks
- SQR3 SQ13 LL_ADC_REG_SetSequencerRanks
- SQR3 SQ14 LL_ADC_REG_SetSequencerRanks
- SQR4 SQ15 LL_ADC_REG_SetSequencerRanks
- SQR4 SQ16 LL_ADC_REG_SetSequencerRanks

LL_ADC_REG_GetSequencerRanks

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerRanks (ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group regular sequence: channel on the selected scan sequence rank.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_REG_RANK_1
  - LL_ADC_REG_RANK_2
  - LL_ADC_REG_RANK_3
  - LL_ADC_REG_RANK_4
  - LL_ADC_REG_RANK_5
  - LL_ADC_REG_RANK_6
  - LL_ADC_REG_RANK_7
  - LL_ADC_REG_RANK_8
  - LL_ADC_REG_RANK_9
  - LL_ADC_REG_RANK_10
  - LL_ADC_REG_RANK_11
  - LL_ADC_REG_RANK_12
  - LL_ADC_REG_RANK_13
  - LL_ADC_REG_RANK_14
  - LL_ADC_REG_RANK_15
  - LL_ADC_REG_RANK_16

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)

  (1) On STM32H7, parameter available only on ADC instance: ADC3.
- (2) On STM32H7, parameter available only on ADC instance: ADC2.
- (3) On STM32H7, fast channel (0.125 us for 14-bit resolution (ADC conversion rate up to 8 Ms/s)). Other channels are slow channels (conversion rate: refer to reference manual).
- (1, 2) For ADC channel read back from ADC register, comparison with internal channel parameter to be done using helper macro __LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL().

Notes

- On this STM32 series, ADC group regular sequencer is fully configurable: sequencer length and each rank affectation to a channel are configurable. Refer to description of function LL_ADC_REG_SetSequencerLength().
- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- Usage of the returned channel number: To reinject this channel into another function LL_ADC_xxx: the returned channel number is only partly formatted on definition of literals LL_ADC_CHANNEL_x. Therefore, it has to be compared with parts of literals LL_ADC_CHANNEL_x or using helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Then the selected literal LL_ADC_CHANNEL_x can be used as parameter for another function.To get the channel number in decimal format: process the returned value with the helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB().

Reference Manual to LL API cross reference:

- SQR1 SQ1 LL_ADC_REG_GetSequencerRanks
- SQR1 SQ2 LL_ADC_REG_GetSequencerRanks
- SQR1 SQ3 LL_ADC_REG_GetSequencerRanks
- SQR1 SQ4 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ5 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ6 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ7 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ8 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ9 LL_ADC_REG_GetSequencerRanks
- SQR3 SQ10 LL_ADC_REG_GetSequencerRanks
- SQR3 SQ11 LL_ADC_REG_GetSequencerRanks
- SQR3 SQ12 LL_ADC_REG_GetSequencerRanks
- SQR3 SQ13 LL_ADC_REG_GetSequencerRanks
- SQR3 SQ14 LL_ADC_REG_GetSequencerRanks
- SQR4 SQ15 LL_ADC_REG_GetSequencerRanks
- SQR4 SQ16 LL_ADC_REG_GetSequencerRanks

LL_ADC_REG_SetContinuousMode

Function name

__STATIC_INLINE void LL_ADC_REG_SetContinuousMode (ADC_TypeDef * ADCx, uint32_t Continuous)

Function description

Set ADC continuous conversion mode on ADC group regular.

Parameters

- **ADCx:** ADC instance
- **Continuous:** This parameter can be one of the following values:
  - LL_ADC_REG_CONV_SINGLE
  - LL_ADC_REG_CONV_CONTINUOUS

Return values

- **None:**

Notes

- Description of ADC continuous conversion mode: single mode: one conversion per triggercontinuous mode: after the first trigger, following conversions launched successively automatically.
- It is not possible to enable both ADC group regular continuous mode and sequencer discontinuous mode.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR CONT LL_ADC_REG_SetContinuousMode

LL_ADC_REG_GetContinuousMode

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetContinuousMode (ADC_TypeDef * ADCx)

Function description

Get ADC continuous conversion mode on ADC group regular.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_CONV_SINGLE
  - LL_ADC_REG_CONV_CONTINUOUS

Notes

- Description of ADC continuous conversion mode: single mode: one conversion per triggercontinuous mode: after the first trigger, following conversions launched successively automatically.

Reference Manual to LL API cross reference:

- CFGR CONT LL_ADC_REG_GetContinuousMode

LL_ADC_REG_SetDataTransferMode

Function name

__STATIC_INLINE void LL_ADC_REG_SetDataTransferMode (ADC_TypeDef * ADCx, uint32_t DataTransferMode)

Function description

Set ADC data transfer mode.

Parameters

- **ADCx:** ADC instance
- **DataTransferMode:** Select Data Management configuration

Return values

- **None:**

Notes

- Conversion data can be either: Available in Data RegisterTransferred by DMA in one shot modeTransferred by DMA in circular modeTransferred to DFSDM data register

Reference Manual to LL API cross reference:

- CFGR DMNGT LL_ADC_REG_SetDataTransferMode

LL_ADC_REG_GetDataTransferMode

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetDataTransferMode (ADC_TypeDef * ADCx)

Function description

Get ADC data transfer mode.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_DR_TRANSFER
  - LL_ADC_REG_DMA_TRANSFER_LIMITED
  - LL_ADC_REG_DMA_TRANSFER_UNLIMITED
  - LL_ADC_REG_DFSDM_TRANSFER

Notes

- Conversion data can be either: Available in Data RegisterTransferred by DMA in one shot modeTransferred by DMA in circular modeTransferred to DFSDM data register

Reference Manual to LL API cross reference:

- CFGR DMNGT LL_ADC_REG_GetDataTransferMode

LL_ADC_REG_SetOverrun

Function name

__STATIC_INLINE void LL_ADC_REG_SetOverrun (ADC_TypeDef * ADCx, uint32_t Overrun)

Function description

Set ADC group regular behavior in case of overrun: data preserved or overwritten.

Parameters

- **ADCx:** ADC instance
- **Overrun:** This parameter can be one of the following values:
  - LL_ADC_REG_OVR_DATA_PRESERVED
  - LL_ADC_REG_OVR_DATA_OVERWRITTEN

Return values

- **None:**

Notes

- Compatibility with devices without feature overrun: other devices without this feature have a behavior equivalent to data overwritten. The default setting of overrun is data preserved. Therefore, for compatibility with all devices, parameter overrun should be set to data overwritten.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR OVRMOD LL_ADC_REG_SetOverrun

LL_ADC_REG_GetOverrun

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetOverrun (ADC_TypeDef * ADCx)

Function description

Get ADC group regular behavior in case of overrun: data preserved or overwritten.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_OVR_DATA_PRESERVED
  - LL_ADC_REG_OVR_DATA_OVERWRITTEN

Reference Manual to LL API cross reference:

- CFGR OVRMOD LL_ADC_REG_GetOverrun

LL_ADC_INJ_SetTriggerSource

Function name

__STATIC_INLINE void LL_ADC_INJ_SetTriggerSource (ADC_TypeDef * ADCx, uint32_t TriggerSource)

Function description

Set ADC group injected conversion trigger source: internal (SW start) or from external peripheral (timer event, external interrupt line).

Parameters

- **ADCx:** ADC instance
- **TriggerSource:** This parameter can be one of the following values:
  - LL_ADC_INJ_TRIG_SOFTWARE
  - LL_ADC_INJ_TRIG_EXT_TIM1_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM1_CH4
  - LL_ADC_INJ_TRIG_EXT_TIM2_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM2_CH1
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH4
  - LL_ADC_INJ_TRIG_EXT_TIM4_TRGO
  - LL_ADC_INJ_TRIG_EXT_EXTI_LINE15
  - LL_ADC_INJ_TRIG_EXT_TIM8_CH4
  - LL_ADC_INJ_TRIG_EXT_TIM1_TRGO2
  - LL_ADC_INJ_TRIG_EXT_TIM8_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM8_TRGO2
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH3
  - LL_ADC_INJ_TRIG_EXT_TIM3_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH1
  - LL_ADC_INJ_TRIG_EXT_TIM6_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM15_TRGO
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG2
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG4
  - LL_ADC_INJ_TRIG_EXT_LPTIM1_OUT
  - LL_ADC_INJ_TRIG_EXT_LPTIM2_OUT
  - LL_ADC_INJ_TRIG_EXT_LPTIM3_OUT

Return values

- **None:**

Notes

- On this STM32 series, setting trigger source to external trigger also set trigger polarity to rising edge (default setting for compatibility with some ADC on other STM32 families having this setting set by HW default value). In case of need to modify trigger edge, use function LL_ADC_INJ_SetTriggerEdge().
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must not be disabled. Can be enabled with or without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- JSQR JEXTSEL LL_ADC_INJ_SetTriggerSource
- JSQR JEXTEN LL_ADC_INJ_SetTriggerSource

LL_ADC_INJ_GetTriggerSource

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetTriggerSource (ADC_TypeDef * ADCx)

Function description

Get ADC group injected conversion trigger source: internal (SW start) or from external peripheral (timer event, external interrupt line).

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_INJ_TRIG_SOFTWARE
  - LL_ADC_INJ_TRIG_EXT_TIM1_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM1_CH4
  - LL_ADC_INJ_TRIG_EXT_TIM2_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM2_CH1
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH4
  - LL_ADC_INJ_TRIG_EXT_TIM4_TRGO
  - LL_ADC_INJ_TRIG_EXT_EXTI_LINE15
  - LL_ADC_INJ_TRIG_EXT_TIM8_CH4
  - LL_ADC_INJ_TRIG_EXT_TIM1_TRGO2
  - LL_ADC_INJ_TRIG_EXT_TIM8_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM8_TRGO2
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH3
  - LL_ADC_INJ_TRIG_EXT_TIM3_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH1
  - LL_ADC_INJ_TRIG_EXT_TIM6_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM15_TRGO
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG2
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG4
  - LL_ADC_INJ_TRIG_EXT_LPTIM1_OUT
  - LL_ADC_INJ_TRIG_EXT_LPTIM2_OUT
  - LL_ADC_INJ_TRIG_EXT_LPTIM3_OUT

Notes

- To determine whether group injected trigger source is internal (SW start) or external, without detail of which peripheral is selected as external trigger, (equivalent to "if(LL_ADC_INJ_GetTriggerSource(ADC1) == LL_ADC_INJ_TRIG_SOFTWARE)") use function LL_ADC_INJ_IsTriggerSourceSWStart.
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.

Reference Manual to LL API cross reference:

- JSQR JEXTSEL LL_ADC_INJ_GetTriggerSource
- JSQR JEXTEN LL_ADC_INJ_GetTriggerSource

LL_ADC_INJ_IsTriggerSourceSWStart

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_IsTriggerSourceSWStart (ADC_TypeDef * ADCx)

Function description

Get ADC group injected conversion trigger source internal (SW start) or external.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** "0" if trigger source external trigger Value "1" if trigger source SW start.

Notes

- In case of group injected trigger source set to external trigger, to determine which peripheral is selected as external trigger, use function LL_ADC_INJ_GetTriggerSource.

Reference Manual to LL API cross reference:

- JSQR JEXTEN LL_ADC_INJ_IsTriggerSourceSWStart

LL_ADC_INJ_SetTriggerEdge

Function name

__STATIC_INLINE void LL_ADC_INJ_SetTriggerEdge (ADC_TypeDef * ADCx, uint32_t ExternalTriggerEdge)

Function description

Set ADC group injected conversion trigger polarity.

Parameters

- **ADCx:** ADC instance
- **ExternalTriggerEdge:** This parameter can be one of the following values:
  - LL_ADC_INJ_TRIG_EXT_RISING
  - LL_ADC_INJ_TRIG_EXT_FALLING
  - LL_ADC_INJ_TRIG_EXT_RISINGFALLING

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must not be disabled. Can be enabled with or without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- JSQR JEXTEN LL_ADC_INJ_SetTriggerEdge

LL_ADC_INJ_GetTriggerEdge

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetTriggerEdge (ADC_TypeDef * ADCx)

Function description

Get ADC group injected conversion trigger polarity.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_INJ_TRIG_EXT_RISING
  - LL_ADC_INJ_TRIG_EXT_FALLING
  - LL_ADC_INJ_TRIG_EXT_RISINGFALLING

Reference Manual to LL API cross reference:

- JSQR JEXTEN LL_ADC_INJ_GetTriggerEdge

LL_ADC_INJ_SetSequencerLength

Function name

__STATIC_INLINE void LL_ADC_INJ_SetSequencerLength (ADC_TypeDef * ADCx, uint32_t SequencerNbRanks)

Function description

Set ADC group injected sequencer length and scan direction.

Parameters

- **ADCx:** ADC instance
- **SequencerNbRanks:** This parameter can be one of the following values:
  - LL_ADC_INJ_SEQ_SCAN_DISABLE
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_2RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_3RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_4RANKS

Return values

- **None:**

Notes

- This function performs configuration of: Sequence length: Number of ranks in the scan sequence.Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from rank 1 to rank n).
- Sequencer disabled is equivalent to sequencer of 1 rank: ADC conversion on only 1 channel.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must not be disabled. Can be enabled with or without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- JSQR JL LL_ADC_INJ_SetSequencerLength

LL_ADC_INJ_GetSequencerLength

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetSequencerLength (ADC_TypeDef * ADCx)

Function description

Get ADC group injected sequencer length and scan direction.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_INJ_SEQ_SCAN_DISABLE
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_2RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_3RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_4RANKS

Notes

- This function retrieves: Sequence length: Number of ranks in the scan sequence.Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from rank 1 to rank n).
- Sequencer disabled is equivalent to sequencer of 1 rank: ADC conversion on only 1 channel.

Reference Manual to LL API cross reference:

- JSQR JL LL_ADC_INJ_GetSequencerLength

LL_ADC_INJ_SetSequencerDiscont

Function name

__STATIC_INLINE void LL_ADC_INJ_SetSequencerDiscont (ADC_TypeDef * ADCx, uint32_t SeqDiscont)

Function description

Set ADC group injected sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.

Parameters

- **ADCx:** ADC instance
- **SeqDiscont:** This parameter can be one of the following values:
  - LL_ADC_INJ_SEQ_DISCONT_DISABLE
  - LL_ADC_INJ_SEQ_DISCONT_1RANK

Return values

- **None:**

Notes

- It is not possible to enable both ADC group injected auto-injected mode and sequencer discontinuous mode.

Reference Manual to LL API cross reference:

- CFGR JDISCEN LL_ADC_INJ_SetSequencerDiscont

LL_ADC_INJ_GetSequencerDiscont

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetSequencerDiscont (ADC_TypeDef * ADCx)

Function description

Get ADC group injected sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_INJ_SEQ_DISCONT_DISABLE
  - LL_ADC_INJ_SEQ_DISCONT_1RANK

Reference Manual to LL API cross reference:

- CFGR JDISCEN LL_ADC_INJ_GetSequencerDiscont

LL_ADC_INJ_SetSequencerRanks

Function name

__STATIC_INLINE void LL_ADC_INJ_SetSequencerRanks (ADC_TypeDef * ADCx, uint32_t Rank, uint32_t Channel)

Function description

Set ADC group injected sequence: channel on the selected sequence rank.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)

  (1) On STM32H7, parameter available only on ADC instance: ADC3.
- (2) On STM32H7, parameter available only on ADC instance: ADC2.
- (3) On STM32H7, fast channel (0.125 us for 14-bit resolution (ADC conversion rate up to 8 Ms/s)). Other channels are slow channels (conversion rate: refer to reference manual).

Return values

- **None:**

Notes

- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- On this STM32 series, to measure internal channels (VrefInt, TempSensor, ...), measurement paths to internal channels must be enabled separately. This can be done using function LL_ADC_SetCommonPathInternalCh().
- On STM32H7, some fast channels are available: fast analog inputs coming from GPIO pads (ADC_IN0..5).
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must not be disabled. Can be enabled with or without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- JSQR JSQ1 LL_ADC_INJ_SetSequencerRanks
- JSQR JSQ2 LL_ADC_INJ_SetSequencerRanks
- JSQR JSQ3 LL_ADC_INJ_SetSequencerRanks
- JSQR JSQ4 LL_ADC_INJ_SetSequencerRanks

LL_ADC_INJ_GetSequencerRanks

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetSequencerRanks (ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected sequence: channel on the selected sequence rank.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)

  (1) On STM32H7, parameter available only on ADC instance: ADC3.
- (2) On STM32H7, parameter available only on ADC instance: ADC2.
- (3) On STM32H7, fast channel (0.125 us for 14-bit resolution (ADC conversion rate up to 8 Ms/s)). Other channels are slow channels (conversion rate: refer to reference manual).
- (1, 2) For ADC channel read back from ADC register, comparison with internal channel parameter to be done using helper macro __LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL().

Notes

- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- Usage of the returned channel number: To reinject this channel into another function LL_ADC_xxx: the returned channel number is only partly formatted on definition of literals LL_ADC_CHANNEL_x. Therefore, it has to be compared with parts of literals LL_ADC_CHANNEL_x or using helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Then the selected literal LL_ADC_CHANNEL_x can be used as parameter for another function.To get the channel number in decimal format: process the returned value with the helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB().

Reference Manual to LL API cross reference:

- JSQR JSQ1 LL_ADC_INJ_GetSequencerRanks
- JSQR JSQ2 LL_ADC_INJ_GetSequencerRanks
- JSQR JSQ3 LL_ADC_INJ_GetSequencerRanks
- JSQR JSQ4 LL_ADC_INJ_GetSequencerRanks

LL_ADC_INJ_SetTrigAuto

Function name

__STATIC_INLINE void LL_ADC_INJ_SetTrigAuto (ADC_TypeDef * ADCx, uint32_t TrigAuto)

Function description

Set ADC group injected conversion trigger: independent or from ADC group regular.

Parameters

- **ADCx:** ADC instance
- **TrigAuto:** This parameter can be one of the following values:
  - LL_ADC_INJ_TRIG_INDEPENDENT
  - LL_ADC_INJ_TRIG_FROM_GRP_REGULAR

Return values

- **None:**

Notes

- This mode can be used to extend number of data registers updated after one ADC conversion trigger and with data permanently kept (not erased by successive conversions of scan of ADC sequencer ranks), up to 5 data registers: 1 data register on ADC group regular, 4 data registers on ADC group injected.
- If ADC group injected injected trigger source is set to an external trigger, this feature must be must be set to independent trigger. ADC group injected automatic trigger is compliant only with group injected trigger source set to SW start, without any further action on ADC group injected conversion start or stop: in this case, ADC group injected is controlled only from ADC group regular.
- It is not possible to enable both ADC group injected auto-injected mode and sequencer discontinuous mode.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR JAUTO LL_ADC_INJ_SetTrigAuto

LL_ADC_INJ_GetTrigAuto

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetTrigAuto (ADC_TypeDef * ADCx)

Function description

Get ADC group injected conversion trigger: independent or from ADC group regular.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_INJ_TRIG_INDEPENDENT
  - LL_ADC_INJ_TRIG_FROM_GRP_REGULAR

Reference Manual to LL API cross reference:

- CFGR JAUTO LL_ADC_INJ_GetTrigAuto

LL_ADC_INJ_SetQueueMode

Function name

__STATIC_INLINE void LL_ADC_INJ_SetQueueMode (ADC_TypeDef * ADCx, uint32_t QueueMode)

Function description

Set ADC group injected contexts queue mode.

Parameters

- **ADCx:** ADC instance
- **QueueMode:** This parameter can be one of the following values:
  - LL_ADC_INJ_QUEUE_DISABLE
  - LL_ADC_INJ_QUEUE_2CONTEXTS_LAST_ACTIVE
  - LL_ADC_INJ_QUEUE_2CONTEXTS_END_EMPTY

Return values

- **None:**

Notes

- A context is a setting of group injected sequencer: group injected triggersequencer lengthsequencer ranks If contexts queue is disabled:only 1 sequence can be configured and is active perpetually. If contexts queue is enabled:up to 2 contexts can be queued and are checked in and out as a FIFO stack (first-in, first-out).If a new context is set when queues is full, error is triggered by interruption "Injected Queue Overflow".Two behaviors are possible when all contexts have been processed: the contexts queue can maintain the last context active perpetually or can be empty and injected group triggers are disabled.Triggers can be only external (not internal SW start)Caution: The sequence must be fully configured in one time (one write of register JSQR makes a check-in of a new context into the queue). Therefore functions to set separately injected trigger and sequencer channels cannot be used, register JSQR must be set using function LL_ADC_INJ_ConfigQueueContext().
- This parameter can be modified only when no conversion is on going on either groups regular or injected.
- A modification of the context mode (bit JQDIS) causes the contexts queue to be flushed and the register JSQR is cleared.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR JQM LL_ADC_INJ_SetQueueMode
- CFGR JQDIS LL_ADC_INJ_SetQueueMode

LL_ADC_INJ_GetQueueMode

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetQueueMode (ADC_TypeDef * ADCx)

Function description

Get ADC group injected context queue mode.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_INJ_QUEUE_DISABLE
  - LL_ADC_INJ_QUEUE_2CONTEXTS_LAST_ACTIVE
  - LL_ADC_INJ_QUEUE_2CONTEXTS_END_EMPTY

Reference Manual to LL API cross reference:

- CFGR JQM LL_ADC_INJ_GetQueueMode
- CFGR JQDIS LL_ADC_INJ_GetQueueMode

LL_ADC_INJ_ConfigQueueContext

Function name

__STATIC_INLINE void LL_ADC_INJ_ConfigQueueContext (ADC_TypeDef * ADCx, uint32_t TriggerSource, uint32_t ExternalTriggerEdge, uint32_t SequencerNbRanks, uint32_t Rank1_Channel, uint32_t Rank2_Channel, uint32_t Rank3_Channel, uint32_t Rank4_Channel)

Function description

Set one context on ADC group injected that will be checked in contexts queue.

Parameters

- **ADCx:** ADC instance
- **TriggerSource:** This parameter can be one of the following values:
  - LL_ADC_INJ_TRIG_SOFTWARE
  - LL_ADC_INJ_TRIG_EXT_TIM1_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM1_CH4
  - LL_ADC_INJ_TRIG_EXT_TIM2_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM2_CH1
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH4
  - LL_ADC_INJ_TRIG_EXT_TIM4_TRGO
  - LL_ADC_INJ_TRIG_EXT_EXTI_LINE15
  - LL_ADC_INJ_TRIG_EXT_TIM8_CH4
  - LL_ADC_INJ_TRIG_EXT_TIM1_TRGO2
  - LL_ADC_INJ_TRIG_EXT_TIM8_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM8_TRGO2
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH3
  - LL_ADC_INJ_TRIG_EXT_TIM3_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH1
  - LL_ADC_INJ_TRIG_EXT_TIM6_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM15_TRGO
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG2
  - LL_ADC_INJ_TRIG_EXT_HRTIM_TRG4
  - LL_ADC_INJ_TRIG_EXT_LPTIM1_OUT
  - LL_ADC_INJ_TRIG_EXT_LPTIM2_OUT
  - LL_ADC_INJ_TRIG_EXT_LPTIM3_OUT
- **ExternalTriggerEdge:** This parameter can be one of the following values:
  - LL_ADC_INJ_TRIG_EXT_RISING
  - LL_ADC_INJ_TRIG_EXT_FALLING
  - LL_ADC_INJ_TRIG_EXT_RISINGFALLING

  Note: This parameter is discarded in case of SW start: parameter "TriggerSource" set to "LL_ADC_INJ_TRIG_SOFTWARE".
- **SequencerNbRanks:** This parameter can be one of the following values:
  - LL_ADC_INJ_SEQ_SCAN_DISABLE
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_2RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_3RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_4RANKS

- **Rank1_Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)

  (1) On STM32H7, parameter available only on ADC instance: ADC3.
- (2) On STM32H7, parameter available only on ADC instance: ADC2.
- (3) On STM32H7, fast channel (0.125 us for 14-bit resolution (ADC conversion rate up to 8 Ms/s)). Other channels are slow channels (conversion rate: refer to reference manual).

- **Rank2_Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)

  (1) On STM32H7, parameter available only on ADC instance: ADC3.
- (2) On STM32H7, parameter available only on ADC instance: ADC2.
- (3) On STM32H7, fast channel (0.125 us for 14-bit resolution (ADC conversion rate up to 8 Ms/s)). Other channels are slow channels (conversion rate: refer to reference manual).

- **Rank3_Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)

  (1) On STM32H7, parameter available only on ADC instance: ADC3.
- (2) On STM32H7, parameter available only on ADC instance: ADC2.
- (3) On STM32H7, fast channel (0.125 us for 14-bit resolution (ADC conversion rate up to 8 Ms/s)). Other channels are slow channels (conversion rate: refer to reference manual).

- **Rank4_Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)

  (1) On STM32H7, parameter available only on ADC instance: ADC3.
- (2) On STM32H7, parameter available only on ADC instance: ADC2.
- (3) On STM32H7, fast channel (0.125 us for 14-bit resolution (ADC conversion rate up to 8 Ms/s)). Other channels are slow channels (conversion rate: refer to reference manual).

Return values

- **None:**

Notes

- A context is a setting of group injected sequencer: group injected triggersequencer lengthsequencer ranks This function is intended to be used when contexts queue is enabled, because the sequence must be fully configured in one time (functions to set separately injected trigger and sequencer channels cannot be used): Refer to function LL_ADC_INJ_SetQueueMode().
- In the contexts queue, only the active context can be read. The parameters of this function can be read using functions: LL_ADC_INJ_GetTriggerSource() LL_ADC_INJ_GetTriggerEdge() LL_ADC_INJ_GetSequencerRanks()
- On this STM32 series, to measure internal channels (VrefInt, TempSensor, ...), measurement paths to internal channels must be enabled separately. This can be done using function LL_ADC_SetCommonPathInternalCh().
- On STM32H7, some fast channels are available: fast analog inputs coming from GPIO pads (ADC_IN0..5).
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must not be disabled. Can be enabled with or without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- JSQR JEXTSEL LL_ADC_INJ_ConfigQueueContext
- JSQR JEXTEN LL_ADC_INJ_ConfigQueueContext
- JSQR JL LL_ADC_INJ_ConfigQueueContext
- JSQR JSQ1 LL_ADC_INJ_ConfigQueueContext
- JSQR JSQ2 LL_ADC_INJ_ConfigQueueContext
- JSQR JSQ3 LL_ADC_INJ_ConfigQueueContext
- JSQR JSQ4 LL_ADC_INJ_ConfigQueueContext

LL_ADC_SetChannelSamplingTime

Function name

__STATIC_INLINE void LL_ADC_SetChannelSamplingTime (ADC_TypeDef * ADCx, uint32_t Channel, uint32_t SamplingTime)

Function description

Set sampling time of the selected ADC channel Unit: ADC clock cycles.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)

  (1) On STM32H7, parameter available only on ADC instance: ADC3.
- (2) On STM32H7, parameter available only on ADC instance: ADC2.
- (3) On STM32H7, fast channel (0.125 us for 14-bit resolution (ADC conversion rate up to 8 Ms/s)). Other channels are slow channels (conversion rate: refer to reference manual).
- **SamplingTime:** This parameter can be one of the following values:
  - LL_ADC_SAMPLINGTIME_1CYCLE_5
  - LL_ADC_SAMPLINGTIME_2CYCLES_5
  - LL_ADC_SAMPLINGTIME_8CYCLES_5
  - LL_ADC_SAMPLINGTIME_16CYCLES_5
  - LL_ADC_SAMPLINGTIME_32CYCLES_5
  - LL_ADC_SAMPLINGTIME_64CYCLES_5
  - LL_ADC_SAMPLINGTIME_387CYCLES_5
  - LL_ADC_SAMPLINGTIME_810CYCLES_5

Return values

- **None:**

Notes

- On this device, sampling time is on channel scope: independently of channel mapped on ADC group regular or injected.
- In case of internal channel (VrefInt, TempSensor, ...) to be converted: sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting). Refer to device datasheet for timings values (parameters TS_vrefint, TS_temp, ...).
- Conversion time is the addition of sampling time and processing time. On this STM32 series, ADC processing time is: 12.5 ADC clock cycles at ADC resolution 12 bits10.5 ADC clock cycles at ADC resolution 10 bits8.5 ADC clock cycles at ADC resolution 8 bits6.5 ADC clock cycles at ADC resolution 6 bits
- In case of ADC conversion of internal channel (VrefInt, temperature sensor, ...), a sampling time minimum value is required. Refer to device datasheet.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- SMPR1 SMP0 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP1 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP2 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP3 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP4 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP5 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP6 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP7 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP8 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP9 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP10 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP11 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP12 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP13 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP14 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP15 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP16 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP17 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP18 LL_ADC_SetChannelSamplingTime

LL_ADC_GetChannelSamplingTime

Function name

__STATIC_INLINE uint32_t LL_ADC_GetChannelSamplingTime (ADC_TypeDef * ADCx, uint32_t Channel)

Function description

Get sampling time of the selected ADC channel Unit: ADC clock cycles.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)

  (1) On STM32H7, parameter available only on ADC instance: ADC3.
- (2) On STM32H7, parameter available only on ADC instance: ADC2.
- (3) On STM32H7, fast channel (0.125 us for 14-bit resolution (ADC conversion rate up to 8 Ms/s)). Other channels are slow channels (conversion rate: refer to reference manual).

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_SAMPLINGTIME_1CYCLE_5
  - LL_ADC_SAMPLINGTIME_2CYCLES_5
  - LL_ADC_SAMPLINGTIME_8CYCLES_5
  - LL_ADC_SAMPLINGTIME_16CYCLES_5
  - LL_ADC_SAMPLINGTIME_32CYCLES_5
  - LL_ADC_SAMPLINGTIME_64CYCLES_5
  - LL_ADC_SAMPLINGTIME_387CYCLES_5
  - LL_ADC_SAMPLINGTIME_810CYCLES_5

Notes

- On this device, sampling time is on channel scope: independently of channel mapped on ADC group regular or injected.
- Conversion time is the addition of sampling time and processing time. On this STM32 series, ADC processing time is: 12.5 ADC clock cycles at ADC resolution 12 bits10.5 ADC clock cycles at ADC resolution 10 bits8.5 ADC clock cycles at ADC resolution 8 bits6.5 ADC clock cycles at ADC resolution 6 bits

Reference Manual to LL API cross reference:

- SMPR1 SMP0 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP1 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP2 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP3 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP4 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP5 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP6 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP7 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP8 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP9 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP10 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP11 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP12 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP13 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP14 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP15 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP16 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP17 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP18 LL_ADC_GetChannelSamplingTime

LL_ADC_SetChannelSingleDiff

Function name

__STATIC_INLINE void LL_ADC_SetChannelSingleDiff (ADC_TypeDef * ADCx, uint32_t Channel, uint32_t SingleDiff)

Function description

Set mode single-ended or differential input of the selected ADC channel.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
- **SingleDiff:** This parameter can be a combination of the following values:
  - LL_ADC_SINGLE_ENDED
  - LL_ADC_DIFFERENTIAL_ENDED

Return values

- **None:**

Notes

- Channel ending is on channel scope: independently of channel mapped on ADC group regular or injected. In differential mode: Differential measurement is carried out between the selected channel 'i' (positive input) and channel 'i+1' (negative input). Only channel 'i' has to be configured, channel 'i+1' is configured automatically.
- Refer to Reference Manual to ensure the selected channel is available in differential mode. For example, internal channels (VrefInt, TempSensor, ...) are not available in differential mode.
- When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately.
- On STM32H7, some channels are internally fixed to single-ended inputs configuration: ADC1: Channels 0, 6, 7, 8, 9, 13, 14, 15, 17, and 19ADC2: Channels 0, 6, 7, 8, 9, 13, 14, 15 and 19ADC3: Channels 0, 6, 7, 8, 9, 12, 16, 17, and 19
- For ADC channels configured in differential mode, both inputs should be biased at (Vref+)/2 +/-200mV. (Vref+ is the analog voltage reference)
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.
- One or several values can be selected. Example: (LL_ADC_CHANNEL_4 | LL_ADC_CHANNEL_12 | ...)

Reference Manual to LL API cross reference:

- DIFSEL DIFSEL LL_ADC_SetChannelSingleDiff

LL_ADC_GetChannelSingleDiff

Function name

__STATIC_INLINE uint32_t LL_ADC_GetChannelSingleDiff (ADC_TypeDef * ADCx, uint32_t Channel)

Function description

Get mode single-ended or differential input of the selected ADC channel.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be a combination of the following values:
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19

Return values

- **0:** channel in single-ended mode, else: channel in differential mode

Notes

- When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately. Therefore, to ensure a channel is configured in single-ended mode, the configuration of channel itself and the channel 'i-1' must be read back (to ensure that the selected channel channel has not been configured in differential mode by the previous channel).
- Refer to Reference Manual to ensure the selected channel is available in differential mode. For example, internal channels (VrefInt, TempSensor, ...) are not available in differential mode.
- When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately.
- On STM32H7, some channels are internally fixed to single-ended inputs configuration: ADC1: Channels 0, 6, 7, 8, 9, 13, 14, 15, 17, and 19ADC2: Channels 0, 6, 7, 8, 9, 13, 14, 15 and 19ADC3: Channels 0, 6, 7, 8, 9, 12, 16, 17, and 19
- One or several values can be selected. In this case, the value returned is null if all channels are in single ended-mode. Example: (LL_ADC_CHANNEL_4 | LL_ADC_CHANNEL_12 | ...)

Reference Manual to LL API cross reference:

- DIFSEL DIFSEL LL_ADC_GetChannelSingleDiff

LL_ADC_SetAnalogWDMonitChannels

Function name

__STATIC_INLINE void LL_ADC_SetAnalogWDMonitChannels (ADC_TypeDef * ADCx, uint32_t AWDy, uint32_t AWDChannelGroup)

Function description

Set ADC analog watchdog monitored channels: a single channel, multiple channels or all channels, on ADC groups regular and-or injected.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  - LL_ADC_AWD1
  - LL_ADC_AWD2
  - LL_ADC_AWD3

- **AWDChannelGroup:** This parameter can be one of the following values:
  - LL_ADC_AWD_DISABLE
  - LL_ADC_AWD_ALL_CHANNELS_REG (0)
  - LL_ADC_AWD_ALL_CHANNELS_INJ (0)
  - LL_ADC_AWD_ALL_CHANNELS_REG_INJ
  - LL_ADC_AWD_CHANNEL_0_REG (0)
  - LL_ADC_AWD_CHANNEL_0_INJ (0)
  - LL_ADC_AWD_CHANNEL_0_REG_INJ
  - LL_ADC_AWD_CHANNEL_1_REG (0)
  - LL_ADC_AWD_CHANNEL_1_INJ (0)
  - LL_ADC_AWD_CHANNEL_1_REG_INJ
  - LL_ADC_AWD_CHANNEL_2_REG (0)
  - LL_ADC_AWD_CHANNEL_2_INJ (0)
  - LL_ADC_AWD_CHANNEL_2_REG_INJ
  - LL_ADC_AWD_CHANNEL_3_REG (0)
  - LL_ADC_AWD_CHANNEL_3_INJ (0)
  - LL_ADC_AWD_CHANNEL_3_REG_INJ
  - LL_ADC_AWD_CHANNEL_4_REG (0)
  - LL_ADC_AWD_CHANNEL_4_INJ (0)
  - LL_ADC_AWD_CHANNEL_4_REG_INJ
  - LL_ADC_AWD_CHANNEL_5_REG (0)
  - LL_ADC_AWD_CHANNEL_5_INJ (0)
  - LL_ADC_AWD_CHANNEL_5_REG_INJ
  - LL_ADC_AWD_CHANNEL_6_REG (0)
  - LL_ADC_AWD_CHANNEL_6_INJ (0)
  - LL_ADC_AWD_CHANNEL_6_REG_INJ
  - LL_ADC_AWD_CHANNEL_7_REG (0)
  - LL_ADC_AWD_CHANNEL_7_INJ (0)
  - LL_ADC_AWD_CHANNEL_7_REG_INJ
  - LL_ADC_AWD_CHANNEL_8_REG (0)
  - LL_ADC_AWD_CHANNEL_8_INJ (0)
  - LL_ADC_AWD_CHANNEL_8_REG_INJ
  - LL_ADC_AWD_CHANNEL_9_REG (0)
  - LL_ADC_AWD_CHANNEL_9_INJ (0)
  - LL_ADC_AWD_CHANNEL_9_REG_INJ
  - LL_ADC_AWD_CHANNEL_10_REG (0)
  - LL_ADC_AWD_CHANNEL_10_INJ (0)
  - LL_ADC_AWD_CHANNEL_10_REG_INJ
  - LL_ADC_AWD_CHANNEL_11_REG (0)
  - LL_ADC_AWD_CHANNEL_11_INJ (0)
  - LL_ADC_AWD_CHANNEL_11_REG_INJ
  - LL_ADC_AWD_CHANNEL_12_REG (0)
  - LL_ADC_AWD_CHANNEL_12_INJ (0)
  - LL_ADC_AWD_CHANNEL_12_REG_INJ
  - LL_ADC_AWD_CHANNEL_13_REG (0)
  - LL_ADC_AWD_CHANNEL_13_INJ (0)
  - LL_ADC_AWD_CHANNEL_13_REG_INJ
  - LL_ADC_AWD_CHANNEL_14_REG (0)
  - LL_ADC_AWD_CHANNEL_14_INJ (0)
  - LL_ADC_AWD_CHANNEL_14_REG_INJ
  - LL_ADC_AWD_CHANNEL_15_REG (0)
  - LL_ADC_AWD_CHANNEL_15_INJ (0)
  - LL_ADC_AWD_CHANNEL_15_REG_INJ

- (1) On STM32H7, parameter available only on ADC instance: ADC3.
- (2) On STM32H7, parameter available only on ADC instance: ADC2.

Return values

- **None:**

Notes

- Once monitored channels are selected, analog watchdog is enabled.
- In case of need to define a single channel to monitor with analog watchdog from sequencer channel definition, use helper macro __LL_ADC_ANALOGWD_CHANNEL_GROUP().
- On this STM32 series, there are 2 kinds of analog watchdog instance: AWD standard (instance AWD1): channels monitored: can monitor 1 channel or all channels.groups monitored: ADC groups regular and-or injected.resolution: resolution is not limited (corresponds to ADC resolution configured). AWD flexible (instances AWD2, AWD3): channels monitored: flexible on channels monitored, selection is channel wise, from from 1 to all channels. Specificity of this analog watchdog: Multiple channels can be selected. For example: (LL_ADC_AWD_CHANNEL4_REG_INJ | LL_ADC_AWD_CHANNEL5_REG_INJ | ...)groups monitored: not selection possible (monitoring on both groups regular and injected). Channels selected are monitored on groups regular and injected: LL_ADC_AWD_CHANNELxx_REG_INJ (do not use parameters LL_ADC_AWD_CHANNELxx_REG and LL_ADC_AWD_CHANNELxx_INJ)resolution: resolution is limited to 8 bits: if ADC resolution is 12 bits the 4 LSB are ignored, if ADC resolution is 10 bits the 2 LSB are ignored.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR AWD1CH LL_ADC_SetAnalogWDMonitChannels
- CFGR AWD1SGL LL_ADC_SetAnalogWDMonitChannels
- CFGR AWD1EN LL_ADC_SetAnalogWDMonitChannels
- CFGR JAWD1EN LL_ADC_SetAnalogWDMonitChannels
- AWD2CR AWD2CH LL_ADC_SetAnalogWDMonitChannels
- AWD3CR AWD3CH LL_ADC_SetAnalogWDMonitChannels

LL_ADC_GetAnalogWDMonitChannels

Function name

__STATIC_INLINE uint32_t LL_ADC_GetAnalogWDMonitChannels (ADC_TypeDef * ADCx, uint32_t AWDy)

Function description

Get ADC analog watchdog monitored channel.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  - LL_ADC_AWD1
  - LL_ADC_AWD2 (1)
  - LL_ADC_AWD3 (1)

  (1) On this AWD number, monitored channel can be retrieved if only 1 channel is programmed (or none or all channels). This function cannot retrieve monitored channel if multiple channels are programmed simultaneously by bitfield.

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_AWD_DISABLE
  - LL_ADC_AWD_ALL_CHANNELS_REG (0)
  - LL_ADC_AWD_ALL_CHANNELS_INJ (0)
  - LL_ADC_AWD_ALL_CHANNELS_REG_INJ
  - LL_ADC_AWD_CHANNEL_0_REG (0)
  - LL_ADC_AWD_CHANNEL_0_INJ (0)
  - LL_ADC_AWD_CHANNEL_0_REG_INJ
  - LL_ADC_AWD_CHANNEL_1_REG (0)
  - LL_ADC_AWD_CHANNEL_1_INJ (0)
  - LL_ADC_AWD_CHANNEL_1_REG_INJ
  - LL_ADC_AWD_CHANNEL_2_REG (0)
  - LL_ADC_AWD_CHANNEL_2_INJ (0)
  - LL_ADC_AWD_CHANNEL_2_REG_INJ
  - LL_ADC_AWD_CHANNEL_3_REG (0)
  - LL_ADC_AWD_CHANNEL_3_INJ (0)
  - LL_ADC_AWD_CHANNEL_3_REG_INJ
  - LL_ADC_AWD_CHANNEL_4_REG (0)
  - LL_ADC_AWD_CHANNEL_4_INJ (0)
  - LL_ADC_AWD_CHANNEL_4_REG_INJ
  - LL_ADC_AWD_CHANNEL_5_REG (0)
  - LL_ADC_AWD_CHANNEL_5_INJ (0)
  - LL_ADC_AWD_CHANNEL_5_REG_INJ
  - LL_ADC_AWD_CHANNEL_6_REG (0)
  - LL_ADC_AWD_CHANNEL_6_INJ (0)
  - LL_ADC_AWD_CHANNEL_6_REG_INJ
  - LL_ADC_AWD_CHANNEL_7_REG (0)
  - LL_ADC_AWD_CHANNEL_7_INJ (0)
  - LL_ADC_AWD_CHANNEL_7_REG_INJ
  - LL_ADC_AWD_CHANNEL_8_REG (0)
  - LL_ADC_AWD_CHANNEL_8_INJ (0)
  - LL_ADC_AWD_CHANNEL_8_REG_INJ
  - LL_ADC_AWD_CHANNEL_9_REG (0)
  - LL_ADC_AWD_CHANNEL_9_INJ (0)
  - LL_ADC_AWD_CHANNEL_9_REG_INJ
  - LL_ADC_AWD_CHANNEL_10_REG (0)
  - LL_ADC_AWD_CHANNEL_10_INJ (0)
  - LL_ADC_AWD_CHANNEL_10_REG_INJ
  - LL_ADC_AWD_CHANNEL_11_REG (0)
  - LL_ADC_AWD_CHANNEL_11_INJ (0)
  - LL_ADC_AWD_CHANNEL_11_REG_INJ
  - LL_ADC_AWD_CHANNEL_12_REG (0)
  - LL_ADC_AWD_CHANNEL_12_INJ (0)
  - LL_ADC_AWD_CHANNEL_12_REG_INJ
  - LL_ADC_AWD_CHANNEL_13_REG (0)
  - LL_ADC_AWD_CHANNEL_13_INJ (0)
  - LL_ADC_AWD_CHANNEL_13_REG_INJ
  - LL_ADC_AWD_CHANNEL_14_REG (0)
  - LL_ADC_AWD_CHANNEL_14_INJ (0)
  - LL_ADC_AWD_CHANNEL_14_REG_INJ
  - LL_ADC_AWD_CHANNEL_15_REG (0)

Notes

- Usage of the returned channel number: To reinject this channel into another function LL_ADC_xxx: the returned channel number is only partly formatted on definition of literals LL_ADC_CHANNEL_x. Therefore, it has to be compared with parts of literals LL_ADC_CHANNEL_x or using helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Then the selected literal LL_ADC_CHANNEL_x can be used as parameter for another function.To get the channel number in decimal format: process the returned value with the helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Applicable only when the analog watchdog is set to monitor one channel.
- On this STM32 series, there are 2 kinds of analog watchdog instance: AWD standard (instance AWD1): channels monitored: can monitor 1 channel or all channels.groups monitored: ADC groups regular and-or injected.resolution: resolution is not limited (corresponds to ADC resolution configured). AWD flexible (instances AWD2, AWD3): channels monitored: flexible on channels monitored, selection is channel wise, from from 1 to all channels. Specificity of this analog watchdog: Multiple channels can be selected. For example: (LL_ADC_AWD_CHANNEL4_REG_INJ | LL_ADC_AWD_CHANNEL5_REG_INJ | ...)groups monitored: not selection possible (monitoring on both groups regular and injected). Channels selected are monitored on groups regular and injected: LL_ADC_AWD_CHANNELxx_REG_INJ (do not use parameters LL_ADC_AWD_CHANNELxx_REG and LL_ADC_AWD_CHANNELxx_INJ)resolution: resolution is limited to 8 bits: if ADC resolution is 12 bits the 4 LSB are ignored, if ADC resolution is 10 bits the 2 LSB are ignored.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR AWD1CH LL_ADC_GetAnalogWDMonitChannels
- CFGR AWD1SGL LL_ADC_GetAnalogWDMonitChannels
- CFGR AWD1EN LL_ADC_GetAnalogWDMonitChannels
- CFGR JAWD1EN LL_ADC_GetAnalogWDMonitChannels
- AWD2CR AWD2CH LL_ADC_GetAnalogWDMonitChannels
- AWD3CR AWD3CH LL_ADC_GetAnalogWDMonitChannels

LL_ADC_SetAnalogWDThresholds

Function name

__STATIC_INLINE void LL_ADC_SetAnalogWDThresholds (ADC_TypeDef * ADCx, uint32_t AWDy, uint32_t AWDThresholdsHighLow, uint32_t AWDThresholdValue)

Function description

Set ADC analog watchdog threshold value of threshold high or low.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  - LL_ADC_AWD1
  - LL_ADC_AWD2
  - LL_ADC_AWD3
- **AWDThresholdsHighLow:** This parameter can be one of the following values:
  - LL_ADC_AWD_THRESHOLD_HIGH
  - LL_ADC_AWD_THRESHOLD_LOW
- **AWDThresholdValue:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values

- **None:**

Notes

- In case of ADC resolution different of 12 bits, analog watchdog thresholds data require a specific shift. Use helper macro __LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION().
- On this STM32 series, there are 2 kinds of analog watchdog instance: AWD standard (instance AWD1): channels monitored: can monitor 1 channel or all channels.groups monitored: ADC groups regular and-or injected.resolution: resolution is not limited (corresponds to ADC resolution configured). AWD flexible (instances AWD2, AWD3): channels monitored: flexible on channels monitored, selection is channel wise, from from 1 to all channels. Specificity of this analog watchdog: Multiple channels can be selected. For example: (LL_ADC_AWD_CHANNEL4_REG_INJ | LL_ADC_AWD_CHANNEL5_REG_INJ | ...)groups monitored: not selection possible (monitoring on both groups regular and injected). Channels selected are monitored on groups regular and injected: LL_ADC_AWD_CHANNELxx_REG_INJ (do not use parameters LL_ADC_AWD_CHANNELxx_REG and LL_ADC_AWD_CHANNELxx_INJ)resolution: resolution is limited to 8 bits: if ADC resolution is 12 bits the 4 LSB are ignored, if ADC resolution is 10 bits the 2 LSB are ignored.
- If ADC oversampling is enabled, ADC analog watchdog thresholds are impacted: the comparison of analog watchdog thresholds is done on oversampling intermediate computation (after ratio, before shift application): intermediate register bitfield [32:7] (26 most significant bits).
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either ADC groups regular or injected.

Reference Manual to LL API cross reference:

- TR1 HT1 LL_ADC_SetAnalogWDThresholds
- TR2 HT2 LL_ADC_SetAnalogWDThresholds
- TR3 HT3 LL_ADC_SetAnalogWDThresholds
- TR1 LT1 LL_ADC_SetAnalogWDThresholds
- TR2 LT2 LL_ADC_SetAnalogWDThresholds
- TR3 LT3 LL_ADC_SetAnalogWDThresholds

LL_ADC_GetAnalogWDThresholds

Function name

__STATIC_INLINE uint32_t LL_ADC_GetAnalogWDThresholds (ADC_TypeDef * ADCx, uint32_t AWDy, uint32_t AWDThresholdsHighLow)

Function description

Get ADC analog watchdog threshold value of threshold high, threshold low or raw data with ADC thresholds high and low concatenated.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  - LL_ADC_AWD1
  - LL_ADC_AWD2
  - LL_ADC_AWD3
- **AWDThresholdsHighLow:** This parameter can be one of the following values:
  - LL_ADC_AWD_THRESHOLD_HIGH
  - LL_ADC_AWD_THRESHOLD_LOW

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0x3FFFFFF

Notes

- In case of ADC resolution different of 12 bits, analog watchdog thresholds data require a specific shift. Use helper macro __LL_ADC_ANALOGWD_GET_THRESHOLD_RESOLUTION().

Reference Manual to LL API cross reference:

- TR1 HT1 LL_ADC_GetAnalogWDThresholds
- TR2 HT2 LL_ADC_GetAnalogWDThresholds
- TR3 HT3 LL_ADC_GetAnalogWDThresholds
- TR1 LT1 LL_ADC_GetAnalogWDThresholds
- TR2 LT2 LL_ADC_GetAnalogWDThresholds
- TR3 LT3 LL_ADC_GetAnalogWDThresholds

LL_ADC_SetOverSamplingScope

Function name

__STATIC_INLINE void LL_ADC_SetOverSamplingScope (ADC_TypeDef * ADCx, uint32_t OvsScope)

Function description

Set ADC oversampling scope: ADC groups regular and-or injected (availability of ADC group injected depends on STM32 families).

Parameters

- **ADCx:** ADC instance
- **OvsScope:** This parameter can be one of the following values:
  - LL_ADC_OVS_DISABLE
  - LL_ADC_OVS_GRP_REGULAR_CONTINUED
  - LL_ADC_OVS_GRP_REGULAR_RESUMED
  - LL_ADC_OVS_GRP_INJECTED
  - LL_ADC_OVS_GRP_INJ_REG_RESUMED

Return values

- **None:**

Notes

- If both groups regular and injected are selected, specify behavior of ADC group injected interrupting group regular: when ADC group injected is triggered, the oversampling on ADC group regular is either temporary stopped and continued, or resumed from start (oversampler buffer reset).
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR2 ROVSE LL_ADC_SetOverSamplingScope
- CFGR2 JOVSE LL_ADC_SetOverSamplingScope
- CFGR2 ROVSM LL_ADC_SetOverSamplingScope

LL_ADC_GetOverSamplingScope

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOverSamplingScope (ADC_TypeDef * ADCx)

Function description

Get ADC oversampling scope: ADC groups regular and-or injected (availability of ADC group injected depends on STM32 families).

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_OVS_DISABLE
  - LL_ADC_OVS_GRP_REGULAR_CONTINUED
  - LL_ADC_OVS_GRP_REGULAR_RESUMED
  - LL_ADC_OVS_GRP_INJECTED
  - LL_ADC_OVS_GRP_INJ_REG_RESUMED

Notes

- If both groups regular and injected are selected, specify behavior of ADC group injected interrupting group regular: when ADC group injected is triggered, the oversampling on ADC group regular is either temporary stopped and continued, or resumed from start (oversampler buffer reset).

Reference Manual to LL API cross reference:

- CFGR2 ROVSE LL_ADC_GetOverSamplingScope
- CFGR2 JOVSE LL_ADC_GetOverSamplingScope
- CFGR2 ROVSM LL_ADC_GetOverSamplingScope

LL_ADC_SetOverSamplingDiscont

Function name

__STATIC_INLINE void LL_ADC_SetOverSamplingDiscont (ADC_TypeDef * ADCx, uint32_t OverSamplingDiscont)

Function description

Set ADC oversampling discontinuous mode (triggered mode) on the selected ADC group.

Parameters

- **ADCx:** ADC instance
- **OverSamplingDiscont:** This parameter can be one of the following values:
  - LL_ADC_OVS_REG_CONT
  - LL_ADC_OVS_REG_DISCONT

Return values

- **None:**

Notes

- Number of oversampled conversions are done either in: continuous mode (all conversions of oversampling ratio are done from 1 trigger)discontinuous mode (each conversion of oversampling ratio needs a trigger)
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
- On this STM32 series, oversampling discontinuous mode (triggered mode) can be used only when oversampling is set on group regular only and in resumed mode.

Reference Manual to LL API cross reference:

- CFGR2 TROVS LL_ADC_SetOverSamplingDiscont

LL_ADC_GetOverSamplingDiscont

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOverSamplingDiscont (ADC_TypeDef * ADCx)

Function description

Get ADC oversampling discontinuous mode (triggered mode) on the selected ADC group.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_OVS_REG_CONT
  - LL_ADC_OVS_REG_DISCONT

Notes

- Number of oversampled conversions are done either in: continuous mode (all conversions of oversampling ratio are done from 1 trigger)discontinuous mode (each conversion of oversampling ratio needs a trigger)

Reference Manual to LL API cross reference:

- CFGR2 TROVS LL_ADC_GetOverSamplingDiscont

LL_ADC_ConfigOverSamplingRatioShift

Function name

__STATIC_INLINE void LL_ADC_ConfigOverSamplingRatioShift (ADC_TypeDef * ADCx, uint32_t Ratio, uint32_t Shift)

Function description

Set ADC oversampling (impacting both ADC groups regular and injected)

Parameters

- **ADCx:** ADC instance
- **Ratio:** This parameter can be in the range from 1 to 1024. In the case of ADC3 can be one of the following values:
  - LL_ADC_OVS_RATIO_2
  - LL_ADC_OVS_RATIO_4
  - LL_ADC_OVS_RATIO_8
  - LL_ADC_OVS_RATIO_16
  - LL_ADC_OVS_RATIO_32
  - LL_ADC_OVS_RATIO_64
  - LL_ADC_OVS_RATIO_128
  - LL_ADC_OVS_RATIO_256
- **Shift:** This parameter can be one of the following values:
  - LL_ADC_OVS_SHIFT_NONE
  - LL_ADC_OVS_SHIFT_RIGHT_1
  - LL_ADC_OVS_SHIFT_RIGHT_2
  - LL_ADC_OVS_SHIFT_RIGHT_3
  - LL_ADC_OVS_SHIFT_RIGHT_4
  - LL_ADC_OVS_SHIFT_RIGHT_5
  - LL_ADC_OVS_SHIFT_RIGHT_6
  - LL_ADC_OVS_SHIFT_RIGHT_7
  - LL_ADC_OVS_SHIFT_RIGHT_8
  - LL_ADC_OVS_SHIFT_RIGHT_9
  - LL_ADC_OVS_SHIFT_RIGHT_10
  - LL_ADC_OVS_SHIFT_RIGHT_11

Return values

- **None:**

Notes

- This function set the 2 items of oversampling configuration: ratioshift
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR2 OVSS LL_ADC_ConfigOverSamplingRatioShift
- CFGR2 OVSR LL_ADC_ConfigOverSamplingRatioShift

LL_ADC_GetOverSamplingRatio

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOverSamplingRatio (ADC_TypeDef * ADCx)

Function description

Get ADC oversampling ratio (impacting both ADC groups regular and injected)

Parameters

- **ADCx:** ADC instance

Return values

- **Ratio:** This parameter can be in the from 1 to 1024. In the case of ADC3 can be one of the following values:
  - LL_ADC_OVS_RATIO_2
  - LL_ADC_OVS_RATIO_4
  - LL_ADC_OVS_RATIO_8
  - LL_ADC_OVS_RATIO_16
  - LL_ADC_OVS_RATIO_32
  - LL_ADC_OVS_RATIO_64
  - LL_ADC_OVS_RATIO_128
  - LL_ADC_OVS_RATIO_256

Reference Manual to LL API cross reference:

- CFGR2 OVSR LL_ADC_GetOverSamplingRatio

LL_ADC_GetOverSamplingShift

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOverSamplingShift (ADC_TypeDef * ADCx)

Function description

Get ADC oversampling shift (impacting both ADC groups regular and injected)

Parameters

- **ADCx:** ADC instance

Return values

- **Shift:** This parameter can be one of the following values:
  - LL_ADC_OVS_SHIFT_NONE
  - LL_ADC_OVS_SHIFT_RIGHT_1
  - LL_ADC_OVS_SHIFT_RIGHT_2
  - LL_ADC_OVS_SHIFT_RIGHT_3
  - LL_ADC_OVS_SHIFT_RIGHT_4
  - LL_ADC_OVS_SHIFT_RIGHT_5
  - LL_ADC_OVS_SHIFT_RIGHT_6
  - LL_ADC_OVS_SHIFT_RIGHT_7
  - LL_ADC_OVS_SHIFT_RIGHT_8
  - LL_ADC_OVS_SHIFT_RIGHT_9
  - LL_ADC_OVS_SHIFT_RIGHT_10
  - LL_ADC_OVS_SHIFT_RIGHT_11

Reference Manual to LL API cross reference:

- CFGR2 OVSS LL_ADC_GetOverSamplingShift

LL_ADC_SetBoostMode

Function name

__STATIC_INLINE void LL_ADC_SetBoostMode (ADC_TypeDef * ADCx, uint32_t BoostMode)

Function description

Set ADC boost mode.

Parameters

- **ADCx:** ADC instance
- **BoostMode:** This parameter can be one of the following values:
  - LL_ADC_BOOST_MODE_6MHZ25
  - LL_ADC_BOOST_MODE_12MHZ5
  - LL_ADC_BOOST_MODE_20MHZ
  - LL_ADC_BOOST_MODE_25MHZ
  - LL_ADC_BOOST_MODE_50MHZ

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC boost must be configured, without calibration on going, without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CR BOOST LL_ADC_SetBoostMode

LL_ADC_GetBoostMode

Function name

__STATIC_INLINE uint32_t LL_ADC_GetBoostMode (ADC_TypeDef * ADCx)

Function description

Get ADC boost mode.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** Boost disabled 1: Boost enabled

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC boost must be configured, without calibration on going, without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CR BOOST LL_ADC_GetBoostMode

LL_ADC_SetMultimode

Function name

__STATIC_INLINE void LL_ADC_SetMultimode (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t Multimode)

Function description

Set ADC multimode configuration to operate in independent mode or multimode (for devices with several ADC instances).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **Multimode:** This parameter can be one of the following values:
  - LL_ADC_MULTI_INDEPENDENT
  - LL_ADC_MULTI_DUAL_REG_SIMULT
  - LL_ADC_MULTI_DUAL_REG_INTERL
  - LL_ADC_MULTI_DUAL_INJ_SIMULT
  - LL_ADC_MULTI_DUAL_INJ_ALTERN
  - LL_ADC_MULTI_DUAL_REG_SIM_INJ_SIM
  - LL_ADC_MULTI_DUAL_REG_SIM_INJ_ALT
  - LL_ADC_MULTI_DUAL_REG_INT_INJ_SIM

Return values

- **None:**

Notes

- If multimode configuration: the selected ADC instance is either master or slave depending on hardware. Refer to reference manual.
- On this STM32 series, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled. This check can be done with function LL_ADC_IsEnabled() for each ADC instance or by using helper macro __LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE().

Reference Manual to LL API cross reference:

- CCR DUAL LL_ADC_SetMultimode

LL_ADC_GetMultimode

Function name

__STATIC_INLINE uint32_t LL_ADC_GetMultimode (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get ADC multimode configuration to operate in independent mode or multimode (for devices with several ADC instances).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_MULTI_INDEPENDENT
  - LL_ADC_MULTI_DUAL_REG_SIMULT
  - LL_ADC_MULTI_DUAL_REG_INTERL
  - LL_ADC_MULTI_DUAL_INJ_SIMULT
  - LL_ADC_MULTI_DUAL_INJ_ALTERN
  - LL_ADC_MULTI_DUAL_REG_SIM_INJ_SIM
  - LL_ADC_MULTI_DUAL_REG_SIM_INJ_ALT
  - LL_ADC_MULTI_DUAL_REG_INT_INJ_SIM

Notes

- If multimode configuration: the selected ADC instance is either master or slave depending on hardware. Refer to reference manual.

Reference Manual to LL API cross reference:

- CCR DUAL LL_ADC_GetMultimode

LL_ADC_SetMultiDMATransfer

Function name

__STATIC_INLINE void LL_ADC_SetMultiDMATransfer (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t MultiDMATransfer)

Function description

Set ADC multimode conversion data transfer: no transfer or transfer by DMA.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **MultiDMATransfer:** This parameter can be one of the following values:
  - LL_ADC_MULTI_REG_DMA_EACH_ADC
  - LL_ADC_MULTI_REG_DMA_RES_32_10B
  - LL_ADC_MULTI_REG_DMA_RES_8B

Return values

- **None:**

Notes

- If ADC multimode transfer by DMA is not selected: each ADC uses its own DMA channel, with its individual DMA transfer settings. If ADC multimode transfer by DMA is selected: One DMA channel is used for both ADC (DMA of ADC master) Specifies the DMA requests mode: Limited mode (One shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.Unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transfers (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular.
- If ADC DMA requests mode is set to unlimited and DMA is set to mode non-circular: when DMA transfers size will be reached, DMA will stop transfers of ADC conversions data ADC will raise an overrun error (overrun flag and interruption if enabled).
- How to retrieve multimode conversion data: Whatever multimode transfer by DMA setting: using function LL_ADC_REG_ReadMultiConversionData32(). If ADC multimode transfer by DMA is selected: conversion data is a raw data with ADC master and slave concatenated. A macro is available to get the conversion data of ADC master or ADC slave: see helper macro __LL_ADC_MULTI_CONV_DATA_MASTER_SLAVE().
- On this STM32 series, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CCR DAMDF LL_ADC_GetMultiDMATransfer
-

LL_ADC_GetMultiDMATransfer

Function name

__STATIC_INLINE uint32_t LL_ADC_GetMultiDMATransfer (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get ADC multimode conversion data transfer: no transfer or transfer by DMA.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_MULTI_REG_DMA_EACH_ADC
  - LL_ADC_MULTI_REG_DMA_RES_32_10B
  - LL_ADC_MULTI_REG_DMA_RES_8B

Notes

- If ADC multimode transfer by DMA is not selected: each ADC uses its own DMA channel, with its individual DMA transfer settings. If ADC multimode transfer by DMA is selected: One DMA channel is used for both ADC (DMA of ADC master) Specifies the DMA requests mode: Limited mode (One shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.Unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transfers (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular.
- If ADC DMA requests mode is set to unlimited and DMA is set to mode non-circular: when DMA transfers size will be reached, DMA will stop transfers of ADC conversions data ADC will raise an overrun error (overrun flag and interruption if enabled).
- How to retrieve multimode conversion data: Whatever multimode transfer by DMA setting: using function LL_ADC_REG_ReadMultiConversionData32(). If ADC multimode transfer by DMA is selected: conversion data is a raw data with ADC master and slave concatenated. A macro is available to get the conversion data of ADC master or ADC slave: see helper macro __LL_ADC_MULTI_CONV_DATA_MASTER_SLAVE().

Reference Manual to LL API cross reference:

- CCR DAMDF LL_ADC_GetMultiDMATransfer
-

LL_ADC_SetMultiTwoSamplingDelay

Function name

__STATIC_INLINE void LL_ADC_SetMultiTwoSamplingDelay (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t MultiTwoSamplingDelay)

Function description

Set ADC multimode delay between 2 sampling phases.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **MultiTwoSamplingDelay:** This parameter can be one of the following values:
  - LL_ADC_MULTI_TWOSMP_DELAY_1CYCLE_5
  - LL_ADC_MULTI_TWOSMP_DELAY_2CYCLES_5
  - LL_ADC_MULTI_TWOSMP_DELAY_3CYCLES_5
  - LL_ADC_MULTI_TWOSMP_DELAY_4CYCLES_5 (1)
  - LL_ADC_MULTI_TWOSMP_DELAY_4CYCLES_5_8_BITS
  - LL_ADC_MULTI_TWOSMP_DELAY_5CYCLES_5 (2)
  - LL_ADC_MULTI_TWOSMP_DELAY_5CYCLES_5_10_BITS
  - LL_ADC_MULTI_TWOSMP_DELAY_6CYCLES (3)
  - LL_ADC_MULTI_TWOSMP_DELAY_6CYCLES_5 (4)
  - LL_ADC_MULTI_TWOSMP_DELAY_6CYCLES_5_12_BITS
  - LL_ADC_MULTI_TWOSMP_DELAY_7CYCLES_5 (5)
  - LL_ADC_MULTI_TWOSMP_DELAY_8CYCLES (6)
  - LL_ADC_MULTI_TWOSMP_DELAY_9CYCLES (7)

  (1) Parameter available only if ADC resolution is 16, 14, 12 or 10 bits. (2) Parameter available only if ADC resolution is 16, 14 or 12 bits. (3) Parameter available only if ADC resolution is 10 or 8 bits. (4) Parameter available only if ADC resolution is 16 or 14 bits. (5) Parameter available only if ADC resolution is 16 bits. (6) Parameter available only if ADC resolution is 12 bits. (7) Parameter available only if ADC resolution is 16 or 14 bits.

Return values

- **None:**

Notes

- The sampling delay range depends on ADC resolution: ADC resolution 12 bits can have maximum delay of 12 cycles.ADC resolution 10 bits can have maximum delay of 10 cycles.ADC resolution 8 bits can have maximum delay of 8 cycles.ADC resolution 6 bits can have maximum delay of 6 cycles.
- On this STM32 series, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled. This check can be done with function LL_ADC_IsEnabled() for each ADC instance or by using helper macro helper macro __LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE().

Reference Manual to LL API cross reference:

- CCR DELAY LL_ADC_SetMultiTwoSamplingDelay

LL_ADC_GetMultiTwoSamplingDelay

Function name

__STATIC_INLINE uint32_t LL_ADC_GetMultiTwoSamplingDelay (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get ADC multimode delay between 2 sampling phases.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_MULTI_TWOSMP_DELAY_1CYCLE_5
  - LL_ADC_MULTI_TWOSMP_DELAY_2CYCLES_5
  - LL_ADC_MULTI_TWOSMP_DELAY_3CYCLES_5
  - LL_ADC_MULTI_TWOSMP_DELAY_4CYCLES_5 (1)
  - LL_ADC_MULTI_TWOSMP_DELAY_4CYCLES_5_8_BITS
  - LL_ADC_MULTI_TWOSMP_DELAY_5CYCLES_5 (2)
  - LL_ADC_MULTI_TWOSMP_DELAY_5CYCLES_5_10_BITS
  - LL_ADC_MULTI_TWOSMP_DELAY_6CYCLES (3)
  - LL_ADC_MULTI_TWOSMP_DELAY_6CYCLES_5 (4)
  - LL_ADC_MULTI_TWOSMP_DELAY_6CYCLES_5_12_BITS
  - LL_ADC_MULTI_TWOSMP_DELAY_7CYCLES_5 (5)
  - LL_ADC_MULTI_TWOSMP_DELAY_8CYCLES (6)
  - LL_ADC_MULTI_TWOSMP_DELAY_9CYCLES (7)

  (1) Parameter available only if ADC resolution is 16, 14, 12 or 10 bits. (2) Parameter available only if ADC resolution is 16, 14 or 12 bits. (3) Parameter available only if ADC resolution is 10 or 8 bits. (4) Parameter available only if ADC resolution is 16 or 14 bits. (5) Parameter available only if ADC resolution is 16 bits. (6) Parameter available only if ADC resolution is 12 bits. (7) Parameter available only if ADC resolution is 16 or 14 bits.

Reference Manual to LL API cross reference:

- CCR DELAY LL_ADC_GetMultiTwoSamplingDelay

LL_ADC_EnableDeepPowerDown

Function name

__STATIC_INLINE void LL_ADC_EnableDeepPowerDown (ADC_TypeDef * ADCx)

Function description

Put ADC instance in deep power down state.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- In case of ADC calibration necessary: When ADC is in deep-power-down state, the internal analog calibration is lost. After exiting from deep power down, calibration must be relaunched or calibration factor (preliminarily saved) must be set back into calibration register.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.

Reference Manual to LL API cross reference:

- CR DEEPPWD LL_ADC_EnableDeepPowerDown

LL_ADC_DisableDeepPowerDown

Function name

__STATIC_INLINE void LL_ADC_DisableDeepPowerDown (ADC_TypeDef * ADCx)

Function description

Disable ADC deep power down mode.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- In case of ADC calibration necessary: When ADC is in deep-power-down state, the internal analog calibration is lost. After exiting from deep power down, calibration must be relaunched or calibration factor (preliminarily saved) must be set back into calibration register.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.

Reference Manual to LL API cross reference:

- CR DEEPPWD LL_ADC_DisableDeepPowerDown

LL_ADC_IsDeepPowerDownEnabled

Function name

__STATIC_INLINE uint32_t LL_ADC_IsDeepPowerDownEnabled (ADC_TypeDef * ADCx)

Function description

Get the selected ADC instance deep power down state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** deep power down is disabled, 1: deep power down is enabled.

Reference Manual to LL API cross reference:

- CR DEEPPWD LL_ADC_IsDeepPowerDownEnabled

LL_ADC_EnableInternalRegulator

Function name

__STATIC_INLINE void LL_ADC_EnableInternalRegulator (ADC_TypeDef * ADCx)

Function description

Enable ADC instance internal voltage regulator.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, after ADC internal voltage regulator enable, a delay for ADC internal voltage regulator stabilization is required before performing a ADC calibration or ADC enable. Refer to device datasheet, parameter tADCVREG_STUP. Refer to literal LL_ADC_DELAY_INTERNAL_REGUL_STAB_US.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.

Reference Manual to LL API cross reference:

- CR ADVREGEN LL_ADC_EnableInternalRegulator

LL_ADC_DisableInternalRegulator

Function name

__STATIC_INLINE void LL_ADC_DisableInternalRegulator (ADC_TypeDef * ADCx)

Function description

Disable ADC internal voltage regulator.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.

Reference Manual to LL API cross reference:

- CR ADVREGEN LL_ADC_DisableInternalRegulator

LL_ADC_IsInternalRegulatorEnabled

Function name

__STATIC_INLINE uint32_t LL_ADC_IsInternalRegulatorEnabled (ADC_TypeDef * ADCx)

Function description

Get the selected ADC instance internal voltage regulator state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** internal regulator is disabled, 1: internal regulator is enabled.

Reference Manual to LL API cross reference:

- CR ADVREGEN LL_ADC_IsInternalRegulatorEnabled

LL_ADC_Enable

Function name

__STATIC_INLINE void LL_ADC_Enable (ADC_TypeDef * ADCx)

Function description

Enable the selected ADC instance.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, after ADC enable, a delay for ADC internal analog stabilization is required before performing a ADC conversion start. Refer to device datasheet, parameter tSTAB.
- On this STM32 series, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be ADC disabled and ADC internal voltage regulator enabled.

Reference Manual to LL API cross reference:

- CR ADEN LL_ADC_Enable

LL_ADC_Disable

Function name

__STATIC_INLINE void LL_ADC_Disable (ADC_TypeDef * ADCx)

Function description

Disable the selected ADC instance.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be not disabled. Must be enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CR ADDIS LL_ADC_Disable

LL_ADC_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabled (ADC_TypeDef * ADCx)

Function description

Get the selected ADC instance enable state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** ADC is disabled, 1: ADC is enabled.

Notes

- On this STM32 series, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)

Reference Manual to LL API cross reference:

- CR ADEN LL_ADC_IsEnabled

LL_ADC_IsDisableOngoing

Function name

__STATIC_INLINE uint32_t LL_ADC_IsDisableOngoing (ADC_TypeDef * ADCx)

Function description

Get the selected ADC instance disable state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** no ADC disable command on going.

Reference Manual to LL API cross reference:

- CR ADDIS LL_ADC_IsDisableOngoing

LL_ADC_StartCalibration

Function name

__STATIC_INLINE void LL_ADC_StartCalibration (ADC_TypeDef * ADCx, uint32_t CalibrationMode, uint32_t SingleDiff)

Function description

Start ADC calibration in the mode single-ended or differential (for devices with differential mode available).

Parameters

- **ADCx:** ADC instance
- **CalibrationMode:** This parameter can be one of the following values:
  - LL_ADC_CALIB_OFFSET
  - LL_ADC_CALIB_OFFSET_LINEARITY
- **SingleDiff:** This parameter can be one of the following values:
  - LL_ADC_SINGLE_ENDED
  - LL_ADC_DIFFERENTIAL_ENDED

Return values

- **None:**

Notes

- On this STM32 series, a minimum number of ADC clock cycles are required between ADC end of calibration and ADC enable. Refer to literal LL_ADC_DELAY_CALIB_ENABLE_ADC_CYCLES.
- Calibration duration: Calibration of offset: 520 ADC clock cyclesCalibration of linearity: 131072 ADC clock cycles
- For devices with differential mode available: Calibration of offset is specific to each of single-ended and differential modes (calibration run must be performed for each of these differential modes, if used afterwards and if the application requires their calibration). Calibration of linearity is common to both single-ended and differential modes (calibration run can be performed only once).
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.

Reference Manual to LL API cross reference:

- CR ADCAL LL_ADC_StartCalibration
- CR ADCALDIF LL_ADC_StartCalibration
- CR ADCALLIN LL_ADC_StartCalibration

LL_ADC_IsCalibrationOnGoing

Function name

__STATIC_INLINE uint32_t LL_ADC_IsCalibrationOnGoing (ADC_TypeDef * ADCx)

Function description

Get ADC calibration state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** calibration complete, 1: calibration in progress.

Reference Manual to LL API cross reference:

- CR ADCAL LL_ADC_IsCalibrationOnGoing

LL_ADC_REG_StartConversion

Function name

__STATIC_INLINE void LL_ADC_REG_StartConversion (ADC_TypeDef * ADCx)

Function description

Start ADC group regular conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, this function is relevant for both internal trigger (SW start) and external trigger: If ADC trigger has been set to software start, ADC conversion starts immediately.If ADC trigger has been set to external trigger, ADC conversion will start at next trigger event (on the selected trigger edge) following the ADC start conversion command.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be enabled without conversion on going on group regular, without conversion stop command on going on group regular, without ADC disable command on going.

Reference Manual to LL API cross reference:

- CR ADSTART LL_ADC_REG_StartConversion

LL_ADC_REG_StopConversion

Function name

__STATIC_INLINE void LL_ADC_REG_StopConversion (ADC_TypeDef * ADCx)

Function description

Stop ADC group regular conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be enabled with conversion on going on group regular, without ADC disable command on going.

Reference Manual to LL API cross reference:

- CR ADSTP LL_ADC_REG_StopConversion

LL_ADC_REG_IsConversionOngoing

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_IsConversionOngoing (ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** no conversion is on going on ADC group regular.

Reference Manual to LL API cross reference:

- CR ADSTART LL_ADC_REG_IsConversionOngoing

LL_ADC_REG_IsStopConversionOngoing

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_IsStopConversionOngoing (ADC_TypeDef * ADCx)

Function description

Get ADC group regular command of conversion stop state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** no command of conversion stop is on going on ADC group regular.

Reference Manual to LL API cross reference:

- CR ADSTP LL_ADC_REG_IsStopConversionOngoing

LL_ADC_REG_ReadConversionData32

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_ReadConversionData32 (ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data, range fit for all ADC configurations: all ADC resolutions and all oversampling increased data width (for devices with feature oversampling).

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_REG_ReadConversionData32

LL_ADC_REG_ReadConversionData16

Function name

__STATIC_INLINE uint16_t LL_ADC_REG_ReadConversionData16 (ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data, range fit for ADC resolution 16 bits.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFFFF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_REG_ReadConversionData16

LL_ADC_REG_ReadConversionData14

Function name

__STATIC_INLINE uint16_t LL_ADC_REG_ReadConversionData14 (ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data, range fit for ADC resolution 14 bits.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x3FF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_REG_ReadConversionData14

LL_ADC_REG_ReadConversionData12

Function name

__STATIC_INLINE uint16_t LL_ADC_REG_ReadConversionData12 (ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data, range fit for ADC resolution 12 bits.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0xFFF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_REG_ReadConversionData12

LL_ADC_REG_ReadConversionData10

Function name

__STATIC_INLINE uint16_t LL_ADC_REG_ReadConversionData10 (ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data, range fit for ADC resolution 10 bits.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0x3FF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_REG_ReadConversionData10

LL_ADC_REG_ReadConversionData8

Function name

__STATIC_INLINE uint8_t LL_ADC_REG_ReadConversionData8 (ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data, range fit for ADC resolution 8 bits.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_REG_ReadConversionData8

LL_ADC_REG_ReadMultiConversionData32

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_ReadMultiConversionData32 (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t ConversionData)

Function description

Get ADC multimode conversion data of ADC master, ADC slave or raw data with ADC master and slave concatenated.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **ConversionData:** This parameter can be one of the following values:
  - LL_ADC_MULTI_MASTER
  - LL_ADC_MULTI_SLAVE
  - LL_ADC_MULTI_MASTER_SLAVE

Return values

- **Value:** between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Notes

- If raw data with ADC master and slave concatenated is retrieved, a macro is available to get the conversion data of ADC master or ADC slave: see helper macro __LL_ADC_MULTI_CONV_DATA_MASTER_SLAVE(). (however this macro is mainly intended for multimode transfer by DMA, because this function can do the same by getting multimode conversion data of ADC master or ADC slave separately).

Reference Manual to LL API cross reference:

- CDR RDATA_MST LL_ADC_REG_ReadMultiConversionData32
- CDR RDATA_SLV LL_ADC_REG_ReadMultiConversionData32

LL_ADC_INJ_StartConversion

Function name

__STATIC_INLINE void LL_ADC_INJ_StartConversion (ADC_TypeDef * ADCx)

Function description

Start ADC group injected conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, this function is relevant for both internal trigger (SW start) and external trigger: If ADC trigger has been set to software start, ADC conversion starts immediately.If ADC trigger has been set to external trigger, ADC conversion will start at next trigger event (on the selected trigger edge) following the ADC start conversion command.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be enabled without conversion on going on group injected, without conversion stop command on going on group injected, without ADC disable command on going.

Reference Manual to LL API cross reference:

- CR JADSTART LL_ADC_INJ_StartConversion

LL_ADC_INJ_StopConversion

Function name

__STATIC_INLINE void LL_ADC_INJ_StopConversion (ADC_TypeDef * ADCx)

Function description

Stop ADC group injected conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be enabled with conversion on going on group injected, without ADC disable command on going.

Reference Manual to LL API cross reference:

- CR JADSTP LL_ADC_INJ_StopConversion

LL_ADC_INJ_IsConversionOngoing

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_IsConversionOngoing (ADC_TypeDef * ADCx)

Function description

Get ADC group injected conversion state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** no conversion is on going on ADC group injected.

Reference Manual to LL API cross reference:

- CR JADSTART LL_ADC_INJ_IsConversionOngoing

LL_ADC_INJ_IsStopConversionOngoing

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_IsStopConversionOngoing (ADC_TypeDef * ADCx)

Function description

Get ADC group injected command of conversion stop state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** no command of conversion stop is on going on ADC group injected.

Reference Manual to LL API cross reference:

- CR JADSTP LL_ADC_INJ_IsStopConversionOngoing

LL_ADC_INJ_ReadConversionData32

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_ReadConversionData32 (ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected conversion data, range fit for all ADC configurations: all ADC resolutions and all oversampling increased data width (for devices with feature oversampling).

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values

- **Value:** between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData32
- JDR2 JDATA LL_ADC_INJ_ReadConversionData32
- JDR3 JDATA LL_ADC_INJ_ReadConversionData32
- JDR4 JDATA LL_ADC_INJ_ReadConversionData32

LL_ADC_INJ_ReadConversionData16

Function name

__STATIC_INLINE uint16_t LL_ADC_INJ_ReadConversionData16 (ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected conversion data, range fit for ADC resolution 16 bits.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0xFFFF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData16
- JDR2 JDATA LL_ADC_INJ_ReadConversionData16
- JDR3 JDATA LL_ADC_INJ_ReadConversionData16
- JDR4 JDATA LL_ADC_INJ_ReadConversionData16

LL_ADC_INJ_ReadConversionData14

Function name

__STATIC_INLINE uint16_t LL_ADC_INJ_ReadConversionData14 (ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected conversion data, range fit for ADC resolution 14 bits.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0x3FFF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData14
- JDR2 JDATA LL_ADC_INJ_ReadConversionData14
- JDR3 JDATA LL_ADC_INJ_ReadConversionData14
- JDR4 JDATA LL_ADC_INJ_ReadConversionData14

LL_ADC_INJ_ReadConversionData12

Function name

__STATIC_INLINE uint16_t LL_ADC_INJ_ReadConversionData12 (ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected conversion data, range fit for ADC resolution 12 bits.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0xFFF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData12
- JDR2 JDATA LL_ADC_INJ_ReadConversionData12
- JDR3 JDATA LL_ADC_INJ_ReadConversionData12
- JDR4 JDATA LL_ADC_INJ_ReadConversionData12

LL_ADC_INJ_ReadConversionData10

Function name

__STATIC_INLINE uint16_t LL_ADC_INJ_ReadConversionData10 (ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected conversion data, range fit for ADC resolution 10 bits.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0x3FF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData10
- JDR2 JDATA LL_ADC_INJ_ReadConversionData10
- JDR3 JDATA LL_ADC_INJ_ReadConversionData10
- JDR4 JDATA LL_ADC_INJ_ReadConversionData10

LL_ADC_INJ_ReadConversionData8

Function name

__STATIC_INLINE uint8_t LL_ADC_INJ_ReadConversionData8 (ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected conversion data, range fit for ADC resolution 8 bits.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData8
- JDR2 JDATA LL_ADC_INJ_ReadConversionData8
- JDR3 JDATA LL_ADC_INJ_ReadConversionData8
- JDR4 JDATA LL_ADC_INJ_ReadConversionData8

LL_ADC_IsActiveFlag_ADRDY

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_ADRDY (ADC_TypeDef * ADCx)

Function description

Get flag ADC ready.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Notes

- On this STM32 series, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)

Reference Manual to LL API cross reference:

- ISR ADRDY LL_ADC_IsActiveFlag_ADRDY

LL_ADC_IsActiveFlag_EOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOC (ADC_TypeDef * ADCx)

Function description

Get flag ADC group regular end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR EOC LL_ADC_IsActiveFlag_EOC

LL_ADC_IsActiveFlag_EOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOS (ADC_TypeDef * ADCx)

Function description

Get flag ADC group regular end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR EOS LL_ADC_IsActiveFlag_EOS

LL_ADC_IsActiveFlag_OVR

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_OVR (ADC_TypeDef * ADCx)

Function description

Get flag ADC group regular overrun.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR OVR LL_ADC_IsActiveFlag_OVR

LL_ADC_IsActiveFlag_EOSMP

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOSMP (ADC_TypeDef * ADCx)

Function description

Get flag ADC group regular end of sampling phase.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR EOSMP LL_ADC_IsActiveFlag_EOSMP

LL_ADC_IsActiveFlag_JEOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_JEOC (ADC_TypeDef * ADCx)

Function description

Get flag ADC group injected end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR JEOC LL_ADC_IsActiveFlag_JEOC

LL_ADC_IsActiveFlag_JEOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_JEOS (ADC_TypeDef * ADCx)

Function description

Get flag ADC group injected end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR JEOS LL_ADC_IsActiveFlag_JEOS

LL_ADC_IsActiveFlag_JQOVF

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_JQOVF (ADC_TypeDef * ADCx)

Function description

Get flag ADC group injected contexts queue overflow.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR JQOVF LL_ADC_IsActiveFlag_JQOVF

LL_ADC_IsActiveFlag_LDORDY

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_LDORDY (ADC_TypeDef * ADCx)

Function description

Get flag ADC LDO output voltage ready bit.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR LDORDY LL_ADC_IsActiveFlag_LDORDY

LL_ADC_IsActiveFlag_AWD1

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_AWD1 (ADC_TypeDef * ADCx)

Function description

Get flag ADC analog watchdog 1 flag.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR AWD1 LL_ADC_IsActiveFlag_AWD1

LL_ADC_IsActiveFlag_AWD2

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_AWD2 (ADC_TypeDef * ADCx)

Function description

Get flag ADC analog watchdog 2.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR AWD2 LL_ADC_IsActiveFlag_AWD2

LL_ADC_IsActiveFlag_AWD3

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_AWD3 (ADC_TypeDef * ADCx)

Function description

Get flag ADC analog watchdog 3.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR AWD3 LL_ADC_IsActiveFlag_AWD3

LL_ADC_ClearFlag_ADRDY

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_ADRDY (ADC_TypeDef * ADCx)

Function description

Clear flag ADC ready.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)

Reference Manual to LL API cross reference:

- ISR ADRDY LL_ADC_ClearFlag_ADRDY

LL_ADC_ClearFlag_EOC

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_EOC (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group regular end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR EOC LL_ADC_ClearFlag_EOC

LL_ADC_ClearFlag_EOS

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_EOS (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group regular end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR EOS LL_ADC_ClearFlag_EOS

LL_ADC_ClearFlag_OVR

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_OVR (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group regular overrun.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR OVR LL_ADC_ClearFlag_OVR

LL_ADC_ClearFlag_EOSMP

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_EOSMP (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group regular end of sampling phase.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR EOSMP LL_ADC_ClearFlag_EOSMP

LL_ADC_ClearFlag_JEOC

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_JEOC (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group injected end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR JEOC LL_ADC_ClearFlag_JEOC

LL_ADC_ClearFlag_JEOS

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_JEOS (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group injected end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR JEOS LL_ADC_ClearFlag_JEOS

LL_ADC_ClearFlag_JQOVF

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_JQOVF (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group injected contexts queue overflow.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR JQOVF LL_ADC_ClearFlag_JQOVF

LL_ADC_ClearFlag_AWD1

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_AWD1 (ADC_TypeDef * ADCx)

Function description

Clear flag ADC analog watchdog 1.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR AWD1 LL_ADC_ClearFlag_AWD1

LL_ADC_ClearFlag_AWD2

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_AWD2 (ADC_TypeDef * ADCx)

Function description

Clear flag ADC analog watchdog 2.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR AWD2 LL_ADC_ClearFlag_AWD2

LL_ADC_ClearFlag_AWD3

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_AWD3 (ADC_TypeDef * ADCx)

Function description

Clear flag ADC analog watchdog 3.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR AWD3 LL_ADC_ClearFlag_AWD3

LL_ADC_IsActiveFlag_MST_ADRDY

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_ADRDY (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC ready of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR ADRDY_MST LL_ADC_IsActiveFlag_MST_ADRDY

LL_ADC_IsActiveFlag_SLV_ADRDY

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_ADRDY (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC ready of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR ADRDY_SLV LL_ADC_IsActiveFlag_SLV_ADRDY

LL_ADC_IsActiveFlag_MST_EOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_EOC (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular end of unitary conversion of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR EOC_MST LL_ADC_IsActiveFlag_MST_EOC

LL_ADC_IsActiveFlag_SLV_EOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_EOC (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular end of unitary conversion of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR EOC_SLV LL_ADC_IsActiveFlag_SLV_EOC

LL_ADC_IsActiveFlag_MST_EOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_EOS (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular end of sequence conversions of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR EOS_MST LL_ADC_IsActiveFlag_MST_EOS

LL_ADC_IsActiveFlag_SLV_EOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_EOS (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular end of sequence conversions of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR EOS_SLV LL_ADC_IsActiveFlag_SLV_EOS

LL_ADC_IsActiveFlag_MST_OVR

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_OVR (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular overrun of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR OVR_MST LL_ADC_IsActiveFlag_MST_OVR

LL_ADC_IsActiveFlag_SLV_OVR

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_OVR (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular overrun of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR OVR_SLV LL_ADC_IsActiveFlag_SLV_OVR

LL_ADC_IsActiveFlag_MST_EOSMP

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_EOSMP (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular end of sampling of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR EOSMP_MST LL_ADC_IsActiveFlag_MST_EOSMP

LL_ADC_IsActiveFlag_SLV_EOSMP

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_EOSMP (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group regular end of sampling of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR EOSMP_SLV LL_ADC_IsActiveFlag_SLV_EOSMP

LL_ADC_IsActiveFlag_MST_JEOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_JEOC (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group injected end of unitary conversion of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR JEOC_MST LL_ADC_IsActiveFlag_MST_JEOC

LL_ADC_IsActiveFlag_SLV_JEOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_JEOC (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group injected end of unitary conversion of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR JEOC_SLV LL_ADC_IsActiveFlag_SLV_JEOC

LL_ADC_IsActiveFlag_MST_JEOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_JEOS (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group injected end of sequence conversions of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR JEOS_MST LL_ADC_IsActiveFlag_MST_JEOS

LL_ADC_IsActiveFlag_SLV_JEOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_JEOS (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group injected end of sequence conversions of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR JEOS_SLV LL_ADC_IsActiveFlag_SLV_JEOS

LL_ADC_IsActiveFlag_MST_JQOVF

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_JQOVF (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group injected context queue overflow of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR JQOVF_MST LL_ADC_IsActiveFlag_MST_JQOVF

LL_ADC_IsActiveFlag_SLV_JQOVF

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_JQOVF (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC group injected context queue overflow of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR JQOVF_SLV LL_ADC_IsActiveFlag_SLV_JQOVF

LL_ADC_IsActiveFlag_MST_AWD1

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_AWD1 (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC analog watchdog 1 of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR AWD1_MST LL_ADC_IsActiveFlag_MST_AWD1

LL_ADC_IsActiveFlag_SLV_AWD1

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_AWD1 (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode analog watchdog 1 of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR AWD1_SLV LL_ADC_IsActiveFlag_SLV_AWD1

LL_ADC_IsActiveFlag_MST_AWD2

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_AWD2 (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC analog watchdog 2 of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR AWD2_MST LL_ADC_IsActiveFlag_MST_AWD2

LL_ADC_IsActiveFlag_SLV_AWD2

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_AWD2 (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC analog watchdog 2 of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR AWD2_SLV LL_ADC_IsActiveFlag_SLV_AWD2

LL_ADC_IsActiveFlag_MST_AWD3

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_MST_AWD3 (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC analog watchdog 3 of the ADC master.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR AWD3_MST LL_ADC_IsActiveFlag_MST_AWD3

LL_ADC_IsActiveFlag_SLV_AWD3

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_SLV_AWD3 (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get flag multimode ADC analog watchdog 3 of the ADC slave.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR AWD3_SLV LL_ADC_IsActiveFlag_SLV_AWD3

LL_ADC_EnableIT_ADRDY

Function name

__STATIC_INLINE void LL_ADC_EnableIT_ADRDY (ADC_TypeDef * ADCx)

Function description

Enable ADC ready.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ADRDYIE LL_ADC_EnableIT_ADRDY

LL_ADC_EnableIT_EOC

Function name

__STATIC_INLINE void LL_ADC_EnableIT_EOC (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC group regular end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOCIE LL_ADC_EnableIT_EOC

LL_ADC_EnableIT_EOS

Function name

__STATIC_INLINE void LL_ADC_EnableIT_EOS (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC group regular end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOSIE LL_ADC_EnableIT_EOS

LL_ADC_EnableIT_OVR

Function name

__STATIC_INLINE void LL_ADC_EnableIT_OVR (ADC_TypeDef * ADCx)

Function description

Enable ADC group regular interruption overrun.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER OVRIE LL_ADC_EnableIT_OVR

LL_ADC_EnableIT_EOSMP

Function name

__STATIC_INLINE void LL_ADC_EnableIT_EOSMP (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC group regular end of sampling.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOSMPIE LL_ADC_EnableIT_EOSMP

LL_ADC_EnableIT_JEOC

Function name

__STATIC_INLINE void LL_ADC_EnableIT_JEOC (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC group injected end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER JEOCIE LL_ADC_EnableIT_JEOC

LL_ADC_EnableIT_JEOS

Function name

__STATIC_INLINE void LL_ADC_EnableIT_JEOS (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC group injected end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER JEOSIE LL_ADC_EnableIT_JEOS

LL_ADC_EnableIT_JQOVF

Function name

__STATIC_INLINE void LL_ADC_EnableIT_JQOVF (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC group injected context queue overflow.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER JQOVFIE LL_ADC_EnableIT_JQOVF

LL_ADC_EnableIT_AWD1

Function name

__STATIC_INLINE void LL_ADC_EnableIT_AWD1 (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC analog watchdog 1.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER AWD1IE LL_ADC_EnableIT_AWD1

LL_ADC_EnableIT_AWD2

Function name

__STATIC_INLINE void LL_ADC_EnableIT_AWD2 (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC analog watchdog 2.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER AWD2IE LL_ADC_EnableIT_AWD2

LL_ADC_EnableIT_AWD3

Function name

__STATIC_INLINE void LL_ADC_EnableIT_AWD3 (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC analog watchdog 3.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER AWD3IE LL_ADC_EnableIT_AWD3

LL_ADC_DisableIT_ADRDY

Function name

__STATIC_INLINE void LL_ADC_DisableIT_ADRDY (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC ready.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ADRDYIE LL_ADC_DisableIT_ADRDY

LL_ADC_DisableIT_EOC

Function name

__STATIC_INLINE void LL_ADC_DisableIT_EOC (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group regular end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOCIE LL_ADC_DisableIT_EOC

LL_ADC_DisableIT_EOS

Function name

__STATIC_INLINE void LL_ADC_DisableIT_EOS (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group regular end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOSIE LL_ADC_DisableIT_EOS

LL_ADC_DisableIT_OVR

Function name

__STATIC_INLINE void LL_ADC_DisableIT_OVR (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group regular overrun.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER OVRIE LL_ADC_DisableIT_OVR

LL_ADC_DisableIT_EOSMP

Function name

__STATIC_INLINE void LL_ADC_DisableIT_EOSMP (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group regular end of sampling.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOSMPIE LL_ADC_DisableIT_EOSMP

LL_ADC_DisableIT_JEOC

Function name

__STATIC_INLINE void LL_ADC_DisableIT_JEOC (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group regular end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER JEOCIE LL_ADC_DisableIT_JEOC

LL_ADC_DisableIT_JEOS

Function name

__STATIC_INLINE void LL_ADC_DisableIT_JEOS (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group injected end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER JEOSIE LL_ADC_DisableIT_JEOS

LL_ADC_DisableIT_JQOVF

Function name

__STATIC_INLINE void LL_ADC_DisableIT_JQOVF (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group injected context queue overflow.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER JQOVFIE LL_ADC_DisableIT_JQOVF

LL_ADC_DisableIT_AWD1

Function name

__STATIC_INLINE void LL_ADC_DisableIT_AWD1 (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC analog watchdog 1.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER AWD1IE LL_ADC_DisableIT_AWD1

LL_ADC_DisableIT_AWD2

Function name

__STATIC_INLINE void LL_ADC_DisableIT_AWD2 (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC analog watchdog 2.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER AWD2IE LL_ADC_DisableIT_AWD2

LL_ADC_DisableIT_AWD3

Function name

__STATIC_INLINE void LL_ADC_DisableIT_AWD3 (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC analog watchdog 3.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER AWD3IE LL_ADC_DisableIT_AWD3

LL_ADC_IsEnabledIT_ADRDY

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_ADRDY (ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC ready (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER ADRDYIE LL_ADC_IsEnabledIT_ADRDY

LL_ADC_IsEnabledIT_EOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOC (ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group regular end of unitary conversion (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER EOCIE LL_ADC_IsEnabledIT_EOC

LL_ADC_IsEnabledIT_EOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOS (ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group regular end of sequence conversions (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER EOSIE LL_ADC_IsEnabledIT_EOS

LL_ADC_IsEnabledIT_OVR

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_OVR (ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group regular overrun (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER OVRIE LL_ADC_IsEnabledIT_OVR

LL_ADC_IsEnabledIT_EOSMP

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOSMP (ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group regular end of sampling (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER EOSMPIE LL_ADC_IsEnabledIT_EOSMP

LL_ADC_IsEnabledIT_JEOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_JEOC (ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group injected end of unitary conversion (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER JEOCIE LL_ADC_IsEnabledIT_JEOC

LL_ADC_IsEnabledIT_JEOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_JEOS (ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group injected end of sequence conversions (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER JEOSIE LL_ADC_IsEnabledIT_JEOS

LL_ADC_IsEnabledIT_JQOVF

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_JQOVF (ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group injected context queue overflow interrupt state (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER JQOVFIE LL_ADC_IsEnabledIT_JQOVF

LL_ADC_IsEnabledIT_AWD1

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_AWD1 (ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC analog watchdog 1 (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER AWD1IE LL_ADC_IsEnabledIT_AWD1

LL_ADC_IsEnabledIT_AWD2

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_AWD2 (ADC_TypeDef * ADCx)

Function description

Get state of interruption Get ADC analog watchdog 2 (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER AWD2IE LL_ADC_IsEnabledIT_AWD2

LL_ADC_IsEnabledIT_AWD3

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_AWD3 (ADC_TypeDef * ADCx)

Function description

Get state of interruption Get ADC analog watchdog 3 (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER AWD3IE LL_ADC_IsEnabledIT_AWD3

LL_ADC_CommonDeInit

Function name

ErrorStatus LL_ADC_CommonDeInit (ADC_Common_TypeDef * ADCxy_COMMON)

Function description

De-initialize registers of all ADC instances belonging to the same ADC common instance to their default reset values.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ADC common registers are de-initialized
  - ERROR: not applicable

Notes

- This function is performing a hard reset, using high level clock source RCC ADC reset. Caution: On this STM32 series, if several ADC instances are available on the selected device, RCC ADC reset will reset all ADC instances belonging to the common ADC instance. To de-initialize only 1 ADC instance, use function LL_ADC_DeInit().

LL_ADC_CommonInit

Function name

ErrorStatus LL_ADC_CommonInit (ADC_Common_TypeDef * ADCxy_COMMON, LL_ADC_CommonInitTypeDef * ADC_CommonInitStruct)

Function description

Initialize some features of ADC common parameters (all ADC instances belonging to the same ADC common instance) and multimode (for devices with several ADC instances available).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **ADC_CommonInitStruct:** Pointer to a LL_ADC_CommonInitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ADC common registers are initialized
  - ERROR: ADC common registers are not initialized

Notes

- The setting of ADC common parameters is conditioned to ADC instances state: All ADC instances belonging to the same ADC common instance must be disabled.

LL_ADC_CommonStructInit

Function name

void LL_ADC_CommonStructInit (LL_ADC_CommonInitTypeDef * ADC_CommonInitStruct)

Function description

Set each LL_ADC_CommonInitTypeDef field to default value.

Parameters

- **ADC_CommonInitStruct:** Pointer to a LL_ADC_CommonInitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_ADC_DeInit

Function name

ErrorStatus LL_ADC_DeInit (ADC_TypeDef * ADCx)

Function description

De-initialize registers of the selected ADC instance to their default reset values.

Parameters

- **ADCx:** ADC instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ADC registers are de-initialized
  - ERROR: ADC registers are not de-initialized

Notes

- To reset all ADC instances quickly (perform a hard reset), use function LL_ADC_CommonDeInit().
- If this functions returns error status, it means that ADC instance is in an unknown state. In this case, perform a hard reset using high level clock source RCC ADC reset. Caution: On this STM32 series, if several ADC instances are available on the selected device, RCC ADC reset will reset all ADC instances belonging to the common ADC instance. Refer to function LL_ADC_CommonDeInit().

LL_ADC_Init

Function name

ErrorStatus LL_ADC_Init (ADC_TypeDef * ADCx, LL_ADC_InitTypeDef * ADC_InitStruct)

Function description

Initialize some features of ADC instance.

Parameters

- **ADCx:** ADC instance
- **ADC_InitStruct:** Pointer to a LL_ADC_REG_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ADC registers are initialized
  - ERROR: ADC registers are not initialized

Notes

- These parameters have an impact on ADC scope: ADC instance. Affects both group regular and group injected (availability of ADC group injected depends on STM32 families). Refer to corresponding unitary functions into Configuration of ADC hierarchical scope: ADC instance .
- The setting of these parameters by function LL_ADC_Init() is conditioned to ADC state: ADC instance must be disabled. This condition is applied to all ADC features, for efficiency and compatibility over all STM32 families. However, the different features can be set under different ADC state conditions (setting possible with ADC enabled without conversion on going, ADC enabled with conversion on going, ...) Each feature can be updated afterwards with a unitary function and potentially with ADC in a different state than disabled, refer to description of each function for setting conditioned to ADC state.
- After using this function, some other features must be configured using LL unitary functions. The minimum configuration remaining to be done is: Set ADC group regular or group injected sequencer: map channel on the selected sequencer rank. Refer to function LL_ADC_REG_SetSequencerRanks().Set ADC channel sampling time Refer to function LL_ADC_SetChannelSamplingTime();

LL_ADC_StructInit

Function name

void LL_ADC_StructInit (LL_ADC_InitTypeDef * ADC_InitStruct)

Function description

Set each LL_ADC_InitTypeDef field to default value.

Parameters

- **ADC_InitStruct:** Pointer to a LL_ADC_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_ADC_REG_Init

Function name

ErrorStatus LL_ADC_REG_Init (ADC_TypeDef * ADCx, LL_ADC_REG_InitTypeDef * ADC_REG_InitStruct)

Function description

Initialize some features of ADC group regular.

Parameters

- **ADCx:** ADC instance
- **ADC_REG_InitStruct:** Pointer to a LL_ADC_REG_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ADC registers are initialized
  - ERROR: ADC registers are not initialized

Notes

- These parameters have an impact on ADC scope: ADC group regular. Refer to corresponding unitary functions into Configuration of ADC hierarchical scope: group regular (functions with prefix "REG").
- The setting of these parameters by function LL_ADC_Init() is conditioned to ADC state: ADC instance must be disabled. This condition is applied to all ADC features, for efficiency and compatibility over all STM32 families. However, the different features can be set under different ADC state conditions (setting possible with ADC enabled without conversion on going, ADC enabled with conversion on going, ...) Each feature can be updated afterwards with a unitary function and potentially with ADC in a different state than disabled, refer to description of each function for setting conditioned to ADC state.
- After using this function, other features must be configured using LL unitary functions. The minimum configuration remaining to be done is: Set ADC group regular or group injected sequencer: map channel on the selected sequencer rank. Refer to function LL_ADC_REG_SetSequencerRanks().Set ADC channel sampling time Refer to function LL_ADC_SetChannelSamplingTime();

LL_ADC_REG_StructInit

Function name

void LL_ADC_REG_StructInit (LL_ADC_REG_InitTypeDef * ADC_REG_InitStruct)

Function description

Set each LL_ADC_REG_InitTypeDef field to default value.

Parameters

- **ADC_REG_InitStruct:** Pointer to a LL_ADC_REG_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_ADC_INJ_Init

Function name

ErrorStatus LL_ADC_INJ_Init (ADC_TypeDef * ADCx, LL_ADC_INJ_InitTypeDef * ADC_INJ_InitStruct)

Function description

Initialize some features of ADC group injected.

Parameters

- **ADCx:** ADC instance
- **ADC_INJ_InitStruct:** Pointer to a LL_ADC_INJ_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ADC registers are initialized
  - ERROR: ADC registers are not initialized

Notes

- These parameters have an impact on ADC scope: ADC group injected. Refer to corresponding unitary functions into Configuration of ADC hierarchical scope: group regular (functions with prefix "INJ").
- The setting of these parameters by function LL_ADC_Init() is conditioned to ADC state: ADC instance must be disabled. This condition is applied to all ADC features, for efficiency and compatibility over all STM32 families. However, the different features can be set under different ADC state conditions (setting possible with ADC enabled without conversion on going, ADC enabled with conversion on going, ...) Each feature can be updated afterwards with a unitary function and potentially with ADC in a different state than disabled, refer to description of each function for setting conditioned to ADC state.
- After using this function, other features must be configured using LL unitary functions. The minimum configuration remaining to be done is: Set ADC group injected sequencer: map channel on the selected sequencer rank. Refer to function LL_ADC_INJ_SetSequencerRanks().Set ADC channel sampling time Refer to function LL_ADC_SetChannelSamplingTime();

LL_ADC_INJ_StructInit

Function name

void LL_ADC_INJ_StructInit (LL_ADC_INJ_InitTypeDef * ADC_INJ_InitStruct)

Function description

Set each LL_ADC_INJ_InitTypeDef field to default value.

Parameters

- **ADC_INJ_InitStruct:** Pointer to a LL_ADC_INJ_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

95.3 ADC Firmware driver defines

The following section lists the various define and macros of the module.

95.3.1 ADC

ADC

ADC Alias definition

LL_ADC_SetChannelPreSelection

Analog watchdog - Monitored channels

LL_ADC_AWD_DISABLE

ADC analog watchdog monitoring disabled

LL_ADC_AWD_ALL_CHANNELS_REG

ADC analog watchdog monitoring of all channels, converted by group regular only

LL_ADC_AWD_ALL_CHANNELS_INJ

ADC analog watchdog monitoring of all channels, converted by group injected only

LL_ADC_AWD_ALL_CHANNELS_REG_INJ

ADC analog watchdog monitoring of all channels, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_0_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN0, converted by group regular only

LL_ADC_AWD_CHANNEL_0_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN0, converted by group injected only

LL_ADC_AWD_CHANNEL_0_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN0, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_1_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN1, converted by group regular only

LL_ADC_AWD_CHANNEL_1_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN1, converted by group injected only

LL_ADC_AWD_CHANNEL_1_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN1, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_2_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN2, converted by group regular only

LL_ADC_AWD_CHANNEL_2_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN2, converted by group injected only

LL_ADC_AWD_CHANNEL_2_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN2, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_3_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN3, converted by group regular only

LL_ADC_AWD_CHANNEL_3_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN3, converted by group injected only

LL_ADC_AWD_CHANNEL_3_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN3, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_4_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN4, converted by group regular only

LL_ADC_AWD_CHANNEL_4_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN4, converted by group injected only

LL_ADC_AWD_CHANNEL_4_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN4, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_5_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN5, converted by group regular only

LL_ADC_AWD_CHANNEL_5_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN5, converted by group injected only

LL_ADC_AWD_CHANNEL_5_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN5, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_6_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN6, converted by group regular only

LL_ADC_AWD_CHANNEL_6_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN6, converted by group injected only

LL_ADC_AWD_CHANNEL_6_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN6, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_7_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN7, converted by group regular only

LL_ADC_AWD_CHANNEL_7_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN7, converted by group injected only

LL_ADC_AWD_CHANNEL_7_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN7, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_8_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN8, converted by group regular only

LL_ADC_AWD_CHANNEL_8_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN8, converted by group injected only

LL_ADC_AWD_CHANNEL_8_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN8, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_9_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN9, converted by group regular only

LL_ADC_AWD_CHANNEL_9_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN9, converted by group injected only

LL_ADC_AWD_CHANNEL_9_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN9, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_10_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN10, converted by group regular only

LL_ADC_AWD_CHANNEL_10_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN10, converted by group injected only

LL_ADC_AWD_CHANNEL_10_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN10, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_11_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN11, converted by group regular only

LL_ADC_AWD_CHANNEL_11_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN11, converted by group injected only

LL_ADC_AWD_CHANNEL_11_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN11, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_12_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN12, converted by group regular only

LL_ADC_AWD_CHANNEL_12_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN12, converted by group injected only

LL_ADC_AWD_CHANNEL_12_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN12, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_13_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN13, converted by group regular only

LL_ADC_AWD_CHANNEL_13_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN13, converted by group injected only

LL_ADC_AWD_CHANNEL_13_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN13, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_14_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN14, converted by group regular only

LL_ADC_AWD_CHANNEL_14_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN14, converted by group injected only

LL_ADC_AWD_CHANNEL_14_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN14, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_15_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN15, converted by group regular only

LL_ADC_AWD_CHANNEL_15_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN15, converted by group injected only

LL_ADC_AWD_CHANNEL_15_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN15, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_16_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN16, converted by group regular only

LL_ADC_AWD_CHANNEL_16_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN16, converted by group injected only

LL_ADC_AWD_CHANNEL_16_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN16, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_17_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN17, converted by group regular only

LL_ADC_AWD_CHANNEL_17_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN17, converted by group injected only

LL_ADC_AWD_CHANNEL_17_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN17, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_18_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN18, converted by group regular only

LL_ADC_AWD_CHANNEL_18_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN18, converted by group injected only

LL_ADC_AWD_CHANNEL_18_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN18, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_19_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN19, converted by group regular only

LL_ADC_AWD_CHANNEL_19_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN19, converted by group injected only

LL_ADC_AWD_CHANNEL_19_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN19, converted by either group regular or injected

LL_ADC_AWD_CH_VREFINT_REG

ADC analog watchdog monitoring of ADC internal channel connected to VrefInt: Internal voltage reference, converted by group regular only

LL_ADC_AWD_CH_VREFINT_INJ

ADC analog watchdog monitoring of ADC internal channel connected to VrefInt: Internal voltage reference, converted by group injected only

LL_ADC_AWD_CH_VREFINT_REG_INJ

ADC analog watchdog monitoring of ADC internal channel connected to VrefInt: Internal voltage reference, converted by either group regular or injected

LL_ADC_AWD_CH_TEMPSENSOR_REG

ADC analog watchdog monitoring of ADC internal channel connected to Temperature sensor, converted by group regular only

LL_ADC_AWD_CH_TEMPSENSOR_INJ

ADC analog watchdog monitoring of ADC internal channel connected to Temperature sensor, converted by group injected only

LL_ADC_AWD_CH_TEMPSENSOR_REG_INJ

ADC analog watchdog monitoring of ADC internal channel connected to Temperature sensor, converted by either group regular or injected

LL_ADC_AWD_CH_VBAT_REG

ADC analog watchdog monitoring of ADC internal channel connected to Vbat/4: Vbat voltage through a divider ladder of factor 1/4 to have Vbat always below Vdda, converted by group regular only

LL_ADC_AWD_CH_VBAT_INJ

ADC analog watchdog monitoring of ADC internal channel connected to Vbat/4: Vbat voltage through a divider ladder of factor 1/4 to have Vbat always below Vdda, converted by group injected only

LL_ADC_AWD_CH_VBAT_REG_INJ

ADC analog watchdog monitoring of ADC internal channel connected to Vbat/4: Vbat voltage through a divider ladder of factor 1/4 to have Vbat always below Vdda

LL_ADC_AWD_CH_DAC1CH1_ADC2_REG

ADC analog watchdog monitoring of ADC internal channel connected to DAC1 channel 1, channel specific to ADC2, converted by group regular only

LL_ADC_AWD_CH_DAC1CH1_ADC2_INJ

ADC analog watchdog monitoring of ADC internal channel connected to DAC1 channel 1, channel specific to ADC2, converted by group injected only

LL_ADC_AWD_CH_DAC1CH1_ADC2_REG_INJ

ADC analog watchdog monitoring of ADC internal channel connected to DAC1 channel 1, channel specific to ADC2, converted by either group regular or injected

LL_ADC_AWD_CH_DAC1CH2_ADC2_REG

ADC analog watchdog monitoring of ADC internal channel connected to DAC1 channel 1, channel specific to ADC2, converted by group regular only

LL_ADC_AWD_CH_DAC1CH2_ADC2_INJ

ADC analog watchdog monitoring of ADC internal channel connected to DAC1 channel 1, channel specific to ADC2, converted by group injected only

LL_ADC_AWD_CH_DAC1CH2_ADC2_REG_INJ

ADC analog watchdog monitoring of ADC internal channel connected to DAC1 channel 1, channel specific to ADC2, converted by either group regular or injected

Analog watchdog - Analog watchdog number

LL_ADC_AWD1

ADC analog watchdog number 1

LL_ADC_AWD2

ADC analog watchdog number 2

LL_ADC_AWD3

ADC analog watchdog number 3

Analog watchdog - Thresholds

LL_ADC_AWD_THRESHOLD_HIGH

ADC analog watchdog threshold high

LL_ADC_AWD_THRESHOLD_LOW

ADC analog watchdog threshold low

ADC instance - Boost mode

LL_ADC_BOOST_MODE_6MHZ25

Boost mode is configured for frequency <= 6.25Mhz

LL_ADC_BOOST_MODE_12MHZ5

Boost mode is configured for 6.25Mhz < frequency <= 12.5Mhz

LL_ADC_BOOST_MODE_20MHZ

Boost mode is configured for 12.5Mhz < frequency <= 20Mhz

LL_ADC_BOOST_MODE_25MHZ

Boost mode is configured for 20Mhz < frequency <= 25Mhz

LL_ADC_BOOST_MODE_50MHZ

Boost mode is configured for frequency > 25Mhz

ADC instance - Calibration linearity words

LL_ADC_CALIB_LINEARITY_WORD1

ADC calibration linearity word 1

LL_ADC_CALIB_LINEARITY_WORD2

ADC calibration linearity word 2

LL_ADC_CALIB_LINEARITY_WORD3

ADC calibration linearity word 3

LL_ADC_CALIB_LINEARITY_WORD4

ADC calibration linearity word 4

LL_ADC_CALIB_LINEARITY_WORD5

ADC calibration linearity word 5

LL_ADC_CALIB_LINEARITY_WORD6

ADC calibration linearity word 6

ADC instance - Calibration mode for offset and linearity

LL_ADC_CALIB_OFFSET

Calibration of ADC offset. Duration of calibration of offset duration: 1280 ADC clock cycles. For devices with differential mode available: Calibration of offset is specific to each of single-ended and differential modes.

LL_ADC_CALIB_LINEARITY

Calibration of ADC linearity. Duration of calibration of linearity: 15104 ADC clock cycles. For devices with differential mode available: Calibration of linearity is common to both single-ended and differential modes.

LL_ADC_CALIB_OFFSET_LINEARITY

Calibration of ADC offset and linearity. Duration of calibration of offset and linearity: 16384 ADC clock cycles. For devices with differential mode available: Calibration of offset is specific to each of single-ended and differential modes, calibration of linearity is common to both single-ended and differential modes.

ADC instance - Channel number

LL_ADC_CHANNEL_0

ADC external channel (channel connected to GPIO pin) ADCx_IN0

LL_ADC_CHANNEL_1

ADC external channel (channel connected to GPIO pin) ADCx_IN1

LL_ADC_CHANNEL_2

ADC external channel (channel connected to GPIO pin) ADCx_IN2

LL_ADC_CHANNEL_3

ADC external channel (channel connected to GPIO pin) ADCx_IN3

LL_ADC_CHANNEL_4

ADC external channel (channel connected to GPIO pin) ADCx_IN4

LL_ADC_CHANNEL_5

ADC external channel (channel connected to GPIO pin) ADCx_IN5

LL_ADC_CHANNEL_6

ADC external channel (channel connected to GPIO pin) ADCx_IN6

LL_ADC_CHANNEL_7

ADC external channel (channel connected to GPIO pin) ADCx_IN7

LL_ADC_CHANNEL_8

ADC external channel (channel connected to GPIO pin) ADCx_IN8

LL_ADC_CHANNEL_9

ADC external channel (channel connected to GPIO pin) ADCx_IN9

LL_ADC_CHANNEL_10

ADC external channel (channel connected to GPIO pin) ADCx_IN10

LL_ADC_CHANNEL_11

ADC external channel (channel connected to GPIO pin) ADCx_IN11

LL_ADC_CHANNEL_12

ADC external channel (channel connected to GPIO pin) ADCx_IN12

LL_ADC_CHANNEL_13

ADC external channel (channel connected to GPIO pin) ADCx_IN13

LL_ADC_CHANNEL_14

ADC external channel (channel connected to GPIO pin) ADCx_IN14

LL_ADC_CHANNEL_15

ADC external channel (channel connected to GPIO pin) ADCx_IN15

LL_ADC_CHANNEL_16

ADC external channel (channel connected to GPIO pin) ADCx_IN16

LL_ADC_CHANNEL_17

ADC external channel (channel connected to GPIO pin) ADCx_IN17

LL_ADC_CHANNEL_18

ADC external channel (channel connected to GPIO pin) ADCx_IN18

LL_ADC_CHANNEL_19

ADC external channel (channel connected to GPIO pin) ADCx_IN19

LL_ADC_CHANNEL_VREFINT

ADC internal channel connected to VrefInt: Internal voltage reference. On STM32H7, ADC channel available only on ADC instance: ADC3.

LL_ADC_CHANNEL_TEMPSENSOR

ADC internal channel connected to Temperature sensor. On STM32H7, ADC channel available only on ADC instance: ADC3.

LL_ADC_CHANNEL_VBAT

ADC internal channel connected to Vbat/3: Vbat voltage through a divider ladder of factor 1/4 to have Vbat always below Vdda. On STM32H7, ADC channel available only on ADC instance: ADC3.

LL_ADC_CHANNEL_DAC1CH1_ADC2

ADC internal channel connected to DAC1 channel 1, channel specific to ADC2

LL_ADC_CHANNEL_DAC1CH2_ADC2

ADC internal channel connected to DAC1 channel 2, channel specific to ADC2

Channel - Sampling time

LL_ADC_SAMPLINGTIME_1CYCLE_5

Sampling time 1.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_2CYCLES_5

Sampling time 2.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_8CYCLES_5

Sampling time 8.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_16CYCLES_5

Sampling time 16.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_32CYCLES_5

Sampling time 32.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_64CYCLES_5

Sampling time 64.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_387CYCLES_5

Sampling time 387.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_810CYCLES_5

Sampling time 810.5 ADC clock cycles

Channel - Single or differential ending

LL_ADC_SINGLE_ENDED

ADC channel ending set to single ended (literal also used to set calibration mode)

LL_ADC_DIFFERENTIAL_ENDED

ADC channel ending set to differential (literal also used to set calibration mode)

LL_ADC_BOTH_SINGLE_DIFF_ENDED

ADC channel ending set to both single ended and differential (literal used only to set calibration factors)

ADC common - Clock source

LL_ADC_CLOCK_SYNC_PCLK_DIV1

ADC synchronous clock derived from AHB clock without prescaler

LL_ADC_CLOCK_SYNC_PCLK_DIV2

ADC synchronous clock derived from AHB clock with prescaler division by 2

LL_ADC_CLOCK_SYNC_PCLK_DIV4

ADC synchronous clock derived from AHB clock with prescaler division by 4

LL_ADC_CLOCK_ASYNC_DIV1

ADC asynchronous clock without prescaler

LL_ADC_CLOCK_ASYNC_DIV2

ADC asynchronous clock with prescaler division by 2

LL_ADC_CLOCK_ASYNC_DIV4

ADC asynchronous clock with prescaler division by 4

LL_ADC_CLOCK_ASYNC_DIV6

ADC asynchronous clock with prescaler division by 6

LL_ADC_CLOCK_ASYNC_DIV8

ADC asynchronous clock with prescaler division by 8

LL_ADC_CLOCK_ASYNC_DIV10

ADC asynchronous clock with prescaler division by 10

LL_ADC_CLOCK_ASYNC_DIV12

ADC asynchronous clock with prescaler division by 12

LL_ADC_CLOCK_ASYNC_DIV16

ADC asynchronous clock with prescaler division by 16

LL_ADC_CLOCK_ASYNC_DIV32

ADC asynchronous clock with prescaler division by 32

LL_ADC_CLOCK_ASYNC_DIV64

ADC asynchronous clock with prescaler division by 64

LL_ADC_CLOCK_ASYNC_DIV128

ADC asynchronous clock with prescaler division by 128

LL_ADC_CLOCK_ASYNC_DIV256

ADC asynchronous clock with prescaler division by 256

ADC common - Measurement path to internal channels

LL_ADC_PATH_INTERNAL_NONE

ADC measurement paths all disabled

LL_ADC_PATH_INTERNAL_VREFINT

ADC measurement path to internal channel VrefInt

LL_ADC_PATH_INTERNAL_TEMPSENSOR

ADC measurement path to internal channel temperature sensor

LL_ADC_PATH_INTERNAL_VBAT

ADC measurement path to internal channel Vbat

ADC flags

LL_ADC_FLAG_ADRDY

ADC flag ADC instance ready

LL_ADC_FLAG_EOC

ADC flag ADC group regular end of unitary conversion

LL_ADC_FLAG_EOS

ADC flag ADC group regular end of sequence conversions

LL_ADC_FLAG_OVR

ADC flag ADC group regular overrun

LL_ADC_FLAG_EOSMP

ADC flag ADC group regular end of sampling phase

LL_ADC_FLAG_JEOC

ADC flag ADC group injected end of unitary conversion

LL_ADC_FLAG_JEOS

ADC flag ADC group injected end of sequence conversions

LL_ADC_FLAG_JQOVF

ADC flag ADC group injected contexts queue overflow

LL_ADC_FLAG_AWD1

ADC flag ADC analog watchdog 1

LL_ADC_FLAG_AWD2

ADC flag ADC analog watchdog 2

LL_ADC_FLAG_AWD3

ADC flag ADC analog watchdog 3

LL_ADC_FLAG_LDORDY

ADC flag ADC LDO output voltage ready bit

LL_ADC_FLAG_ADRDY_MST

ADC flag ADC multimode master instance ready

LL_ADC_FLAG_ADRDY_SLV

ADC flag ADC multimode slave instance ready

LL_ADC_FLAG_EOC_MST

ADC flag ADC multimode master group regular end of unitary conversion

LL_ADC_FLAG_EOC_SLV

ADC flag ADC multimode slave group regular end of unitary conversion

LL_ADC_FLAG_EOS_MST

ADC flag ADC multimode master group regular end of sequence conversions

LL_ADC_FLAG_EOS_SLV

ADC flag ADC multimode slave group regular end of sequence conversions

LL_ADC_FLAG_OVR_MST

ADC flag ADC multimode master group regular overrun

LL_ADC_FLAG_OVR_SLV

ADC flag ADC multimode slave group regular overrun

LL_ADC_FLAG_EOSMP_MST

ADC flag ADC multimode master group regular end of sampling phase

LL_ADC_FLAG_EOSMP_SLV

ADC flag ADC multimode slave group regular end of sampling phase

LL_ADC_FLAG_JEOC_MST

ADC flag ADC multimode master group injected end of unitary conversion

LL_ADC_FLAG_JEOC_SLV

ADC flag ADC multimode slave group injected end of unitary conversion

LL_ADC_FLAG_JEOS_MST

ADC flag ADC multimode master group injected end of sequence conversions

LL_ADC_FLAG_JEOS_SLV

ADC flag ADC multimode slave group injected end of sequence conversions

LL_ADC_FLAG_JQOVF_MST

ADC flag ADC multimode master group injected contexts queue overflow

LL_ADC_FLAG_JQOVF_SLV

ADC flag ADC multimode slave group injected contexts queue overflow

LL_ADC_FLAG_AWD1_MST

ADC flag ADC multimode master analog watchdog 1 of the ADC master

LL_ADC_FLAG_AWD1_SLV

ADC flag ADC multimode slave analog watchdog 1 of the ADC slave

LL_ADC_FLAG_AWD2_MST

ADC flag ADC multimode master analog watchdog 2 of the ADC master

LL_ADC_FLAG_AWD2_SLV

ADC flag ADC multimode slave analog watchdog 2 of the ADC slave

LL_ADC_FLAG_AWD3_MST

ADC flag ADC multimode master analog watchdog 3 of the ADC master

LL_ADC_FLAG_AWD3_SLV

ADC flag ADC multimode slave analog watchdog 3 of the ADC slave

ADC instance - Groups

LL_ADC_GROUP_REGULAR

ADC group regular (available on all STM32 devices)

LL_ADC_GROUP_INJECTED

ADC group injected (not available on all STM32 devices)

LL_ADC_GROUP_REGULAR_INJECTED

ADC both groups regular and injected

Definitions of ADC hardware constraints delays

LL_ADC_DELAY_INTERNAL_REGUL_STAB_US

Delay for ADC stabilization time (ADC voltage regulator start-up time)

LL_ADC_DELAY_VREFINT_STAB_US

Delay for internal voltage reference stabilization time

LL_ADC_DELAY_TEMPSENSOR_STAB_US

Delay for temperature sensor stabilization time

LL_ADC_DELAY_CALIB_ENABLE_ADC_CYCLES

Delay required between ADC end of calibration and ADC enable

ADC_LINEARITY_BIT_TOGGLE_TIMEOUT

ADC linearity set/clear bit delay

ADC group injected - Context queue mode

LL_ADC_INJ_QUEUE_2CONTEXTS_LAST_ACTIVE

LL_ADC_INJ_QUEUE_2CONTEXTS_END_EMPTY

LL_ADC_INJ_QUEUE_DISABLE

ADC group injected - Sequencer discontinuous mode

LL_ADC_INJ_SEQ_DISCONT_DISABLE

ADC group injected sequencer discontinuous mode disable

LL_ADC_INJ_SEQ_DISCONT_1RANK

ADC group injected sequencer discontinuous mode enable with sequence interruption every rank

ADC group injected - Sequencer ranks

LL_ADC_INJ_RANK_1

ADC group injected sequencer rank 1

LL_ADC_INJ_RANK_2

ADC group injected sequencer rank 2

LL_ADC_INJ_RANK_3

ADC group injected sequencer rank 3

LL_ADC_INJ_RANK_4

ADC group injected sequencer rank 4

ADC group injected - Sequencer scan length

LL_ADC_INJ_SEQ_SCAN_DISABLE

ADC group injected sequencer disable (equivalent to sequencer of 1 rank: ADC conversion on only 1 channel)

LL_ADC_INJ_SEQ_SCAN_ENABLE_2RANKS

ADC group injected sequencer enable with 2 ranks in the sequence

LL_ADC_INJ_SEQ_SCAN_ENABLE_3RANKS

ADC group injected sequencer enable with 3 ranks in the sequence

LL_ADC_INJ_SEQ_SCAN_ENABLE_4RANKS

ADC group injected sequencer enable with 4 ranks in the sequence

ADC group injected - Trigger edge

LL_ADC_INJ_TRIG_EXT_RISING

ADC group injected conversion trigger polarity set to rising edge

LL_ADC_INJ_TRIG_EXT_FALLING

ADC group injected conversion trigger polarity set to falling edge

LL_ADC_INJ_TRIG_EXT_RISINGFALLING

ADC group injected conversion trigger polarity set to both rising and falling edges

ADC group injected - Trigger source

LL_ADC_INJ_TRIG_SOFTWARE

ADC group injected conversion trigger internal: SW start.

LL_ADC_INJ_TRIG_EXT_TIM1_TRGO

ADC group injected conversion trigger from external peripheral: TIM1 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM1_CH4

ADC group injected conversion trigger from external peripheral: TIM1 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM2_TRGO

ADC group injected conversion trigger from external peripheral: TIM2 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM2_CH1

ADC group injected conversion trigger from external peripheral: TIM2 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM3_CH4

ADC group injected conversion trigger from external peripheral: TIM3 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM4_TRGO

ADC group injected conversion trigger from external peripheral: TIM4 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_EXTI_LINE15

ADC group injected conversion trigger from external peripheral: external interrupt line 15. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM8_CH4

ADC group injected conversion trigger from external peripheral: TIM8 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM1_TRGO2

ADC group injected conversion trigger from external peripheral: TIM1 TRGO2 event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM8_TRGO

ADC group injected conversion trigger from external peripheral: TIM8 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM8_TRGO2

ADC group injected conversion trigger from external peripheral: TIM8 TRGO2 event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM3_CH3

ADC group injected conversion trigger from external peripheral: TIM3 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM3_TRGO

ADC group injected conversion trigger from external peripheral: TIM3 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM3_CH1

ADC group injected conversion trigger from external peripheral: TIM3 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM6_TRGO

ADC group injected conversion trigger from external peripheral: TIM6 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM15_TRGO

ADC group injected conversion trigger from external peripheral: TIM15 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_HRTIM_TRG2

ADC group injected conversion trigger from external peripheral: HRTIM1 TRG2 event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_HRTIM_TRG4

ADC group injected conversion trigger from external peripheral: HRTIM1 TRG4 event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_LPTIM1_OUT

ADC group injected conversion trigger from external peripheral: LPTIM1 OUT event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_LPTIM2_OUT

ADC group injected conversion trigger from external peripheral: LPTIM2 OUT event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_LPTIM3_OUT

ADC group injected conversion trigger from external peripheral: LPTIM3 OUT event. 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM23_TRGO

ADC group regular conversion trigger from external peripheral: TIM23 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM24_TRGO

ADC group regular conversion trigger from external peripheral: TIM24 TRGO event. Trigger edge set to rising edge (default setting).

ADC group injected - Automatic trigger mode

LL_ADC_INJ_TRIG_INDEPENDENT

ADC group injected conversion trigger independent. Setting mandatory if ADC group injected injected trigger source is set to an external trigger.

LL_ADC_INJ_TRIG_FROM_GRP_REGULAR

ADC group injected conversion trigger from ADC group regular. Setting compliant only with group injected trigger source set to SW start, without any further action on ADC group injected conversion start or stop: in this case, ADC group injected is controlled only from ADC group regular.

ADC interruptions for configuration (interruption enable or disable)

LL_ADC_IT_ADRDY

ADC interruption ADC instance ready

LL_ADC_IT_EOC

ADC interruption ADC group regular end of unitary conversion

LL_ADC_IT_EOS

ADC interruption ADC group regular end of sequence conversions

LL_ADC_IT_OVR

ADC interruption ADC group regular overrun

LL_ADC_IT_EOSMP

ADC interruption ADC group regular end of sampling phase

LL_ADC_IT_JEOC

ADC interruption ADC group injected end of unitary conversion

LL_ADC_IT_JEOS

ADC interruption ADC group injected end of sequence conversions

LL_ADC_IT_JQOVF

ADC interruption ADC group injected contexts queue overflow

LL_ADC_IT_AWD1

ADC interruption ADC analog watchdog 1

LL_ADC_IT_AWD2

ADC interruption ADC analog watchdog 2

LL_ADC_IT_AWD3

ADC interruption ADC analog watchdog 3

ADC left Shift

LL_ADC_LEFT_BIT_SHIFT_NONE

ADC no bit shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_1

ADC 1 bit shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_2

ADC 2 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_3

ADC 3 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_4

ADC 4 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_5

ADC 5 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_6

ADC 6 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_7

ADC 7 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_8

ADC 8 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_9

ADC 9 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_10

ADC 10 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_11

ADC 11 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_12

ADC 12 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_13

ADC 13 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_14

ADC 14 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_15

ADC 15 bits shift left applied on the final ADC conversion data

ADC instance - Low power mode

LL_ADC_LP_MODE_NONE

No ADC low power mode activated

LL_ADC_LP_AUTOWAIT

ADC low power mode auto delay: Dynamic low power mode, ADC conversions are performed only when necessary (when previous ADC conversion data is read). See description with function

Multimode - DMA transfer

LL_ADC_MULTI_REG_DMA_EACH_ADC

ADC multimode group regular conversions are transferred by DMA: each ADC uses its own DMA channel, with its individual DMA transfer settings

LL_ADC_MULTI_REG_DMA_RES_32_10B

ADC multimode group regular conversions are transferred by DMA, one DMA channel for both ADC (DMA of ADC master). Setting for ADC resolution of 32 (16x2) down to 10 bits

LL_ADC_MULTI_REG_DMA_RES_8B

ADC multimode group regular conversions are transferred by DMA, one DMA channel for both ADC (DMA of ADC master). Setting for ADC resolution of 8 bits

Multimode - ADC master or slave

LL_ADC_MULTI_MASTER

In multimode, selection among several ADC instances: ADC master

LL_ADC_MULTI_SLAVE

In multimode, selection among several ADC instances: ADC slave

LL_ADC_MULTI_MASTER_SLAVE

In multimode, selection among several ADC instances: both ADC master and ADC slave

Multimode - Mode

LL_ADC_MULTI_INDEPENDENT

ADC dual mode disabled (ADC independent mode)

LL_ADC_MULTI_DUAL_REG_SIMULT

ADC dual mode enabled: group regular simultaneous

LL_ADC_MULTI_DUAL_REG_INTERL

ADC dual mode enabled: Combined group regular interleaved

LL_ADC_MULTI_DUAL_INJ_SIMULT

ADC dual mode enabled: group injected simultaneous

LL_ADC_MULTI_DUAL_INJ_ALTERN

ADC dual mode enabled: group injected alternate trigger. Works only with external triggers (not internal SW start)

LL_ADC_MULTI_DUAL_REG_SIM_INJ_SIM

ADC dual mode enabled: Combined group regular simultaneous + group injected simultaneous

LL_ADC_MULTI_DUAL_REG_SIM_INJ_ALT

ADC dual mode enabled: Combined group regular simultaneous + group injected alternate trigger

LL_ADC_MULTI_DUAL_REG_INT_INJ_SIM

ADC dual mode enabled: Combined group regular interleaved + group injected simultaneous

Multimode - Delay between two sampling phases

LL_ADC_MULTI_TWOSMP_DELAY_1CYCLE_5

ADC multimode delay between two sampling phases: 1.5 ADC clock cycle for all resolution

LL_ADC_MULTI_TWOSMP_DELAY_2CYCLES_5

ADC multimode delay between two sampling phases: 2.5 ADC clock cycles for all resolution

LL_ADC_MULTI_TWOSMP_DELAY_3CYCLES_5

ADC multimode delay between two sampling phases: 3.5 ADC clock cycles for all resolution

LL_ADC_MULTI_TWOSMP_DELAY_4CYCLES_5

ADC multimode delay between two sampling phases: 4.5 ADC clock cycles for 16, 14, 12 or 10 bits resolution

LL_ADC_MULTI_TWOSMP_DELAY_4CYCLES_5_8_BITS

ADC multimode delay between two sampling phases: 4.5 ADC clock cycles for 8 bits resolution

LL_ADC_MULTI_TWOSMP_DELAY_5CYCLES_5

ADC multimode delay between two sampling phases: 5.5 ADC clock cycles for 16, 14, 12 bits resolution

LL_ADC_MULTI_TWOSMP_DELAY_5CYCLES_5_10_BITS

ADC multimode delay between two sampling phases: 5.5 ADC clock cycles for 10 bits resolution

LL_ADC_MULTI_TWOSMP_DELAY_6CYCLES

ADC multimode delay between two sampling phases: 6 ADC clock cycles for 10 or 8 bits resolution

LL_ADC_MULTI_TWOSMP_DELAY_6CYCLES_5

ADC multimode delay between two sampling phases: 6.5 ADC clock cycles for 16 or 14 bits resolution

LL_ADC_MULTI_TWOSMP_DELAY_6CYCLES_5_12_BITS

ADC multimode delay between two sampling phases: 6.5 ADC clock cycles for 12 bits resolution

LL_ADC_MULTI_TWOSMP_DELAY_7CYCLES_5

ADC multimode delay between two sampling phases: 7.5 ADC clock cycles for 16 bits resolution

LL_ADC_MULTI_TWOSMP_DELAY_8CYCLES

ADC multimode delay between two sampling phases: 8 ADC clock cycles for 12 bits resolution

LL_ADC_MULTI_TWOSMP_DELAY_9CYCLES

ADC multimode delay between two sampling phases: 9 ADC clock cycles for 16 or 14 bits resolution

ADC instance - Offset number

LL_ADC_OFFSET_1

ADC offset number 1: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

LL_ADC_OFFSET_2

ADC offset number 2: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

LL_ADC_OFFSET_3

ADC offset number 3: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

LL_ADC_OFFSET_4

ADC offset number 4: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

ADC instance - Offset right shift

LL_ADC_OFFSET_RSHIFT_DISABLE

ADC offset right shift is disabled (among ADC selected offset number 1, 2, 3 or 4)

LL_ADC_OFFSET_RSHIFT_ENABLE

ADC offset right shift is enabled (among ADC selected offset number 1, 2, 3 or 4)

ADC instance - Offset signed saturation mode

LL_ADC_OFFSET_SIGNED_SATURATION_DISABLE

ADC offset signed saturation is disabled (among ADC selected offset number 1, 2, 3 or 4)

LL_ADC_OFFSET_SIGNED_SATURATION_ENABLE

ADC offset signed saturation is enabled (among ADC selected offset number 1, 2, 3 or 4)

Oversampling - Discontinuous mode

LL_ADC_OVS_REG_CONT

ADC oversampling discontinuous mode: continuous mode (all conversions of oversampling ratio are done from 1 trigger)

LL_ADC_OVS_REG_DISCONT

ADC oversampling discontinuous mode: discontinuous mode (each conversion of oversampling ratio needs a trigger)

Oversampling - Oversampling scope

LL_ADC_OVS_DISABLE

ADC oversampling disabled.

LL_ADC_OVS_GRP_REGULAR_CONTINUED

ADC oversampling on conversions of ADC group regular. If group injected interrupts group regular: when ADC group injected is triggered, the oversampling on ADC group regular is temporary stopped and continued afterwards.

LL_ADC_OVS_GRP_REGULAR_RESUMED

ADC oversampling on conversions of ADC group regular. If group injected interrupts group regular: when ADC group injected is triggered, the oversampling on ADC group regular is resumed from start (oversampler buffer reset).

LL_ADC_OVS_GRP_INJECTED

ADC oversampling on conversions of ADC group injected.

LL_ADC_OVS_GRP_INJ_REG_RESUMED

ADC oversampling on conversions of both ADC groups regular and injected. If group injected interrupting group regular: when ADC group injected is triggered, the oversampling on ADC group regular is resumed from start (oversampler buffer reset).

Oversampling - Data shift

LL_ADC_OVS_SHIFT_NONE

ADC oversampling no shift (sum of the ADC conversions data is not divided to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_1

ADC oversampling shift of 1 (sum of the ADC conversions data is divided by 2 to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_2

ADC oversampling shift of 2 (sum of the ADC conversions data is divided by 4 to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_3

ADC oversampling shift of 3 (sum of the ADC conversions data is divided by 8 to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_4

ADC oversampling shift of 4 (sum of the ADC conversions data is divided by 16 to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_5

ADC oversampling shift of 5 (sum of the ADC conversions data is divided by 32 to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_6

ADC oversampling shift of 6 (sum of the ADC conversions data is divided by 64 to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_7

ADC oversampling shift of 7 (sum of the ADC conversions data is divided by 128 to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_8

ADC oversampling shift of 8 (sum of the ADC conversions data is divided by 256 to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_9

ADC oversampling shift of 9 (sum of the ADC conversions data is divided by 512 to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_10

ADC oversampling shift of 10 (sum of the ADC conversions data is divided by 1024 to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_11

ADC oversampling shift of 11 (sum of the ADC conversions data is divided by 2048 to result as the ADC oversampling conversion data)

ADC registers compliant with specific purpose

LL_ADC_DMA_REG_REGULAR_DATA

LL_ADC_DMA_REG_REGULAR_DATA_MULTI

ADC group regular - Continuous mode

LL_ADC_REG_CONV_SINGLE

ADC conversions are performed in single mode: one conversion per trigger

LL_ADC_REG_CONV_CONTINUOUS

ADC conversions are performed in continuous mode: after the first trigger, following conversions launched successively automatically

ADC group regular - Data transfer mode of ADC conversion data

LL_ADC_REG_DR_TRANSFER

ADC conversions are transferred to DR rigister

LL_ADC_REG_DMA_TRANSFER_LIMITED

ADC conversion data are transferred by DMA, in limited mode (one shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.

LL_ADC_REG_DMA_TRANSFER_UNLIMITED

ADC conversion data are transferred by DMA, in unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transferred (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular.

LL_ADC_REG_DFSDM_TRANSFER

ADC conversion data are transferred to DFSDM

ADC group regular - Overrun behavior on conversion data

LL_ADC_REG_OVR_DATA_PRESERVED

ADC group regular behavior in case of overrun: data preserved

LL_ADC_REG_OVR_DATA_OVERWRITTEN

ADC group regular behavior in case of overrun: data overwritten

ADC group regular - Sequencer discontinuous mode

LL_ADC_REG_SEQ_DISCONT_DISABLE

ADC group regular sequencer discontinuous mode disable

LL_ADC_REG_SEQ_DISCONT_1RANK

ADC group regular sequencer discontinuous mode enable with sequence interruption every rank

LL_ADC_REG_SEQ_DISCONT_2RANKS

ADC group regular sequencer discontinuous mode enabled with sequence interruption every 2 ranks

LL_ADC_REG_SEQ_DISCONT_3RANKS

ADC group regular sequencer discontinuous mode enable with sequence interruption every 3 ranks

LL_ADC_REG_SEQ_DISCONT_4RANKS

ADC group regular sequencer discontinuous mode enable with sequence interruption every 4 ranks

LL_ADC_REG_SEQ_DISCONT_5RANKS

ADC group regular sequencer discontinuous mode enable with sequence interruption every 5 ranks

LL_ADC_REG_SEQ_DISCONT_6RANKS

ADC group regular sequencer discontinuous mode enable with sequence interruption every 6 ranks

LL_ADC_REG_SEQ_DISCONT_7RANKS

ADC group regular sequencer discontinuous mode enable with sequence interruption every 7 ranks

LL_ADC_REG_SEQ_DISCONT_8RANKS

ADC group regular sequencer discontinuous mode enable with sequence interruption every 8 ranks

ADC group regular - Sequencer ranks

LL_ADC_REG_RANK_1

ADC group regular sequencer rank 1

LL_ADC_REG_RANK_2

ADC group regular sequencer rank 2

LL_ADC_REG_RANK_3

ADC group regular sequencer rank 3

LL_ADC_REG_RANK_4

ADC group regular sequencer rank 4

LL_ADC_REG_RANK_5

ADC group regular sequencer rank 5

LL_ADC_REG_RANK_6

ADC group regular sequencer rank 6

LL_ADC_REG_RANK_7

ADC group regular sequencer rank 7

LL_ADC_REG_RANK_8

ADC group regular sequencer rank 8

LL_ADC_REG_RANK_9

ADC group regular sequencer rank 9

LL_ADC_REG_RANK_10

ADC group regular sequencer rank 10

LL_ADC_REG_RANK_11

ADC group regular sequencer rank 11

LL_ADC_REG_RANK_12

ADC group regular sequencer rank 12

LL_ADC_REG_RANK_13

ADC group regular sequencer rank 13

LL_ADC_REG_RANK_14

ADC group regular sequencer rank 14

LL_ADC_REG_RANK_15

ADC group regular sequencer rank 15

LL_ADC_REG_RANK_16

ADC group regular sequencer rank 16

ADC group regular - Sequencer scan length

LL_ADC_REG_SEQ_SCAN_DISABLE

ADC group regular sequencer disable (equivalent to sequencer of 1 rank: ADC conversion on only 1 channel)

LL_ADC_REG_SEQ_SCAN_ENABLE_2RANKS

ADC group regular sequencer enable with 2 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_3RANKS

ADC group regular sequencer enable with 3 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_4RANKS

ADC group regular sequencer enable with 4 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_5RANKS

ADC group regular sequencer enable with 5 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_6RANKS

ADC group regular sequencer enable with 6 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_7RANKS

ADC group regular sequencer enable with 7 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_8RANKS

ADC group regular sequencer enable with 8 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_9RANKS

ADC group regular sequencer enable with 9 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_10RANKS

ADC group regular sequencer enable with 10 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_11RANKS

ADC group regular sequencer enable with 11 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_12RANKS

ADC group regular sequencer enable with 12 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_13RANKS

ADC group regular sequencer enable with 13 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_14RANKS

ADC group regular sequencer enable with 14 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_15RANKS

ADC group regular sequencer enable with 15 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_16RANKS

ADC group regular sequencer enable with 16 ranks in the sequence

ADC group regular - Trigger edge

LL_ADC_REG_TRIG_EXT_RISING

ADC group regular conversion trigger polarity set to rising edge

LL_ADC_REG_TRIG_EXT_FALLING

ADC group regular conversion trigger polarity set to falling edge

LL_ADC_REG_TRIG_EXT_RISINGFALLING

ADC group regular conversion trigger polarity set to both rising and falling edges

ADC group regular - Trigger source

LL_ADC_REG_TRIG_SOFTWARE

ADC group regular conversion trigger internal: SW start.

LL_ADC_REG_TRIG_EXT_TIM1_CH1

ADC group regular conversion trigger from external peripheral: TIM1 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM1_CH2

ADC group regular conversion trigger from external peripheral: TIM1 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM1_CH3

ADC group regular conversion trigger from external peripheral: TIM1 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM2_CH2

ADC group regular conversion trigger from external peripheral: TIM2 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM3_TRGO

ADC group regular conversion trigger from external peripheral: TIM3 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM4_CH4

ADC group regular conversion trigger from external peripheral: TIM4 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_EXTI_LINE11

ADC group regular conversion trigger from external peripheral: external interrupt line 11 event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM8_TRGO

ADC group regular conversion trigger from external peripheral: TIM8 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM8_TRGO2

ADC group regular conversion trigger from external peripheral: TIM8 TRGO2 event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM1_TRGO

ADC group regular conversion trigger from external peripheral: TIM1 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM1_TRGO2

ADC group regular conversion trigger from external peripheral: TIM1 TRGO2 event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM2_TRGO

ADC group regular conversion trigger from external peripheral: TIM2 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM4_TRGO

ADC group regular conversion trigger from external peripheral: TIM4 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM6_TRGO

ADC group regular conversion trigger from external peripheral: TIM6 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM15_TRGO

ADC group regular conversion trigger from external peripheral: TIM15 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM3_CH4

ADC group regular conversion trigger from external peripheral: TIM3 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_HRTIM_TRG1

ADC group regular conversion trigger from external peripheral: HRTIM TRG1 event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_HRTIM_TRG3

ADC group regular conversion trigger from external peripheral: HRTIM TRG2 event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_LPTIM1_OUT

ADC group regular conversion trigger from external peripheral: LPTIM1 OUT event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_LPTIM2_OUT

ADC group regular conversion trigger from external peripheral: LPTIM2 OUT event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_LPTIM3_OUT

ADC group regular conversion trigger from external peripheral: LPTIM3 event OUT. Trigger edge set to rising edge (default setting).

ADC instance - Resolution

LL_ADC_RESOLUTION_16B

ADC resolution 16 bits

LL_ADC_RESOLUTION_14B

ADC resolution 12 bits

LL_ADC_RESOLUTION_12B

ADC resolution 12 bits

LL_ADC_RESOLUTION_10B

ADC resolution 10 bits

LL_ADC_RESOLUTION_14B_OPT

ADC resolution 14 bits optimized for power consumption, available on for devices revision V only

LL_ADC_RESOLUTION_12B_OPT

ADC resolution 12 bits optimized for power consumption, available on for devices revision V only

LL_ADC_RESOLUTION_8B

ADC resolution 8 bits The resolution setting is managed internally in the driver: "LL_ADC_RESOLUTION_8B" definition: keep using the "100b" value (corresponding to STM32H74x/5x rev Y). Rev.V value "111b" is handled through functions "LL_ADC_SetResolution/LL_ADC_GetResolution" with a dedicated check on DBGMCU IDCODE register

ADC helper macro

__LL_ADC_CHANNEL_TO_DECIMAL_NB

Description:

- Helper macro to get ADC channel number in decimal format from literals LL_ADC_CHANNEL_x.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)

Return value:

- Value: between Min_Data=0 and Max_Data=18

Notes:

- Example: __LL_ADC_CHANNEL_TO_DECIMAL_NB(LL_ADC_CHANNEL_4) will return decimal number "4". The input can be a value from functions where a channel number is returned, either defined with number or with bitfield (only one bit must be set).

__LL_ADC_DECIMAL_NB_TO_CHANNEL

Description:

- Helper macro to get ADC channel in literal format LL_ADC_CHANNEL_x from number in decimal format.

Parameters:

- __DECIMAL_NB__: Value between Min_Data=0 and Max_Data=18

Return value:

- Returned: value can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)

Notes:

- Example: __LL_ADC_DECIMAL_NB_TO_CHANNEL(4) will return a data equivalent to "LL_ADC_CHANNEL_4".

__LL_ADC_IS_CHANNEL_INTERNAL

Description:

- Helper macro to determine whether the selected channel corresponds to literal definitions of driver.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)

Return value:

- Value: "0" if the channel corresponds to a parameter definition of a ADC external channel (channel connected to a GPIO pin). Value "1" if the channel corresponds to a parameter definition of a ADC internal channel.

Notes:

- The different literal definitions of ADC channels are: ADC internal channel: LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...ADC external channel (channel connected to a GPIO pin): LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ... The channel parameter must be a value defined from literal definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...), ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...), must not be a value from functions where a channel number is returned from ADC registers, because internal and external channels share the same channel number in ADC registers. The differentiation is made only with parameters definitions of driver.

__LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL

Description:

- Helper macro to convert a channel defined from parameter definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...), to its equivalent parameter definition of a ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...).

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)

Return value:

- Returned: value can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19

Notes:

- The channel parameter can be, additionally to a value defined from parameter definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...), a value defined from parameter definition of ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...) or a value from functions where a channel number is returned from ADC registers.

__LL_ADC_IS_CHANNEL_INTERNAL_AVAILABLE

Description:

- Helper macro to determine whether the internal channel selected is available on the ADC instance selected.

Parameters:

- __ADC_INSTANCE__: ADC instance
- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)

Return value:

- Value: "0" if the internal channel selected is not available on the ADC instance selected. Value "1" if the internal channel selected is available on the ADC instance selected.

Notes:

- The channel parameter must be a value defined from parameter definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...), must not be a value defined from parameter definition of ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...) or a value from functions where a channel number is returned from ADC registers, because internal and external channels share the same channel number in ADC registers. The differentiation is made only with parameters definitions of driver.

__LL_ADC_ANALOGWD_CHANNEL_GROUP

Description:

- Helper macro to define ADC analog watchdog parameter: define a single channel to monitor with analog watchdog from sequencer channel and groups definition.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT (1)
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC2 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC2 (2)
- __GROUP__: This parameter can be one of the following values:
  - LL_ADC_GROUP_REGULAR
  - LL_ADC_GROUP_INJECTED
  - LL_ADC_GROUP_REGULAR_INJECTED

Return value:

- Returned: value can be one of the following values:
  - LL_ADC_AWD_DISABLE
  - LL_ADC_AWD_ALL_CHANNELS_REG (0)
  - LL_ADC_AWD_ALL_CHANNELS_INJ (0)
  - LL_ADC_AWD_ALL_CHANNELS_REG_INJ
  - LL_ADC_AWD_CHANNEL_0_REG (0)
  - LL_ADC_AWD_CHANNEL_0_INJ (0)
  - LL_ADC_AWD_CHANNEL_0_REG_INJ
  - LL_ADC_AWD_CHANNEL_1_REG (0)
  - LL_ADC_AWD_CHANNEL_1_INJ (0)
  - LL_ADC_AWD_CHANNEL_1_REG_INJ
  - LL_ADC_AWD_CHANNEL_2_REG (0)
  - LL_ADC_AWD_CHANNEL_2_INJ (0)
  - LL_ADC_AWD_CHANNEL_2_REG_INJ
  - LL_ADC_AWD_CHANNEL_3_REG (0)
  - LL_ADC_AWD_CHANNEL_3_INJ (0)
  - LL_ADC_AWD_CHANNEL_3_REG_INJ
  - LL_ADC_AWD_CHANNEL_4_REG (0)
  - LL_ADC_AWD_CHANNEL_4_INJ (0)
  - LL_ADC_AWD_CHANNEL_4_REG_INJ
  - LL_ADC_AWD_CHANNEL_5_REG (0)
  - LL_ADC_AWD_CHANNEL_5_INJ (0)
  - LL_ADC_AWD_CHANNEL_5_REG_INJ
  - LL_ADC_AWD_CHANNEL_6_REG (0)
  - LL_ADC_AWD_CHANNEL_6_INJ (0)
  - LL_ADC_AWD_CHANNEL_6_REG_INJ
  - LL_ADC_AWD_CHANNEL_7_REG (0)
  - LL_ADC_AWD_CHANNEL_7_INJ (0)
  - LL_ADC_AWD_CHANNEL_7_REG_INJ
  - LL_ADC_AWD_CHANNEL_8_REG (0)
  - LL_ADC_AWD_CHANNEL_8_INJ (0)
  - LL_ADC_AWD_CHANNEL_8_REG_INJ
  - LL_ADC_AWD_CHANNEL_9_REG (0)
  - LL_ADC_AWD_CHANNEL_9_INJ (0)
  - LL_ADC_AWD_CHANNEL_9_REG_INJ
  - LL_ADC_AWD_CHANNEL_10_REG (0)
  - LL_ADC_AWD_CHANNEL_10_INJ (0)
  - LL_ADC_AWD_CHANNEL_10_REG_INJ
  - LL_ADC_AWD_CHANNEL_11_REG (0)
  - LL_ADC_AWD_CHANNEL_11_INJ (0)
  - LL_ADC_AWD_CHANNEL_11_REG_INJ
  - LL_ADC_AWD_CHANNEL_12_REG (0)
  - LL_ADC_AWD_CHANNEL_12_INJ (0)
  - LL_ADC_AWD_CHANNEL_12_REG_INJ
  - LL_ADC_AWD_CHANNEL_13_REG (0)
  - LL_ADC_AWD_CHANNEL_13_INJ (0)
  - LL_ADC_AWD_CHANNEL_13_REG_INJ
  - LL_ADC_AWD_CHANNEL_14_REG (0)
  - LL_ADC_AWD_CHANNEL_14_INJ (0)
  - LL_ADC_AWD_CHANNEL_14_REG_INJ
  - LL_ADC_AWD_CHANNEL_15_REG (0)
  - LL_ADC_AWD_CHANNEL_15_INJ (0)

Notes:

- To be used with function LL_ADC_SetAnalogWDMonitChannels().
  Example: LL_ADC_SetAnalogWDMonitChannels(ADC1, LL_ADC_AWD1,
  __LL_ADC_ANALOGWD_CHANNEL_GROUP(LL_ADC_CHANNEL4, LL_ADC_GROUP_REGULAR))

__LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION

Description:

- Helper macro to set the value of ADC analog watchdog threshold high or low in function of ADC resolution, when ADC resolution is different of 16 bits.

Parameters:

- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_16B
  - LL_ADC_RESOLUTION_14B
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
- __AWD_THRESHOLD__: Value between Min_Data=0x000000 and Max_Data=0xFFFFFF

Return value:

- Value: between Min_Data=0x000000 and Max_Data=0xFFFFFF

Notes:

- To be used with function LL_ADC_SetAnalogWDThresholds(). Example, with a ADC resolution of 8 bits, to set the value of analog watchdog threshold high (on 18 bits): LL_ADC_SetAnalogWDThresholds (< ADCx param >, __LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION(LL_ADC_RESOLUTION_8B, <threshold_value_18_bits>));

__LL_ADC_ANALOGWD_GET_THRESHOLD_RESOLUTION

Description:

- Helper macro to get the value of ADC analog watchdog threshold high or low in function of ADC resolution, when ADC resolution is different of 16 bits.

Parameters:

- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_16B
  - LL_ADC_RESOLUTION_14B
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
- __AWD_THRESHOLD_16_BITS__: Value between Min_Data=0x000000 and Max_Data=0xFFFFFF

Return value:

- Value: between Min_Data=0x000000 and Max_Data=0xFFFFFF

Notes:

- To be used with function LL_ADC_GetAnalogWDThresholds(). Example, with a ADC resolution of 8 bits, to get the value of analog watchdog threshold high (on 18 bits): < threshold_value_18_bits > = __LL_ADC_ANALOGWD_GET_THRESHOLD_RESOLUTION (LL_ADC_RESOLUTION_8B, LL_ADC_GetAnalogWDThresholds(<ADCx param>, LL_ADC_AWD_THRESHOLD_HIGH));

__LL_ADC_CALIB_FACTOR_SINGLE_DIFF

Description:

- Helper macro to set the ADC calibration value with both single ended and differential modes calibration factors concatenated.

Parameters:

- __CALIB_FACTOR_SINGLE_ENDED__: Value between Min_Data=0x00 and Max_Data=0x7F
- __CALIB_FACTOR_DIFFERENTIAL__: Value between Min_Data=0x00 and Max_Data=0x7F

Return value:

- Value: between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Notes:

- To be used with function LL_ADC_SetCalibrationOffsetFactor(). Example, to set calibration factors single ended to 0x55 and differential ended to 0x2A: LL_ADC_SetCalibrationOffsetFactor(ADC1, __LL_ADC_CALIB_FACTOR_SINGLE_DIFF(0x55, 0x2A))

__LL_ADC_MULTI_CONV_DATA_MASTER_SLAVE

Description:

- Helper macro to get the ADC multimode conversion data of ADC master or ADC slave from raw value with both ADC conversion data concatenated.

Parameters:

- __ADC_MULTI_MASTER_SLAVE__: This parameter can be one of the following values:
  - LL_ADC_MULTI_MASTER
  - LL_ADC_MULTI_SLAVE
- __ADC_MULTI_CONV_DATA__: Value between Min_Data=0x000 and Max_Data=0xFFF

Return value:

- Value: between Min_Data=0x000 and Max_Data=0xFFF

Notes:

- This macro is intended to be used when multimode transfer by DMA is enabled: refer to function LL_ADC_SetMultiDMATransfer(). In this case the transferred data need to processed with this macro to separate the conversion data of ADC master and ADC slave.

__LL_ADC_MULTI_INSTANCE_MASTER

Description:

- Helper macro to select, from a ADC instance, to which ADC instance it has a dependence in multimode (ADC master of the corresponding ADC common instance).

Parameters:

- __ADCx__: ADC instance

Return value:

- __ADCx__: ADC instance master of the corresponding ADC common instance

Notes:

- In case of device with multimode available and a mix of ADC instances compliant and not compliant with multimode feature, ADC instances not compliant with multimode feature are considered as master instances (do not depend to any other ADC instance).

__LL_ADC_COMMON_INSTANCE

Description:

- Helper macro to select the ADC common instance to which is belonging the selected ADC instance.

Parameters:

- __ADCx__: ADC instance

Return value:

- ADC: common register instance

Notes:

- ADC common register instance can be used for: Set parameters common to several ADC instancesMultimode (for devices with several ADC instances) Refer to functions having argument "ADCxy_COMMON" as parameter.

__LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE

Description:

- Helper macro to check if all ADC instances sharing the same ADC common instance are disabled.

Parameters:

- __ADCXY_COMMON__: ADC common instance (can be set directly from CMSIS definition or by using helper macro

Return value:

- Value: "0" if all ADC instances sharing the same ADC common instance are disabled. Value "1" if at least one ADC instance sharing the same ADC common instance is enabled.

Notes:

- This check is required by functions with setting conditioned to ADC state: All ADC instances of the ADC common group must be disabled. Refer to functions having argument "ADCxy_COMMON" as parameter. On devices with only 1 ADC common instance, parameter of this macro is useless and can be ignored (parameter kept for compatibility with devices featuring several ADC common instances).

__LL_ADC_DIGITAL_SCALE

Description:

- Helper macro to define the ADC conversion data full-scale digital value corresponding to the selected ADC resolution.

Parameters:

- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_16B
  - LL_ADC_RESOLUTION_14B
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B

Return value:

- ADC: conversion data full-scale digital value (unit: digital value of ADC conversion data)

Notes:

- ADC conversion data full-scale corresponds to voltage range determined by analog voltage references Vref+ and Vref- (refer to reference manual).

__LL_ADC_CONVERT_DATA_RESOLUTION

Description:

- Helper macro to convert the ADC conversion data from a resolution to another resolution.

Parameters:

- __DATA__: ADC conversion data to be converted
- __ADC_RESOLUTION_CURRENT__: Resolution of the data to be converted This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_16B
  - LL_ADC_RESOLUTION_14B
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
- __ADC_RESOLUTION_TARGET__: Resolution of the data after conversion This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_16B
  - LL_ADC_RESOLUTION_14B
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B

Return value:

- ADC: conversion data to the requested resolution

__LL_ADC_CALC_DATA_TO_VOLTAGE

Description:

- Helper macro to calculate the voltage (unit: mVolt) corresponding to a ADC conversion data (unit: digital value).

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __ADC_DATA__: ADC conversion data (resolution 16 bits) (unit: digital value).
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_16B
  - LL_ADC_RESOLUTION_14B
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B

Return value:

- ADC: conversion data equivalent voltage value (unit: mVolt)

Notes:

- Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE().

__LL_ADC_CALC_VREFANALOG_VOLTAGE

Description:

- Helper macro to calculate analog reference voltage (Vref+) (unit: mVolt) from ADC conversion data of internal voltage reference VrefInt.

Parameters:

- __VREFINT_ADC_DATA__: ADC conversion data (resolution 16 bits) of internal voltage reference VrefInt (unit: digital value).
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_16B
  - LL_ADC_RESOLUTION_14B
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B

Return value:

- Analog: reference voltage (unit: mV)

Notes:

- Computation is using VrefInt calibration value stored in system memory for each device during production. This voltage depends on user board environment: voltage level connected to pin Vref+. On devices with small package, the pin Vref+ is not present and internally bonded to pin Vdda. On this STM32 series, calibration data of internal voltage reference VrefInt corresponds to a resolution of 16 bits, this is the recommended ADC resolution to convert voltage of internal voltage reference VrefInt. Otherwise, this macro performs the processing to scale ADC conversion data to 16 bits.

__LL_ADC_CALC_TEMPERATURE

Description:

- Helper macro to calculate the temperature (unit: degree Celsius) from ADC conversion data of internal temperature sensor.

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __TEMPSENSOR_ADC_DATA__: ADC conversion data of internal temperature sensor (unit: digital value).
- __ADC_RESOLUTION__: ADC resolution at which internal temperature sensor voltage has been measured. This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_16B
  - LL_ADC_RESOLUTION_14B
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B

Return value:

- Temperature: (unit: degree Celsius)

Notes:

- Computation is using temperature sensor calibration values stored in system memory for each device during production. Calculation formula: Temperature = ((TS_ADC_DATA - TS_CAL1) * (TS_CAL2_TEMP - TS_CAL1_TEMP)) / (TS_CAL2 - TS_CAL1) + TS_CAL1_TEMP with TS_ADC_DATA = temperature sensor raw data measured by ADC Avg_Slope = (TS_CAL2 - TS_CAL1) / (TS_CAL2_TEMP - TS_CAL1_TEMP) TS_CAL1 = equivalent TS_ADC_DATA at temperature TEMP_DEGC_CAL1 (calibrated in factory) TS_CAL2 = equivalent TS_ADC_DATA at temperature TEMP_DEGC_CAL2 (calibrated in factory) Caution: Calculation relevancy under reserve that calibration parameters are correct (address and data). To calculate temperature using temperature sensor datasheet typical values (generic values less, therefore less accurate than calibrated values), use helper macro __LL_ADC_CALC_TEMPERATURE_TYP_PARAMS(). As calculation input, the analog reference voltage (Vref+) must be defined as it impacts the ADC LSB equivalent voltage. Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE(). On this STM32 series, calibration data of temperature sensor corresponds to a resolution of 16 bits, this is the recommended ADC resolution to convert voltage of temperature sensor. Otherwise, this macro performs the processing to scale ADC conversion data to 16 bits.

__LL_ADC_CALC_TEMPERATURE_TYP_PARAMS

Description:

- Helper macro to calculate the temperature (unit: degree Celsius) from ADC conversion data of internal temperature sensor.

Parameters:

- __TEMPSENSOR_TYP_AVGSLOPE__: Device datasheet data: Temperature sensor slope typical value (unit: uV/DegCelsius). On STM32H7, refer to device datasheet parameter "Avg_Slope".
- __TEMPSENSOR_TYP_CALX_V__: Device datasheet data: Temperature sensor voltage typical value (at temperature and Vref+ defined in parameters below) (unit: mV). On STM32H7, refer to device datasheet parameter "V30" (corresponding to TS_CAL1).
- __TEMPSENSOR_CALX_TEMP__: Device datasheet data: Temperature at which temperature sensor voltage (see parameter above) is corresponding (unit: mV)
- __VREFANALOG_VOLTAGE__: Analog voltage reference (Vref+) voltage (unit: mV)
- __TEMPSENSOR_ADC_DATA__: ADC conversion data of internal temperature sensor (unit: digital value).
- __ADC_RESOLUTION__: ADC resolution at which internal temperature sensor voltage has been measured. This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_16B
  - LL_ADC_RESOLUTION_14B
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B

Return value:

- Temperature: (unit: degree Celsius)

Notes:

- Computation is using temperature sensor typical values (refer to device datasheet). Calculation formula: Temperature = (TS_TYP_CALx_VOLT(uV) - TS_ADC_DATA * Conversion_uV) / Avg_Slope + CALx_TEMP with TS_ADC_DATA = temperature sensor raw data measured by ADC (unit: digital value) Avg_Slope = temperature sensor slope (unit: uV/Degree Celsius) TS_TYP_CALx_VOLT = temperature sensor digital value at temperature CALx_TEMP (unit: mV) Caution: Calculation relevancy under reserve the temperature sensor of the current device has characteristics in line with datasheet typical values. If temperature sensor calibration values are available on on this device (presence of macro __LL_ADC_CALC_TEMPERATURE()), temperature calculation will be more accurate using helper macro __LL_ADC_CALC_TEMPERATURE(). As calculation input, the analog reference voltage (Vref+) must be defined as it impacts the ADC LSB equivalent voltage. Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE(). ADC measurement data must correspond to a resolution of 16 bits (full scale digital value 4095). If not the case, the data must be preliminarily rescaled to an equivalent resolution of 16 bits.

Common write and read registers Macros

LL_ADC_WriteReg

Description:

- Write a value in ADC register.

Parameters:

- __INSTANCE__: ADC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_ADC_ReadReg

Description:

- Read a value in ADC register.

Parameters:

- __INSTANCE__: ADC Instance
- __REG__: Register to be read

Return value:

- Register: value

96 LL BDMA Generic Driver

96.1 BDMA Firmware driver registers structures

96.1.1 LL_BDMA_InitTypeDef

LL_BDMA_InitTypeDef is defined in the stm32h7xx_ll_bdma.h

Data Fields

- ***uint32_t PeriphOrM2MSrcAddress***
- ***uint32_t MemoryOrM2MDstAddress***
- ***uint32_t Direction***
- ***uint32_t Mode***
- ***uint32_t PeriphOrM2MSrcIncMode***
- ***uint32_t MemoryOrM2MDstIncMode***
- ***uint32_t PeriphOrM2MSrcDataSize***
- ***uint32_t MemoryOrM2MDstDataSize***
- ***uint32_t NbData***
- ***uint32_t PeriphRequest***
- ***uint32_t Priority***

Field Documentation

- ***uint32_t LL_BDMA_InitTypeDef::PeriphOrM2MSrcAddress***

  Specifies the peripheral base address for BDMA transfer or as Source base address in case of memory to memory transfer direction.This parameter must be a value between Min_Data = 0 and Max_Data = 0xFFFFFFFF.

- ***uint32_t LL_BDMA_InitTypeDef::MemoryOrM2MDstAddress***

  Specifies the memory base address for DMA transfer or as Destination base address in case of memory to memory transfer direction.This parameter must be a value between Min_Data = 0 and Max_Data = 0xFFFFFFFF.

- ***uint32_t LL_BDMA_InitTypeDef::Direction***

  Specifies if the data will be transferred from memory to peripheral, from memory to memory or from peripheral to memory. This parameter can be a value of ***BDMA_LL_EC_DIRECTION***This feature can be modified afterwards using unitary function **LL_BDMA_SetDataTransferDirection()**.

- ***uint32_t LL_BDMA_InitTypeDef::Mode***

  Specifies the normal or circular operation mode. This parameter can be a value of ***BDMA_LL_EC_MODE***

  Note:

  – : The circular buffer mode cannot be used if the memory to memory data transfer direction is configured on the selected Channel

  This feature can be modified afterwards using unitary function **LL_BDMA_SetMode()**.

- ***uint32_t LL_BDMA_InitTypeDef::PeriphOrM2MSrcIncMode***

  Specifies whether the Peripheral address or Source address in case of memory to memory transfer direction is incremented or not. This parameter can be a value of ***BDMA_LL_EC_PERIPH***This feature can be modified afterwards using unitary function **LL_BDMA_SetPeriphIncMode()**.

- ***uint32_t LL_BDMA_InitTypeDef::MemoryOrM2MDstIncMode***

  Specifies whether the Memory address or Destination address in case of memory to memory transfer direction is incremented or not. This parameter can be a value of ***BDMA_LL_EC_MEMORY***This feature can be modified afterwards using unitary function **LL_BDMA_SetMemoryIncMode()**.

- ***uint32_t LL_BDMA_InitTypeDef::PeriphOrM2MSrcDataSize***

  Specifies the Peripheral data size alignment or Source data size alignment (byte, half word, word) in case of memory to memory transfer direction. This parameter can be a value of ***BDMA_LL_EC_PDATAALIGN***This feature can be modified afterwards using unitary function **LL_BDMA_SetPeriphSize()**.

- ***uint32_t LL_BDMA_InitTypeDef::MemoryOrM2MDstDataSize***

  Specifies the Memory data size alignment or Destination data size alignment (byte, half word, word) in case of memory to memory transfer direction. This parameter can be a value of ***BDMA_LL_EC_MDATAALIGN***This feature can be modified afterwards using unitary function **LL_BDMA_SetMemorySize()**.
- ***uint32_t LL_BDMA_InitTypeDef::NbData***

  Specifies the number of data to transfer, in data unit. The data unit is equal to the source buffer configuration set in PeripheralSize or MemorySize parameters depending in the transfer direction. This parameter must be a value between Min_Data = 0 and Max_Data = 0x0000FFFFThis feature can be modified afterwards using unitary function **LL_BDMA_SetDataLength()**.
- ***uint32_t LL_BDMA_InitTypeDef::PeriphRequest***

  Specifies the peripheral request. This parameter can be a value of ***DMAMUX2_Request_selection***This feature can be modified afterwards using unitary function **LL_BDMA_SetPeriphRequest()**.
- ***uint32_t LL_BDMA_InitTypeDef::Priority***

  Specifies the channel priority level. This parameter can be a value of ***BDMA_LL_EC_PRIORITY***This feature can be modified afterwards using unitary function **LL_BDMA_SetChannelPriorityLevel()**.

96.2 BDMA Firmware driver API description

The following section lists the various functions of the BDMA library.

96.2.1 Detailed description of functions

LL_BDMA_EnableChannel

Function name

__STATIC_INLINE void LL_BDMA_EnableChannel (BDMA_TypeDef * BDMAx, uint32_t Channel)

Function description

Enable BDMA channel.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR EN LL_BDMA_EnableChannel

LL_BDMA_DisableChannel

Function name

__STATIC_INLINE void LL_BDMA_DisableChannel (BDMA_TypeDef * BDMAx, uint32_t Channel)

Function description

Disable BDMA channel.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR EN LL_BDMA_DisableChannel

LL_BDMA_IsEnabledChannel

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsEnabledChannel (BDMA_TypeDef * BDMAx, uint32_t Channel)

Function description

Check if BDMA channel is enabled or disabled.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCR EN LL_BDMA_IsEnabledChannel

LL_BDMA_ConfigTransfer

Function name

__STATIC_INLINE void LL_BDMA_ConfigTransfer (BDMA_TypeDef * BDMAx, uint32_t Channel, uint32_t Configuration)

Function description

Configure all parameters link to BDMA transfer.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_BDMA_DIRECTION_PERIPH_TO_MEMORY or LL_BDMA_DIRECTION_MEMORY_TO_PERIPH or LL_BDMA_DIRECTION_MEMORY_TO_MEMORY
  - LL_BDMA_MODE_NORMAL or LL_BDMA_MODE_CIRCULAR
  - LL_BDMA_PERIPH_INCREMENT or LL_BDMA_PERIPH_NOINCREMENT
  - LL_BDMA_MEMORY_INCREMENT or LL_BDMA_MEMORY_NOINCREMENT
  - LL_BDMA_PDATAALIGN_BYTE or LL_BDMA_PDATAALIGN_HALFWORD or LL_BDMA_PDATAALIGN_WORD
  - LL_BDMA_MDATAALIGN_BYTE or LL_BDMA_MDATAALIGN_HALFWORD or LL_BDMA_MDATAALIGN_WORD
  - LL_BDMA_PRIORITY_LOW or LL_BDMA_PRIORITY_MEDIUM or LL_BDMA_PRIORITY_HIGH or LL_BDMA_PRIORITY_VERYHIGH

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR DIR LL_BDMA_ConfigTransfer
- CCR MEM2MEM LL_BDMA_ConfigTransfer
- CCR CIRC LL_BDMA_ConfigTransfer
- CCR PINC LL_BDMA_ConfigTransfer
- CCR MINC LL_BDMA_ConfigTransfer
- CCR PSIZE LL_BDMA_ConfigTransfer
- CCR MSIZE LL_BDMA_ConfigTransfer
- CCR PL LL_BDMA_ConfigTransfer

LL_BDMA_SetDataTransferDirection

Function name

__STATIC_INLINE void LL_BDMA_SetDataTransferDirection (BDMA_TypeDef * BDMAx, uint32_t Channel, uint32_t Direction)

Function description

Set Data transfer direction (read from peripheral or from memory).

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7
- **Direction:** This parameter can be one of the following values:
  - LL_BDMA_DIRECTION_PERIPH_TO_MEMORY
  - LL_BDMA_DIRECTION_MEMORY_TO_PERIPH
  - LL_BDMA_DIRECTION_MEMORY_TO_MEMORY

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR DIR LL_BDMA_SetDataTransferDirection
- CCR MEM2MEM LL_BDMA_SetDataTransferDirection

LL_BDMA_GetDataTransferDirection

Function name

__STATIC_INLINE uint32_t LL_BDMA_GetDataTransferDirection (BDMA_TypeDef * BDMAx, uint32_t Channel)

Function description

Get Data transfer direction (read from peripheral or from memory).

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_BDMA_DIRECTION_PERIPH_TO_MEMORY
  - LL_BDMA_DIRECTION_MEMORY_TO_PERIPH
  - LL_BDMA_DIRECTION_MEMORY_TO_MEMORY

Reference Manual to LL API cross reference:

- CCR DIR LL_BDMA_GetDataTransferDirection
- CCR MEM2MEM LL_BDMA_GetDataTransferDirection

LL_BDMA_SetMode

Function name

__STATIC_INLINE void LL_BDMA_SetMode (BDMA_TypeDef * BDMAx, uint32_t Channel, uint32_t Mode)

Function description

Set BDMA mode circular or normal.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7
- **Mode:** This parameter can be one of the following values:
  - LL_BDMA_MODE_NORMAL
  - LL_BDMA_MODE_CIRCULAR

Return values

- **None:**

Notes

- The circular buffer mode cannot be used if the memory-to-memory data transfer is configured on the selected Channel.

Reference Manual to LL API cross reference:

- CCR CIRC LL_BDMA_SetMode

LL_BDMA_GetMode

Function name

__STATIC_INLINE uint32_t LL_BDMA_GetMode (BDMA_TypeDef * BDMAx, uint32_t Channel)

Function description

Get BDMA mode circular or normal.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_BDMA_MODE_NORMAL
  - LL_BDMA_MODE_CIRCULAR

Reference Manual to LL API cross reference:

- CCR CIRC LL_BDMA_GetMode

LL_BDMA_SetPeriphIncMode

Function name

__STATIC_INLINE void LL_BDMA_SetPeriphIncMode (BDMA_TypeDef * BDMAx, uint32_t Channel, uint32_t PeriphOrM2MSrcIncMode)

Function description

Set Peripheral increment mode.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7
- **PeriphOrM2MSrcIncMode:** This parameter can be one of the following values:
  - LL_BDMA_PERIPH_INCREMENT
  - LL_BDMA_PERIPH_NOINCREMENT

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR PINC LL_BDMA_SetPeriphIncMode

LL_BDMA_GetPeriphIncMode

Function name

__STATIC_INLINE uint32_t LL_BDMA_GetPeriphIncMode (BDMA_TypeDef * BDMAx, uint32_t Channel)

Function description

Get Peripheral increment mode.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_BDMA_PERIPH_INCREMENT
  - LL_BDMA_PERIPH_NOINCREMENT

Reference Manual to LL API cross reference:

- CCR PINC LL_BDMA_GetPeriphIncMode

LL_BDMA_SetMemoryIncMode

Function name

__STATIC_INLINE void LL_BDMA_SetMemoryIncMode (BDMA_TypeDef * BDMAx, uint32_t Channel, uint32_t MemoryOrM2MDstIncMode)

Function description

Set Memory increment mode.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7
- **MemoryOrM2MDstIncMode:** This parameter can be one of the following values:
  - LL_BDMA_MEMORY_INCREMENT
  - LL_BDMA_MEMORY_NOINCREMENT

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR MINC LL_BDMA_SetMemoryIncMode

LL_BDMA_GetMemoryIncMode

Function name

__STATIC_INLINE uint32_t LL_BDMA_GetMemoryIncMode (BDMA_TypeDef * BDMAx, uint32_t Channel)

Function description

Get Memory increment mode.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_BDMA_MEMORY_INCREMENT
  - LL_BDMA_MEMORY_NOINCREMENT

Reference Manual to LL API cross reference:

- CCR MINC LL_BDMA_GetMemoryIncMode

LL_BDMA_SetPeriphSize

Function name

__STATIC_INLINE void LL_BDMA_SetPeriphSize (BDMA_TypeDef * BDMAx, uint32_t Channel, uint32_t PeriphOrM2MSrcDataSize)

Function description

Set Peripheral size.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7
- **PeriphOrM2MSrcDataSize:** This parameter can be one of the following values:
  - LL_BDMA_PDATAALIGN_BYTE
  - LL_BDMA_PDATAALIGN_HALFWORD
  - LL_BDMA_PDATAALIGN_WORD

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR PSIZE LL_BDMA_SetPeriphSize

LL_BDMA_GetPeriphSize

Function name

__STATIC_INLINE uint32_t LL_BDMA_GetPeriphSize (BDMA_TypeDef * BDMAx, uint32_t Channel)

Function description

Get Peripheral size.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_BDMA_PDATAALIGN_BYTE
  - LL_BDMA_PDATAALIGN_HALFWORD
  - LL_BDMA_PDATAALIGN_WORD

Reference Manual to LL API cross reference:

- CCR PSIZE LL_BDMA_GetPeriphSize

LL_BDMA_SetMemorySize

Function name

__STATIC_INLINE void LL_BDMA_SetMemorySize (BDMA_TypeDef * BDMAx, uint32_t Channel, uint32_t MemoryOrM2MDstDataSize)

Function description

Set Memory size.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7
- **MemoryOrM2MDstDataSize:** This parameter can be one of the following values:
  - LL_BDMA_MDATAALIGN_BYTE
  - LL_BDMA_MDATAALIGN_HALFWORD
  - LL_BDMA_MDATAALIGN_WORD

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR MSIZE LL_BDMA_SetMemorySize

LL_BDMA_GetMemorySize

Function name

__STATIC_INLINE uint32_t LL_BDMA_GetMemorySize (BDMA_TypeDef * BDMAx, uint32_t Channel)

Function description

Get Memory size.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_BDMA_MDATAALIGN_BYTE
  - LL_BDMA_MDATAALIGN_HALFWORD
  - LL_BDMA_MDATAALIGN_WORD

Reference Manual to LL API cross reference:

- CCR MSIZE LL_BDMA_GetMemorySize

LL_BDMA_SetChannelPriorityLevel

Function name

__STATIC_INLINE void LL_BDMA_SetChannelPriorityLevel (BDMA_TypeDef * BDMAx, uint32_t Channel, uint32_t Priority)

Function description

Set Channel priority level.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7
- **Priority:** This parameter can be one of the following values:
  - LL_BDMA_PRIORITY_LOW
  - LL_BDMA_PRIORITY_MEDIUM
  - LL_BDMA_PRIORITY_HIGH
  - LL_BDMA_PRIORITY_VERYHIGH

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR PL LL_BDMA_SetChannelPriorityLevel

LL_BDMA_GetChannelPriorityLevel

Function name

__STATIC_INLINE uint32_t LL_BDMA_GetChannelPriorityLevel (BDMA_TypeDef * BDMAx, uint32_t Channel)

Function description

Get Channel priority level.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_BDMA_PRIORITY_LOW
  - LL_BDMA_PRIORITY_MEDIUM
  - LL_BDMA_PRIORITY_HIGH
  - LL_BDMA_PRIORITY_VERYHIGH

Reference Manual to LL API cross reference:

- CCR PL LL_BDMA_GetChannelPriorityLevel

LL_BDMA_SetDataLength

Function name

__STATIC_INLINE void LL_BDMA_SetDataLength (BDMA_TypeDef * BDMAx, uint32_t Channel, uint32_t NbData)

Function description

Set Number of data to transfer.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7
- **NbData:** Between Min_Data = 0 and Max_Data = 0x0000FFFF

Return values

- **None:**

Notes

- This action has no effect if channel is enabled.

Reference Manual to LL API cross reference:

- CNDTR NDT LL_BDMA_SetDataLength

LL_BDMA_GetDataLength

Function name

__STATIC_INLINE uint32_t LL_BDMA_GetDataLength (BDMA_TypeDef * BDMAx, uint32_t Channel)

Function description

Get Number of data to transfer.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7

Return values

- **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF

Notes

- Once the channel is enabled, the return value indicate the remaining bytes to be transmitted.

Reference Manual to LL API cross reference:

- CNDTR NDT LL_BDMA_GetDataLength

LL_BDMA_SetCurrentTargetMem

Function name

__STATIC_INLINE void LL_BDMA_SetCurrentTargetMem (BDMA_TypeDef * BDMAx, uint32_t Channel, uint32_t CurrentMemory)

Function description

Set Current target (only in double buffer mode) to Memory 1 or Memory 0.

Parameters

- **BDMAx:** BDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7
- **CurrentMemory:** This parameter can be one of the following values:
  - LL_BDMA_CURRENTTARGETMEM0
  - LL_BDMA_CURRENTTARGETMEM1

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CT LL_BDMA_SetCurrentTargetMem

LL_BDMA_GetCurrentTargetMem

Function name

__STATIC_INLINE uint32_t LL_BDMA_GetCurrentTargetMem (BDMA_TypeDef * BDMAx, uint32_t Channel)

Function description

Set Current target (only in double buffer mode) to Memory 1 or Memory 0.

Parameters

- **BDMAx:** BDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_BDMA_CURRENTTARGETMEM0
  - LL_BDMA_CURRENTTARGETMEM1

Reference Manual to LL API cross reference:

- CR CT LL_BDMA_GetCurrentTargetMem

LL_BDMA_EnableDoubleBufferMode

Function name

__STATIC_INLINE void LL_BDMA_EnableDoubleBufferMode (BDMA_TypeDef * BDMAx, uint32_t Channel)

Function description

Enable the double buffer mode.

Parameters

- **BDMAx:** BDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DBM LL_BDMA_EnableDoubleBufferMode

LL_BDMA_DisableDoubleBufferMode

Function name

__STATIC_INLINE void LL_BDMA_DisableDoubleBufferMode (BDMA_TypeDef * BDMAx, uint32_t Channel)

Function description

Disable the double buffer mode.

Parameters

- **BDMAx:** BDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DBM LL_BDMA_DisableDoubleBufferMode

LL_BDMA_ConfigAddresses

Function name

__STATIC_INLINE void LL_BDMA_ConfigAddresses (BDMA_TypeDef * BDMAx, uint32_t Channel, uint32_t SrcAddress, uint32_t DstAddress, uint32_t Direction)

Function description

Configure the Source and Destination addresses.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7
- **SrcAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF
- **DstAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF
- **Direction:** This parameter can be one of the following values:
  - LL_BDMA_DIRECTION_PERIPH_TO_MEMORY
  - LL_BDMA_DIRECTION_MEMORY_TO_PERIPH
  - LL_BDMA_DIRECTION_MEMORY_TO_MEMORY

Return values

- **None:**

Notes

- This API must not be called when the BDMA channel is enabled.
- Each IP using BDMA provides an API to get directly the register address (LL_PPP_BDMA_GetRegAddr).

Reference Manual to LL API cross reference:

- CPAR PA LL_BDMA_ConfigAddresses
- CMAR MA LL_BDMA_ConfigAddresses

LL_BDMA_SetMemoryAddress

Function name

__STATIC_INLINE void LL_BDMA_SetMemoryAddress (BDMA_TypeDef * BDMAx, uint32_t Channel, uint32_t MemoryAddress)

Function description

Set the Memory address.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7
- **MemoryAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF

Return values

- **None:**

Notes

- Interface used for direction LL_BDMA_DIRECTION_PERIPH_TO_MEMORY or LL_BDMA_DIRECTION_MEMORY_TO_PERIPH only.
- This API must not be called when the BDMA channel is enabled.

Reference Manual to LL API cross reference:

- CMAR MA LL_BDMA_SetMemoryAddress

LL_BDMA_SetPeriphAddress

Function name

__STATIC_INLINE void LL_BDMA_SetPeriphAddress (BDMA_TypeDef * BDMAx, uint32_t Channel, uint32_t PeriphAddress)

Function description

Set the Peripheral address.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7
- **PeriphAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF

Return values

- **None:**

Notes

- Interface used for direction LL_BDMA_DIRECTION_PERIPH_TO_MEMORY or LL_BDMA_DIRECTION_MEMORY_TO_PERIPH only.
- This API must not be called when the BDMA channel is enabled.

Reference Manual to LL API cross reference:

- CPAR PA LL_BDMA_SetPeriphAddress

LL_BDMA_GetMemoryAddress

Function name

__STATIC_INLINE uint32_t LL_BDMA_GetMemoryAddress (BDMA_TypeDef * BDMAx, uint32_t Channel)

Function description

Get Memory address.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7

Return values

- **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF

Notes

- Interface used for direction LL_BDMA_DIRECTION_PERIPH_TO_MEMORY or LL_BDMA_DIRECTION_MEMORY_TO_PERIPH only.

Reference Manual to LL API cross reference:

- CMAR MA LL_BDMA_GetMemoryAddress

LL_BDMA_GetPeriphAddress

Function name

__STATIC_INLINE uint32_t LL_BDMA_GetPeriphAddress (BDMA_TypeDef * BDMAx, uint32_t Channel)

Function description

Get Peripheral address.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7

Return values

- **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF

Notes

- Interface used for direction LL_BDMA_DIRECTION_PERIPH_TO_MEMORY or LL_BDMA_DIRECTION_MEMORY_TO_PERIPH only.

Reference Manual to LL API cross reference:

- CPAR PA LL_BDMA_GetPeriphAddress

LL_BDMA_SetM2MSrcAddress

Function name

__STATIC_INLINE void LL_BDMA_SetM2MSrcAddress (BDMA_TypeDef * BDMAx, uint32_t Channel, uint32_t MemoryAddress)

Function description

Set the Memory to Memory Source address.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7
- **MemoryAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF

Return values

- **None:**

Notes

- Interface used for direction LL_BDMA_DIRECTION_MEMORY_TO_MEMORY only.
- This API must not be called when the BDMA channel is enabled.

Reference Manual to LL API cross reference:

- CPAR PA LL_BDMA_SetM2MSrcAddress

LL_BDMA_SetM2MDstAddress

Function name

__STATIC_INLINE void LL_BDMA_SetM2MDstAddress (BDMA_TypeDef * BDMAx, uint32_t Channel, uint32_t MemoryAddress)

Function description

Set the Memory to Memory Destination address.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7
- **MemoryAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF

Return values

- **None:**

Notes

- Interface used for direction LL_BDMA_DIRECTION_MEMORY_TO_MEMORY only.
- This API must not be called when the BDMA channel is enabled.

Reference Manual to LL API cross reference:

- CMAR MA LL_BDMA_SetM2MDstAddress

LL_BDMA_GetM2MSrcAddress

Function name

__STATIC_INLINE uint32_t LL_BDMA_GetM2MSrcAddress (BDMA_TypeDef * BDMAx, uint32_t Channel)

Function description

Get the Memory to Memory Source address.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7

Return values

- **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF

Notes

- Interface used for direction LL_BDMA_DIRECTION_MEMORY_TO_MEMORY only.

Reference Manual to LL API cross reference:

- CPAR PA LL_BDMA_GetM2MSrcAddress

LL_BDMA_GetM2MDstAddress

Function name

__STATIC_INLINE uint32_t LL_BDMA_GetM2MDstAddress (BDMA_TypeDef * BDMAx, uint32_t Channel)

Function description

Get the Memory to Memory Destination address.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7

Return values

- **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF

Notes

- Interface used for direction LL_BDMA_DIRECTION_MEMORY_TO_MEMORY only.

Reference Manual to LL API cross reference:

- CMAR MA LL_BDMA_GetM2MDstAddress

LL_BDMA_SetMemory1Address

Function name

__STATIC_INLINE void LL_BDMA_SetMemory1Address (BDMA_TypeDef * BDMAx, uint32_t Channel, uint32_t Address)

Function description

Set Memory 1 address (used in case of Double buffer mode).

Parameters

- **BDMAx:** BDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7
- **Address:** Between 0 to 0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- M1AR M1A LL_BDMA_SetMemory1Address

LL_BDMA_GetMemory1Address

Function name

__STATIC_INLINE uint32_t LL_BDMA_GetMemory1Address (BDMA_TypeDef * BDMAx, uint32_t Channel)

Function description

Get Memory 1 address (used in case of Double buffer mode).

Parameters

- **BDMAx:** BDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7

Return values

- **Between:** 0 to 0xFFFFFFFF

Reference Manual to LL API cross reference:

- M1AR M1A LL_BDMA_GetMemory1Address

LL_BDMA_SetPeriphRequest

Function name

__STATIC_INLINE void LL_BDMA_SetPeriphRequest (BDMA_TypeDef * BDMAx, uint32_t Channel, uint32_t Request)

Function description

Set BDMA request for BDMA Channels on DMAMUX Channel x.

Parameters

- **BDMAx:** BDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7
- **Request:** This parameter can be one of the following values:
  - LL_DMAMUX2_REQ_MEM2MEM
  - LL_DMAMUX2_REQ_GENERATOR0
  - LL_DMAMUX2_REQ_GENERATOR1
  - LL_DMAMUX2_REQ_GENERATOR2
  - LL_DMAMUX2_REQ_GENERATOR3
  - LL_DMAMUX2_REQ_GENERATOR4
  - LL_DMAMUX2_REQ_GENERATOR5
  - LL_DMAMUX2_REQ_GENERATOR6
  - LL_DMAMUX2_REQ_GENERATOR7
  - LL_DMAMUX2_REQ_LPUART1_RX
  - LL_DMAMUX2_REQ_LPUART1_TX
  - LL_DMAMUX2_REQ_SPI6_RX
  - LL_DMAMUX2_REQ_SPI6_TX
  - LL_DMAMUX2_REQ_I2C4_RX
  - LL_DMAMUX2_REQ_I2C4_TX
  - LL_DMAMUX2_REQ_SAI4_A (*)
  - LL_DMAMUX2_REQ_SAI4_B (*)
  - LL_DMAMUX2_REQ_ADC3 (*)
  - LL_DMAMUX2_REQ_DAC2_CH1 (*)
  - LL_DMAMUX2_REQ_DFSDM2_FLT0 (*)

Return values

- **None:**

Notes

- DMAMUX2 channel 0 to 7 are mapped to BDMA channel 0 to 7.
- (*) Availability depends on devices.

Reference Manual to LL API cross reference:

- CxCR DMAREQ_ID LL_BDMA_SetPeriphRequest

LL_BDMA_GetPeriphRequest

Function name

__STATIC_INLINE uint32_t LL_BDMA_GetPeriphRequest (BDMA_TypeDef * BDMAx, uint32_t Channel)

Function description

Get BDMA request for BDMA Channels on DMAMUX Channel x.

Parameters

- **BDMAx:** BDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMAMUX2_REQ_MEM2MEM
  - LL_DMAMUX2_REQ_GENERATOR0
  - LL_DMAMUX2_REQ_GENERATOR1
  - LL_DMAMUX2_REQ_GENERATOR2
  - LL_DMAMUX2_REQ_GENERATOR3
  - LL_DMAMUX2_REQ_GENERATOR4
  - LL_DMAMUX2_REQ_GENERATOR5
  - LL_DMAMUX2_REQ_GENERATOR6
  - LL_DMAMUX2_REQ_GENERATOR7
  - LL_DMAMUX2_REQ_LPUART1_RX
  - LL_DMAMUX2_REQ_LPUART1_TX
  - LL_DMAMUX2_REQ_SPI6_RX
  - LL_DMAMUX2_REQ_SPI6_TX
  - LL_DMAMUX2_REQ_I2C4_RX
  - LL_DMAMUX2_REQ_I2C4_TX
  - LL_DMAMUX2_REQ_SAI4_A (*)
  - LL_DMAMUX2_REQ_SAI4_B (*)
  - LL_DMAMUX2_REQ_ADC3 (*)
  - LL_DMAMUX2_REQ_DAC2_CH1 (*)
  - LL_DMAMUX2_REQ_DFSDM2_FLT0 (*)

Notes

- DMAMUX channel 0 to 7 are mapped to BDMA channel 0 to 7.
- (*) Availability depends on devices.

Reference Manual to LL API cross reference:

- CxCR DMAREQ_ID LL_BDMA_GetPeriphRequest

LL_BDMA_IsActiveFlag_GI0

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_GI0 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 0 global interrupt flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR GIF0 LL_BDMA_IsActiveFlag_GI0

LL_BDMA_IsActiveFlag_GI1

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_GI1 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 1 global interrupt flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR GIF1 LL_BDMA_IsActiveFlag_GI1

LL_BDMA_IsActiveFlag_GI2

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_GI2 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 2 global interrupt flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR GIF2 LL_BDMA_IsActiveFlag_GI2

LL_BDMA_IsActiveFlag_GI3

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_GI3 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 3 global interrupt flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR GIF3 LL_BDMA_IsActiveFlag_GI3

LL_BDMA_IsActiveFlag_GI4

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_GI4 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 4 global interrupt flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR GIF4 LL_BDMA_IsActiveFlag_GI4

LL_BDMA_IsActiveFlag_GI5

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_GI5 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 5 global interrupt flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR GIF5 LL_BDMA_IsActiveFlag_GI5

LL_BDMA_IsActiveFlag_GI6

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_GI6 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 6 global interrupt flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR GIF6 LL_BDMA_IsActiveFlag_GI6

LL_BDMA_IsActiveFlag_GI7

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_GI7 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 7 global interrupt flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR GIF7 LL_BDMA_IsActiveFlag_GI7

LL_BDMA_IsActiveFlag_TC0

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_TC0 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 0 transfer complete flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCIF0 LL_BDMA_IsActiveFlag_TC0

LL_BDMA_IsActiveFlag_TC1

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_TC1 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 1 transfer complete flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCIF1 LL_BDMA_IsActiveFlag_TC1

LL_BDMA_IsActiveFlag_TC2

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_TC2 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 2 transfer complete flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCIF2 LL_BDMA_IsActiveFlag_TC2

LL_BDMA_IsActiveFlag_TC3

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_TC3 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 3 transfer complete flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCIF3 LL_BDMA_IsActiveFlag_TC3

LL_BDMA_IsActiveFlag_TC4

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_TC4 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 4 transfer complete flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCIF4 LL_BDMA_IsActiveFlag_TC4

LL_BDMA_IsActiveFlag_TC5

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_TC5 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 5 transfer complete flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCIF5 LL_BDMA_IsActiveFlag_TC5

LL_BDMA_IsActiveFlag_TC6

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_TC6 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 6 transfer complete flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCIF6 LL_BDMA_IsActiveFlag_TC6

LL_BDMA_IsActiveFlag_TC7

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_TC7 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 7 transfer complete flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCIF7 LL_BDMA_IsActiveFlag_TC7

LL_BDMA_IsActiveFlag_HT0

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_HT0 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 0 half transfer flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR HTIF0 LL_BDMA_IsActiveFlag_HT0

LL_BDMA_IsActiveFlag_HT1

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_HT1 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 1 half transfer flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR HTIF1 LL_BDMA_IsActiveFlag_HT1

LL_BDMA_IsActiveFlag_HT2

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_HT2 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 2 half transfer flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR HTIF2 LL_BDMA_IsActiveFlag_HT2

LL_BDMA_IsActiveFlag_HT3

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_HT3 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 3 half transfer flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR HTIF3 LL_BDMA_IsActiveFlag_HT3

LL_BDMA_IsActiveFlag_HT4

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_HT4 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 4 half transfer flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR HTIF4 LL_BDMA_IsActiveFlag_HT4

LL_BDMA_IsActiveFlag_HT5

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_HT5 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 5 half transfer flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR HTIF5 LL_BDMA_IsActiveFlag_HT5

LL_BDMA_IsActiveFlag_HT6

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_HT6 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 6 half transfer flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR HTIF6 LL_BDMA_IsActiveFlag_HT6

LL_BDMA_IsActiveFlag_HT7

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_HT7 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 7 half transfer flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR HTIF7 LL_BDMA_IsActiveFlag_HT7

LL_BDMA_IsActiveFlag_TE0

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_TE0 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 0 transfer error flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEIF0 LL_BDMA_IsActiveFlag_TE0

LL_BDMA_IsActiveFlag_TE1

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_TE1 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 1 transfer error flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEIF1 LL_BDMA_IsActiveFlag_TE1

LL_BDMA_IsActiveFlag_TE2

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_TE2 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 2 transfer error flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEIF2 LL_BDMA_IsActiveFlag_TE2

LL_BDMA_IsActiveFlag_TE3

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_TE3 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 3 transfer error flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEIF3 LL_BDMA_IsActiveFlag_TE3

LL_BDMA_IsActiveFlag_TE4

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_TE4 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 4 transfer error flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEIF4 LL_BDMA_IsActiveFlag_TE4

LL_BDMA_IsActiveFlag_TE5

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_TE5 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 5 transfer error flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEIF5 LL_BDMA_IsActiveFlag_TE5

LL_BDMA_IsActiveFlag_TE6

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_TE6 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 6 transfer error flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEIF6 LL_BDMA_IsActiveFlag_TE6

LL_BDMA_IsActiveFlag_TE7

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsActiveFlag_TE7 (BDMA_TypeDef * BDMAx)

Function description

Get Channel 7 transfer error flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEIF7 LL_BDMA_IsActiveFlag_TE7

LL_BDMA_ClearFlag_GI0

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_GI0 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 0 global interrupt flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Notes

- Do not Clear Channel 0 global interrupt flag when the channel in ON. Instead clear specific flags transfer complete, half transfer & transfer error flag with LL_DMA_ClearFlag_TC0, LL_DMA_ClearFlag_HT0, LL_DMA_ClearFlag_TE0. bug id 2.3.1 in Product Errata Sheet.

Reference Manual to LL API cross reference:

- IFCR CGIF0 LL_BDMA_ClearFlag_GI0

LL_BDMA_ClearFlag_GI1

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_GI1 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 1 global interrupt flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Notes

- Do not Clear Channel 1 global interrupt flag when the channel in ON. Instead clear specific flags transfer complete, half transfer & transfer error flag with LL_DMA_ClearFlag_TC1, LL_DMA_ClearFlag_HT1, LL_DMA_ClearFlag_TE1. bug id 2.3.1 in Product Errata Sheet.

Reference Manual to LL API cross reference:

- IFCR CGIF1 LL_BDMA_ClearFlag_GI1

LL_BDMA_ClearFlag_GI2

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_GI2 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 2 global interrupt flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Notes

- Do not Clear Channel 2 global interrupt flag when the channel in ON. Instead clear specific flags transfer complete, half transfer & transfer error flag with LL_DMA_ClearFlag_TC2, LL_DMA_ClearFlag_HT2, LL_DMA_ClearFlag_TE2. bug id 2.3.1 in Product Errata Sheet.

Reference Manual to LL API cross reference:

- IFCR CGIF2 LL_BDMA_ClearFlag_GI2

LL_BDMA_ClearFlag_GI3

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_GI3 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 3 global interrupt flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Notes

- Do not Clear Channel 3 global interrupt flag when the channel in ON. Instead clear specific flags transfer complete, half transfer & transfer error flag with LL_DMA_ClearFlag_TC3, LL_DMA_ClearFlag_HT3, LL_DMA_ClearFlag_TE3. bug id 2.3.1 in Product Errata Sheet.

Reference Manual to LL API cross reference:

- IFCR CGIF3 LL_BDMA_ClearFlag_GI3

LL_BDMA_ClearFlag_GI4

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_GI4 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 4 global interrupt flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Notes

- Do not Clear Channel 4 global interrupt flag when the channel in ON. Instead clear specific flags transfer complete, half transfer & transfer error flag with LL_DMA_ClearFlag_TC4, LL_DMA_ClearFlag_HT4, LL_DMA_ClearFlag_TE4. bug id 2.3.1 in Product Errata Sheet.

Reference Manual to LL API cross reference:

- IFCR CGIF4 LL_BDMA_ClearFlag_GI4

LL_BDMA_ClearFlag_GI5

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_GI5 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 5 global interrupt flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Notes

- Do not Clear Channel 5 global interrupt flag when the channel in ON. Instead clear specific flags transfer complete, half transfer & transfer error flag with LL_DMA_ClearFlag_TC5, LL_DMA_ClearFlag_HT5, LL_DMA_ClearFlag_TE5. bug id 2.3.1 in Product Errata Sheet.

Reference Manual to LL API cross reference:

- IFCR CGIF5 LL_BDMA_ClearFlag_GI5

LL_BDMA_ClearFlag_GI6

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_GI6 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 6 global interrupt flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Notes

- Do not Clear Channel 6 global interrupt flag when the channel in ON. Instead clear specific flags transfer complete, half transfer & transfer error flag with LL_DMA_ClearFlag_TC6, LL_DMA_ClearFlag_HT6, LL_DMA_ClearFlag_TE6. bug id 2.3.1 in Product Errata Sheet.

Reference Manual to LL API cross reference:

- IFCR CGIF6 LL_BDMA_ClearFlag_GI6

LL_BDMA_ClearFlag_GI7

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_GI7 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 7 global interrupt flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Notes

- Do not Clear Channel 7 global interrupt flag when the channel in ON. Instead clear specific flags transfer complete, half transfer & transfer error flag with LL_DMA_ClearFlag_TC7, LL_DMA_ClearFlag_HT7, LL_DMA_ClearFlag_TE7. bug id 2.3.1 in Product Errata Sheet.

Reference Manual to LL API cross reference:

- IFCR CGIF7 LL_BDMA_ClearFlag_GI7

LL_BDMA_ClearFlag_TC0

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_TC0 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 0 transfer complete flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTCIF0 LL_BDMA_ClearFlag_TC0

LL_BDMA_ClearFlag_TC1

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_TC1 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 1 transfer complete flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTCIF1 LL_BDMA_ClearFlag_TC1

LL_BDMA_ClearFlag_TC2

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_TC2 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 2 transfer complete flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTCIF2 LL_BDMA_ClearFlag_TC2

LL_BDMA_ClearFlag_TC3

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_TC3 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 3 transfer complete flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTCIF3 LL_BDMA_ClearFlag_TC3

LL_BDMA_ClearFlag_TC4

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_TC4 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 4 transfer complete flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTCIF4 LL_BDMA_ClearFlag_TC4

LL_BDMA_ClearFlag_TC5

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_TC5 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 5 transfer complete flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTCIF5 LL_BDMA_ClearFlag_TC5

LL_BDMA_ClearFlag_TC6

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_TC6 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 6 transfer complete flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTCIF6 LL_BDMA_ClearFlag_TC6

LL_BDMA_ClearFlag_TC7

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_TC7 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 7 transfer complete flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTCIF7 LL_BDMA_ClearFlag_TC7

LL_BDMA_ClearFlag_HT0

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_HT0 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 0 half transfer flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CHTIF0 LL_BDMA_ClearFlag_HT0

LL_BDMA_ClearFlag_HT1

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_HT1 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 1 half transfer flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CHTIF1 LL_BDMA_ClearFlag_HT1

LL_BDMA_ClearFlag_HT2

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_HT2 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 2 half transfer flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CHTIF2 LL_BDMA_ClearFlag_HT2

LL_BDMA_ClearFlag_HT3

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_HT3 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 3 half transfer flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CHTIF3 LL_BDMA_ClearFlag_HT3

LL_BDMA_ClearFlag_HT4

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_HT4 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 4 half transfer flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CHTIF4 LL_BDMA_ClearFlag_HT4

LL_BDMA_ClearFlag_HT5

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_HT5 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 5 half transfer flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CHTIF5 LL_BDMA_ClearFlag_HT5

LL_BDMA_ClearFlag_HT6

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_HT6 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 6 half transfer flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CHTIF6 LL_BDMA_ClearFlag_HT6

LL_BDMA_ClearFlag_HT7

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_HT7 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 7 half transfer flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CHTIF7 LL_BDMA_ClearFlag_HT7

LL_BDMA_ClearFlag_TE0

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_TE0 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 0 transfer error flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTEIF0 LL_BDMA_ClearFlag_TE0

LL_BDMA_ClearFlag_TE1

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_TE1 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 1 transfer error flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTEIF1 LL_BDMA_ClearFlag_TE1

LL_BDMA_ClearFlag_TE2

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_TE2 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 2 transfer error flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTEIF2 LL_BDMA_ClearFlag_TE2

LL_BDMA_ClearFlag_TE3

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_TE3 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 3 transfer error flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTEIF3 LL_BDMA_ClearFlag_TE3

LL_BDMA_ClearFlag_TE4

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_TE4 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 4 transfer error flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTEIF4 LL_BDMA_ClearFlag_TE4

LL_BDMA_ClearFlag_TE5

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_TE5 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 5 transfer error flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTEIF5 LL_BDMA_ClearFlag_TE5

LL_BDMA_ClearFlag_TE6

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_TE6 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 6 transfer error flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTEIF6 LL_BDMA_ClearFlag_TE6

LL_BDMA_ClearFlag_TE7

Function name

__STATIC_INLINE void LL_BDMA_ClearFlag_TE7 (BDMA_TypeDef * BDMAx)

Function description

Clear Channel 7 transfer error flag.

Parameters

- **BDMAx:** BDMA Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTEIF7 LL_BDMA_ClearFlag_TE7

LL_BDMA_EnableIT_TC

Function name

__STATIC_INLINE void LL_BDMA_EnableIT_TC (BDMA_TypeDef * BDMAx, uint32_t Channel)

Function description

Enable Transfer complete interrupt.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR TCIE LL_BDMA_EnableIT_TC

LL_BDMA_EnableIT_HT

Function name

__STATIC_INLINE void LL_BDMA_EnableIT_HT (BDMA_TypeDef * BDMAx, uint32_t Channel)

Function description

Enable Half transfer interrupt.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR HTIE LL_BDMA_EnableIT_HT

LL_BDMA_EnableIT_TE

Function name

__STATIC_INLINE void LL_BDMA_EnableIT_TE (BDMA_TypeDef * BDMAx, uint32_t Channel)

Function description

Enable Transfer error interrupt.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR TEIE LL_BDMA_EnableIT_TE

LL_BDMA_DisableIT_TC

Function name

__STATIC_INLINE void LL_BDMA_DisableIT_TC (BDMA_TypeDef * BDMAx, uint32_t Channel)

Function description

Disable Transfer complete interrupt.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR TCIE LL_BDMA_DisableIT_TC

LL_BDMA_DisableIT_HT

Function name

__STATIC_INLINE void LL_BDMA_DisableIT_HT (BDMA_TypeDef * BDMAx, uint32_t Channel)

Function description

Disable Half transfer interrupt.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR HTIE LL_BDMA_DisableIT_HT

LL_BDMA_DisableIT_TE

Function name

__STATIC_INLINE void LL_BDMA_DisableIT_TE (BDMA_TypeDef * BDMAx, uint32_t Channel)

Function description

Disable Transfer error interrupt.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR TEIE LL_BDMA_DisableIT_TE

LL_BDMA_IsEnabledIT_TC

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsEnabledIT_TC (BDMA_TypeDef * BDMAx, uint32_t Channel)

Function description

Check if Transfer complete Interrupt is enabled.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCR TCIE LL_BDMA_IsEnabledIT_TC

LL_BDMA_IsEnabledIT_HT

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsEnabledIT_HT (BDMA_TypeDef * BDMAx, uint32_t Channel)

Function description

Check if Half transfer Interrupt is enabled.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCR HTIE LL_BDMA_IsEnabledIT_HT

LL_BDMA_IsEnabledIT_TE

Function name

__STATIC_INLINE uint32_t LL_BDMA_IsEnabledIT_TE (BDMA_TypeDef * BDMAx, uint32_t Channel)

Function description

Check if Transfer error Interrupt is enabled.

Parameters

- **BDMAx:** BDMA Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCR TEIE LL_BDMA_IsEnabledIT_TE

LL_BDMA_Init

Function name

uint32_t LL_BDMA_Init (BDMA_TypeDef * BDMAx, uint32_t Channel, LL_BDMA_InitTypeDef * BDMA_InitStruct)

Function description

Initialize the BDMA registers according to the specified parameters in BDMA_InitStruct.

Parameters

- **BDMAx:** BDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7
- **BDMA_InitStruct:** pointer to a LL_BDMA_InitTypeDef structure.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: DMA registers are initialized
  - ERROR: Not applicable

Notes

- To convert BDMAx_Channely Instance to BDMAx Instance and Channely, use helper macros : __LL_BDMA_GET_INSTANCE __LL_BDMA_GET_CHANNEL

LL_BDMA_DeInit

Function name

uint32_t LL_BDMA_DeInit (BDMA_TypeDef * BDMAx, uint32_t Channel)

Function description

De-initialize the DMA registers to their default reset values.

Parameters

- **BDMAx:** BDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_BDMA_CHANNEL_0
  - LL_BDMA_CHANNEL_1
  - LL_BDMA_CHANNEL_2
  - LL_BDMA_CHANNEL_3
  - LL_BDMA_CHANNEL_4
  - LL_BDMA_CHANNEL_5
  - LL_BDMA_CHANNEL_6
  - LL_BDMA_CHANNEL_7
  - LL_BDMA_CHANNEL_ALL

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: DMA registers are de-initialized
  - ERROR: DMA registers are not de-initialized

LL_BDMA_StructInit

Function name

void LL_BDMA_StructInit (LL_BDMA_InitTypeDef * BDMA_InitStruct)

Function description

Set each LL_BDMA_InitTypeDef field to default value.

Parameters

- **BDMA_InitStruct:** Pointer to a LL_BDMA_InitTypeDef structure.

Return values

- **None:**

96.3 BDMA Firmware driver defines

The following section lists the various define and macros of the module.

96.3.1 BDMA

BDMA

CHANNEL

LL_BDMA_CHANNEL_0

DMA Channel 0

LL_BDMA_CHANNEL_1

BDMA Channel 1

LL_BDMA_CHANNEL_2

BDMA Channel 2

LL_BDMA_CHANNEL_3

BDMA Channel 3

LL_BDMA_CHANNEL_4

BDMA Channel 4

LL_BDMA_CHANNEL_5

BDMA Channel 5

LL_BDMA_CHANNEL_6

BDMA Channel 6

LL_BDMA_CHANNEL_7

BDMA Channel 7

LL_BDMA_CHANNEL_ALL

BDMA Channel all (used only for function

Clear Flags Defines

LL_BDMA_IFCR_CGIF1

Channel 1 global flag

LL_BDMA_IFCR_CTCIF1

Channel 1 transfer complete flag

LL_BDMA_IFCR_CHTIF1

Channel 1 half transfer flag

LL_BDMA_IFCR_CTEIF1

Channel 1 transfer error flag

LL_BDMA_IFCR_CGIF2

Channel 2 global flag

LL_BDMA_IFCR_CTCIF2

Channel 2 transfer complete flag

LL_BDMA_IFCR_CHTIF2

Channel 2 half transfer flag

LL_BDMA_IFCR_CTEIF2

Channel 2 transfer error flag

LL_BDMA_IFCR_CGIF3

Channel 3 global flag

LL_BDMA_IFCR_CTCIF3

Channel 3 transfer complete flag

LL_BDMA_IFCR_CHTIF3

Channel 3 half transfer flag

LL_BDMA_IFCR_CTEIF3

Channel 3 transfer error flag

LL_BDMA_IFCR_CGIF4

Channel 4 global flag

LL_BDMA_IFCR_CTCIF4

Channel 4 transfer complete flag

LL_BDMA_IFCR_CHTIF4

Channel 4 half transfer flag

LL_BDMA_IFCR_CTEIF4

Channel 4 transfer error flag

LL_BDMA_IFCR_CGIF5

Channel 5 global flag

LL_BDMA_IFCR_CTCIF5

Channel 5 transfer complete flag

LL_BDMA_IFCR_CHTIF5

Channel 5 half transfer flag

LL_BDMA_IFCR_CTEIF5

Channel 5 transfer error flag

LL_BDMA_IFCR_CGIF6

Channel 6 global flag

LL_BDMA_IFCR_CTCIF6

Channel 6 transfer complete flag

LL_BDMA_IFCR_CHTIF6

Channel 6 half transfer flag

LL_BDMA_IFCR_CTEIF6

Channel 6 transfer error flag

LL_BDMA_IFCR_CGIF7

Channel 7 global flag

LL_BDMA_IFCR_CTCIF7

Channel 7 transfer complete flag

LL_BDMA_IFCR_CHTIF7

Channel 7 half transfer flag

LL_BDMA_IFCR_CTEIF7

Channel 7 transfer error flag

Transfer Direction

LL_BDMA_DIRECTION_PERIPH_TO_MEMORY

Peripheral to memory direction

LL_BDMA_DIRECTION_MEMORY_TO_PERIPH

Memory to peripheral direction

LL_BDMA_DIRECTION_MEMORY_TO_MEMORY

Memory to memory direction

Get Flags Defines

LL_BDMA_ISR_GIF0

Channel 1 global flag

LL_BDMA_ISR_TCIF0

Channel 1 transfer complete flag

LL_BDMA_ISR_HTIF0

Channel 1 half transfer flag

LL_BDMA_ISR_TEIF0

Channel 1 transfer error flag

LL_BDMA_ISR_GIF1

Channel 1 global flag

LL_BDMA_ISR_TCIF1

Channel 1 transfer complete flag

LL_BDMA_ISR_HTIF1

Channel 1 half transfer flag

LL_BDMA_ISR_TEIF1

Channel 1 transfer error flag

LL_BDMA_ISR_GIF2

Channel 2 global flag

LL_BDMA_ISR_TCIF2

Channel 2 transfer complete flag

LL_BDMA_ISR_HTIF2

Channel 2 half transfer flag

LL_BDMA_ISR_TEIF2

Channel 2 transfer error flag

LL_BDMA_ISR_GIF3

Channel 3 global flag

LL_BDMA_ISR_TCIF3

Channel 3 transfer complete flag

LL_BDMA_ISR_HTIF3

Channel 3 half transfer flag

LL_BDMA_ISR_TEIF3

Channel 3 transfer error flag

LL_BDMA_ISR_GIF4

Channel 4 global flag

LL_BDMA_ISR_TCIF4

Channel 4 transfer complete flag

LL_BDMA_ISR_HTIF4

Channel 4 half transfer flag

LL_BDMA_ISR_TEIF4

Channel 4 transfer error flag

LL_BDMA_ISR_GIF5

Channel 5 global flag

LL_BDMA_ISR_TCIF5

Channel 5 transfer complete flag

LL_BDMA_ISR_HTIF5

Channel 5 half transfer flag

LL_BDMA_ISR_TEIF5

Channel 5 transfer error flag

LL_BDMA_ISR_GIF6

Channel 6 global flag

LL_BDMA_ISR_TCIF6

Channel 6 transfer complete flag

LL_BDMA_ISR_HTIF6

Channel 6 half transfer flag

LL_BDMA_ISR_TEIF6

Channel 6 transfer error flag

LL_BDMA_ISR_GIF7

Channel 7 global flag

LL_BDMA_ISR_TCIF7

Channel 7 transfer complete flag

LL_BDMA_ISR_HTIF7

Channel 7 half transfer flag

LL_BDMA_ISR_TEIF7

Channel 7 transfer error flag

IT Defines

LL_BDMA_CCR_TCIE

Transfer complete interrupt

LL_BDMA_CCR_HTIE

Half Transfer interrupt

LL_BDMA_CCR_TEIE

Transfer error interrupt

Memory data alignment

LL_BDMA_MDATAALIGN_BYTE

Memory data alignment : Byte

LL_BDMA_MDATAALIGN_HALFWORD

Memory data alignment : HalfWord

LL_BDMA_MDATAALIGN_WORD

Memory data alignment : Word

Memory increment mode

LL_BDMA_MEMORY_INCREMENT

Memory increment mode Enable

LL_BDMA_MEMORY_NOINCREMENT

Memory increment mode Disable

Transfer mode

LL_BDMA_MODE_NORMAL

Normal Mode

LL_BDMA_MODE_CIRCULAR

Circular Mode

Peripheral data alignment

LL_BDMA_PDATAALIGN_BYTE

Peripheral data alignment : Byte

LL_BDMA_PDATAALIGN_HALFWORD

Peripheral data alignment : HalfWord

LL_BDMA_PDATAALIGN_WORD

Peripheral data alignment : Word

Peripheral increment mode

LL_BDMA_PERIPH_INCREMENT

Peripheral increment mode Enable

LL_BDMA_PERIPH_NOINCREMENT

Peripheral increment mode Disable

Transfer Priority level

LL_BDMA_PRIORITY_LOW

Priority level : Low

LL_BDMA_PRIORITY_MEDIUM

Priority level : Medium

LL_BDMA_PRIORITY_HIGH

Priority level : High

LL_BDMA_PRIORITY_VERYHIGH

Priority level : Very_High

Convert BDMAxChannely

__LL_BDMA_GET_INSTANCE

Description:

- Convert BDMAx_Channely into BDMAx.

Parameters:

- __CHANNEL_INSTANCE__: BDMAx_Channely

Return value:

- BDMAx

__LL_BDMA_GET_CHANNEL

Description:

- Convert BDMAx_Channely into LL_BDMA_CHANNEL_y.

Parameters:

- __CHANNEL_INSTANCE__: BDMAx_Channely

Return value:

- LL_BDMA_CHANNEL_y

__LL_BDMA_GET_CHANNEL_INSTANCE

Description:

- Convert BDMA Instance BDMAx and LL_BDMA_CHANNEL_y into BDMAx_Channely.

Parameters:

- __BDMA_INSTANCE__: BDMAx
- __CHANNEL__: LL_BDMA_CHANNEL_y

Return value:

- BDMAx_Channely

Common Write and read registers macros

LL_BDMA_WriteReg

Description:

- Write a value in BDMA register.

Parameters:

- __INSTANCE__: BDMA Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_BDMA_ReadReg

Description:

- Read a value in BDMA register.

Parameters:

- __INSTANCE__: BDMA Instance
- __REG__: Register to be read

Return value:

- Register: value

97 LL BUS Generic Driver

97.1 BUS Firmware driver API description

The following section lists the various functions of the BUS library.

97.1.1 Detailed description of functions

LL_AHB3_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_AHB3_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable AHB3 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_MDMA
  - LL_AHB3_GRP1_PERIPH_DMA2D
  - LL_AHB3_GRP1_PERIPH_JPGDEC (*)
  - LL_AHB3_GRP1_PERIPH_FMC
  - LL_AHB3_GRP1_PERIPH_QSPI (*)
  - LL_AHB3_GRP1_PERIPH_OSPI1 (*)
  - LL_AHB3_GRP1_PERIPH_OSPI2 (*)
  - LL_AHB3_GRP1_PERIPH_OCTOSPIM (*)
  - LL_AHB3_GRP1_PERIPH_OTFDEC1 (*)
  - LL_AHB3_GRP1_PERIPH_OTFDEC2 (*)
  - LL_AHB3_GRP1_PERIPH_GFXMMU (*)
  - LL_AHB3_GRP1_PERIPH_SDMMC1
  - LL_AHB3_GRP1_PERIPH_FLASH (*)
  - LL_AHB3_GRP1_PERIPH_DTCM1 (*)
  - LL_AHB3_GRP1_PERIPH_DTCM2 (*)
  - LL_AHB3_GRP1_PERIPH_ITCM (*)
  - LL_AHB3_GRP1_PERIPH_AXISRAM

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3ENR MDMAEN LL_AHB3_GRP1_EnableClock
- AHB3ENR DMA2DEN LL_AHB3_GRP1_EnableClock
- AHB3ENR JPGDECEN LL_AHB3_GRP1_EnableClock
- AHB3ENR FMCEN LL_AHB3_GRP1_EnableClock
- AHB3ENR QSPIEN LL_AHB3_GRP1_EnableClock
- (*) AHB3ENR OSPI1EN LL_AHB3_GRP1_EnableClock
- (*) AHB3ENR OSPI2EN LL_AHB3_GRP1_EnableClock
- (*) AHB3ENR IOMNGREN LL_AHB3_GRP1_EnableClock
- (*) AHB3ENR OTFDEC1EN LL_AHB3_GRP1_EnableClock
- (*) AHB3ENR OTFDEC2EN LL_AHB3_GRP1_EnableClock
- (*) AHB3ENR GFXMMUEN LL_AHB3_GRP1_EnableClock
- (*) AHB3ENR SDMMC1EN LL_AHB3_GRP1_EnableClock
- AHB3ENR FLASHEN LL_AHB3_GRP1_EnableClock
- (*) AHB3ENR DTCM1EN LL_AHB3_GRP1_EnableClock
- (*) AHB3ENR DTCM2EN LL_AHB3_GRP1_EnableClock
- (*) AHB3ENR ITCMEN LL_AHB3_GRP1_EnableClock
- (*) AHB3ENR AXISRAMEN LL_AHB3_GRP1_EnableClock (*)

LL_AHB3_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_AHB3_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if AHB3 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_MDMA
  - LL_AHB3_GRP1_PERIPH_DMA2D
  - LL_AHB3_GRP1_PERIPH_JPGDEC (*)
  - LL_AHB3_GRP1_PERIPH_FMC
  - LL_AHB3_GRP1_PERIPH_QSPI (*)
  - LL_AHB3_GRP1_PERIPH_OSPI1 (*)
  - LL_AHB3_GRP1_PERIPH_OSPI2 (*)
  - LL_AHB3_GRP1_PERIPH_OCTOSPIM (*)
  - LL_AHB3_GRP1_PERIPH_OTFDEC1 (*)
  - LL_AHB3_GRP1_PERIPH_OTFDEC2 (*)
  - LL_AHB3_GRP1_PERIPH_GFXMMU (*)
  - LL_AHB3_GRP1_PERIPH_SDMMC1
  - LL_AHB3_GRP1_PERIPH_FLASH (*)
  - LL_AHB3_GRP1_PERIPH_DTCM1 (*)
  - LL_AHB3_GRP1_PERIPH_DTCM2 (*)
  - LL_AHB3_GRP1_PERIPH_ITCM (*)
  - LL_AHB3_GRP1_PERIPH_AXISRAM

  (*) value not defined in all devices.

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- AHB3ENR MDMAEN LL_AHB3_GRP1_IsEnabledClock
- AHB3ENR DMA2DEN LL_AHB3_GRP1_IsEnabledClock
- AHB3ENR JPGDECEN LL_AHB3_GRP1_IsEnabledClock
- AHB3ENR FMCEN LL_AHB3_GRP1_IsEnabledClock
- AHB3ENR QSPIEN LL_AHB3_GRP1_IsEnabledClock
- (*) AHB3ENR OSPI1EN LL_AHB3_GRP1_IsEnabledClock
- (*) AHB3ENR OSPI2EN LL_AHB3_GRP1_IsEnabledClock
- (*) AHB3ENR IOMNGREN LL_AHB3_GRP1_IsEnabledClock
- (*) AHB3ENR OTFDEC1EN LL_AHB3_GRP1_IsEnabledClock
- (*) AHB3ENR OTFDEC2EN LL_AHB3_GRP1_IsEnabledClock
- (*) AHB3ENR GFXMMUEN LL_AHB3_GRP1_IsEnabledClock
- (*) AHB3ENR SDMMC1EN LL_AHB3_GRP1_IsEnabledClock
- AHB3ENR FLASHEN LL_AHB3_GRP1_IsEnabledClock
- (*) AHB3ENR DTCM1EN LL_AHB3_GRP1_IsEnabledClock
- (*) AHB3ENR DTCM2EN LL_AHB3_GRP1_IsEnabledClock
- (*) AHB3ENR ITCMEN LL_AHB3_GRP1_IsEnabledClock
- (*) AHB3ENR AXISRAMEN LL_AHB3_GRP1_IsEnabledClock (*)

LL_AHB3_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_AHB3_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable AHB3 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_MDMA
  - LL_AHB3_GRP1_PERIPH_DMA2D
  - LL_AHB3_GRP1_PERIPH_JPGDEC (*)
  - LL_AHB3_GRP1_PERIPH_FMC
  - LL_AHB3_GRP1_PERIPH_QSPI (*)
  - LL_AHB3_GRP1_PERIPH_OSPI1 (*)
  - LL_AHB3_GRP1_PERIPH_OSPI2 (*)
  - LL_AHB3_GRP1_PERIPH_OCTOSPIM (*)
  - LL_AHB3_GRP1_PERIPH_OTFDEC1 (*)
  - LL_AHB3_GRP1_PERIPH_OTFDEC2 (*)
  - LL_AHB3_GRP1_PERIPH_GFXMMU (*)
  - LL_AHB3_GRP1_PERIPH_SDMMC1
  - LL_AHB3_GRP1_PERIPH_FLASH (*)
  - LL_AHB3_GRP1_PERIPH_DTCM1 (*)
  - LL_AHB3_GRP1_PERIPH_DTCM2 (*)
  - LL_AHB3_GRP1_PERIPH_ITCM (*)
  - LL_AHB3_GRP1_PERIPH_AXISRAM

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3ENR MDMAEN LL_AHB3_GRP1_DisableClock
- AHB3ENR DMA2DEN LL_AHB3_GRP1_DisableClock
- AHB3ENR JPGDECEN LL_AHB3_GRP1_DisableClock
- AHB3ENR FMCEN LL_AHB3_GRP1_DisableClock
- AHB3ENR QSPIEN LL_AHB3_GRP1_DisableClock
- (*) AHB3ENR OSPI1EN LL_AHB3_GRP1_DisableClock
- (*) AHB3ENR OSPI2EN LL_AHB3_GRP1_DisableClock
- (*) AHB3ENR IOMNGREN LL_AHB3_GRP1_DisableClock
- (*) AHB3ENR OTFDEC1EN LL_AHB3_GRP1_DisableClock
- (*) AHB3ENR OTFDEC2EN LL_AHB3_GRP1_DisableClock
- (*) AHB3ENR GFXMMUEN LL_AHB3_GRP1_DisableClock
- (*) AHB3ENR SDMMC1EN LL_AHB3_GRP1_DisableClock
- (*) AHB3ENR FLASHEN LL_AHB3_GRP1_DisableClock
- (*) AHB3ENR DTCM1EN LL_AHB3_GRP1_DisableClock
- (*) AHB3ENR DTCM2EN LL_AHB3_GRP1_DisableClock
- (*) AHB3ENR ITCMEN LL_AHB3_GRP1_DisableClock
- (*) AHB3ENR AXISRAMEN LL_AHB3_GRP1_DisableClock

LL_AHB3_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_AHB3_GRP1_ForceReset (uint32_t Periphs)

Function description

Force AHB3 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_MDMA
  - LL_AHB3_GRP1_PERIPH_DMA2D
  - LL_AHB3_GRP1_PERIPH_JPGDEC (*)
  - LL_AHB3_GRP1_PERIPH_FMC
  - LL_AHB3_GRP1_PERIPH_QSPI (*)
  - LL_AHB3_GRP1_PERIPH_OSPI1 (*)
  - LL_AHB3_GRP1_PERIPH_OSPI2 (*)
  - LL_AHB3_GRP1_PERIPH_OCTOSPIM (*)
  - LL_AHB3_GRP1_PERIPH_OTFDEC1 (*)
  - LL_AHB3_GRP1_PERIPH_OTFDEC2 (*)
  - LL_AHB3_GRP1_PERIPH_GFXMMU (*)
  - LL_AHB3_GRP1_PERIPH_SDMMC1

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3RSTR MDMARST LL_AHB3_GRP1_ForceReset
- AHB3RSTR DMA2DRST LL_AHB3_GRP1_ForceReset
- AHB3RSTR JPGDECRST LL_AHB3_GRP1_ForceReset
- AHB3RSTR FMCRST LL_AHB3_GRP1_ForceReset
- AHB3RSTR QSPIRST LL_AHB3_GRP1_ForceReset
- (*) AHB3RSTR OSPI1RST LL_AHB3_GRP1_ForceReset
- (*) AHB3RSTR OSPI2RST LL_AHB3_GRP1_ForceReset
- (*) AHB3RSTR IOMNGRRST LL_AHB3_GRP1_ForceReset
- (*) AHB3RSTR OTFDEC1RST LL_AHB3_GRP1_ForceReset
- (*) AHB3RSTR OTFDEC2RST LL_AHB3_GRP1_ForceReset
- (*) AHB3RSTR GFXMMURST LL_AHB3_GRP1_ForceReset
- (*) AHB3RSTR SDMMC1RST LL_AHB3_GRP1_ForceReset

LL_AHB3_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_AHB3_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release AHB3 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_MDMA
  - LL_AHB3_GRP1_PERIPH_DMA2D
  - LL_AHB3_GRP1_PERIPH_JPGDEC (*)
  - LL_AHB3_GRP1_PERIPH_FMC
  - LL_AHB3_GRP1_PERIPH_QSPI (*)
  - LL_AHB3_GRP1_PERIPH_OSPI1 (*)
  - LL_AHB3_GRP1_PERIPH_OSPI2 (*)
  - LL_AHB3_GRP1_PERIPH_OCTOSPIM (*)
  - LL_AHB3_GRP1_PERIPH_OTFDEC1 (*)
  - LL_AHB3_GRP1_PERIPH_OTFDEC2 (*)
  - LL_AHB3_GRP1_PERIPH_GFXMMU (*)
  - LL_AHB3_GRP1_PERIPH_SDMMC1

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3RSTR MDMARST LL_AHB3_GRP1_ReleaseReset
- AHB3RSTR DMA2DRST LL_AHB3_GRP1_ReleaseReset
- AHB3RSTR JPGDECRST LL_AHB3_GRP1_ReleaseReset
- AHB3RSTR FMCRST LL_AHB3_GRP1_ReleaseReset
- AHB3RSTR QSPIRST LL_AHB3_GRP1_ReleaseReset
- AHB3RSTR OSPI1RST LL_AHB3_GRP1_ReleaseReset
- (*) AHB3RSTR OSPI2RST LL_AHB3_GRP1_ReleaseReset
- (*) AHB3RSTR IOMNGRRST LL_AHB3_GRP1_ReleaseReset
- (*) AHB3RSTR OTFDEC1RST LL_AHB3_GRP1_ReleaseReset
- (*) AHB3RSTR OTFDEC2RST LL_AHB3_GRP1_ReleaseReset
- (*) AHB3RSTR GFXMMURST LL_AHB3_GRP1_ReleaseReset
- (*) AHB3RSTR SDMMC1RST LL_AHB3_GRP1_ReleaseReset

LL_AHB3_GRP1_EnableClockSleep

Function name

__STATIC_INLINE void LL_AHB3_GRP1_EnableClockSleep (uint32_t Periphs)

Function description

Enable AHB3 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_DMA2D
  - LL_AHB3_GRP1_PERIPH_JPGDEC (*)
  - LL_AHB3_GRP1_PERIPH_FMC
  - LL_AHB3_GRP1_PERIPH_QSPI (*)
  - LL_AHB3_GRP1_PERIPH_OSPI1 (*)
  - LL_AHB3_GRP1_PERIPH_OSPI2 (*)
  - LL_AHB3_GRP1_PERIPH_OCTOSPIM (*)
  - LL_AHB3_GRP1_PERIPH_OTFDEC1 (*)
  - LL_AHB3_GRP1_PERIPH_OTFDEC2 (*)
  - LL_AHB3_GRP1_PERIPH_GFXMMU (*)
  - LL_AHB3_GRP1_PERIPH_SDMMC1
  - LL_AHB3_GRP1_PERIPH_FLASH
  - LL_AHB3_GRP1_PERIPH_DTCM1
  - LL_AHB3_GRP1_PERIPH_DTCM2
  - LL_AHB3_GRP1_PERIPH_ITCM
  - LL_AHB3_GRP1_PERIPH_AXISRAM

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3LPENR MDMALPEN LL_AHB3_GRP1_EnableClockSleep
- AHB3LPENR DMA2DLPEN LL_AHB3_GRP1_EnableClockSleep
- AHB3LPENR JPGDECLPEN LL_AHB3_GRP1_EnableClockSleep
- AHB3LPENR FMCLPEN LL_AHB3_GRP1_EnableClockSleep
- AHB3LPENR QSPILPEN LL_AHB3_GRP1_EnableClockSleep
- (*) AHB3LPENR OSPI1LPEN LL_AHB3_GRP1_EnableClockSleep
- (*) AHB3LPENR OSPI2LPEN LL_AHB3_GRP1_EnableClockSleep
- (*) AHB3LPENR IOMNGRLPEN LL_AHB3_GRP1_EnableClockSleep
- (*) AHB3LPENR OTFDEC1LPEN LL_AHB3_GRP1_EnableClockSleep
- (*) AHB3LPENR OTFDEC1LPEN LL_AHB3_GRP1_EnableClockSleep
- (*) AHB3LPENR GFXMMULPEN LL_AHB3_GRP1_EnableClockSleep
- (*) AHB3LPENR SDMMC1LPEN LL_AHB3_GRP1_EnableClockSleep
- AHB3LPENR FLASHLPEN LL_AHB3_GRP1_EnableClockSleep
- AHB3LPENR DTCM1LPEN LL_AHB3_GRP1_EnableClockSleep
- AHB3LPENR DTCM2LPEN LL_AHB3_GRP1_EnableClockSleep
- AHB3LPENR ITCMLPEN LL_AHB3_GRP1_EnableClockSleep
- AHB3LPENR AXISRAMLPEN LL_AHB3_GRP1_EnableClockSleep

LL_AHB3_GRP1_DisableClockSleep

Function name

__STATIC_INLINE void LL_AHB3_GRP1_DisableClockSleep (uint32_t Periphs)

Function description

Disable AHB3 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_DMA2D
  - LL_AHB3_GRP1_PERIPH_JPGDEC (*)
  - LL_AHB3_GRP1_PERIPH_FMC
  - LL_AHB3_GRP1_PERIPH_QSPI (*)
  - LL_AHB3_GRP1_PERIPH_OSPI1 (*)
  - LL_AHB3_GRP1_PERIPH_OSPI2 (*)
  - LL_AHB3_GRP1_PERIPH_OCTOSPIM (*)
  - LL_AHB3_GRP1_PERIPH_OTFDEC1 (*)
  - LL_AHB3_GRP1_PERIPH_OTFDEC2 (*)
  - LL_AHB3_GRP1_PERIPH_GFXMMU (*)
  - LL_AHB3_GRP1_PERIPH_SDMMC1
  - LL_AHB3_GRP1_PERIPH_FLASH
  - LL_AHB3_GRP1_PERIPH_DTCM1
  - LL_AHB3_GRP1_PERIPH_DTCM2
  - LL_AHB3_GRP1_PERIPH_ITCM
  - LL_AHB3_GRP1_PERIPH_AXISRAM

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3LPENR MDMALPEN LL_AHB3_GRP1_DisableClockSleep
- AHB3LPENR DMA2DLPEN LL_AHB3_GRP1_DisableClockSleep
- AHB3LPENR JPGDECLPEN LL_AHB3_GRP1_DisableClockSleep
- AHB3LPENR FMCLPEN LL_AHB3_GRP1_DisableClockSleep
- AHB3LPENR QSPILPEN LL_AHB3_GRP1_DisableClockSleep
- AHB3LPENR OSPI1LPEN LL_AHB3_GRP1_DisableClockSleep
- (*) AHB3LPENR OSPI2LPEN LL_AHB3_GRP1_DisableClockSleep
- (*) AHB3LPENR IOMNGRLPEN LL_AHB3_GRP1_DisableClockSleep
- (*) AHB3LPENR OTFDEC1LPEN LL_AHB3_GRP1_DisableClockSleep
- (*) AHB3LPENR OTFDEC1LPEN LL_AHB3_GRP1_DisableClockSleep
- (*) AHB3LPENR GFXMMULPEN LL_AHB3_GRP1_DisableClockSleep
- (*) AHB3LPENR SDMMC1LPEN LL_AHB3_GRP1_DisableClockSleep
- AHB3LPENR FLASHLPEN LL_AHB3_GRP1_DisableClockSleep
- AHB3LPENR DTCM1LPEN LL_AHB3_GRP1_DisableClockSleep
- AHB3LPENR DTCM2LPEN LL_AHB3_GRP1_DisableClockSleep
- AHB3LPENR ITCMLPEN LL_AHB3_GRP1_DisableClockSleep
- AHB3LPENR AXISRAMLPEN LL_AHB3_GRP1_DisableClockSleep

LL_AHB1_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_AHB1_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable AHB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_DMA1
  - LL_AHB1_GRP1_PERIPH_DMA2
  - LL_AHB1_GRP1_PERIPH_ADC12
  - LL_AHB1_GRP1_PERIPH_ART (*)
  - LL_AHB1_GRP1_PERIPH_CRC (*)
  - LL_AHB1_GRP1_PERIPH_ETH1MAC (*)
  - LL_AHB1_GRP1_PERIPH_ETH1TX (*)
  - LL_AHB1_GRP1_PERIPH_ETH1RX (*)
  - LL_AHB1_GRP1_PERIPH_USB1OTGHS
  - LL_AHB1_GRP1_PERIPH_USB1OTGHSULPI
  - LL_AHB1_GRP1_PERIPH_USB2OTGHS (*)
  - LL_AHB1_GRP1_PERIPH_USB2OTGHSULPI (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1ENR DMA1EN LL_AHB1_GRP1_EnableClock
- AHB1ENR DMA2EN LL_AHB1_GRP1_EnableClock
- AHB1ENR ADC12EN LL_AHB1_GRP1_EnableClock
- AHB1ENR ARTEN LL_AHB1_GRP1_EnableClock
- AHB1ENR CRCEN LL_AHB1_GRP1_EnableClock
- (*) AHB1ENR ETH1MACEN LL_AHB1_GRP1_EnableClock
- (*) AHB1ENR ETH1TXEN LL_AHB1_GRP1_EnableClock
- (*) AHB1ENR ETH1RXEN LL_AHB1_GRP1_EnableClock
- (*) AHB1ENR USB1OTGHSEN LL_AHB1_GRP1_EnableClock
- AHB1ENR USB1OTGHSULPIEN LL_AHB1_GRP1_EnableClock
- AHB1ENR USB2OTGHSEN LL_AHB1_GRP1_EnableClock
- (*) AHB1ENR USB2OTGHSULPIEN LL_AHB1_GRP1_EnableClock (*)

LL_AHB1_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_AHB1_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if AHB1 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_DMA1
  - LL_AHB1_GRP1_PERIPH_DMA2
  - LL_AHB1_GRP1_PERIPH_ADC12
  - LL_AHB1_GRP1_PERIPH_ART (*)
  - LL_AHB1_GRP1_PERIPH_CRC (*)
  - LL_AHB1_GRP1_PERIPH_ETH1MAC (*)
  - LL_AHB1_GRP1_PERIPH_ETH1TX (*)
  - LL_AHB1_GRP1_PERIPH_ETH1RX (*)
  - LL_AHB1_GRP1_PERIPH_USB1OTGHS
  - LL_AHB1_GRP1_PERIPH_USB1OTGHSULPI
  - LL_AHB1_GRP1_PERIPH_USB2OTGHS (*)
  - LL_AHB1_GRP1_PERIPH_USB2OTGHSULPI (*)

  (*) value not defined in all devices.

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- AHB1ENR DMA1EN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR DMA2EN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR ADC12EN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR ARTEN LL_AHB1_GRP1_IsEnabledClock
- (*) AHB1ENR CRCEN LL_AHB1_GRP1_IsEnabledClock
- (*) AHB1ENR ETH1MACEN LL_AHB1_GRP1_IsEnabledClock
- (*) AHB1ENR ETH1TXEN LL_AHB1_GRP1_IsEnabledClock
- (*) AHB1ENR ETH1RXEN LL_AHB1_GRP1_IsEnabledClock
- (*) AHB1ENR USB1OTGHSEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR USB1OTGHSULPIEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR USB2OTGHSEN LL_AHB1_GRP1_IsEnabledClock
- (*) AHB1ENR USB2OTGHSULPIEN LL_AHB1_GRP1_IsEnabledClock (*)

LL_AHB1_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_AHB1_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable AHB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_DMA1
  - LL_AHB1_GRP1_PERIPH_DMA2
  - LL_AHB1_GRP1_PERIPH_ADC12
  - LL_AHB1_GRP1_PERIPH_ART (*)
  - LL_AHB1_GRP1_PERIPH_CRC (*)
  - LL_AHB1_GRP1_PERIPH_ETH1MAC (*)
  - LL_AHB1_GRP1_PERIPH_ETH1TX (*)
  - LL_AHB1_GRP1_PERIPH_ETH1RX (*)
  - LL_AHB1_GRP1_PERIPH_USB1OTGHS
  - LL_AHB1_GRP1_PERIPH_USB1OTGHSULPI
  - LL_AHB1_GRP1_PERIPH_USB2OTGHS (*)
  - LL_AHB1_GRP1_PERIPH_USB2OTGHSULPI (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1ENR DMA1EN LL_AHB1_GRP1_DisableClock
- AHB1ENR DMA2EN LL_AHB1_GRP1_DisableClock
- AHB1ENR ADC12EN LL_AHB1_GRP1_DisableClock
- AHB1ENR ARTEN LL_AHB1_GRP1_DisableClock
- (*) AHB1ENR CRCEN LL_AHB1_GRP1_DisableClock
- (*) AHB1ENR ETH1MACEN LL_AHB1_GRP1_DisableClock
- (*) AHB1ENR ETH1TXEN LL_AHB1_GRP1_DisableClock
- (*) AHB1ENR ETH1RXEN LL_AHB1_GRP1_DisableClock
- (*) AHB1ENR USB1OTGHSEN LL_AHB1_GRP1_DisableClock
- AHB1ENR USB1OTGHSULPIEN LL_AHB1_GRP1_DisableClock
- AHB1ENR USB2OTGHSEN LL_AHB1_GRP1_DisableClock
- (*) AHB1ENR USB2OTGHSULPIEN LL_AHB1_GRP1_DisableClock (*)

LL_AHB1_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_AHB1_GRP1_ForceReset (uint32_t Periphs)

Function description

Force AHB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_DMA1
  - LL_AHB1_GRP1_PERIPH_DMA2
  - LL_AHB1_GRP1_PERIPH_ADC12
  - LL_AHB1_GRP1_PERIPH_ART (*)
  - LL_AHB1_GRP1_PERIPH_CRC (*)
  - LL_AHB1_GRP1_PERIPH_ETH1MAC (*)
  - LL_AHB1_GRP1_PERIPH_USB1OTGHS
  - LL_AHB1_GRP1_PERIPH_USB2OTGHS (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1RSTR DMA1RST LL_AHB1_GRP1_ForceReset
- AHB1RSTR DMA2RST LL_AHB1_GRP1_ForceReset
- AHB1RSTR ADC12RST LL_AHB1_GRP1_ForceReset
- AHB1RSTR ARTRST LL_AHB1_GRP1_ForceReset
- (*) AHB1RSTR CRCRST LL_AHB1_GRP1_ForceReset
- (*) AHB1RSTR ETH1MACRST LL_AHB1_GRP1_ForceReset
- (*) AHB1RSTR USB1OTGHSRST LL_AHB1_GRP1_ForceReset
- AHB1RSTR USB2OTGHSRST LL_AHB1_GRP1_ForceReset (*)

LL_AHB1_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_AHB1_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release AHB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_DMA1
  - LL_AHB1_GRP1_PERIPH_DMA2
  - LL_AHB1_GRP1_PERIPH_ADC12
  - LL_AHB1_GRP1_PERIPH_ART (*)
  - LL_AHB1_GRP1_PERIPH_CRC (*)
  - LL_AHB1_GRP1_PERIPH_ETH1MAC (*)
  - LL_AHB1_GRP1_PERIPH_USB1OTGHS
  - LL_AHB1_GRP1_PERIPH_USB2OTGHS (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1RSTR DMA1RST LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR DMA2RST LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR ADC12RST LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR ARTRST LL_AHB1_GRP1_ReleaseReset
- (*) AHB1RSTR CRCRST LL_AHB1_GRP1_ReleaseReset
- (*) AHB1RSTR ETH1MACRST LL_AHB1_GRP1_ReleaseReset
- (*) AHB1RSTR USB1OTGHSRST LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR USB2OTGHSRST LL_AHB1_GRP1_ReleaseReset (*)

LL_AHB1_GRP1_EnableClockSleep

Function name

__STATIC_INLINE void LL_AHB1_GRP1_EnableClockSleep (uint32_t Periphs)

Function description

Enable AHB1 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_DMA1
  - LL_AHB1_GRP1_PERIPH_DMA2
  - LL_AHB1_GRP1_PERIPH_ADC12
  - LL_AHB1_GRP1_PERIPH_ART (*)
  - LL_AHB1_GRP1_PERIPH_CRC (*)
  - LL_AHB1_GRP1_PERIPH_ETH1MAC (*)
  - LL_AHB1_GRP1_PERIPH_ETH1TX (*)
  - LL_AHB1_GRP1_PERIPH_ETH1RX (*)
  - LL_AHB1_GRP1_PERIPH_USB1OTGHS
  - LL_AHB1_GRP1_PERIPH_USB1OTGHSULPI
  - LL_AHB1_GRP1_PERIPH_USB2OTGHS (*)
  - LL_AHB1_GRP1_PERIPH_USB2OTGHSULPI (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1LPENR DMA1LPEN LL_AHB1_GRP1_EnableClockSleep
- AHB1LPENR DMA2LPEN LL_AHB1_GRP1_EnableClockSleep
- AHB1LPENR ADC12LPEN LL_AHB1_GRP1_EnableClockSleep
- AHB1LPENR ARTLPEN LL_AHB1_GRP1_EnableClockSleep
- (*) AHB1LPENR CRCLPEN LL_AHB1_GRP1_EnableClockSleep
- (*) AHB1LPENR ETH1MACLPEN LL_AHB1_GRP1_EnableClockSleep
- (*) AHB1LPENR ETH1TXLPEN LL_AHB1_GRP1_EnableClockSleep
- (*) AHB1LPENR ETH1RXLPEN LL_AHB1_GRP1_EnableClockSleep
- AHB1LPENR USB1OTGHSLPEN LL_AHB1_GRP1_EnableClockSleep
- AHB1LPENR USB1OTGHSULPILPEN LL_AHB1_GRP1_EnableClockSleep
- AHB1LPENR USB2OTGHSLPEN LL_AHB1_GRP1_EnableClockSleep
- (*) AHB1LPENR USB2OTGHSULPILPEN LL_AHB1_GRP1_EnableClockSleep (*)

LL_AHB1_GRP1_DisableClockSleep

Function name

__STATIC_INLINE void LL_AHB1_GRP1_DisableClockSleep (uint32_t Periphs)

Function description

Disable AHB1 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_DMA1
  - LL_AHB1_GRP1_PERIPH_DMA2
  - LL_AHB1_GRP1_PERIPH_ADC12
  - LL_AHB1_GRP1_PERIPH_ART (*)
  - LL_AHB1_GRP1_PERIPH_CRC (*)
  - LL_AHB1_GRP1_PERIPH_ETH1MAC (*)
  - LL_AHB1_GRP1_PERIPH_ETH1TX (*)
  - LL_AHB1_GRP1_PERIPH_ETH1RX (*)
  - LL_AHB1_GRP1_PERIPH_USB1OTGHS
  - LL_AHB1_GRP1_PERIPH_USB1OTGHSULPI
  - LL_AHB1_GRP1_PERIPH_USB2OTGHS (*)
  - LL_AHB1_GRP1_PERIPH_USB2OTGHSULPI (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1LPENR DMA1LPEN LL_AHB1_GRP1_DisableClockSleep
- AHB1LPENR DMA2LPEN LL_AHB1_GRP1_DisableClockSleep
- AHB1LPENR ADC12LPEN LL_AHB1_GRP1_DisableClockSleep
- AHB1LPENR ARTLPEN LL_AHB1_GRP1_DisableClockSleep
- (*) AHB1LPENR CRCLPEN LL_AHB1_GRP1_DisableClockSleep
- (*) AHB1LPENR ETH1MACLPEN LL_AHB1_GRP1_DisableClockSleep
- (*) AHB1LPENR ETH1TXLPEN LL_AHB1_GRP1_DisableClockSleep
- (*) AHB1LPENR ETH1RXLPEN LL_AHB1_GRP1_DisableClockSleep
- (*) AHB1LPENR USB1OTGHSLPEN LL_AHB1_GRP1_DisableClockSleep
- AHB1LPENR USB1OTGHSULPILPEN LL_AHB1_GRP1_DisableClockSleep
- AHB1LPENR USB2OTGHSLPEN LL_AHB1_GRP1_DisableClockSleep
- (*) AHB1LPENR USB2OTGHSULPILPEN LL_AHB1_GRP1_DisableClockSleep (*)

LL_AHB2_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_AHB2_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable AHB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_DCMI
  - LL_AHB2_GRP1_PERIPH_HSEM (*)
  - LL_AHB2_GRP1_PERIPH_CRYP (*)
  - LL_AHB2_GRP1_PERIPH_HASH (*)
  - LL_AHB2_GRP1_PERIPH_RNG
  - LL_AHB2_GRP1_PERIPH_SDMMC2
  - LL_AHB2_GRP1_PERIPH_BDMA1 (*)
  - LL_AHB2_GRP1_PERIPH_FMAC (*)
  - LL_AHB2_GRP1_PERIPH_CORDIC (*)
  - LL_AHB2_GRP1_PERIPH_D2SRAM1
  - LL_AHB2_GRP1_PERIPH_D2SRAM2
  - LL_AHB2_GRP1_PERIPH_D2SRAM3 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2ENR DCMIEN LL_AHB2_GRP1_EnableClock
- AHB2ENR HSEMEN LL_AHB2_GRP1_EnableClock
- (*) AHB2ENR CRYPEN LL_AHB2_GRP1_EnableClock
- (*) AHB2ENR HASHEN LL_AHB2_GRP1_EnableClock
- (*) AHB2ENR RNGEN LL_AHB2_GRP1_EnableClock
- AHB2ENR SDMMC2EN LL_AHB2_GRP1_EnableClock
- AHB2ENR BDMA1EN LL_AHB2_GRP1_EnableClock
- (*) AHB2ENR FMACEN LL_AHB2_GRP1_EnableClock
- AHB2ENR CORDICEN LL_AHB2_GRP1_EnableClock
- AHB2ENR D2SRAM1EN LL_AHB2_GRP1_EnableClock
- AHB2ENR D2SRAM2EN LL_AHB2_GRP1_EnableClock
- AHB2ENR D2SRAM3EN LL_AHB2_GRP1_EnableClock (*)

LL_AHB2_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_AHB2_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if AHB2 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_DCMI
  - LL_AHB2_GRP1_PERIPH_HSEM (*)
  - LL_AHB2_GRP1_PERIPH_CRYP (*)
  - LL_AHB2_GRP1_PERIPH_HASH (*)
  - LL_AHB2_GRP1_PERIPH_RNG
  - LL_AHB2_GRP1_PERIPH_SDMMC2
  - LL_AHB2_GRP1_PERIPH_BDMA1 (*)
  - LL_AHB2_GRP1_PERIPH_FMAC (*)
  - LL_AHB2_GRP1_PERIPH_CORDIC (*)
  - LL_AHB2_GRP1_PERIPH_D2SRAM1
  - LL_AHB2_GRP1_PERIPH_D2SRAM2
  - LL_AHB2_GRP1_PERIPH_D2SRAM3 (*)

  (*) value not defined in all devices.

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- AHB2ENR DCMIEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR HSEMEN LL_AHB2_GRP1_IsEnabledClock
- (*) AHB2ENR CRYPEN LL_AHB2_GRP1_IsEnabledClock
- (*) AHB2ENR HASHEN LL_AHB2_GRP1_IsEnabledClock
- (*) AHB2ENR RNGEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR SDMMC2EN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR BDMA1EN LL_AHB2_GRP1_IsEnabledClock
- (*) AHB2ENR FMACEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR CORDICEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR D2SRAM1EN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR D2SRAM2EN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR D2SRAM3EN LL_AHB2_GRP1_IsEnabledClock (*)

LL_AHB2_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_AHB2_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable AHB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_DCMI
  - LL_AHB2_GRP1_PERIPH_HSEM (*)
  - LL_AHB2_GRP1_PERIPH_CRYP (*)
  - LL_AHB2_GRP1_PERIPH_HASH (*)
  - LL_AHB2_GRP1_PERIPH_RNG
  - LL_AHB2_GRP1_PERIPH_SDMMC2
  - LL_AHB2_GRP1_PERIPH_BDMA1 (*)
  - LL_AHB2_GRP1_PERIPH_FMAC (*)
  - LL_AHB2_GRP1_PERIPH_CORDIC (*)
  - LL_AHB2_GRP1_PERIPH_D2SRAM1
  - LL_AHB2_GRP1_PERIPH_D2SRAM2
  - LL_AHB2_GRP1_PERIPH_D2SRAM3 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2ENR DCMIEN LL_AHB2_GRP1_DisableClock
- AHB2ENR HSEMEN LL_AHB2_GRP1_DisableClock
- (*) AHB2ENR CRYPEN LL_AHB2_GRP1_DisableClock
- (*) AHB2ENR HASHEN LL_AHB2_GRP1_DisableClock
- (*) AHB2ENR RNGEN LL_AHB2_GRP1_DisableClock
- AHB2ENR SDMMC2EN LL_AHB2_GRP1_DisableClock
- AHB2ENR BDMA1EN LL_AHB2_GRP1_DisableClock
- (*) AHB2ENR FMACEN LL_AHB2_GRP1_DisableClock
- AHB2ENR CORDICEN LL_AHB2_GRP1_DisableClock
- AHB2ENR D2SRAM1EN LL_AHB2_GRP1_DisableClock
- AHB2ENR D2SRAM2EN LL_AHB2_GRP1_DisableClock
- AHB2ENR D2SRAM3EN LL_AHB2_GRP1_DisableClock (*)

LL_AHB2_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_AHB2_GRP1_ForceReset (uint32_t Periphs)

Function description

Force AHB2 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_DCMI
  - LL_AHB2_GRP1_PERIPH_HSEM (*)
  - LL_AHB2_GRP1_PERIPH_CRYP (*)
  - LL_AHB2_GRP1_PERIPH_HASH (*)
  - LL_AHB2_GRP1_PERIPH_RNG
  - LL_AHB2_GRP1_PERIPH_SDMMC2
  - LL_AHB2_GRP1_PERIPH_BDMA1 (*)
  - LL_AHB2_GRP1_PERIPH_FMAC (*)
  - LL_AHB2_GRP1_PERIPH_CORDIC (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2RSTR DCMIRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR HSEMRST LL_AHB2_GRP1_ForceReset
- (*) AHB2RSTR CRYPRST LL_AHB2_GRP1_ForceReset
- (*) AHB2RSTR HASHRST LL_AHB2_GRP1_ForceReset
- (*) AHB2RSTR RNGRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR SDMMC2RST LL_AHB2_GRP1_ForceReset
- AHB2RSTR BDMA1RST LL_AHB2_GRP1_ForceReset
- (*) AHB2RSTR FMACRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR CORDICRST LL_AHB2_GRP1_ForceReset

LL_AHB2_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_AHB2_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release AHB2 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_DCMI
  - LL_AHB2_GRP1_PERIPH_HSEM (*)
  - LL_AHB2_GRP1_PERIPH_CRYP (*)
  - LL_AHB2_GRP1_PERIPH_HASH (*)
  - LL_AHB2_GRP1_PERIPH_RNG
  - LL_AHB2_GRP1_PERIPH_SDMMC2
  - LL_AHB2_GRP1_PERIPH_BDMA1 (*)
  - LL_AHB2_GRP1_PERIPH_FMAC (*)
  - LL_AHB2_GRP1_PERIPH_CORDIC (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2RSTR DCMIRST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR HSEMRST LL_AHB2_GRP1_ReleaseReset
- (*) AHB2RSTR CRYPRST LL_AHB2_GRP1_ReleaseReset
- (*) AHB2RSTR HASHRST LL_AHB2_GRP1_ReleaseReset
- (*) AHB2RSTR RNGRST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR SDMMC2RST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR BDMA1RST LL_AHB2_GRP1_ReleaseReset
- (*) AHB2RSTR FMACRST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR CORDICRST LL_AHB2_GRP1_ReleaseReset

LL_AHB2_GRP1_EnableClockSleep

Function name

__STATIC_INLINE void LL_AHB2_GRP1_EnableClockSleep (uint32_t Periphs)

Function description

Enable AHB2 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_DCMI
  - LL_AHB2_GRP1_PERIPH_CRYP (*)
  - LL_AHB2_GRP1_PERIPH_HASH (*)
  - LL_AHB2_GRP1_PERIPH_RNG
  - LL_AHB2_GRP1_PERIPH_SDMMC2
  - LL_AHB2_GRP1_PERIPH_BDMA1 (*)
  - LL_AHB2_GRP1_PERIPH_FMAC (*)
  - LL_AHB2_GRP1_PERIPH_CORDIC (*)
  - LL_AHB2_GRP1_PERIPH_D2SRAM1
  - LL_AHB2_GRP1_PERIPH_D2SRAM2
  - LL_AHB2_GRP1_PERIPH_D2SRAM3 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2LPENR DCMILPEN LL_AHB2_GRP1_EnableClockSleep
- AHB2LPENR CRYPLPEN LL_AHB2_GRP1_EnableClockSleep
- (*) AHB2LPENR HASHLPEN LL_AHB2_GRP1_EnableClockSleep
- (*) AHB2LPENR RNGLPEN LL_AHB2_GRP1_EnableClockSleep
- AHB2LPENR SDMMC2LPEN LL_AHB2_GRP1_EnableClockSleep
- AHB2LPENR BDMA1LPEN LL_AHB2_GRP1_EnableClockSleep
- (*) AHB2LPENR FMACLPEN LL_AHB2_GRP1_EnableClockSleep
- AHB2LPENR CORDICLPEN LL_AHB2_GRP1_EnableClockSleep
- AHB2LPENR D2SRAM1LPEN LL_AHB2_GRP1_EnableClockSleep
- AHB2LPENR D2SRAM2LPEN LL_AHB2_GRP1_EnableClockSleep
- AHB2LPENR D2SRAM3LPEN LL_AHB2_GRP1_EnableClockSleep (*)

LL_AHB2_GRP1_DisableClockSleep

Function name

__STATIC_INLINE void LL_AHB2_GRP1_DisableClockSleep (uint32_t Periphs)

Function description

Disable AHB2 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_DCMI
  - LL_AHB2_GRP1_PERIPH_CRYP (*)
  - LL_AHB2_GRP1_PERIPH_HASH (*)
  - LL_AHB2_GRP1_PERIPH_RNG
  - LL_AHB2_GRP1_PERIPH_SDMMC2
  - LL_AHB2_GRP1_PERIPH_BDMA1 (*)
  - LL_AHB2_GRP1_PERIPH_FMAC (*)
  - LL_AHB2_GRP1_PERIPH_CORDIC (*)
  - LL_AHB2_GRP1_PERIPH_D2SRAM1
  - LL_AHB2_GRP1_PERIPH_D2SRAM2
  - LL_AHB2_GRP1_PERIPH_D2SRAM3 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2LPENR DCMILPEN LL_AHB2_GRP1_DisableClockSleep
- AHB2LPENR CRYPLPEN LL_AHB2_GRP1_DisableClockSleep
- (*) AHB2LPENR HASHLPEN LL_AHB2_GRP1_DisableClockSleep
- (*) AHB2LPENR RNGLPEN LL_AHB2_GRP1_DisableClockSleep
- AHB2LPENR SDMMC2LPEN LL_AHB2_GRP1_DisableClockSleep
- AHB2LPENR BDMA1LPEN LL_AHB2_GRP1_DisableClockSleep
- (*) AHB2LPENR D2SRAM1LPEN LL_AHB2_GRP1_DisableClockSleep
- AHB2LPENR D2SRAM2LPEN LL_AHB2_GRP1_DisableClockSleep
- AHB2LPENR D2SRAM3LPEN LL_AHB2_GRP1_DisableClockSleep (*)

LL_AHB4_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_AHB4_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable AHB4 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_GPIOA
  - LL_AHB4_GRP1_PERIPH_GPIOB
  - LL_AHB4_GRP1_PERIPH_GPIOC
  - LL_AHB4_GRP1_PERIPH_GPIOD
  - LL_AHB4_GRP1_PERIPH_GPIOE
  - LL_AHB4_GRP1_PERIPH_GPIOF
  - LL_AHB4_GRP1_PERIPH_GPIOG
  - LL_AHB4_GRP1_PERIPH_GPIOH
  - LL_AHB4_GRP1_PERIPH_GPIOI (*)
  - LL_AHB4_GRP1_PERIPH_GPIOJ
  - LL_AHB4_GRP1_PERIPH_GPIOK
  - LL_AHB4_GRP1_PERIPH_CRC (*)
  - LL_AHB4_GRP1_PERIPH_BDMA
  - LL_AHB4_GRP1_PERIPH_ADC3 (*)
  - LL_AHB4_GRP1_PERIPH_HSEM (*)
  - LL_AHB4_GRP1_PERIPH_BKPRAM
  - LL_AHB4_GRP1_PERIPH_SRAM4

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB4ENR GPIOAEN LL_AHB4_GRP1_EnableClock
- AHB4ENR GPIOBEN LL_AHB4_GRP1_EnableClock
- AHB4ENR GPIOCEN LL_AHB4_GRP1_EnableClock
- AHB4ENR GPIODEN LL_AHB4_GRP1_EnableClock
- AHB4ENR GPIOEEN LL_AHB4_GRP1_EnableClock
- AHB4ENR GPIOFEN LL_AHB4_GRP1_EnableClock
- AHB4ENR GPIOGEN LL_AHB4_GRP1_EnableClock
- AHB4ENR GPIOHEN LL_AHB4_GRP1_EnableClock
- AHB4ENR GPIOIEN LL_AHB4_GRP1_EnableClock
- (*) AHB4ENR GPIOJEN LL_AHB4_GRP1_EnableClock
- AHB4ENR GPIOKEN LL_AHB4_GRP1_EnableClock
- AHB4ENR CRCEN LL_AHB4_GRP1_EnableClock
- (*) AHB4ENR BDMAEN LL_AHB4_GRP1_EnableClock
- AHB4ENR ADC3EN LL_AHB4_GRP1_EnableClock
- (*) AHB4ENR HSEMEN LL_AHB4_GRP1_EnableClock
- (*) AHB4ENR BKPRAMEN LL_AHB4_GRP1_EnableClock
- AHB4ENR SRAM4EN LL_AHB4_GRP1_EnableClock

LL_AHB4_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_AHB4_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if AHB4 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_GPIOA
  - LL_AHB4_GRP1_PERIPH_GPIOB
  - LL_AHB4_GRP1_PERIPH_GPIOC
  - LL_AHB4_GRP1_PERIPH_GPIOD
  - LL_AHB4_GRP1_PERIPH_GPIOE
  - LL_AHB4_GRP1_PERIPH_GPIOF
  - LL_AHB4_GRP1_PERIPH_GPIOG
  - LL_AHB4_GRP1_PERIPH_GPIOH
  - LL_AHB4_GRP1_PERIPH_GPIOI (*)
  - LL_AHB4_GRP1_PERIPH_GPIOJ
  - LL_AHB4_GRP1_PERIPH_GPIOK
  - LL_AHB4_GRP1_PERIPH_CRC (*)
  - LL_AHB4_GRP1_PERIPH_BDMA
  - LL_AHB4_GRP1_PERIPH_ADC3 (*)
  - LL_AHB4_GRP1_PERIPH_HSEM (*)
  - LL_AHB4_GRP1_PERIPH_BKPRAM
  - LL_AHB4_GRP1_PERIPH_SRAM4

  (*) value not defined in all devices.

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- AHB4ENR GPIOAEN LL_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOBEN LL_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOCEN LL_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIODEN LL_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOEEN LL_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOFEN LL_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOGEN LL_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOHEN LL_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOIEN LL_AHB4_GRP1_IsEnabledClock
- (*) AHB4ENR GPIOJEN LL_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOKEN LL_AHB4_GRP1_IsEnabledClock
- AHB4ENR CRCEN LL_AHB4_GRP1_IsEnabledClock
- (*) AHB4ENR BDMAEN LL_AHB4_GRP1_IsEnabledClock
- AHB4ENR ADC3EN LL_AHB4_GRP1_IsEnabledClock
- (*) AHB4ENR HSEMEN LL_AHB4_GRP1_IsEnabledClock
- (*) AHB4ENR BKPRAMEN LL_AHB4_GRP1_IsEnabledClock
- AHB4ENR SRAM4EN LL_AHB4_GRP1_IsEnabledClock

LL_AHB4_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_AHB4_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable AHB4 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_GPIOA
  - LL_AHB4_GRP1_PERIPH_GPIOB
  - LL_AHB4_GRP1_PERIPH_GPIOC
  - LL_AHB4_GRP1_PERIPH_GPIOD
  - LL_AHB4_GRP1_PERIPH_GPIOE
  - LL_AHB4_GRP1_PERIPH_GPIOF
  - LL_AHB4_GRP1_PERIPH_GPIOG
  - LL_AHB4_GRP1_PERIPH_GPIOH
  - LL_AHB4_GRP1_PERIPH_GPIOI (*)
  - LL_AHB4_GRP1_PERIPH_GPIOJ
  - LL_AHB4_GRP1_PERIPH_GPIOK
  - LL_AHB4_GRP1_PERIPH_CRC (*)
  - LL_AHB4_GRP1_PERIPH_BDMA
  - LL_AHB4_GRP1_PERIPH_ADC3 (*)
  - LL_AHB4_GRP1_PERIPH_HSEM (*)
  - LL_AHB4_GRP1_PERIPH_BKPRAM
  - LL_AHB4_GRP1_PERIPH_SRAM4

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB4ENR GPIOAEN LL_AHB4_GRP1_DisableClock
- AHB4ENR GPIOBEN LL_AHB4_GRP1_DisableClock
- AHB4ENR GPIOCEN LL_AHB4_GRP1_DisableClock
- AHB4ENR GPIODEN LL_AHB4_GRP1_DisableClock
- AHB4ENR GPIOEEN LL_AHB4_GRP1_DisableClock
- AHB4ENR GPIOFEN LL_AHB4_GRP1_DisableClock
- AHB4ENR GPIOGEN LL_AHB4_GRP1_DisableClock
- AHB4ENR GPIOHEN LL_AHB4_GRP1_DisableClock
- AHB4ENR GPIOIEN LL_AHB4_GRP1_DisableClock
- (*) AHB4ENR GPIOJEN LL_AHB4_GRP1_DisableClock
- AHB4ENR GPIOKEN LL_AHB4_GRP1_DisableClock
- AHB4ENR CRCEN LL_AHB4_GRP1_DisableClock
- (*) AHB4ENR BDMAEN LL_AHB4_GRP1_DisableClock
- AHB4ENR ADC3EN LL_AHB4_GRP1_DisableClock
- (*) AHB4ENR HSEMEN LL_AHB4_GRP1_DisableClock
- (*) AHB4ENR BKPRAMEN LL_AHB4_GRP1_DisableClock
- AHB4ENR SRAM4EN LL_AHB4_GRP1_DisableClock

LL_AHB4_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_AHB4_GRP1_ForceReset (uint32_t Periphs)

Function description

Force AHB4 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_GPIOA
  - LL_AHB4_GRP1_PERIPH_GPIOB
  - LL_AHB4_GRP1_PERIPH_GPIOC
  - LL_AHB4_GRP1_PERIPH_GPIOD
  - LL_AHB4_GRP1_PERIPH_GPIOE
  - LL_AHB4_GRP1_PERIPH_GPIOF
  - LL_AHB4_GRP1_PERIPH_GPIOG
  - LL_AHB4_GRP1_PERIPH_GPIOH
  - LL_AHB4_GRP1_PERIPH_GPIOI (*)
  - LL_AHB4_GRP1_PERIPH_GPIOJ
  - LL_AHB4_GRP1_PERIPH_GPIOK
  - LL_AHB4_GRP1_PERIPH_CRC (*)
  - LL_AHB4_GRP1_PERIPH_BDMA
  - LL_AHB4_GRP1_PERIPH_ADC3 (*)
  - LL_AHB4_GRP1_PERIPH_HSEM (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB4RSTR GPIOARST LL_AHB4_GRP1_ForceReset
- AHB4RSTR GPIOBRST LL_AHB4_GRP1_ForceReset
- AHB4RSTR GPIOCRST LL_AHB4_GRP1_ForceReset
- AHB4RSTR GPIODRST LL_AHB4_GRP1_ForceReset
- AHB4RSTR GPIOERST LL_AHB4_GRP1_ForceReset
- AHB4RSTR GPIOFRST LL_AHB4_GRP1_ForceReset
- AHB4RSTR GPIOGRST LL_AHB4_GRP1_ForceReset
- AHB4RSTR GPIOHRST LL_AHB4_GRP1_ForceReset
- AHB4RSTR GPIOIRST LL_AHB4_GRP1_ForceReset
- (*) AHB4RSTR GPIOJRST LL_AHB4_GRP1_ForceReset
- AHB4RSTR GPIOKRST LL_AHB4_GRP1_ForceReset
- AHB4RSTR CRCRST LL_AHB4_GRP1_ForceReset
- (*) AHB4RSTR BDMARST LL_AHB4_GRP1_ForceReset
- AHB4RSTR ADC3RST LL_AHB4_GRP1_ForceReset
- (*) AHB4RSTR HSEMRST LL_AHB4_GRP1_ForceReset (*)

LL_AHB4_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_AHB4_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release AHB4 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_GPIOA
  - LL_AHB4_GRP1_PERIPH_GPIOB
  - LL_AHB4_GRP1_PERIPH_GPIOC
  - LL_AHB4_GRP1_PERIPH_GPIOD
  - LL_AHB4_GRP1_PERIPH_GPIOE
  - LL_AHB4_GRP1_PERIPH_GPIOF
  - LL_AHB4_GRP1_PERIPH_GPIOG
  - LL_AHB4_GRP1_PERIPH_GPIOH
  - LL_AHB4_GRP1_PERIPH_GPIOI (*)
  - LL_AHB4_GRP1_PERIPH_GPIOJ
  - LL_AHB4_GRP1_PERIPH_GPIOK
  - LL_AHB4_GRP1_PERIPH_CRC (*)
  - LL_AHB4_GRP1_PERIPH_BDMA
  - LL_AHB4_GRP1_PERIPH_ADC3 (*)
  - LL_AHB4_GRP1_PERIPH_HSEM (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB4RSTR GPIOARST LL_AHB4_GRP1_ReleaseReset
- AHB4RSTR GPIOBRST LL_AHB4_GRP1_ReleaseReset
- AHB4RSTR GPIOCRST LL_AHB4_GRP1_ReleaseReset
- AHB4RSTR GPIODRST LL_AHB4_GRP1_ReleaseReset
- AHB4RSTR GPIOERST LL_AHB4_GRP1_ReleaseReset
- AHB4RSTR GPIOFRST LL_AHB4_GRP1_ReleaseReset
- AHB4RSTR GPIOGRST LL_AHB4_GRP1_ReleaseReset
- AHB4RSTR GPIOHRST LL_AHB4_GRP1_ReleaseReset
- AHB4RSTR GPIOIRST LL_AHB4_GRP1_ReleaseReset
- (*) AHB4RSTR GPIOJRST LL_AHB4_GRP1_ReleaseReset
- AHB4RSTR GPIOKRST LL_AHB4_GRP1_ReleaseReset
- AHB4RSTR CRCRST LL_AHB4_GRP1_ReleaseReset
- (*) AHB4RSTR BDMARST LL_AHB4_GRP1_ReleaseReset
- AHB4RSTR ADC3RST LL_AHB4_GRP1_ReleaseReset
- (*) AHB4RSTR HSEMRST LL_AHB4_GRP1_ReleaseReset (*)

LL_AHB4_GRP1_EnableClockSleep

Function name

__STATIC_INLINE void LL_AHB4_GRP1_EnableClockSleep (uint32_t Periphs)

Function description

Enable AHB4 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_GPIOA
  - LL_AHB4_GRP1_PERIPH_GPIOB
  - LL_AHB4_GRP1_PERIPH_GPIOC
  - LL_AHB4_GRP1_PERIPH_GPIOD
  - LL_AHB4_GRP1_PERIPH_GPIOE
  - LL_AHB4_GRP1_PERIPH_GPIOF
  - LL_AHB4_GRP1_PERIPH_GPIOG
  - LL_AHB4_GRP1_PERIPH_GPIOH
  - LL_AHB4_GRP1_PERIPH_GPIOI (*)
  - LL_AHB4_GRP1_PERIPH_GPIOJ
  - LL_AHB4_GRP1_PERIPH_GPIOK
  - LL_AHB4_GRP1_PERIPH_CRC (*)
  - LL_AHB4_GRP1_PERIPH_BDMA
  - LL_AHB4_GRP1_PERIPH_ADC3 (*)
  - LL_AHB4_GRP1_PERIPH_BKPRAM
  - LL_AHB4_GRP1_PERIPH_SRAM4

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB4LPENR GPIOALPEN LL_AHB4_GRP1_EnableClockSleep
- AHB4LPENR GPIOBLPEN LL_AHB4_GRP1_EnableClockSleep
- AHB4LPENR GPIOCLPEN LL_AHB4_GRP1_EnableClockSleep
- AHB4LPENR GPIODLPEN LL_AHB4_GRP1_EnableClockSleep
- AHB4LPENR GPIOELPEN LL_AHB4_GRP1_EnableClockSleep
- AHB4LPENR GPIOFLPEN LL_AHB4_GRP1_EnableClockSleep
- AHB4LPENR GPIOGLPEN LL_AHB4_GRP1_EnableClockSleep
- AHB4LPENR GPIOHLPEN LL_AHB4_GRP1_EnableClockSleep
- AHB4LPENR GPIOILPEN LL_AHB4_GRP1_EnableClockSleep
- (*) AHB4LPENR GPIOJLPEN LL_AHB4_GRP1_EnableClockSleep
- AHB4LPENR GPIOKLPEN LL_AHB4_GRP1_EnableClockSleep
- AHB4LPENR CRCLPEN LL_AHB4_GRP1_EnableClockSleep
- (*) AHB4LPENR BDMALPEN LL_AHB4_GRP1_EnableClockSleep
- AHB4LPENR ADC3LPEN LL_AHB4_GRP1_EnableClockSleep
- (*) AHB4LPENR BKPRAMLPEN LL_AHB4_GRP1_EnableClockSleep
- AHB4LPENR SRAM4LPEN LL_AHB4_GRP1_EnableClockSleep

LL_AHB4_GRP1_DisableClockSleep

Function name

__STATIC_INLINE void LL_AHB4_GRP1_DisableClockSleep (uint32_t Periphs)

Function description

Disable AHB4 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_GPIOA
  - LL_AHB4_GRP1_PERIPH_GPIOB
  - LL_AHB4_GRP1_PERIPH_GPIOC
  - LL_AHB4_GRP1_PERIPH_GPIOD
  - LL_AHB4_GRP1_PERIPH_GPIOE
  - LL_AHB4_GRP1_PERIPH_GPIOF
  - LL_AHB4_GRP1_PERIPH_GPIOG
  - LL_AHB4_GRP1_PERIPH_GPIOH
  - LL_AHB4_GRP1_PERIPH_GPIOI (*)
  - LL_AHB4_GRP1_PERIPH_GPIOJ
  - LL_AHB4_GRP1_PERIPH_GPIOK
  - LL_AHB4_GRP1_PERIPH_CRC (*)
  - LL_AHB4_GRP1_PERIPH_BDMA
  - LL_AHB4_GRP1_PERIPH_ADC3 (*)
  - LL_AHB4_GRP1_PERIPH_BKPRAM
  - LL_AHB4_GRP1_PERIPH_SRAM4

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB4LPENR GPIOALPEN LL_AHB4_GRP1_DisableClockSleep
- AHB4LPENR GPIOBLPEN LL_AHB4_GRP1_DisableClockSleep
- AHB4LPENR GPIOCLPEN LL_AHB4_GRP1_DisableClockSleep
- AHB4LPENR GPIODLPEN LL_AHB4_GRP1_DisableClockSleep
- AHB4LPENR GPIOELPEN LL_AHB4_GRP1_DisableClockSleep
- AHB4LPENR GPIOFLPEN LL_AHB4_GRP1_DisableClockSleep
- AHB4LPENR GPIOGLPEN LL_AHB4_GRP1_DisableClockSleep
- AHB4LPENR GPIOHLPEN LL_AHB4_GRP1_DisableClockSleep
- AHB4LPENR GPIOILPEN LL_AHB4_GRP1_DisableClockSleep
- (*) AHB4LPENR GPIOJLPEN LL_AHB4_GRP1_DisableClockSleep
- AHB4LPENR GPIOKLPEN LL_AHB4_GRP1_DisableClockSleep
- AHB4LPENR CRCLPEN LL_AHB4_GRP1_DisableClockSleep
- (*) AHB4LPENR BDMALPEN LL_AHB4_GRP1_DisableClockSleep
- AHB4LPENR ADC3LPEN LL_AHB4_GRP1_DisableClockSleep
- (*) AHB4LPENR BKPRAMLPEN LL_AHB4_GRP1_DisableClockSleep
- AHB4LPENR SRAM4LPEN LL_AHB4_GRP1_DisableClockSleep

LL_APB3_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_APB3_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable APB3 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB3_GRP1_PERIPH_LTDC (*)
  - LL_APB3_GRP1_PERIPH_DSI (*)
  - LL_APB3_GRP1_PERIPH_WWDG1

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB3ENR LTDCEN LL_APB3_GRP1_EnableClock
- (*) APB3ENR DSIEN LL_APB3_GRP1_EnableClock
- (*) APB3ENR WWDG1EN LL_APB3_GRP1_EnableClock

LL_APB3_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_APB3_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if APB3 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB3_GRP1_PERIPH_LTDC (*)
  - LL_APB3_GRP1_PERIPH_DSI (*)
  - LL_APB3_GRP1_PERIPH_WWDG1

  (*) value not defined in all devices.

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- APB3ENR LTDCEN LL_APB3_GRP1_IsEnabledClock
- (*) APB3ENR DSIEN LL_APB3_GRP1_IsEnabledClock
- (*) APB3ENR WWDG1EN LL_APB3_GRP1_IsEnabledClock

LL_APB3_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_APB3_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable APB3 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB3_GRP1_PERIPH_LTDC (*)
  - LL_APB3_GRP1_PERIPH_DSI (*)
  - LL_APB3_GRP1_PERIPH_WWDG1

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB3ENR LTDCEN LL_APB3_GRP1_DisableClock
- APB3ENR DSIEN LL_APB3_GRP1_DisableClock
- APB3ENR WWDG1EN LL_APB3_GRP1_DisableClock

LL_APB3_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_APB3_GRP1_ForceReset (uint32_t Periphs)

Function description

Force APB3 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB3_GRP1_PERIPH_LTDC (*)
  - LL_APB3_GRP1_PERIPH_DSI (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB3RSTR LTDCRST LL_APB3_GRP1_ForceReset
- (*) APB3RSTR DSIRST LL_APB3_GRP1_ForceReset (*)

LL_APB3_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_APB3_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release APB3 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB3_GRP1_PERIPH_LTDC (*)
  - LL_APB3_GRP1_PERIPH_DSI (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB3RSTR LTDCRST LL_APB3_GRP1_ReleaseReset
- APB3RSTR DSIRST LL_APB3_GRP1_ReleaseReset

LL_APB3_GRP1_EnableClockSleep

Function name

__STATIC_INLINE void LL_APB3_GRP1_EnableClockSleep (uint32_t Periphs)

Function description

Enable APB3 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB3_GRP1_PERIPH_LTDC (*)
  - LL_APB3_GRP1_PERIPH_DSI (*)
  - LL_APB3_GRP1_PERIPH_WWDG1

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB3LPENR LTDCLPEN LL_APB3_GRP1_EnableClockSleep
- (*) APB3LPENR DSILPEN LL_APB3_GRP1_EnableClockSleep
- (*) APB3LPENR WWDG1LPEN LL_APB3_GRP1_EnableClockSleep

LL_APB3_GRP1_DisableClockSleep

Function name

__STATIC_INLINE void LL_APB3_GRP1_DisableClockSleep (uint32_t Periphs)

Function description

Disable APB3 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB3_GRP1_PERIPH_LTDC (*)
  - LL_APB3_GRP1_PERIPH_DSI (*)
  - LL_APB3_GRP1_PERIPH_WWDG1

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB3LPENR LTDCLPEN LL_APB3_GRP1_DisableClockSleep
- (*) APB3LPENR DSILPEN LL_APB3_GRP1_DisableClockSleep
- (*) APB3LPENR WWDG1LPEN LL_APB3_GRP1_DisableClockSleep

LL_APB1_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_APB1_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable APB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM12
  - LL_APB1_GRP1_PERIPH_TIM13
  - LL_APB1_GRP1_PERIPH_TIM14
  - LL_APB1_GRP1_PERIPH_LPTIM1
  - LL_APB1_GRP1_PERIPH_WWDG2 (*)
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_SPDIFRX
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4
  - LL_APB1_GRP1_PERIPH_UART5
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_I2C5 (*)
  - LL_APB1_GRP1_PERIPH_CEC
  - LL_APB1_GRP1_PERIPH_DAC12
  - LL_APB1_GRP1_PERIPH_UART7
  - LL_APB1_GRP1_PERIPH_UART8

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1LENR TIM2EN LL_APB1_GRP1_EnableClock
- APB1LENR TIM3EN LL_APB1_GRP1_EnableClock
- APB1LENR TIM4EN LL_APB1_GRP1_EnableClock
- APB1LENR TIM5EN LL_APB1_GRP1_EnableClock
- APB1LENR TIM6EN LL_APB1_GRP1_EnableClock
- APB1LENR TIM7EN LL_APB1_GRP1_EnableClock
- APB1LENR TIM12EN LL_APB1_GRP1_EnableClock
- APB1LENR TIM13EN LL_APB1_GRP1_EnableClock
- APB1LENR TIM14EN LL_APB1_GRP1_EnableClock
- APB1LENR LPTIM1EN LL_APB1_GRP1_EnableClock
- APB1LENR WWDG2EN LL_APB1_GRP1_EnableClock
- (*) APB1LENR SPI2EN LL_APB1_GRP1_EnableClock
- APB1LENR SPI3EN LL_APB1_GRP1_EnableClock
- APB1LENR SPDIFRXEN LL_APB1_GRP1_EnableClock
- APB1LENR USART2EN LL_APB1_GRP1_EnableClock
- APB1LENR USART3EN LL_APB1_GRP1_EnableClock
- APB1LENR UART4EN LL_APB1_GRP1_EnableClock
- APB1LENR UART5EN LL_APB1_GRP1_EnableClock
- APB1LENR I2C1EN LL_APB1_GRP1_EnableClock
- APB1LENR I2C2EN LL_APB1_GRP1_EnableClock
- APB1LENR I2C3EN LL_APB1_GRP1_EnableClock
- APB1LENR I2C5EN LL_APB1_GRP1_EnableClock
- (*) APB1LENR CECEN LL_APB1_GRP1_EnableClock
- APB1LENR DAC12EN LL_APB1_GRP1_EnableClock
- APB1LENR UART7EN LL_APB1_GRP1_EnableClock
- APB1LENR UART8EN LL_APB1_GRP1_EnableClock

LL_APB1_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_APB1_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if APB1 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM12
  - LL_APB1_GRP1_PERIPH_TIM13
  - LL_APB1_GRP1_PERIPH_TIM14
  - LL_APB1_GRP1_PERIPH_LPTIM1
  - LL_APB1_GRP1_PERIPH_WWDG2 (*)
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_SPDIFRX
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4
  - LL_APB1_GRP1_PERIPH_UART5
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_I2C5 (*)
  - LL_APB1_GRP1_PERIPH_CEC
  - LL_APB1_GRP1_PERIPH_DAC12
  - LL_APB1_GRP1_PERIPH_UART7
  - LL_APB1_GRP1_PERIPH_UART8

  (*) value not defined in all devices.

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- APB1LENR TIM2EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR TIM3EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR TIM4EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR TIM5EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR TIM6EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR TIM7EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR TIM12EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR TIM13EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR TIM14EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR LPTIM1EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR WWDG2EN LL_APB1_GRP1_IsEnabledClock
- (*) APB1LENR SPI2EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR SPI3EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR SPDIFRXEN LL_APB1_GRP1_IsEnabledClock
- APB1LENR USART2EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR USART3EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR UART4EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR UART5EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR I2C1EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR I2C2EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR I2C3EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR I2C5EN LL_APB1_GRP1_IsEnabledClock
- (*) APB1LENR CECEN LL_APB1_GRP1_IsEnabledClock
- APB1LENR DAC12EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR UART7EN LL_APB1_GRP1_IsEnabledClock
- APB1LENR UART8EN LL_APB1_GRP1_IsEnabledClock

LL_APB1_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_APB1_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable APB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM12
  - LL_APB1_GRP1_PERIPH_TIM13
  - LL_APB1_GRP1_PERIPH_TIM14
  - LL_APB1_GRP1_PERIPH_LPTIM1
  - LL_APB1_GRP1_PERIPH_WWDG2 (*)
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_SPDIFRX
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4
  - LL_APB1_GRP1_PERIPH_UART5
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_I2C5 (*)
  - LL_APB1_GRP1_PERIPH_CEC
  - LL_APB1_GRP1_PERIPH_DAC12
  - LL_APB1_GRP1_PERIPH_UART7
  - LL_APB1_GRP1_PERIPH_UART8

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1LENR TIM2EN LL_APB1_GRP1_DisableClock
- APB1LENR TIM3EN LL_APB1_GRP1_DisableClock
- APB1LENR TIM4EN LL_APB1_GRP1_DisableClock
- APB1LENR TIM5EN LL_APB1_GRP1_DisableClock
- APB1LENR TIM6EN LL_APB1_GRP1_DisableClock
- APB1LENR TIM7EN LL_APB1_GRP1_DisableClock
- APB1LENR TIM12EN LL_APB1_GRP1_DisableClock
- APB1LENR TIM13EN LL_APB1_GRP1_DisableClock
- APB1LENR TIM14EN LL_APB1_GRP1_DisableClock
- APB1LENR LPTIM1EN LL_APB1_GRP1_DisableClock
- APB1LENR WWDG2EN LL_APB1_GRP1_DisableClock
- (*) APB1LENR SPI2EN LL_APB1_GRP1_DisableClock
- APB1LENR SPI3EN LL_APB1_GRP1_DisableClock
- APB1LENR SPDIFRXEN LL_APB1_GRP1_DisableClock
- APB1LENR USART2EN LL_APB1_GRP1_DisableClock
- APB1LENR USART3EN LL_APB1_GRP1_DisableClock
- APB1LENR UART4EN LL_APB1_GRP1_DisableClock
- APB1LENR UART5EN LL_APB1_GRP1_DisableClock
- APB1LENR I2C1EN LL_APB1_GRP1_DisableClock
- APB1LENR I2C2EN LL_APB1_GRP1_DisableClock
- APB1LENR I2C3EN LL_APB1_GRP1_DisableClock
- APB1LENR I2C5EN LL_APB1_GRP1_DisableClock
- (*) APB1LENR CECEN LL_APB1_GRP1_DisableClock
- APB1LENR DAC12EN LL_APB1_GRP1_DisableClock
- APB1LENR UART7EN LL_APB1_GRP1_DisableClock
- APB1LENR UART8EN LL_APB1_GRP1_DisableClock

LL_APB1_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_APB1_GRP1_ForceReset (uint32_t Periphs)

Function description

Force APB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM12
  - LL_APB1_GRP1_PERIPH_TIM13
  - LL_APB1_GRP1_PERIPH_TIM14
  - LL_APB1_GRP1_PERIPH_LPTIM1
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_SPDIFRX
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4
  - LL_APB1_GRP1_PERIPH_UART5
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_I2C5 (*)
  - LL_APB1_GRP1_PERIPH_CEC
  - LL_APB1_GRP1_PERIPH_DAC12
  - LL_APB1_GRP1_PERIPH_UART7
  - LL_APB1_GRP1_PERIPH_UART8

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1LRSTR TIM2RST LL_APB1_GRP1_ForceReset
- APB1LRSTR TIM3RST LL_APB1_GRP1_ForceReset
- APB1LRSTR TIM4RST LL_APB1_GRP1_ForceReset
- APB1LRSTR TIM5RST LL_APB1_GRP1_ForceReset
- APB1LRSTR TIM6RST LL_APB1_GRP1_ForceReset
- APB1LRSTR TIM7RST LL_APB1_GRP1_ForceReset
- APB1LRSTR TIM12RST LL_APB1_GRP1_ForceReset
- APB1LRSTR TIM13RST LL_APB1_GRP1_ForceReset
- APB1LRSTR TIM14RST LL_APB1_GRP1_ForceReset
- APB1LRSTR LPTIM1RST LL_APB1_GRP1_ForceReset
- APB1LRSTR SPI2RST LL_APB1_GRP1_ForceReset
- APB1LRSTR SPI3RST LL_APB1_GRP1_ForceReset
- APB1LRSTR SPDIFRXRST LL_APB1_GRP1_ForceReset
- APB1LRSTR USART2RST LL_APB1_GRP1_ForceReset
- APB1LRSTR USART3RST LL_APB1_GRP1_ForceReset
- APB1LRSTR UART4RST LL_APB1_GRP1_ForceReset
- APB1LRSTR UART5RST LL_APB1_GRP1_ForceReset
- APB1LRSTR I2C1RST LL_APB1_GRP1_ForceReset
- APB1LRSTR I2C2RST LL_APB1_GRP1_ForceReset
- APB1LRSTR I2C3RST LL_APB1_GRP1_ForceReset
- APB1LRSTR I2C5RST LL_APB1_GRP5_ForceReset
- (*) APB1LRSTR CECRST LL_APB1_GRP1_ForceReset
- APB1LRSTR DAC12RST LL_APB1_GRP1_ForceReset
- APB1LRSTR UART7RST LL_APB1_GRP1_ForceReset
- APB1LRSTR UART8RST LL_APB1_GRP1_ForceReset

LL_APB1_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_APB1_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release APB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM12
  - LL_APB1_GRP1_PERIPH_TIM13
  - LL_APB1_GRP1_PERIPH_TIM14
  - LL_APB1_GRP1_PERIPH_LPTIM1
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_SPDIFRX
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4
  - LL_APB1_GRP1_PERIPH_UART5
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_I2C5 (*)
  - LL_APB1_GRP1_PERIPH_CEC
  - LL_APB1_GRP1_PERIPH_DAC12
  - LL_APB1_GRP1_PERIPH_UART7
  - LL_APB1_GRP1_PERIPH_UART8

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1LRSTR TIM2RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR TIM3RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR TIM4RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR TIM5RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR TIM6RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR TIM7RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR TIM12RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR TIM13RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR TIM14RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR LPTIM1RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR SPI2RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR SPI3RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR SPDIFRXRST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR USART2RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR USART3RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR UART4RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR UART5RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR I2C1RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR I2C2RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR I2C3RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR I2C5RST LL_APB1_GRP1_ReleaseReset
- (*) APB1LRSTR CECRST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR DAC12RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR UART7RST LL_APB1_GRP1_ReleaseReset
- APB1LRSTR UART8RST LL_APB1_GRP1_ReleaseReset

LL_APB1_GRP1_EnableClockSleep

Function name

__STATIC_INLINE void LL_APB1_GRP1_EnableClockSleep (uint32_t Periphs)

Function description

Enable APB1 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM12
  - LL_APB1_GRP1_PERIPH_TIM13
  - LL_APB1_GRP1_PERIPH_TIM14
  - LL_APB1_GRP1_PERIPH_LPTIM1
  - LL_APB1_GRP1_PERIPH_WWDG2 (*)
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_SPDIFRX
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4
  - LL_APB1_GRP1_PERIPH_UART5
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_I2C5 (*)
  - LL_APB1_GRP1_PERIPH_CEC
  - LL_APB1_GRP1_PERIPH_DAC12
  - LL_APB1_GRP1_PERIPH_UART7
  - LL_APB1_GRP1_PERIPH_UART8

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1LLPENR TIM2LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR TIM3LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR TIM4LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR TIM5LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR TIM6LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR TIM7LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR TIM12LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR TIM13LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR TIM14LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR LPTIM1LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR WWDG2LPEN LL_APB1_GRP1_EnableClockSleep
- (*) APB1LLPENR SPI2LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR SPI3LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR SPDIFRXLPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR USART2LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR USART3LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR UART4LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR UART5LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR I2C1LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR I2C2LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR I2C3LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR I2C5LPEN LL_APB1_GRP1_EnableClockSleep
- (*) APB1LLPENR CECLPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR DAC12LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR UART7LPEN LL_APB1_GRP1_EnableClockSleep
- APB1LLPENR UART8LPEN LL_APB1_GRP1_EnableClockSleep

LL_APB1_GRP1_DisableClockSleep

Function name

__STATIC_INLINE void LL_APB1_GRP1_DisableClockSleep (uint32_t Periphs)

Function description

Disable APB1 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM12
  - LL_APB1_GRP1_PERIPH_TIM13
  - LL_APB1_GRP1_PERIPH_TIM14
  - LL_APB1_GRP1_PERIPH_LPTIM1
  - LL_APB1_GRP1_PERIPH_WWDG2 (*)
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_SPDIFRX
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4
  - LL_APB1_GRP1_PERIPH_UART5
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_I2C5 (*)
  - LL_APB1_GRP1_PERIPH_CEC
  - LL_APB1_GRP1_PERIPH_DAC12
  - LL_APB1_GRP1_PERIPH_UART7
  - LL_APB1_GRP1_PERIPH_UART8

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1LLPENR TIM2LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR TIM3LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR TIM4LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR TIM5LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR TIM6LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR TIM7LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR TIM12LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR TIM13LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR TIM14LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR LPTIM1LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR WWDG2LPEN LL_APB1_GRP1_DisableClockSleep
- (*) APB1LLPENR SPI2LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR SPI3LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR SPDIFRXLPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR USART2LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR USART3LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR UART4LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR UART5LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR I2C1LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR I2C2LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR I2C3LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR I2C5LPEN LL_APB1_GRP1_DisableClockSleep
- (*) APB1LLPENR CECLPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR DAC12LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR UART7LPEN LL_APB1_GRP1_DisableClockSleep
- APB1LLPENR UART8LPEN LL_APB1_GRP1_DisableClockSleep

LL_APB1_GRP2_EnableClock

Function name

__STATIC_INLINE void LL_APB1_GRP2_EnableClock (uint32_t Periphs)

Function description

Enable APB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_CRS
  - LL_APB1_GRP2_PERIPH_SWPMI1
  - LL_APB1_GRP2_PERIPH_OPAMP
  - LL_APB1_GRP2_PERIPH_MDIOS
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_TIM23 (*)
  - LL_APB1_GRP2_PERIPH_TIM24 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1HENR CRSEN LL_APB1_GRP2_EnableClock
- APB1HENR SWPMIEN LL_APB1_GRP2_EnableClock
- APB1HENR OPAMPEN LL_APB1_GRP2_EnableClock
- APB1HENR MDIOSEN LL_APB1_GRP2_EnableClock
- APB1HENR FDCANEN LL_APB1_GRP2_EnableClock

LL_APB1_GRP2_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_APB1_GRP2_IsEnabledClock (uint32_t Periphs)

Function description

Check if APB1 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_CRS
  - LL_APB1_GRP2_PERIPH_SWPMI1
  - LL_APB1_GRP2_PERIPH_OPAMP
  - LL_APB1_GRP2_PERIPH_MDIOS
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_TIM23 (*)
  - LL_APB1_GRP2_PERIPH_TIM24 (*)

  (*) value not defined in all devices.

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- APB1HENR CRSEN LL_APB1_GRP2_IsEnabledClock
- APB1HENR SWPMIEN LL_APB1_GRP2_IsEnabledClock
- APB1HENR OPAMPEN LL_APB1_GRP2_IsEnabledClock
- APB1HENR MDIOSEN LL_APB1_GRP2_IsEnabledClock
- APB1HENR FDCANEN LL_APB1_GRP2_IsEnabledClock

LL_APB1_GRP2_DisableClock

Function name

__STATIC_INLINE void LL_APB1_GRP2_DisableClock (uint32_t Periphs)

Function description

Disable APB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_CRS
  - LL_APB1_GRP2_PERIPH_SWPMI1
  - LL_APB1_GRP2_PERIPH_OPAMP
  - LL_APB1_GRP2_PERIPH_MDIOS
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_TIM23 (*)
  - LL_APB1_GRP2_PERIPH_TIM24 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1HENR CRSEN LL_APB1_GRP2_DisableClock
- APB1HENR SWPMIEN LL_APB1_GRP2_DisableClock
- APB1HENR OPAMPEN LL_APB1_GRP2_DisableClock
- APB1HENR MDIOSEN LL_APB1_GRP2_DisableClock
- APB1HENR FDCANEN LL_APB1_GRP2_DisableClock

LL_APB1_GRP2_ForceReset

Function name

__STATIC_INLINE void LL_APB1_GRP2_ForceReset (uint32_t Periphs)

Function description

Force APB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_CRS
  - LL_APB1_GRP2_PERIPH_SWPMI1
  - LL_APB1_GRP2_PERIPH_OPAMP
  - LL_APB1_GRP2_PERIPH_MDIOS
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_TIM23 (*)
  - LL_APB1_GRP2_PERIPH_TIM24 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1HRSTR CRSRST LL_APB1_GRP2_ForceReset
- APB1HRSTR SWPMIRST LL_APB1_GRP2_ForceReset
- APB1HRSTR OPAMPRST LL_APB1_GRP2_ForceReset
- APB1HRSTR MDIOSRST LL_APB1_GRP2_ForceReset
- APB1HRSTR FDCANRST LL_APB1_GRP2_ForceReset

LL_APB1_GRP2_ReleaseReset

Function name

__STATIC_INLINE void LL_APB1_GRP2_ReleaseReset (uint32_t Periphs)

Function description

Release APB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_CRS
  - LL_APB1_GRP2_PERIPH_SWPMI1
  - LL_APB1_GRP2_PERIPH_OPAMP
  - LL_APB1_GRP2_PERIPH_MDIOS
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_TIM23 (*)
  - LL_APB1_GRP2_PERIPH_TIM24 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1HRSTR CRSRST LL_APB1_GRP2_ReleaseReset
- APB1HRSTR SWPMIRST LL_APB1_GRP2_ReleaseReset
- APB1HRSTR OPAMPRST LL_APB1_GRP2_ReleaseReset
- APB1HRSTR MDIOSRST LL_APB1_GRP2_ReleaseReset
- APB1HRSTR FDCANRST LL_APB1_GRP2_ReleaseReset

LL_APB1_GRP2_EnableClockSleep

Function name

__STATIC_INLINE void LL_APB1_GRP2_EnableClockSleep (uint32_t Periphs)

Function description

Enable APB1 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_CRS
  - LL_APB1_GRP2_PERIPH_SWPMI1
  - LL_APB1_GRP2_PERIPH_OPAMP
  - LL_APB1_GRP2_PERIPH_MDIOS
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_TIM23 (*)
  - LL_APB1_GRP2_PERIPH_TIM24 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1HLPENR CRSLPEN LL_APB1_GRP2_EnableClockSleep
- APB1HLPENR SWPMILPEN LL_APB1_GRP2_EnableClockSleep
- APB1HLPENR OPAMPLPEN LL_APB1_GRP2_EnableClockSleep
- APB1HLPENR MDIOSLPEN LL_APB1_GRP2_EnableClockSleep
- APB1HLPENR FDCANLPEN LL_APB1_GRP2_EnableClockSleep

LL_APB1_GRP2_DisableClockSleep

Function name

__STATIC_INLINE void LL_APB1_GRP2_DisableClockSleep (uint32_t Periphs)

Function description

Disable APB1 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_CRS
  - LL_APB1_GRP2_PERIPH_SWPMI1
  - LL_APB1_GRP2_PERIPH_OPAMP
  - LL_APB1_GRP2_PERIPH_MDIOS
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_TIM23 (*)
  - LL_APB1_GRP2_PERIPH_TIM24 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1HLPENR CRSLPEN LL_APB1_GRP2_DisableClockSleep
- APB1HLPENR SWPMILPEN LL_APB1_GRP2_DisableClockSleep
- APB1HLPENR OPAMPLPEN LL_APB1_GRP2_DisableClockSleep
- APB1HLPENR MDIOSLPEN LL_APB1_GRP2_DisableClockSleep
- APB1HLPENR FDCANLPEN LL_APB1_GRP2_DisableClockSleep

LL_APB2_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_APB2_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable APB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_USART6
  - LL_APB2_GRP1_PERIPH_UART9 (*)
  - LL_APB2_GRP1_PERIPH_USART10 (*)
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_SPI4
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_SPI5
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_SAI2 (*)
  - LL_APB2_GRP1_PERIPH_SAI3 (*)
  - LL_APB2_GRP1_PERIPH_DFSDM1
  - LL_APB2_GRP1_PERIPH_HRTIM (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2ENR TIM1EN LL_APB2_GRP1_EnableClock
- APB2ENR TIM8EN LL_APB2_GRP1_EnableClock
- APB2ENR USART1EN LL_APB2_GRP1_EnableClock
- APB2ENR USART6EN LL_APB2_GRP1_EnableClock
- APB2ENR UART9EN LL_APB2_GRP1_EnableClock
- (*) APB2ENR USART10EN LL_APB2_GRP1_EnableClock
- (*) APB2ENR SPI1EN LL_APB2_GRP1_EnableClock
- APB2ENR SPI4EN LL_APB2_GRP1_EnableClock
- APB2ENR TIM15EN LL_APB2_GRP1_EnableClock
- APB2ENR TIM16EN LL_APB2_GRP1_EnableClock
- APB2ENR TIM17EN LL_APB2_GRP1_EnableClock
- APB2ENR SPI5EN LL_APB2_GRP1_EnableClock
- APB2ENR SAI1EN LL_APB2_GRP1_EnableClock
- APB2ENR SAI2EN LL_APB2_GRP1_EnableClock
- APB2ENR SAI3EN LL_APB2_GRP1_EnableClock
- (*) APB2ENR DFSDM1EN LL_APB2_GRP1_EnableClock
- APB2ENR HRTIMEN LL_APB2_GRP1_EnableClock (*)

LL_APB2_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_APB2_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if APB2 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_USART6
  - LL_APB2_GRP1_PERIPH_UART9 (*)
  - LL_APB2_GRP1_PERIPH_USART10 (*)
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_SPI4
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_SPI5
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_SAI2 (*)
  - LL_APB2_GRP1_PERIPH_SAI3 (*)
  - LL_APB2_GRP1_PERIPH_DFSDM1
  - LL_APB2_GRP1_PERIPH_HRTIM (*)

  (*) value not defined in all devices.

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- APB2ENR TIM1EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM8EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR USART1EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR USART6EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR UART9EN LL_APB2_GRP1_IsEnabledClock
- (*) APB2ENR USART10EN LL_APB2_GRP1_IsEnabledClock
- (*) APB2ENR SPI1EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR SPI4EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM15EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM16EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM17EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR SPI5EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR SAI1EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR SAI2EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR SAI3EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR DFSDM1EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR HRTIMEN LL_APB2_GRP1_IsEnabledClock

LL_APB2_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_APB2_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable APB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_USART6
  - LL_APB2_GRP1_PERIPH_UART9 (*)
  - LL_APB2_GRP1_PERIPH_USART10 (*)
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_SPI4
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_SPI5
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_SAI2 (*)
  - LL_APB2_GRP1_PERIPH_SAI3 (*)
  - LL_APB2_GRP1_PERIPH_DFSDM1
  - LL_APB2_GRP1_PERIPH_HRTIM (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2ENR TIM1EN LL_APB2_GRP1_DisableClock
- APB2ENR TIM8EN LL_APB2_GRP1_DisableClock
- APB2ENR USART1EN LL_APB2_GRP1_DisableClock
- APB2ENR USART6EN LL_APB2_GRP1_DisableClock
- APB2ENR UART9EN LL_APB2_GRP1_DisableClock
- (*) APB2ENR USART10EN LL_APB2_GRP1_DisableClock
- (*) APB2ENR SPI1EN LL_APB2_GRP1_DisableClock
- APB2ENR SPI4EN LL_APB2_GRP1_DisableClock
- APB2ENR TIM15EN LL_APB2_GRP1_DisableClock
- APB2ENR TIM16EN LL_APB2_GRP1_DisableClock
- APB2ENR TIM17EN LL_APB2_GRP1_DisableClock
- APB2ENR SPI5EN LL_APB2_GRP1_DisableClock
- APB2ENR SAI1EN LL_APB2_GRP1_DisableClock
- APB2ENR SAI2EN LL_APB2_GRP1_DisableClock
- APB2ENR SAI3EN LL_APB2_GRP1_DisableClock
- (*) APB2ENR DFSDM1EN LL_APB2_GRP1_DisableClock
- APB2ENR HRTIMEN LL_APB2_GRP1_DisableClock (*)

LL_APB2_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_APB2_GRP1_ForceReset (uint32_t Periphs)

Function description

Force APB2 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_USART6
  - LL_APB2_GRP1_PERIPH_UART9 (*)
  - LL_APB2_GRP1_PERIPH_USART10 (*)
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_SPI4
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_SPI5
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_SAI2 (*)
  - LL_APB2_GRP1_PERIPH_SAI3 (*)
  - LL_APB2_GRP1_PERIPH_DFSDM1
  - LL_APB2_GRP1_PERIPH_HRTIM (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2RSTR TIM1RST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM8RST LL_APB2_GRP1_ForceReset
- APB2RSTR USART1RST LL_APB2_GRP1_ForceReset
- APB2RSTR USART6RST LL_APB2_GRP1_ForceReset
- APB2ENR UART9RST LL_APB2_GRP1_ForceReset
- (*) APB2ENR USART10RST LL_APB2_GRP1_ForceReset
- (*) APB2RSTR SPI1RST LL_APB2_GRP1_ForceReset
- APB2RSTR SPI4RST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM15RST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM16RST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM17RST LL_APB2_GRP1_ForceReset
- APB2RSTR SPI5RST LL_APB2_GRP1_ForceReset
- APB2RSTR SAI1RST LL_APB2_GRP1_ForceReset
- APB2RSTR SAI2RST LL_APB2_GRP1_ForceReset
- APB2RSTR SAI3RST LL_APB2_GRP1_ForceReset
- (*) APB2RSTR DFSDM1RST LL_APB2_GRP1_ForceReset
- APB2RSTR HRTIMRST LL_APB2_GRP1_ForceReset (*)

LL_APB2_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_APB2_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release APB2 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_USART6
  - LL_APB2_GRP1_PERIPH_UART9 (*)
  - LL_APB2_GRP1_PERIPH_USART10 (*)
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_SPI4
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_SPI5
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_SAI2 (*)
  - LL_APB2_GRP1_PERIPH_SAI3 (*)
  - LL_APB2_GRP1_PERIPH_DFSDM1
  - LL_APB2_GRP1_PERIPH_HRTIM (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2RSTR TIM1RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR TIM8RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR USART1RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR USART6RST LL_APB2_GRP1_ReleaseReset
- APB2ENR UART9RST LL_APB2_GRP1_ReleaseReset
- (*) APB2ENR USART10RST LL_APB2_GRP1_ReleaseReset
- (*) APB2RSTR SPI1RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR SPI4RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR TIM15RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR TIM16RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR TIM17RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR SPI5RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR SAI1RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR SAI2RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR SAI3RST LL_APB2_GRP1_ReleaseReset
- (*) APB2RSTR DFSDM1RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR HRTIMRST LL_APB2_GRP1_ReleaseReset (*)

LL_APB2_GRP1_EnableClockSleep

Function name

__STATIC_INLINE void LL_APB2_GRP1_EnableClockSleep (uint32_t Periphs)

Function description

Enable APB2 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_USART6
  - LL_APB2_GRP1_PERIPH_UART9 (*)
  - LL_APB2_GRP1_PERIPH_USART10 (*)
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_SPI4
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_SPI5
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_SAI2 (*)
  - LL_APB2_GRP1_PERIPH_SAI3 (*)
  - LL_APB2_GRP1_PERIPH_DFSDM1
  - LL_APB2_GRP1_PERIPH_HRTIM (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2LPENR TIM1LPEN LL_APB2_GRP1_EnableClockSleep
- APB2LPENR TIM8LPEN LL_APB2_GRP1_EnableClockSleep
- APB2LPENR USART1LPEN LL_APB2_GRP1_EnableClockSleep
- APB2LPENR USART6LPEN LL_APB2_GRP1_EnableClockSleep
- APB2ENR UART9LPEN LL_APB2_GRP1_EnableClockSleep
- (*) APB2ENR USART10LPEN LL_APB2_GRP1_EnableClockSleep
- (*) APB2LPENR SPI1LPEN LL_APB2_GRP1_EnableClockSleep
- APB2LPENR SPI4LPEN LL_APB2_GRP1_EnableClockSleep
- APB2LPENR TIM15LPEN LL_APB2_GRP1_EnableClockSleep
- APB2LPENR TIM16LPEN LL_APB2_GRP1_EnableClockSleep
- APB2LPENR TIM17LPEN LL_APB2_GRP1_EnableClockSleep
- APB2LPENR SPI5LPEN LL_APB2_GRP1_EnableClockSleep
- APB2LPENR SAI1LPEN LL_APB2_GRP1_EnableClockSleep
- APB2LPENR SAI2LPEN LL_APB2_GRP1_EnableClockSleep
- APB2LPENR SAI3LPEN LL_APB2_GRP1_EnableClockSleep
- (*) APB2LPENR DFSDM1LPEN LL_APB2_GRP1_EnableClockSleep
- APB2LPENR HRTIMLPEN LL_APB2_GRP1_EnableClockSleep (*)

LL_APB2_GRP1_DisableClockSleep

Function name

__STATIC_INLINE void LL_APB2_GRP1_DisableClockSleep (uint32_t Periphs)

Function description

Disable APB2 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_USART6
  - LL_APB2_GRP1_PERIPH_UART9 (*)
  - LL_APB2_GRP1_PERIPH_USART10 (*)
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_SPI4
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_SPI5
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_SAI2 (*)
  - LL_APB2_GRP1_PERIPH_SAI3 (*)
  - LL_APB2_GRP1_PERIPH_DFSDM1
  - LL_APB2_GRP1_PERIPH_HRTIM (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2LPENR TIM1LPEN LL_APB2_GRP1_DisableClockSleep
- APB2LPENR TIM8LPEN LL_APB2_GRP1_DisableClockSleep
- APB2LPENR USART1LPEN LL_APB2_GRP1_DisableClockSleep
- APB2LPENR USART6LPEN LL_APB2_GRP1_DisableClockSleep
- APB2ENR UART9LPEN LL_APB2_GRP1_DisableClockSleep
- (*) APB2ENR USART10LPEN LL_APB2_GRP1_DisableClockSleep
- (*) APB2LPENR SPI1LPEN LL_APB2_GRP1_DisableClockSleep
- APB2LPENR SPI4LPEN LL_APB2_GRP1_DisableClockSleep
- APB2LPENR TIM15LPEN LL_APB2_GRP1_DisableClockSleep
- APB2LPENR TIM16LPEN LL_APB2_GRP1_DisableClockSleep
- APB2LPENR TIM17LPEN LL_APB2_GRP1_DisableClockSleep
- APB2LPENR SPI5LPEN LL_APB2_GRP1_DisableClockSleep
- APB2LPENR SAI1LPEN LL_APB2_GRP1_DisableClockSleep
- APB2LPENR SAI2LPEN LL_APB2_GRP1_DisableClockSleep
- APB2LPENR SAI3LPEN LL_APB2_GRP1_DisableClockSleep
- (*) APB2LPENR DFSDM1LPEN LL_APB2_GRP1_DisableClockSleep
- APB2LPENR HRTIMLPEN LL_APB2_GRP1_DisableClockSleep (*)

LL_APB4_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_APB4_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable APB4 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP1_PERIPH_SYSCFG
  - LL_APB4_GRP1_PERIPH_LPUART1
  - LL_APB4_GRP1_PERIPH_SPI6
  - LL_APB4_GRP1_PERIPH_I2C4
  - LL_APB4_GRP1_PERIPH_LPTIM2
  - LL_APB4_GRP1_PERIPH_LPTIM3
  - LL_APB4_GRP1_PERIPH_LPTIM4 (*)
  - LL_APB4_GRP1_PERIPH_LPTIM5 (*)
  - LL_APB4_GRP1_PERIPH_DAC2 (*)
  - LL_APB4_GRP1_PERIPH_COMP12
  - LL_APB4_GRP1_PERIPH_VREF
  - LL_APB4_GRP1_PERIPH_RTCAPB
  - LL_APB4_GRP1_PERIPH_SAI4 (*)
  - LL_APB4_GRP1_PERIPH_DTS (*)
  - LL_APB4_GRP1_PERIPH_DFSDM2 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB4ENR SYSCFGEN LL_APB4_GRP1_EnableClock
- APB4ENR LPUART1EN LL_APB4_GRP1_EnableClock
- APB4ENR SPI6EN LL_APB4_GRP1_EnableClock
- APB4ENR I2C4EN LL_APB4_GRP1_EnableClock
- APB4ENR LPTIM2EN LL_APB4_GRP1_EnableClock
- APB4ENR LPTIM3EN LL_APB4_GRP1_EnableClock
- APB4ENR LPTIM4EN LL_APB4_GRP1_EnableClock
- (*) APB4ENR LPTIM5EN LL_APB4_GRP1_EnableClock
- (*) APB4ENR DAC2EN LL_APB4_GRP1_EnableClock
- (*) APB4ENR COMP12EN LL_APB4_GRP1_EnableClock
- APB4ENR VREFEN LL_APB4_GRP1_EnableClock
- APB4ENR RTCAPBEN LL_APB4_GRP1_EnableClock
- APB4ENR SAI4EN LL_APB4_GRP1_EnableClock
- (*) APB4ENR DTSEN LL_APB4_GRP1_EnableClock
- (*) APB4ENR DFSDM2EN LL_APB4_GRP1_EnableClock (*)

LL_APB4_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_APB4_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if APB4 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP1_PERIPH_SYSCFG
  - LL_APB4_GRP1_PERIPH_LPUART1
  - LL_APB4_GRP1_PERIPH_SPI6
  - LL_APB4_GRP1_PERIPH_I2C4
  - LL_APB4_GRP1_PERIPH_LPTIM2
  - LL_APB4_GRP1_PERIPH_LPTIM3
  - LL_APB4_GRP1_PERIPH_LPTIM4 (*)
  - LL_APB4_GRP1_PERIPH_LPTIM5 (*)
  - LL_APB4_GRP1_PERIPH_DAC2 (*)
  - LL_APB4_GRP1_PERIPH_COMP12
  - LL_APB4_GRP1_PERIPH_VREF
  - LL_APB4_GRP1_PERIPH_RTCAPB
  - LL_APB4_GRP1_PERIPH_SAI4 (*)
  - LL_APB4_GRP1_PERIPH_DTS (*)
  - LL_APB4_GRP1_PERIPH_DFSDM2 (*)

  (*) value not defined in all devices.

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- APB4ENR SYSCFGEN LL_APB4_GRP1_IsEnabledClock
- APB4ENR LPUART1EN LL_APB4_GRP1_IsEnabledClock
- APB4ENR SPI6EN LL_APB4_GRP1_IsEnabledClock
- APB4ENR I2C4EN LL_APB4_GRP1_IsEnabledClock
- APB4ENR LPTIM2EN LL_APB4_GRP1_IsEnabledClock
- APB4ENR LPTIM3EN LL_APB4_GRP1_IsEnabledClock
- APB4ENR LPTIM4EN LL_APB4_GRP1_IsEnabledClock
- (*) APB4ENR LPTIM5EN LL_APB4_GRP1_IsEnabledClock
- (*) APB4ENR DAC2EN LL_APB4_GRP1_IsEnabledClock
- (*) APB4ENR COMP12EN LL_APB4_GRP1_IsEnabledClock
- APB4ENR VREFEN LL_APB4_GRP1_IsEnabledClock
- APB4ENR RTCAPBEN LL_APB4_GRP1_IsEnabledClock
- APB4ENR SAI4EN LL_APB4_GRP1_IsEnabledClock
- (*) APB4ENR DTSEN LL_APB4_GRP1_IsEnabledClock
- (*) APB4ENR DFSDM2EN LL_APB4_GRP1_IsEnabledClock (*)

LL_APB4_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_APB4_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable APB4 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP1_PERIPH_SYSCFG
  - LL_APB4_GRP1_PERIPH_LPUART1
  - LL_APB4_GRP1_PERIPH_SPI6
  - LL_APB4_GRP1_PERIPH_I2C4
  - LL_APB4_GRP1_PERIPH_LPTIM2
  - LL_APB4_GRP1_PERIPH_LPTIM3
  - LL_APB4_GRP1_PERIPH_LPTIM4 (*)
  - LL_APB4_GRP1_PERIPH_LPTIM5 (*)
  - LL_APB4_GRP1_PERIPH_DAC2 (*)
  - LL_APB4_GRP1_PERIPH_COMP12
  - LL_APB4_GRP1_PERIPH_VREF
  - LL_APB4_GRP1_PERIPH_RTCAPB
  - LL_APB4_GRP1_PERIPH_SAI4 (*)
  - LL_APB4_GRP1_PERIPH_DTS (*)
  - LL_APB4_GRP1_PERIPH_DFSDM2 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB4ENR SYSCFGEN LL_APB4_GRP1_DisableClock
- APB4ENR LPUART1EN LL_APB4_GRP1_DisableClock
- APB4ENR SPI6EN LL_APB4_GRP1_DisableClock
- APB4ENR I2C4EN LL_APB4_GRP1_DisableClock
- APB4ENR LPTIM2EN LL_APB4_GRP1_DisableClock
- APB4ENR LPTIM3EN LL_APB4_GRP1_DisableClock
- APB4ENR LPTIM4EN LL_APB4_GRP1_DisableClock
- (*) APB4ENR LPTIM5EN LL_APB4_GRP1_DisableClock
- (*) APB4ENR DAC2EN LL_APB4_GRP1_DisableClock
- (*) APB4ENR COMP12EN LL_APB4_GRP1_DisableClock
- APB4ENR VREFEN LL_APB4_GRP1_DisableClock
- APB4ENR RTCAPBEN LL_APB4_GRP1_DisableClock
- APB4ENR SAI4EN LL_APB4_GRP1_DisableClock
- (*) APB4ENR DTSEN LL_APB4_GRP1_DisableClock
- (*) APB4ENR DFSDM2EN LL_APB4_GRP1_DisableClock (*)

LL_APB4_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_APB4_GRP1_ForceReset (uint32_t Periphs)

Function description

Force APB4 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP1_PERIPH_SYSCFG
  - LL_APB4_GRP1_PERIPH_LPUART1
  - LL_APB4_GRP1_PERIPH_SPI6
  - LL_APB4_GRP1_PERIPH_I2C4
  - LL_APB4_GRP1_PERIPH_LPTIM2
  - LL_APB4_GRP1_PERIPH_LPTIM3
  - LL_APB4_GRP1_PERIPH_LPTIM4 (*)
  - LL_APB4_GRP1_PERIPH_LPTIM5 (*)
  - LL_APB4_GRP1_PERIPH_DAC2 (*)
  - LL_APB4_GRP1_PERIPH_COMP12
  - LL_APB4_GRP1_PERIPH_VREF
  - LL_APB4_GRP1_PERIPH_SAI4 (*)
  - LL_APB4_GRP1_PERIPH_DTS (*)
  - LL_APB4_GRP1_PERIPH_DFSDM2 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB4RSTR SYSCFGRST LL_APB4_GRP1_ForceReset
- APB4RSTR LPUART1RST LL_APB4_GRP1_ForceReset
- APB4RSTR SPI6RST LL_APB4_GRP1_ForceReset
- APB4RSTR I2C4RST LL_APB4_GRP1_ForceReset
- APB4RSTR LPTIM2RST LL_APB4_GRP1_ForceReset
- APB4RSTR LPTIM3RST LL_APB4_GRP1_ForceReset
- APB4RSTR LPTIM4RST LL_APB4_GRP1_ForceReset
- (*) APB4RSTR LPTIM5RST LL_APB4_GRP1_ForceReset
- (*) APB4RSTR DAC2EN LL_APB4_GRP1_ForceReset
- (*) APB4RSTR COMP12RST LL_APB4_GRP1_ForceReset
- APB4RSTR VREFRST LL_APB4_GRP1_ForceReset
- APB4RSTR SAI4RST LL_APB4_GRP1_ForceReset
- (*) APB4RSTR DTSRST LL_APB4_GRP1_ForceReset
- (*) APB4RSTR DFSDM2RST LL_APB4_GRP1_ForceReset (*)

LL_APB4_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_APB4_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release APB4 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP1_PERIPH_SYSCFG
  - LL_APB4_GRP1_PERIPH_LPUART1
  - LL_APB4_GRP1_PERIPH_SPI6
  - LL_APB4_GRP1_PERIPH_I2C4
  - LL_APB4_GRP1_PERIPH_LPTIM2
  - LL_APB4_GRP1_PERIPH_LPTIM3
  - LL_APB4_GRP1_PERIPH_LPTIM4 (*)
  - LL_APB4_GRP1_PERIPH_LPTIM5 (*)
  - LL_APB4_GRP1_PERIPH_DAC2 (*)
  - LL_APB4_GRP1_PERIPH_COMP12
  - LL_APB4_GRP1_PERIPH_VREF
  - LL_APB4_GRP1_PERIPH_SAI4 (*)
  - LL_APB4_GRP1_PERIPH_DTS (*)
  - LL_APB4_GRP1_PERIPH_DFSDM2 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB4RSTR SYSCFGRST LL_APB4_GRP1_ReleaseReset
- APB4RSTR LPUART1RST LL_APB4_GRP1_ReleaseReset
- APB4RSTR SPI6RST LL_APB4_GRP1_ReleaseReset
- APB4RSTR I2C4RST LL_APB4_GRP1_ReleaseReset
- APB4RSTR LPTIM2RST LL_APB4_GRP1_ReleaseReset
- APB4RSTR LPTIM3RST LL_APB4_GRP1_ReleaseReset
- APB4RSTR LPTIM4RST LL_APB4_GRP1_ReleaseReset
- (*) APB4RSTR LPTIM5RST LL_APB4_GRP1_ReleaseReset
- (*) APB4RSTR DAC2RST LL_APB4_GRP1_ReleaseReset
- (*) APB4RSTR COMP12RST LL_APB4_GRP1_ReleaseReset
- APB4RSTR VREFRST LL_APB4_GRP1_ReleaseReset
- APB4RSTR SAI4RST LL_APB4_GRP1_ReleaseReset
- APB4RSTR DTSRST LL_APB4_GRP1_ReleaseReset
- (*) APB4RSTR DFSDM2RST LL_APB4_GRP1_ReleaseReset (*)

LL_APB4_GRP1_EnableClockSleep

Function name

__STATIC_INLINE void LL_APB4_GRP1_EnableClockSleep (uint32_t Periphs)

Function description

Enable APB4 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP1_PERIPH_SYSCFG
  - LL_APB4_GRP1_PERIPH_LPUART1
  - LL_APB4_GRP1_PERIPH_SPI6
  - LL_APB4_GRP1_PERIPH_I2C4
  - LL_APB4_GRP1_PERIPH_LPTIM2
  - LL_APB4_GRP1_PERIPH_LPTIM3
  - LL_APB4_GRP1_PERIPH_LPTIM4 (*)
  - LL_APB4_GRP1_PERIPH_LPTIM5 (*)
  - LL_APB4_GRP1_PERIPH_DAC2 (*)
  - LL_APB4_GRP1_PERIPH_COMP12
  - LL_APB4_GRP1_PERIPH_VREF
  - LL_APB4_GRP1_PERIPH_RTCAPB
  - LL_APB4_GRP1_PERIPH_SAI4 (*)
  - LL_APB4_GRP1_PERIPH_DTS (*)
  - LL_APB4_GRP1_PERIPH_DFSDM2 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB4LPENR SYSCFGLPEN LL_APB4_GRP1_EnableClockSleep
- APB4LPENR LPUART1LPEN LL_APB4_GRP1_EnableClockSleep
- APB4LPENR SPI6LPEN LL_APB4_GRP1_EnableClockSleep
- APB4LPENR I2C4LPEN LL_APB4_GRP1_EnableClockSleep
- APB4LPENR LPTIM2LPEN LL_APB4_GRP1_EnableClockSleep
- APB4LPENR LPTIM3LPEN LL_APB4_GRP1_EnableClockSleep
- APB4LPENR LPTIM4LPEN LL_APB4_GRP1_EnableClockSleep
- (*) APB4LPENR LPTIM5LPEN LL_APB4_GRP1_EnableClockSleep
- (*) APB4LPENR DAC2LPEN LL_APB4_GRP1_EnableClockSleep
- (*) APB4LPENR COMP12LPEN LL_APB4_GRP1_EnableClockSleep
- APB4LPENR VREFLPEN LL_APB4_GRP1_EnableClockSleep
- APB4LPENR RTCAPBLPEN LL_APB4_GRP1_EnableClockSleep
- APB4LPENR SAI4LPEN LL_APB4_GRP1_EnableClockSleep
- (*) APB4LPENR DTSLPEN LL_APB4_GRP1_EnableClockSleep
- (*) APB4LPENR DFSDM2LPEN LL_APB4_GRP1_EnableClockSleep (*)

LL_APB4_GRP1_DisableClockSleep

Function name

__STATIC_INLINE void LL_APB4_GRP1_DisableClockSleep (uint32_t Periphs)

Function description

Disable APB4 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP1_PERIPH_SYSCFG
  - LL_APB4_GRP1_PERIPH_LPUART1
  - LL_APB4_GRP1_PERIPH_SPI6
  - LL_APB4_GRP1_PERIPH_I2C4
  - LL_APB4_GRP1_PERIPH_LPTIM2
  - LL_APB4_GRP1_PERIPH_LPTIM3
  - LL_APB4_GRP1_PERIPH_LPTIM4 (*)
  - LL_APB4_GRP1_PERIPH_LPTIM5 (*)
  - LL_APB4_GRP1_PERIPH_DAC2 (*)
  - LL_APB4_GRP1_PERIPH_COMP12
  - LL_APB4_GRP1_PERIPH_VREF
  - LL_APB4_GRP1_PERIPH_RTCAPB
  - LL_APB4_GRP1_PERIPH_SAI4 (*)
  - LL_APB4_GRP1_PERIPH_DTS (*)
  - LL_APB4_GRP1_PERIPH_DFSDM2 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB4LPENR SYSCFGLPEN LL_APB4_GRP1_DisableClockSleep
- APB4LPENR LPUART1LPEN LL_APB4_GRP1_DisableClockSleep
- APB4LPENR SPI6LPEN LL_APB4_GRP1_DisableClockSleep
- APB4LPENR I2C4LPEN LL_APB4_GRP1_DisableClockSleep
- APB4LPENR LPTIM2LPEN LL_APB4_GRP1_DisableClockSleep
- APB4LPENR LPTIM3LPEN LL_APB4_GRP1_DisableClockSleep
- APB4LPENR LPTIM4LPEN LL_APB4_GRP1_DisableClockSleep
- (*) APB4LPENR LPTIM5LPEN LL_APB4_GRP1_DisableClockSleep
- (*) APB4LPENR DAC2LPEN LL_APB4_GRP1_DisableClockSleep
- (*) APB4LPENR COMP12LPEN LL_APB4_GRP1_DisableClockSleep
- APB4LPENR VREFLPEN LL_APB4_GRP1_DisableClockSleep
- APB4LPENR RTCAPBLPEN LL_APB4_GRP1_DisableClockSleep
- APB4LPENR SAI4LPEN LL_APB4_GRP1_DisableClockSleep
- (*) APB4LPENR DTSLPEN LL_APB4_GRP1_DisableClockSleep
- (*) APB4LPENR DFSDM2LPEN LL_APB4_GRP1_DisableClockSleep (*)

LL_CLKAM_Enable

Function name

__STATIC_INLINE void LL_CLKAM_Enable (uint32_t Periphs)

Function description

Enable peripherals clock for CLKAM Mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_CLKAM_PERIPH_BDMA
  - LL_CLKAM_PERIPH_GPIO (*)
  - LL_CLKAM_PERIPH_LPUART1
  - LL_CLKAM_PERIPH_SPI6
  - LL_CLKAM_PERIPH_I2C4
  - LL_CLKAM_PERIPH_LPTIM2
  - LL_CLKAM_PERIPH_LPTIM3
  - LL_CLKAM_PERIPH_LPTIM4 (*)
  - LL_CLKAM_PERIPH_LPTIM5 (*)
  - LL_CLKAM_PERIPH_DAC2 (*)
  - LL_CLKAM_PERIPH_COMP12
  - LL_CLKAM_PERIPH_VREF
  - LL_CLKAM_PERIPH_RTC
  - LL_CLKAM_PERIPH_CRC (*)
  - LL_CLKAM_PERIPH_SAI4 (*)
  - LL_CLKAM_PERIPH_ADC3 (*)
  - LL_CLKAM_PERIPH_DTS (*)
  - LL_CLKAM_PERIPH_DFSDM2 (*)
  - LL_CLKAM_PERIPH_BKPRAM
  - LL_CLKAM_PERIPH_SRAM4

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- D3AMR / SRDAMR BDMA LL_CLKAM_Enable
- D3AMR / SRDAMR LPUART1 LL_CLKAM_Enable
- D3AMR / SRDAMR SPI6 LL_CLKAM_Enable
- D3AMR / SRDAMR I2C4 LL_CLKAM_Enable
- D3AMR / SRDAMR LPTIM2 LL_CLKAM_Enable
- D3AMR / SRDAMR LPTIM3 LL_CLKAM_Enable
- D3AMR / SRDAMR LPTIM4 LL_CLKAM_Enable
- (*) D3AMR / SRDAMR LPTIM5 LL_CLKAM_Enable
- (*) D3AMR / SRDAMR DAC2 LL_CLKAM_Enable
- (*) D3AMR / SRDAMR COMP12 LL_CLKAM_Enable
- D3AMR / SRDAMR VREF LL_CLKAM_Enable
- D3AMR / SRDAMR RTC LL_CLKAM_Enable
- D3AMR / SRDAMR CRC LL_CLKAM_Enable
- D3AMR / SRDAMR SAI4 LL_CLKAM_Enable
- (*) D3AMR / SRDAMR ADC3 LL_CLKAM_Enable
- (*) D3AMR / SRDAMR DTS LL_CLKAM_Enable
- (*) D3AMR / SRDAMR DFSDM2 LL_CLKAM_Enable
- (*) D3AMR / SRDAMR BKPRAM LL_CLKAM_Enable
- D3AMR / SRDAMR SRAM4 LL_CLKAM_Enable

LL_CLKAM_Disable

Function name

__STATIC_INLINE void LL_CLKAM_Disable (uint32_t Periphs)

Function description

Disable peripherals clock for CLKAM Mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_CLKAM_PERIPH_BDMA
  - LL_CLKAM_PERIPH_GPIO (*)
  - LL_CLKAM_PERIPH_LPUART1
  - LL_CLKAM_PERIPH_SPI6
  - LL_CLKAM_PERIPH_I2C4
  - LL_CLKAM_PERIPH_LPTIM2
  - LL_CLKAM_PERIPH_LPTIM3
  - LL_CLKAM_PERIPH_LPTIM4 (*)
  - LL_CLKAM_PERIPH_LPTIM5 (*)
  - LL_CLKAM_PERIPH_DAC2 (*)
  - LL_CLKAM_PERIPH_COMP12
  - LL_CLKAM_PERIPH_VREF
  - LL_CLKAM_PERIPH_RTC
  - LL_CLKAM_PERIPH_CRC (*)
  - LL_CLKAM_PERIPH_SAI4 (*)
  - LL_CLKAM_PERIPH_ADC3 (*)
  - LL_CLKAM_PERIPH_DTS (*)
  - LL_CLKAM_PERIPH_DFSDM2 (*)
  - LL_CLKAM_PERIPH_BKPRAM
  - LL_CLKAM_PERIPH_SRAM4

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- D3AMR / SRDAMR BDMA LL_CLKAM_Disable
- D3AMR / SRDAMR LPUART1 LL_CLKAM_Disable
- D3AMR / SRDAMR SPI6 LL_CLKAM_Disable
- D3AMR / SRDAMR I2C4 LL_CLKAM_Disable
- D3AMR / SRDAMR LPTIM2 LL_CLKAM_Disable
- D3AMR / SRDAMR LPTIM3 LL_CLKAM_Disable
- D3AMR / SRDAMR LPTIM4 LL_CLKAM_Disable
- (*) D3AMR / SRDAMR LPTIM5 LL_CLKAM_Disable
- (*) D3AMR / SRDAMR DAC2 LL_CLKAM_Disable
- (*) D3AMR / SRDAMR COMP12 LL_CLKAM_Disable
- D3AMR / SRDAMR VREF LL_CLKAM_Disable
- D3AMR / SRDAMR RTC LL_CLKAM_Disable
- D3AMR / SRDAMR CRC LL_CLKAM_Disable
- D3AMR / SRDAMR SAI4 LL_CLKAM_Disable
- (*) D3AMR / SRDAMR ADC3 LL_CLKAM_Disable
- (*) D3AMR / SRDAMR DTS LL_CLKAM_Disable
- (*) D3AMR / SRDAMR DFSDM2 LL_CLKAM_Disable
- (*) D3AMR / SRDAMR BKPRAM LL_CLKAM_Disable
- D3AMR / SRDAMR SRAM4 LL_CLKAM_Disable

LL_C1_AHB3_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_C1_AHB3_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable C1 AHB3 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_MDMA
  - LL_AHB3_GRP1_PERIPH_DMA2D
  - LL_AHB3_GRP1_PERIPH_JPGDEC
  - LL_AHB3_GRP1_PERIPH_FMC
  - LL_AHB3_GRP1_PERIPH_QSPI (*)
  - LL_AHB3_GRP1_PERIPH_OSPI1 (*)
  - LL_AHB3_GRP1_PERIPH_OSPI2 (*)
  - LL_AHB3_GRP1_PERIPH_OCTOSPIM (*)
  - LL_AHB3_GRP1_PERIPH_OTFDEC1 (*)
  - LL_AHB3_GRP1_PERIPH_OTFDEC2 (*)
  - LL_AHB3_GRP1_PERIPH_GFXMMU (*)
  - LL_AHB3_GRP1_PERIPH_SDMMC1

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3ENR MDMAEN LL_C1_AHB3_GRP1_EnableClock
- AHB3ENR DMA2DEN LL_C1_AHB3_GRP1_EnableClock
- AHB3ENR JPGDECEN LL_C1_AHB3_GRP1_EnableClock
- AHB3ENR FMCEN LL_C1_AHB3_GRP1_EnableClock
- AHB3ENR QSPIEN LL_C1_AHB3_GRP1_EnableClock
- (*) AHB3ENR OSPI1EN LL_C1_AHB3_GRP1_EnableClock
- (*) AHB3ENR OSPI2EN LL_C1_AHB3_GRP1_EnableClock
- (*) AHB3ENR IOMNGREN LL_C1_AHB3_GRP1_EnableClock
- (*) AHB3ENR OTFDEC1EN LL_C1_AHB3_GRP1_EnableClock
- (*) AHB3ENR OTFDEC2EN LL_C1_AHB3_GRP1_EnableClock
- (*) AHB3ENR GFXMMUEN LL_C1_AHB3_GRP1_EnableClock
- (*) AHB3ENR SDMMC1EN LL_C1_AHB3_GRP1_EnableClock

LL_C1_AHB3_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_C1_AHB3_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if C1 AHB3 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_MDMA
  - LL_AHB3_GRP1_PERIPH_DMA2D
  - LL_AHB3_GRP1_PERIPH_JPGDEC
  - LL_AHB3_GRP1_PERIPH_FMC
  - LL_AHB3_GRP1_PERIPH_QSPI (*)
  - LL_AHB3_GRP1_PERIPH_OSPI1 (*)
  - LL_AHB3_GRP1_PERIPH_OSPI2 (*)
  - LL_AHB3_GRP1_PERIPH_OCTOSPIM (*)
  - LL_AHB3_GRP1_PERIPH_OTFDEC1 (*)
  - LL_AHB3_GRP1_PERIPH_OTFDEC2 (*)
  - LL_AHB3_GRP1_PERIPH_GFXMMU (*)
  - LL_AHB3_GRP1_PERIPH_SDMMC1

  (*) value not defined in all devices.

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- AHB3ENR MDMAEN LL_C1_AHB3_GRP1_IsEnabledClock
- AHB3ENR DMA2DEN LL_C1_AHB3_GRP1_IsEnabledClock
- AHB3ENR JPGDECEN LL_C1_AHB3_GRP1_IsEnabledClock
- AHB3ENR FMCEN LL_C1_AHB3_GRP1_IsEnabledClock
- AHB3ENR QSPIEN LL_C1_AHB3_GRP1_IsEnabledClock
- (*) AHB3ENR OSPI1EN LL_C1_AHB3_GRP1_IsEnabledClock
- (*) AHB3ENR OSPI2EN LL_C1_AHB3_GRP1_IsEnabledClock
- (*) AHB3ENR IOMNGREN LL_C1_AHB3_GRP1_IsEnabledClock
- (*) AHB3ENR OTFDEC1EN LL_C1_AHB3_GRP1_IsEnabledClock
- (*) AHB3ENR OTFDEC2EN LL_C1_AHB3_GRP1_IsEnabledClock
- (*) AHB3ENR GFXMMUEN LL_C1_AHB3_GRP1_IsEnabledClock
- (*) AHB3ENR SDMMC1EN LL_C1_AHB3_GRP1_IsEnabledClock

LL_C1_AHB3_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_C1_AHB3_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable C1 AHB3 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_MDMA
  - LL_AHB3_GRP1_PERIPH_DMA2D
  - LL_AHB3_GRP1_PERIPH_JPGDEC (*)
  - LL_AHB3_GRP1_PERIPH_FMC
  - LL_AHB3_GRP1_PERIPH_QSPI (*)
  - LL_AHB3_GRP1_PERIPH_OSPI1 (*)
  - LL_AHB3_GRP1_PERIPH_OSPI2 (*)
  - LL_AHB3_GRP1_PERIPH_OCTOSPIM (*)
  - LL_AHB3_GRP1_PERIPH_OTFDEC1 (*)
  - LL_AHB3_GRP1_PERIPH_OTFDEC2 (*)
  - LL_AHB3_GRP1_PERIPH_GFXMMU (*)
  - LL_AHB3_GRP1_PERIPH_SDMMC1

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3ENR MDMAEN LL_C1_AHB3_GRP1_DisableClock
- AHB3ENR DMA2DEN LL_C1_AHB3_GRP1_DisableClock
- AHB3ENR JPGDECEN LL_C1_AHB3_GRP1_DisableClock
- AHB3ENR FMCEN LL_C1_AHB3_GRP1_DisableClock
- AHB3ENR QSPIEN LL_C1_AHB3_GRP1_DisableClock
- (*) AHB3ENR OSPI1EN LL_C1_AHB3_GRP1_DisableClock
- (*) AHB3ENR OSPI2EN LL_C1_AHB3_GRP1_DisableClock
- (*) AHB3ENR IOMNGREN LL_C1_AHB3_GRP1_DisableClock
- (*) AHB3ENR OTFDEC1EN LL_C1_AHB3_GRP1_DisableClock
- (*) AHB3ENR OTFDEC2EN LL_C1_AHB3_GRP1_DisableClock
- (*) AHB3ENR GFXMMUEN LL_C1_AHB3_GRP1_DisableClock
- (*) AHB3ENR SDMMC1EN LL_C1_AHB3_GRP1_DisableClock

LL_C1_AHB3_GRP1_EnableClockSleep

Function name

__STATIC_INLINE void LL_C1_AHB3_GRP1_EnableClockSleep (uint32_t Periphs)

Function description

Enable C1 AHB3 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_DMA2D
  - LL_AHB3_GRP1_PERIPH_JPGDEC (*)
  - LL_AHB3_GRP1_PERIPH_FMC
  - LL_AHB3_GRP1_PERIPH_QSPI (*)
  - LL_AHB3_GRP1_PERIPH_OSPI1 (*)
  - LL_AHB3_GRP1_PERIPH_OSPI2 (*)
  - LL_AHB3_GRP1_PERIPH_OCTOSPIM (*)
  - LL_AHB3_GRP1_PERIPH_OTFDEC1 (*)
  - LL_AHB3_GRP1_PERIPH_OTFDEC2 (*)
  - LL_AHB3_GRP1_PERIPH_GFXMMU (*)
  - LL_AHB3_GRP1_PERIPH_SDMMC1
  - LL_AHB3_GRP1_PERIPH_FLASH
  - LL_AHB3_GRP1_PERIPH_DTCM1
  - LL_AHB3_GRP1_PERIPH_DTCM2
  - LL_AHB3_GRP1_PERIPH_ITCM
  - LL_AHB3_GRP1_PERIPH_AXISRAM

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3LPENR MDMALPEN LL_C1_AHB3_GRP1_EnableClockSleep
- AHB3LPENR DMA2DLPEN LL_C1_AHB3_GRP1_EnableClockSleep
- AHB3LPENR JPGDECLPEN LL_C1_AHB3_GRP1_EnableClockSleep
- AHB3LPENR FMCLPEN LL_C1_AHB3_GRP1_EnableClockSleep
- AHB3LPENR QSPILPEN LL_C1_AHB3_GRP1_EnableClockSleep
- (*) AHB3LPENR OSPI1LPEN LL_C1_AHB3_GRP1_EnableClockSleep
- (*) AHB3LPENR OSPI2LPEN LL_C1_AHB3_GRP1_EnableClockSleep
- (*) AHB3LPENR IOMNGRLPEN LL_C1_AHB3_GRP1_EnableClockSleep
- (*) AHB3LPENR OTFDEC1LPEN LL_C1_AHB3_GRP1_EnableClockSleep
- (*) AHB3LPENR OTFDEC1LPEN LL_C1_AHB3_GRP1_EnableClockSleep
- (*) AHB3LPENR GFXMMULPEN LL_C1_AHB3_GRP1_EnableClockSleep
- (*) AHB3LPENR SDMMC1LPEN LL_C1_AHB3_GRP1_EnableClockSleep
- AHB3LPENR FLASHLPEN LL_C1_AHB3_GRP1_EnableClockSleep
- AHB3LPENR DTCM1LPEN LL_C1_AHB3_GRP1_EnableClockSleep
- AHB3LPENR DTCM2LPEN LL_C1_AHB3_GRP1_EnableClockSleep
- AHB3LPENR ITCMLPEN LL_C1_AHB3_GRP1_EnableClockSleep
- AHB3LPENR AXISRAMLPEN LL_C1_AHB3_GRP1_EnableClockSleep

LL_C1_AHB3_GRP1_DisableClockSleep

Function name

__STATIC_INLINE void LL_C1_AHB3_GRP1_DisableClockSleep (uint32_t Periphs)

Function description

Disable C1 AHB3 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_DMA2D
  - LL_AHB3_GRP1_PERIPH_JPGDEC (*)
  - LL_AHB3_GRP1_PERIPH_FMC
  - LL_AHB3_GRP1_PERIPH_QSPI (*)
  - LL_AHB3_GRP1_PERIPH_OSPI1 (*)
  - LL_AHB3_GRP1_PERIPH_OSPI2 (*)
  - LL_AHB3_GRP1_PERIPH_OCTOSPIM (*)
  - LL_AHB3_GRP1_PERIPH_OTFDEC1 (*)
  - LL_AHB3_GRP1_PERIPH_OTFDEC2 (*)
  - LL_AHB3_GRP1_PERIPH_GFXMMU (*)
  - LL_AHB3_GRP1_PERIPH_SDMMC1
  - LL_AHB3_GRP1_PERIPH_FLASH
  - LL_AHB3_GRP1_PERIPH_DTCM1
  - LL_AHB3_GRP1_PERIPH_DTCM2
  - LL_AHB3_GRP1_PERIPH_ITCM
  - LL_AHB3_GRP1_PERIPH_AXISRAM

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3LPENR MDMALPEN LL_C1_AHB3_GRP1_DisableClockSleep
- AHB3LPENR DMA2DLPEN LL_C1_AHB3_GRP1_DisableClockSleep
- AHB3LPENR JPGDECLPEN LL_C1_AHB3_GRP1_DisableClockSleep
- AHB3LPENR FMCLPEN LL_C1_AHB3_GRP1_DisableClockSleep
- AHB3LPENR QSPILPEN LL_C1_AHB3_GRP1_DisableClockSleep
- (*) AHB3LPENR OSPI1LPEN LL_C1_AHB3_GRP1_DisableClockSleep
- (*) AHB3LPENR OSPI2LPEN LL_C1_AHB3_GRP1_DisableClockSleep
- (*) AHB3LPENR IOMNGRLPEN LL_C1_AHB3_GRP1_DisableClockSleep
- (*) AHB3LPENR OTFDEC1LPEN LL_C1_AHB3_GRP1_DisableClockSleep
- (*) AHB3LPENR OTFDEC1LPEN LL_C1_AHB3_GRP1_DisableClockSleep
- (*) AHB3LPENR GFXMMULPEN LL_C1_AHB3_GRP1_DisableClockSleep
- (*) AHB3LPENR SDMMC1LPEN LL_C1_AHB3_GRP1_DisableClockSleep
- AHB3LPENR FLASHLPEN LL_C1_AHB3_GRP1_DisableClockSleep
- AHB3LPENR DTCM1LPEN LL_C1_AHB3_GRP1_DisableClockSleep
- AHB3LPENR DTCM2LPEN LL_C1_AHB3_GRP1_DisableClockSleep
- AHB3LPENR ITCMLPEN LL_C1_AHB3_GRP1_DisableClockSleep
- AHB3LPENR AXISRAMLPEN LL_C1_AHB3_GRP1_DisableClockSleep

LL_C1_AHB1_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_C1_AHB1_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable C1 AHB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_DMA1
  - LL_AHB1_GRP1_PERIPH_DMA2
  - LL_AHB1_GRP1_PERIPH_ADC12
  - LL_AHB1_GRP1_PERIPH_CRC (*)
  - LL_AHB1_GRP1_PERIPH_ART (*)
  - LL_AHB1_GRP1_PERIPH_ETH1MAC (*)
  - LL_AHB1_GRP1_PERIPH_ETH1TX (*)
  - LL_AHB1_GRP1_PERIPH_ETH1RX (*)
  - LL_AHB1_GRP1_PERIPH_USB1OTGHS
  - LL_AHB1_GRP1_PERIPH_USB1OTGHSULPI
  - LL_AHB1_GRP1_PERIPH_USB2OTGHS (*)
  - LL_AHB1_GRP1_PERIPH_USB2OTGHSULPI (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1ENR DMA1EN LL_C1_AHB1_GRP1_EnableClock
- AHB1ENR DMA2EN LL_C1_AHB1_GRP1_EnableClock
- AHB1ENR ADC12EN LL_C1_AHB1_GRP1_EnableClock
- AHB1ENR CRCEN LL_C1_AHB1_GRP1_EnableClock
- (*) AHB1ENR ARTEN LL_C1_AHB1_GRP1_EnableClock
- (*) AHB1ENR ETH1MACEN LL_C1_AHB1_GRP1_EnableClock
- (*) AHB1ENR ETH1TXEN LL_C1_AHB1_GRP1_EnableClock
- (*) AHB1ENR ETH1RXEN LL_C1_AHB1_GRP1_EnableClock
- (*) AHB1ENR USB1OTGHSEN LL_C1_AHB1_GRP1_EnableClock
- AHB1ENR USB1OTGHSULPIEN LL_C1_AHB1_GRP1_EnableClock
- AHB1ENR USB2OTGHSEN LL_C1_AHB1_GRP1_EnableClock
- (*) AHB1ENR USB2OTGHSULPIEN LL_C1_AHB1_GRP1_EnableClock (*)

LL_C1_AHB1_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_C1_AHB1_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if C1 AHB1 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_DMA1
  - LL_AHB1_GRP1_PERIPH_DMA2
  - LL_AHB1_GRP1_PERIPH_ADC12
  - LL_AHB1_GRP1_PERIPH_CRC (*)
  - LL_AHB1_GRP1_PERIPH_ART (*)
  - LL_AHB1_GRP1_PERIPH_ETH1MAC (*)
  - LL_AHB1_GRP1_PERIPH_ETH1TX (*)
  - LL_AHB1_GRP1_PERIPH_ETH1RX (*)
  - LL_AHB1_GRP1_PERIPH_USB1OTGHS
  - LL_AHB1_GRP1_PERIPH_USB1OTGHSULPI
  - LL_AHB1_GRP1_PERIPH_USB2OTGHS (*)
  - LL_AHB1_GRP1_PERIPH_USB2OTGHSULPI (*)

  (*) value not defined in all devices.

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- AHB1ENR DMA1EN LL_C1_AHB1_GRP1_IsEnabledClock
- AHB1ENR DMA2EN LL_C1_AHB1_GRP1_IsEnabledClock
- AHB1ENR ADC12EN LL_C1_AHB1_GRP1_IsEnabledClock
- AHB1ENR CRCEN LL_C1_AHB1_GRP1_IsEnabledClock
- (*) AHB1ENR ARTEN LL_C1_AHB1_GRP1_IsEnabledClock
- (*) AHB1ENR ETH1MACEN LL_C1_AHB1_GRP1_IsEnabledClock
- (*) AHB1ENR ETH1TXEN LL_C1_AHB1_GRP1_IsEnabledClock
- (*) AHB1ENR ETH1RXEN LL_C1_AHB1_GRP1_IsEnabledClock
- (*) AHB1ENR USB1OTGHSEN LL_C1_AHB1_GRP1_IsEnabledClock
- AHB1ENR USB1OTGHSULPIEN LL_C1_AHB1_GRP1_IsEnabledClock
- AHB1ENR USB2OTGHSEN LL_C1_AHB1_GRP1_IsEnabledClock
- (*) AHB1ENR USB2OTGHSULPIEN LL_C1_AHB1_GRP1_IsEnabledClock (*)

LL_C1_AHB1_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_C1_AHB1_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable C1 AHB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_DMA1
  - LL_AHB1_GRP1_PERIPH_DMA2
  - LL_AHB1_GRP1_PERIPH_ADC12
  - LL_AHB1_GRP1_PERIPH_CRC (*)
  - LL_AHB1_GRP1_PERIPH_ART (*)
  - LL_AHB1_GRP1_PERIPH_ETH1MAC (*)
  - LL_AHB1_GRP1_PERIPH_ETH1TX (*)
  - LL_AHB1_GRP1_PERIPH_ETH1RX (*)
  - LL_AHB1_GRP1_PERIPH_USB1OTGHS
  - LL_AHB1_GRP1_PERIPH_USB1OTGHSULPI
  - LL_AHB1_GRP1_PERIPH_USB2OTGHS (*)
  - LL_AHB1_GRP1_PERIPH_USB2OTGHSULPI (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1ENR DMA1EN LL_C1_AHB1_GRP1_DisableClock
- AHB1ENR DMA2EN LL_C1_AHB1_GRP1_DisableClock
- AHB1ENR ADC12EN LL_C1_AHB1_GRP1_DisableClock
- AHB1ENR CRCEN LL_C1_AHB1_GRP1_DisableClock
- (*) AHB1ENR ARTEN LL_C1_AHB1_GRP1_DisableClock
- (*) AHB1ENR ETH1MACEN LL_C1_AHB1_GRP1_DisableClock
- (*) AHB1ENR ETH1TXEN LL_C1_AHB1_GRP1_DisableClock
- (*) AHB1ENR ETH1RXEN LL_C1_AHB1_GRP1_DisableClock
- (*) AHB1ENR USB1OTGHSEN LL_C1_AHB1_GRP1_DisableClock
- AHB1ENR USB1OTGHSULPIEN LL_C1_AHB1_GRP1_DisableClock
- AHB1ENR USB2OTGHSEN LL_C1_AHB1_GRP1_DisableClock
- (*) AHB1ENR USB2OTGHSULPIEN LL_C1_AHB1_GRP1_DisableClock (*)

LL_C1_AHB1_GRP1_EnableClockSleep

Function name

__STATIC_INLINE void LL_C1_AHB1_GRP1_EnableClockSleep (uint32_t Periphs)

Function description

Enable C1 AHB1 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_DMA1
  - LL_AHB1_GRP1_PERIPH_DMA2
  - LL_AHB1_GRP1_PERIPH_ADC12
  - LL_AHB1_GRP1_PERIPH_CRC (*)
  - LL_AHB1_GRP1_PERIPH_ART (*)
  - LL_AHB1_GRP1_PERIPH_ETH1MAC (*)
  - LL_AHB1_GRP1_PERIPH_ETH1TX (*)
  - LL_AHB1_GRP1_PERIPH_ETH1RX (*)
  - LL_AHB1_GRP1_PERIPH_USB1OTGHS
  - LL_AHB1_GRP1_PERIPH_USB1OTGHSULPI
  - LL_AHB1_GRP1_PERIPH_USB2OTGHS (*)
  - LL_AHB1_GRP1_PERIPH_USB2OTGHSULPI (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1LPENR DMA1LPEN LL_C1_AHB1_GRP1_EnableClockSleep
- AHB1LPENR DMA2LPEN LL_C1_AHB1_GRP1_EnableClockSleep
- AHB1LPENR ADC12LPEN LL_C1_AHB1_GRP1_EnableClockSleep
- AHB1LPENR CRCLPEN LL_C1_AHB1_GRP1_EnableClockSleep
- (*) AHB1LPENR ARTLPEN LL_C1_AHB1_GRP1_EnableClockSleep
- (*) AHB1LPENR ETH1MACLPEN LL_C1_AHB1_GRP1_EnableClockSleep
- (*) AHB1LPENR ETH1TXLPEN LL_C1_AHB1_GRP1_EnableClockSleep
- (*) AHB1LPENR ETH1RXLPEN LL_C1_AHB1_GRP1_EnableClockSleep
- (*) AHB1LPENR USB1OTGHSLPEN LL_C1_AHB1_GRP1_EnableClockSleep
- AHB1LPENR USB1OTGHSULPILPEN LL_C1_AHB1_GRP1_EnableClockSleep
- AHB1LPENR USB2OTGHSLPEN LL_C1_AHB1_GRP1_EnableClockSleep
- (*) AHB1LPENR USB2OTGHSULPILPEN LL_C1_AHB1_GRP1_EnableClockSleep (*)

LL_C1_AHB1_GRP1_DisableClockSleep

Function name

__STATIC_INLINE void LL_C1_AHB1_GRP1_DisableClockSleep (uint32_t Periphs)

Function description

Disable C1 AHB1 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_DMA1
  - LL_AHB1_GRP1_PERIPH_DMA2
  - LL_AHB1_GRP1_PERIPH_ADC12
  - LL_AHB1_GRP1_PERIPH_CRC (*)
  - LL_AHB1_GRP1_PERIPH_ART (*)
  - LL_AHB1_GRP1_PERIPH_ETH1MAC (*)
  - LL_AHB1_GRP1_PERIPH_ETH1TX (*)
  - LL_AHB1_GRP1_PERIPH_ETH1RX (*)
  - LL_AHB1_GRP1_PERIPH_USB1OTGHS
  - LL_AHB1_GRP1_PERIPH_USB1OTGHSULPI
  - LL_AHB1_GRP1_PERIPH_USB2OTGHS (*)
  - LL_AHB1_GRP1_PERIPH_USB2OTGHSULPI (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1LPENR DMA1LPEN LL_C1_AHB1_GRP1_DisableClockSleep
- AHB1LPENR DMA2LPEN LL_C1_AHB1_GRP1_DisableClockSleep
- AHB1LPENR ADC12LPEN LL_C1_AHB1_GRP1_DisableClockSleep
- AHB1LPENR CRCLPEN LL_C1_AHB1_GRP1_DisableClockSleep
- (*) AHB1LPENR ARTLPEN LL_C1_AHB1_GRP1_DisableClockSleep
- (*) AHB1LPENR ETH1MACLPEN LL_C1_AHB1_GRP1_DisableClockSleep
- (*) AHB1LPENR ETH1TXLPEN LL_C1_AHB1_GRP1_DisableClockSleep
- (*) AHB1LPENR ETH1RXLPEN LL_C1_AHB1_GRP1_DisableClockSleep
- (*) AHB1LPENR USB1OTGHSLPEN LL_C1_AHB1_GRP1_DisableClockSleep
- AHB1LPENR USB1OTGHSULPILPEN LL_C1_AHB1_GRP1_DisableClockSleep
- AHB1LPENR USB2OTGHSLPEN LL_C1_AHB1_GRP1_DisableClockSleep
- (*) AHB1LPENR USB2OTGHSULPILPEN LL_C1_AHB1_GRP1_DisableClockSleep (*)

LL_C1_AHB2_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_C1_AHB2_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable C1 AHB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_DCMI
  - LL_AHB2_GRP1_PERIPH_HSEM (*)
  - LL_AHB2_GRP1_PERIPH_CRYP (*)
  - LL_AHB2_GRP1_PERIPH_HASH (*)
  - LL_AHB2_GRP1_PERIPH_RNG
  - LL_AHB2_GRP1_PERIPH_BDMA1 (*)
  - LL_AHB2_GRP1_PERIPH_SDMMC2
  - LL_AHB2_GRP1_PERIPH_D2SRAM1
  - LL_AHB2_GRP1_PERIPH_D2SRAM2
  - LL_AHB2_GRP1_PERIPH_D2SRAM3 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2ENR DCMIEN LL_C1_AHB2_GRP1_EnableClock
- AHB2ENR HSEMEN LL_C1_AHB2_GRP1_EnableClock
- (*) AHB2ENR CRYPEN LL_C1_AHB2_GRP1_EnableClock
- (*) AHB2ENR HASHEN LL_C1_AHB2_GRP1_EnableClock
- (*) AHB2ENR RNGEN LL_C1_AHB2_GRP1_EnableClock
- AHB2ENR SDMMC2EN LL_C1_AHB2_GRP1_EnableClock
- AHB2ENR BDMA1EN LL_C1_AHB2_GRP1_EnableClock
- (*) AHB2ENR D2SRAM1EN LL_C1_AHB2_GRP1_EnableClock
- AHB2ENR D2SRAM2EN LL_C1_AHB2_GRP1_EnableClock
- AHB2ENR D2SRAM3EN LL_C1_AHB2_GRP1_EnableClock (*)

LL_C1_AHB2_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_C1_AHB2_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if C1 AHB2 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_DCMI
  - LL_AHB2_GRP1_PERIPH_HSEM (*)
  - LL_AHB2_GRP1_PERIPH_CRYP (*)
  - LL_AHB2_GRP1_PERIPH_HASH (*)
  - LL_AHB2_GRP1_PERIPH_RNG
  - LL_AHB2_GRP1_PERIPH_BDMA1 (*)
  - LL_AHB2_GRP1_PERIPH_SDMMC2
  - LL_AHB2_GRP1_PERIPH_D2SRAM1
  - LL_AHB2_GRP1_PERIPH_D2SRAM2
  - LL_AHB2_GRP1_PERIPH_D2SRAM3 (*)

  (*) value not defined in all devices.

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- AHB2ENR DCMIEN LL_C1_AHB2_GRP1_IsEnabledClock
- AHB2ENR HSEMEN LL_C1_AHB2_GRP1_IsEnabledClock
- (*) AHB2ENR CRYPEN LL_C1_AHB2_GRP1_IsEnabledClock
- (*) AHB2ENR HASHEN LL_C1_AHB2_GRP1_IsEnabledClock
- (*) AHB2ENR RNGEN LL_C1_AHB2_GRP1_IsEnabledClock
- AHB2ENR SDMMC2EN LL_C1_AHB2_GRP1_IsEnabledClock
- AHB2ENR BDMA1EN LL_C1_AHB2_GRP1_IsEnabledClock
- (*) AHB2ENR D2SRAM1EN LL_C1_AHB2_GRP1_IsEnabledClock
- AHB2ENR D2SRAM2EN LL_C1_AHB2_GRP1_IsEnabledClock
- AHB2ENR D2SRAM3EN LL_C1_AHB2_GRP1_IsEnabledClock (*)

LL_C1_AHB2_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_C1_AHB2_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable C1 AHB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_DCMI
  - LL_AHB2_GRP1_PERIPH_HSEM (*)
  - LL_AHB2_GRP1_PERIPH_CRYP (*)
  - LL_AHB2_GRP1_PERIPH_HASH (*)
  - LL_AHB2_GRP1_PERIPH_RNG
  - LL_AHB2_GRP1_PERIPH_BDMA1 (*)
  - LL_AHB2_GRP1_PERIPH_SDMMC2
  - LL_AHB2_GRP1_PERIPH_D2SRAM1
  - LL_AHB2_GRP1_PERIPH_D2SRAM2
  - LL_AHB2_GRP1_PERIPH_D2SRAM3 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2ENR DCMIEN LL_C1_AHB2_GRP1_DisableClock
- AHB2ENR HSEMEN LL_C1_AHB2_GRP1_DisableClock
- (*) AHB2ENR CRYPEN LL_C1_AHB2_GRP1_DisableClock
- (*) AHB2ENR HASHEN LL_C1_AHB2_GRP1_DisableClock
- (*) AHB2ENR RNGEN LL_C1_AHB2_GRP1_DisableClock
- AHB2ENR SDMMC2EN LL_C1_AHB2_GRP1_DisableClock
- AHB2ENR BDMA1EN LL_C1_AHB2_GRP1_DisableClock
- (*) AHB2ENR D2SRAM1EN LL_C1_AHB2_GRP1_DisableClock
- AHB2ENR D2SRAM2EN LL_C1_AHB2_GRP1_DisableClock
- AHB2ENR D2SRAM3EN LL_C1_AHB2_GRP1_DisableClock (*)

LL_C1_AHB2_GRP1_EnableClockSleep

Function name

__STATIC_INLINE void LL_C1_AHB2_GRP1_EnableClockSleep (uint32_t Periphs)

Function description

Enable C1 AHB2 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_DCMI
  - LL_AHB2_GRP1_PERIPH_CRYP (*)
  - LL_AHB2_GRP1_PERIPH_HASH (*)
  - LL_AHB2_GRP1_PERIPH_RNG
  - LL_AHB2_GRP1_PERIPH_SDMMC2
  - LL_AHB2_GRP1_PERIPH_BDMA1 (*)
  - LL_AHB2_GRP1_PERIPH_D2SRAM1
  - LL_AHB2_GRP1_PERIPH_D2SRAM2
  - LL_AHB2_GRP1_PERIPH_D2SRAM3 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2LPENR DCMILPEN LL_C1_AHB2_GRP1_EnableClockSleep
- AHB2LPENR CRYPLPEN LL_C1_AHB2_GRP1_EnableClockSleep
- (*) AHB2LPENR HASHLPEN LL_C1_AHB2_GRP1_EnableClockSleep
- (*) AHB2LPENR RNGLPEN LL_C1_AHB2_GRP1_EnableClockSleep
- AHB2LPENR SDMMC2LPEN LL_C1_AHB2_GRP1_EnableClockSleep
- AHB2LPENR D2SRAM1LPEN LL_C1_AHB2_GRP1_EnableClockSleep
- AHB2LPENR BDAM1LPEN LL_C1_AHB2_GRP1_EnableClockSleep
- (*) AHB2LPENR D2SRAM2LPEN LL_C1_AHB2_GRP1_EnableClockSleep
- AHB2LPENR D2SRAM3LPEN LL_C1_AHB2_GRP1_EnableClockSleep (*)

LL_C1_AHB2_GRP1_DisableClockSleep

Function name

__STATIC_INLINE void LL_C1_AHB2_GRP1_DisableClockSleep (uint32_t Periphs)

Function description

Disable C1 AHB2 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_DCMI
  - LL_AHB2_GRP1_PERIPH_CRYP (*)
  - LL_AHB2_GRP1_PERIPH_HASH (*)
  - LL_AHB2_GRP1_PERIPH_RNG
  - LL_AHB2_GRP1_PERIPH_SDMMC2
  - LL_AHB2_GRP1_PERIPH_BDMA1 (*)
  - LL_AHB2_GRP1_PERIPH_D2SRAM1
  - LL_AHB2_GRP1_PERIPH_D2SRAM2
  - LL_AHB2_GRP1_PERIPH_D2SRAM3 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2LPENR DCMILPEN LL_C1_AHB2_GRP1_DisableClockSleep
- AHB2LPENR CRYPLPEN LL_C1_AHB2_GRP1_DisableClockSleep
- (*) AHB2LPENR HASHLPEN LL_C1_AHB2_GRP1_DisableClockSleep
- (*) AHB2LPENR RNGLPEN LL_C1_AHB2_GRP1_DisableClockSleep
- AHB2LPENR SDMMC2LPEN LL_C1_AHB2_GRP1_DisableClockSleep
- AHB2LPENR BDAM1LPEN LL_C1_AHB2_GRP1_DisableClockSleep
- (*) AHB2LPENR D2SRAM1LPEN LL_C1_AHB2_GRP1_DisableClockSleep
- AHB2LPENR D2SRAM2LPEN LL_C1_AHB2_GRP1_DisableClockSleep
- AHB2LPENR D2SRAM3LPEN LL_C1_AHB2_GRP1_DisableClockSleep

LL_C1_AHB4_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_C1_AHB4_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable C1 AHB4 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_GPIOA
  - LL_AHB4_GRP1_PERIPH_GPIOB
  - LL_AHB4_GRP1_PERIPH_GPIOC
  - LL_AHB4_GRP1_PERIPH_GPIOD
  - LL_AHB4_GRP1_PERIPH_GPIOE
  - LL_AHB4_GRP1_PERIPH_GPIOF
  - LL_AHB4_GRP1_PERIPH_GPIOG
  - LL_AHB4_GRP1_PERIPH_GPIOH
  - LL_AHB4_GRP1_PERIPH_GPIOI (*)
  - LL_AHB4_GRP1_PERIPH_GPIOJ
  - LL_AHB4_GRP1_PERIPH_GPIOK
  - LL_AHB4_GRP1_PERIPH_CRC (*)
  - LL_AHB4_GRP1_PERIPH_BDMA
  - LL_AHB4_GRP1_PERIPH_ADC3 (*)
  - LL_AHB4_GRP1_PERIPH_HSEM (*)
  - LL_AHB4_GRP1_PERIPH_BKPRAM
  - LL_AHB4_GRP1_PERIPH_SRAM4

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB4ENR GPIOAEN LL_C1_AHB4_GRP1_EnableClock
- AHB4ENR GPIOBEN LL_C1_AHB4_GRP1_EnableClock
- AHB4ENR GPIOCEN LL_C1_AHB4_GRP1_EnableClock
- AHB4ENR GPIODEN LL_C1_AHB4_GRP1_EnableClock
- AHB4ENR GPIOEEN LL_C1_AHB4_GRP1_EnableClock
- AHB4ENR GPIOFEN LL_C1_AHB4_GRP1_EnableClock
- AHB4ENR GPIOGEN LL_C1_AHB4_GRP1_EnableClock
- AHB4ENR GPIOHEN LL_C1_AHB4_GRP1_EnableClock
- AHB4ENR GPIOIEN LL_C1_AHB4_GRP1_EnableClock
- AHB4ENR GPIOJEN LL_C1_AHB4_GRP1_EnableClock
- AHB4ENR GPIOKEN LL_C1_AHB4_GRP1_EnableClock
- AHB4ENR CRCEN LL_C1_AHB4_GRP1_EnableClock
- (*) AHB4ENR BDMAEN LL_C1_AHB4_GRP1_EnableClock
- AHB4ENR ADC3EN LL_C1_AHB4_GRP1_EnableClock
- (*) AHB4ENR HSEMEN LL_C1_AHB4_GRP1_EnableClock
- (*) AHB4ENR BKPRAMEN LL_C1_AHB4_GRP1_EnableClock
- AHB4ENR SRAM4EN LL_C1_AHB4_GRP1_EnableClock

LL_C1_AHB4_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_C1_AHB4_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if C1 AHB4 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_GPIOA
  - LL_AHB4_GRP1_PERIPH_GPIOB
  - LL_AHB4_GRP1_PERIPH_GPIOC
  - LL_AHB4_GRP1_PERIPH_GPIOD
  - LL_AHB4_GRP1_PERIPH_GPIOE
  - LL_AHB4_GRP1_PERIPH_GPIOF
  - LL_AHB4_GRP1_PERIPH_GPIOG
  - LL_AHB4_GRP1_PERIPH_GPIOH
  - LL_AHB4_GRP1_PERIPH_GPIOI (*)
  - LL_AHB4_GRP1_PERIPH_GPIOJ
  - LL_AHB4_GRP1_PERIPH_GPIOK
  - LL_AHB4_GRP1_PERIPH_CRC (*)
  - LL_AHB4_GRP1_PERIPH_BDMA
  - LL_AHB4_GRP1_PERIPH_ADC3 (*)
  - LL_AHB4_GRP1_PERIPH_HSEM (*)
  - LL_AHB4_GRP1_PERIPH_BKPRAM
  - LL_AHB4_GRP1_PERIPH_SRAM4

  (*) value not defined in all devices.

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- AHB4ENR GPIOAEN LL_C1_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOBEN LL_C1_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOCEN LL_C1_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIODEN LL_C1_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOEEN LL_C1_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOFEN LL_C1_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOGEN LL_C1_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOHEN LL_C1_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOIEN LL_C1_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOJEN LL_C1_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOKEN LL_C1_AHB4_GRP1_IsEnabledClock
- AHB4ENR CRCEN LL_C1_AHB4_GRP1_IsEnabledClock
- (*) AHB4ENR BDMAEN LL_C1_AHB4_GRP1_IsEnabledClock
- AHB4ENR ADC3EN LL_C1_AHB4_GRP1_IsEnabledClock
- (*) AHB4ENR HSEMEN LL_C1_AHB4_GRP1_IsEnabledClock
- (*) AHB4ENR BKPRAMEN LL_C1_AHB4_GRP1_IsEnabledClock
- AHB4ENR SRAM4EN LL_C1_AHB4_GRP1_IsEnabledClock

LL_C1_AHB4_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_C1_AHB4_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable C1 AHB4 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_GPIOA
  - LL_AHB4_GRP1_PERIPH_GPIOB
  - LL_AHB4_GRP1_PERIPH_GPIOC
  - LL_AHB4_GRP1_PERIPH_GPIOD
  - LL_AHB4_GRP1_PERIPH_GPIOE
  - LL_AHB4_GRP1_PERIPH_GPIOF
  - LL_AHB4_GRP1_PERIPH_GPIOG
  - LL_AHB4_GRP1_PERIPH_GPIOH
  - LL_AHB4_GRP1_PERIPH_GPIOI (*)
  - LL_AHB4_GRP1_PERIPH_GPIOJ
  - LL_AHB4_GRP1_PERIPH_GPIOK
  - LL_AHB4_GRP1_PERIPH_CRC (*)
  - LL_AHB4_GRP1_PERIPH_BDMA
  - LL_AHB4_GRP1_PERIPH_ADC3 (*)
  - LL_AHB4_GRP1_PERIPH_HSEM (*)
  - LL_AHB4_GRP1_PERIPH_BKPRAM
  - LL_AHB4_GRP1_PERIPH_SRAM4

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB4ENR GPIOAEN LL_C1_AHB4_GRP1_DisableClock
- AHB4ENR GPIOBEN LL_C1_AHB4_GRP1_DisableClock
- AHB4ENR GPIOCEN LL_C1_AHB4_GRP1_DisableClock
- AHB4ENR GPIODEN LL_C1_AHB4_GRP1_DisableClock
- AHB4ENR GPIOEEN LL_C1_AHB4_GRP1_DisableClock
- AHB4ENR GPIOFEN LL_C1_AHB4_GRP1_DisableClock
- AHB4ENR GPIOGEN LL_C1_AHB4_GRP1_DisableClock
- AHB4ENR GPIOHEN LL_C1_AHB4_GRP1_DisableClock
- AHB4ENR GPIOIEN LL_C1_AHB4_GRP1_DisableClock
- AHB4ENR GPIOJEN LL_C1_AHB4_GRP1_DisableClock
- AHB4ENR GPIOKEN LL_C1_AHB4_GRP1_DisableClock
- AHB4ENR CRCEN LL_C1_AHB4_GRP1_DisableClock
- (*) AHB4ENR BDMAEN LL_C1_AHB4_GRP1_DisableClock
- AHB4ENR ADC3EN LL_C1_AHB4_GRP1_DisableClock
- (*) AHB4ENR HSEMEN LL_C1_AHB4_GRP1_DisableClock
- (*) AHB4ENR BKPRAMEN LL_C1_AHB4_GRP1_DisableClock
- AHB4ENR SRAM4EN LL_C1_AHB4_GRP1_DisableClock

LL_C1_AHB4_GRP1_EnableClockSleep

Function name

__STATIC_INLINE void LL_C1_AHB4_GRP1_EnableClockSleep (uint32_t Periphs)

Function description

Enable C1 AHB4 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_GPIOA
  - LL_AHB4_GRP1_PERIPH_GPIOB
  - LL_AHB4_GRP1_PERIPH_GPIOC
  - LL_AHB4_GRP1_PERIPH_GPIOD
  - LL_AHB4_GRP1_PERIPH_GPIOE
  - LL_AHB4_GRP1_PERIPH_GPIOF
  - LL_AHB4_GRP1_PERIPH_GPIOG
  - LL_AHB4_GRP1_PERIPH_GPIOH
  - LL_AHB4_GRP1_PERIPH_GPIOI (*)
  - LL_AHB4_GRP1_PERIPH_GPIOJ
  - LL_AHB4_GRP1_PERIPH_GPIOK
  - LL_AHB4_GRP1_PERIPH_CRC (*)
  - LL_AHB4_GRP1_PERIPH_BDMA
  - LL_AHB4_GRP1_PERIPH_ADC3 (*)
  - LL_AHB4_GRP1_PERIPH_BKPRAM
  - LL_AHB4_GRP1_PERIPH_SRAM4

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB4LPENR GPIOALPEN LL_C1_AHB4_GRP1_EnableClockSleep
- AHB4LPENR GPIOBLPEN LL_C1_AHB4_GRP1_EnableClockSleep
- AHB4LPENR GPIOCLPEN LL_C1_AHB4_GRP1_EnableClockSleep
- AHB4LPENR GPIODLPEN LL_C1_AHB4_GRP1_EnableClockSleep
- AHB4LPENR GPIOELPEN LL_C1_AHB4_GRP1_EnableClockSleep
- AHB4LPENR GPIOFLPEN LL_C1_AHB4_GRP1_EnableClockSleep
- AHB4LPENR GPIOGLPEN LL_C1_AHB4_GRP1_EnableClockSleep
- AHB4LPENR GPIOHLPEN LL_C1_AHB4_GRP1_EnableClockSleep
- AHB4LPENR GPIOILPEN LL_C1_AHB4_GRP1_EnableClockSleep
- AHB4LPENR GPIOJLPEN LL_C1_AHB4_GRP1_EnableClockSleep
- AHB4LPENR GPIOKLPEN LL_C1_AHB4_GRP1_EnableClockSleep
- AHB4LPENR CRCLPEN LL_C1_AHB4_GRP1_EnableClockSleep
- (*) AHB4LPENR BDMALPEN LL_C1_AHB4_GRP1_EnableClockSleep
- AHB4LPENR ADC3LPEN LL_C1_AHB4_GRP1_EnableClockSleep
- (*) AHB4LPENR BKPRAMLPEN LL_C1_AHB4_GRP1_EnableClockSleep
- AHB4LPENR SRAM4LPEN LL_C1_AHB4_GRP1_EnableClockSleep

LL_C1_AHB4_GRP1_DisableClockSleep

Function name

__STATIC_INLINE void LL_C1_AHB4_GRP1_DisableClockSleep (uint32_t Periphs)

Function description

Disable C1 AHB4 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_GPIOA
  - LL_AHB4_GRP1_PERIPH_GPIOB
  - LL_AHB4_GRP1_PERIPH_GPIOC
  - LL_AHB4_GRP1_PERIPH_GPIOD
  - LL_AHB4_GRP1_PERIPH_GPIOE
  - LL_AHB4_GRP1_PERIPH_GPIOF
  - LL_AHB4_GRP1_PERIPH_GPIOG
  - LL_AHB4_GRP1_PERIPH_GPIOH
  - LL_AHB4_GRP1_PERIPH_GPIOI (*)
  - LL_AHB4_GRP1_PERIPH_GPIOJ
  - LL_AHB4_GRP1_PERIPH_GPIOK
  - LL_AHB4_GRP1_PERIPH_CRC (*)
  - LL_AHB4_GRP1_PERIPH_BDMA
  - LL_AHB4_GRP1_PERIPH_ADC3 (*)
  - LL_AHB4_GRP1_PERIPH_BKPRAM
  - LL_AHB4_GRP1_PERIPH_SRAM4

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB4LPENR GPIOALPEN LL_C1_AHB4_GRP1_DisableClockSleep
- AHB4LPENR GPIOBLPEN LL_C1_AHB4_GRP1_DisableClockSleep
- AHB4LPENR GPIOCLPEN LL_C1_AHB4_GRP1_DisableClockSleep
- AHB4LPENR GPIODLPEN LL_C1_AHB4_GRP1_DisableClockSleep
- AHB4LPENR GPIOELPEN LL_C1_AHB4_GRP1_DisableClockSleep
- AHB4LPENR GPIOFLPEN LL_C1_AHB4_GRP1_DisableClockSleep
- AHB4LPENR GPIOGLPEN LL_C1_AHB4_GRP1_DisableClockSleep
- AHB4LPENR GPIOHLPEN LL_C1_AHB4_GRP1_DisableClockSleep
- AHB4LPENR GPIOILPEN LL_C1_AHB4_GRP1_DisableClockSleep
- AHB4LPENR GPIOJLPEN LL_C1_AHB4_GRP1_DisableClockSleep
- AHB4LPENR GPIOKLPEN LL_C1_AHB4_GRP1_DisableClockSleep
- AHB4LPENR CRCLPEN LL_C1_AHB4_GRP1_DisableClockSleep
- (*) AHB4LPENR BDMALPEN LL_C1_AHB4_GRP1_DisableClockSleep
- AHB4LPENR ADC3LPEN LL_C1_AHB4_GRP1_DisableClockSleep
- (*) AHB4LPENR BKPRAMLPEN LL_C1_AHB4_GRP1_DisableClockSleep
- AHB4LPENR SRAM4LPEN LL_C1_AHB4_GRP1_DisableClockSleep

LL_C1_APB3_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_C1_APB3_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable C1 APB3 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB3_GRP1_PERIPH_LTDC (*)
  - LL_APB3_GRP1_PERIPH_DSI (*)
  - LL_APB3_GRP1_PERIPH_WWDG1

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB3ENR LTDCEN LL_C1_APB3_GRP1_EnableClock
- (*) APB3ENR DSIEN LL_C1_APB3_GRP1_EnableClock
- (*) APB3ENR WWDG1EN LL_C1_APB3_GRP1_EnableClock

LL_C1_APB3_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_C1_APB3_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if C1 APB3 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB3_GRP1_PERIPH_LTDC (*)
  - LL_APB3_GRP1_PERIPH_DSI (*)
  - LL_APB3_GRP1_PERIPH_WWDG1

  (*) value not defined in all devices.

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- APB3ENR LTDCEN LL_C1_APB3_GRP1_IsEnabledClock
- (*) APB3ENR DSIEN LL_C1_APB3_GRP1_IsEnabledClock
- (*) APB3ENR WWDG1EN LL_C1_APB3_GRP1_IsEnabledClock

LL_C1_APB3_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_C1_APB3_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable C1 APB3 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB3_GRP1_PERIPH_LTDC (*)
  - LL_APB3_GRP1_PERIPH_DSI (*)
  - LL_APB3_GRP1_PERIPH_WWDG1

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB3ENR LTDCEN LL_C1_APB3_GRP1_DisableClock
- (*) APB3ENR DSIEN LL_C1_APB3_GRP1_DisableClock
- (*) APB3ENR WWDG1EN LL_C1_APB3_GRP1_DisableClock

LL_C1_APB3_GRP1_EnableClockSleep

Function name

__STATIC_INLINE void LL_C1_APB3_GRP1_EnableClockSleep (uint32_t Periphs)

Function description

Enable C1 APB3 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB3_GRP1_PERIPH_LTDC (*)
  - LL_APB3_GRP1_PERIPH_DSI (*)
  - LL_APB3_GRP1_PERIPH_WWDG1

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB3LPENR LTDCLPEN LL_C1_APB3_GRP1_EnableClockSleep
- (*) APB3LPENR DSILPEN LL_C1_APB3_GRP1_EnableClockSleep
- (*) APB3LPENR WWDG1LPEN LL_C1_APB3_GRP1_EnableClockSleep

LL_C1_APB3_GRP1_DisableClockSleep

Function name

__STATIC_INLINE void LL_C1_APB3_GRP1_DisableClockSleep (uint32_t Periphs)

Function description

Disable C1 APB3 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB3_GRP1_PERIPH_LTDC (*)
  - LL_APB3_GRP1_PERIPH_DSI (*)
  - LL_APB3_GRP1_PERIPH_WWDG1

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB3LPENR LTDCLPEN LL_C1_APB3_GRP1_DisableClockSleep
- (*) APB3LPENR DSILPEN LL_C1_APB3_GRP1_DisableClockSleep
- (*) APB3LPENR WWDG1LPEN LL_C1_APB3_GRP1_DisableClockSleep

LL_C1_APB1_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_C1_APB1_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable C1 APB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM12
  - LL_APB1_GRP1_PERIPH_TIM13
  - LL_APB1_GRP1_PERIPH_TIM14
  - LL_APB1_GRP1_PERIPH_LPTIM1
  - LL_APB1_GRP1_PERIPH_WWDG2 (*)
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_SPDIFRX
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4
  - LL_APB1_GRP1_PERIPH_UART5
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_CEC
  - LL_APB1_GRP1_PERIPH_DAC12
  - LL_APB1_GRP1_PERIPH_UART7
  - LL_APB1_GRP1_PERIPH_UART8

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1LENR TIM2EN LL_C1_APB1_GRP1_EnableClock
- APB1LENR TIM3EN LL_C1_APB1_GRP1_EnableClock
- APB1LENR TIM4EN LL_C1_APB1_GRP1_EnableClock
- APB1LENR TIM5EN LL_C1_APB1_GRP1_EnableClock
- APB1LENR TIM6EN LL_C1_APB1_GRP1_EnableClock
- APB1LENR TIM7EN LL_C1_APB1_GRP1_EnableClock
- APB1LENR TIM12EN LL_C1_APB1_GRP1_EnableClock
- APB1LENR TIM13EN LL_C1_APB1_GRP1_EnableClock
- APB1LENR TIM14EN LL_C1_APB1_GRP1_EnableClock
- APB1LENR LPTIM1EN LL_C1_APB1_GRP1_EnableClock
- APB1LENR WWDG2EN LL_C1_APB1_GRP1_EnableClock
- (*) APB1LENR SPI2EN LL_C1_APB1_GRP1_EnableClock
- APB1LENR SPI3EN LL_C1_APB1_GRP1_EnableClock
- APB1LENR SPDIFRXEN LL_C1_APB1_GRP1_EnableClock
- APB1LENR USART2EN LL_C1_APB1_GRP1_EnableClock
- APB1LENR USART3EN LL_C1_APB1_GRP1_EnableClock
- APB1LENR UART4EN LL_C1_APB1_GRP1_EnableClock
- APB1LENR UART5EN LL_C1_APB1_GRP1_EnableClock
- APB1LENR I2C1EN LL_C1_APB1_GRP1_EnableClock
- APB1LENR I2C2EN LL_C1_APB1_GRP1_EnableClock
- APB1LENR I2C3EN LL_C1_APB1_GRP1_EnableClock
- APB1LENR CECEN LL_C1_APB1_GRP1_EnableClock
- APB1LENR DAC12EN LL_C1_APB1_GRP1_EnableClock
- APB1LENR UART7EN LL_C1_APB1_GRP1_EnableClock
- APB1LENR UART8EN LL_C1_APB1_GRP1_EnableClock

LL_C1_APB1_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_C1_APB1_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if C1 APB1 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM12
  - LL_APB1_GRP1_PERIPH_TIM13
  - LL_APB1_GRP1_PERIPH_TIM14
  - LL_APB1_GRP1_PERIPH_LPTIM1
  - LL_APB1_GRP1_PERIPH_WWDG2 (*)
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_SPDIFRX
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4
  - LL_APB1_GRP1_PERIPH_UART5
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_CEC
  - LL_APB1_GRP1_PERIPH_DAC12
  - LL_APB1_GRP1_PERIPH_UART7
  - LL_APB1_GRP1_PERIPH_UART8

  (*) value not defined in all devices.

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- APB1LENR TIM2EN LL_C1_APB1_GRP1_IsEnabledClock
- APB1LENR TIM3EN LL_C1_APB1_GRP1_IsEnabledClock
- APB1LENR TIM4EN LL_C1_APB1_GRP1_IsEnabledClock
- APB1LENR TIM5EN LL_C1_APB1_GRP1_IsEnabledClock
- APB1LENR TIM6EN LL_C1_APB1_GRP1_IsEnabledClock
- APB1LENR TIM7EN LL_C1_APB1_GRP1_IsEnabledClock
- APB1LENR TIM12EN LL_C1_APB1_GRP1_IsEnabledClock
- APB1LENR TIM13EN LL_C1_APB1_GRP1_IsEnabledClock
- APB1LENR TIM14EN LL_C1_APB1_GRP1_IsEnabledClock
- APB1LENR LPTIM1EN LL_C1_APB1_GRP1_IsEnabledClock
- APB1LENR WWDG2EN LL_C1_APB1_GRP1_IsEnabledClock
- (*) APB1LENR SPI2EN LL_C1_APB1_GRP1_IsEnabledClock
- APB1LENR SPI3EN LL_C1_APB1_GRP1_IsEnabledClock
- APB1LENR SPDIFRXEN LL_C1_APB1_GRP1_IsEnabledClock
- APB1LENR USART2EN LL_C1_APB1_GRP1_IsEnabledClock
- APB1LENR USART3EN LL_C1_APB1_GRP1_IsEnabledClock
- APB1LENR UART4EN LL_C1_APB1_GRP1_IsEnabledClock
- APB1LENR UART5EN LL_C1_APB1_GRP1_IsEnabledClock
- APB1LENR I2C1EN LL_C1_APB1_GRP1_IsEnabledClock
- APB1LENR I2C2EN LL_C1_APB1_GRP1_IsEnabledClock
- APB1LENR I2C3EN LL_C1_APB1_GRP1_IsEnabledClock
- APB1LENR CECEN LL_C1_APB1_GRP1_IsEnabledClock
- APB1LENR DAC12EN LL_C1_APB1_GRP1_IsEnabledClock
- APB1LENR UART7EN LL_C1_APB1_GRP1_IsEnabledClock
- APB1LENR UART8EN LL_C1_APB1_GRP1_IsEnabledClock

LL_C1_APB1_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_C1_APB1_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable C1 APB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM12
  - LL_APB1_GRP1_PERIPH_TIM13
  - LL_APB1_GRP1_PERIPH_TIM14
  - LL_APB1_GRP1_PERIPH_LPTIM1
  - LL_APB1_GRP1_PERIPH_WWDG2 (*)
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_SPDIFRX
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4
  - LL_APB1_GRP1_PERIPH_UART5
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_CEC
  - LL_APB1_GRP1_PERIPH_DAC12
  - LL_APB1_GRP1_PERIPH_UART7
  - LL_APB1_GRP1_PERIPH_UART8

  (*) value not defined in all devices.

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- APB1LENR TIM2EN LL_C1_APB1_GRP1_DisableClock
- APB1LENR TIM3EN LL_C1_APB1_GRP1_DisableClock
- APB1LENR TIM4EN LL_C1_APB1_GRP1_DisableClock
- APB1LENR TIM5EN LL_C1_APB1_GRP1_DisableClock
- APB1LENR TIM6EN LL_C1_APB1_GRP1_DisableClock
- APB1LENR TIM7EN LL_C1_APB1_GRP1_DisableClock
- APB1LENR TIM12EN LL_C1_APB1_GRP1_DisableClock
- APB1LENR TIM13EN LL_C1_APB1_GRP1_DisableClock
- APB1LENR TIM14EN LL_C1_APB1_GRP1_DisableClock
- APB1LENR LPTIM1EN LL_C1_APB1_GRP1_DisableClock
- APB1LENR WWDG2EN LL_C1_APB1_GRP1_DisableClock
- (*) APB1LENR SPI2EN LL_C1_APB1_GRP1_DisableClock
- APB1LENR SPI3EN LL_C1_APB1_GRP1_DisableClock
- APB1LENR SPDIFRXEN LL_C1_APB1_GRP1_DisableClock
- APB1LENR USART2EN LL_C1_APB1_GRP1_DisableClock
- APB1LENR USART3EN LL_C1_APB1_GRP1_DisableClock
- APB1LENR UART4EN LL_C1_APB1_GRP1_DisableClock
- APB1LENR UART5EN LL_C1_APB1_GRP1_DisableClock
- APB1LENR I2C1EN LL_C1_APB1_GRP1_DisableClock
- APB1LENR I2C2EN LL_C1_APB1_GRP1_DisableClock
- APB1LENR I2C3EN LL_C1_APB1_GRP1_DisableClock
- APB1LENR CECEN LL_C1_APB1_GRP1_DisableClock
- APB1LENR DAC12EN LL_C1_APB1_GRP1_DisableClock
- APB1LENR UART7EN LL_C1_APB1_GRP1_DisableClock
- APB1LENR UART8EN LL_C1_APB1_GRP1_DisableClock

LL_C1_APB1_GRP1_EnableClockSleep

Function name

__STATIC_INLINE void LL_C1_APB1_GRP1_EnableClockSleep (uint32_t Periphs)

Function description

Enable C1 APB1 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM12
  - LL_APB1_GRP1_PERIPH_TIM13
  - LL_APB1_GRP1_PERIPH_TIM14
  - LL_APB1_GRP1_PERIPH_LPTIM1
  - LL_APB1_GRP1_PERIPH_WWDG2 (*)
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_SPDIFRX
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4
  - LL_APB1_GRP1_PERIPH_UART5
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_CEC
  - LL_APB1_GRP1_PERIPH_DAC12
  - LL_APB1_GRP1_PERIPH_UART7
  - LL_APB1_GRP1_PERIPH_UART8

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1LLPENR TIM2LPEN LL_C1_APB1_GRP1_EnableClockSleep
- APB1LLPENR TIM3LPEN LL_C1_APB1_GRP1_EnableClockSleep
- APB1LLPENR TIM4LPEN LL_C1_APB1_GRP1_EnableClockSleep
- APB1LLPENR TIM5LPEN LL_C1_APB1_GRP1_EnableClockSleep
- APB1LLPENR TIM6LPEN LL_C1_APB1_GRP1_EnableClockSleep
- APB1LLPENR TIM7LPEN LL_C1_APB1_GRP1_EnableClockSleep
- APB1LLPENR TIM12LPEN LL_C1_APB1_GRP1_EnableClockSleep
- APB1LLPENR TIM13LPEN LL_C1_APB1_GRP1_EnableClockSleep
- APB1LLPENR TIM14LPEN LL_C1_APB1_GRP1_EnableClockSleep
- APB1LLPENR LPTIM1LPEN LL_C1_APB1_GRP1_EnableClockSleep
- APB1LLPENR WWDG2LPEN LL_C1_APB1_GRP1_EnableClockSleep
- (*) APB1LLPENR SPI2LPEN LL_C1_APB1_GRP1_EnableClockSleep
- APB1LLPENR SPI3LPEN LL_C1_APB1_GRP1_EnableClockSleep
- APB1LLPENR SPDIFRXLPEN LL_C1_APB1_GRP1_EnableClockSleep
- APB1LLPENR USART2LPEN LL_C1_APB1_GRP1_EnableClockSleep
- APB1LLPENR USART3LPEN LL_C1_APB1_GRP1_EnableClockSleep
- APB1LLPENR UART4LPEN LL_C1_APB1_GRP1_EnableClockSleep
- APB1LLPENR UART5LPEN LL_C1_APB1_GRP1_EnableClockSleep
- APB1LLPENR I2C1LPEN LL_C1_APB1_GRP1_EnableClockSleep
- APB1LLPENR I2C2LPEN LL_C1_APB1_GRP1_EnableClockSleep
- APB1LLPENR I2C3LPEN LL_C1_APB1_GRP1_EnableClockSleep
- APB1LLPENR CECLPEN LL_C1_APB1_GRP1_EnableClockSleep
- APB1LLPENR DAC12LPEN LL_C1_APB1_GRP1_EnableClockSleep
- APB1LLPENR UART7LPEN LL_C1_APB1_GRP1_EnableClockSleep
- APB1LLPENR UART8LPEN LL_C1_APB1_GRP1_EnableClockSleep

LL_C1_APB1_GRP1_DisableClockSleep

Function name

__STATIC_INLINE void LL_C1_APB1_GRP1_DisableClockSleep (uint32_t Periphs)

Function description

Disable C1 APB1 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM12
  - LL_APB1_GRP1_PERIPH_TIM13
  - LL_APB1_GRP1_PERIPH_TIM14
  - LL_APB1_GRP1_PERIPH_LPTIM1
  - LL_APB1_GRP1_PERIPH_WWDG2 (*)
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_SPDIFRX
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4
  - LL_APB1_GRP1_PERIPH_UART5
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_CEC
  - LL_APB1_GRP1_PERIPH_DAC12
  - LL_APB1_GRP1_PERIPH_UART7
  - LL_APB1_GRP1_PERIPH_UART8

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1LLPENR TIM2LPEN LL_C1_APB1_GRP1_DisableClockSleep
- APB1LLPENR TIM3LPEN LL_C1_APB1_GRP1_DisableClockSleep
- APB1LLPENR TIM4LPEN LL_C1_APB1_GRP1_DisableClockSleep
- APB1LLPENR TIM5LPEN LL_C1_APB1_GRP1_DisableClockSleep
- APB1LLPENR TIM6LPEN LL_C1_APB1_GRP1_DisableClockSleep
- APB1LLPENR TIM7LPEN LL_C1_APB1_GRP1_DisableClockSleep
- APB1LLPENR TIM12LPEN LL_C1_APB1_GRP1_DisableClockSleep
- APB1LLPENR TIM13LPEN LL_C1_APB1_GRP1_DisableClockSleep
- APB1LLPENR TIM14LPEN LL_C1_APB1_GRP1_DisableClockSleep
- APB1LLPENR LPTIM1LPEN LL_C1_APB1_GRP1_DisableClockSleep
- APB1LLPENR WWDG2LPEN LL_C1_APB1_GRP1_DisableClockSleep
- (*) APB1LLPENR SPI2LPEN LL_C1_APB1_GRP1_DisableClockSleep
- APB1LLPENR SPI3LPEN LL_C1_APB1_GRP1_DisableClockSleep
- APB1LLPENR SPDIFRXLPEN LL_C1_APB1_GRP1_DisableClockSleep
- APB1LLPENR USART2LPEN LL_C1_APB1_GRP1_DisableClockSleep
- APB1LLPENR USART3LPEN LL_C1_APB1_GRP1_DisableClockSleep
- APB1LLPENR UART4LPEN LL_C1_APB1_GRP1_DisableClockSleep
- APB1LLPENR UART5LPEN LL_C1_APB1_GRP1_DisableClockSleep
- APB1LLPENR I2C1LPEN LL_C1_APB1_GRP1_DisableClockSleep
- APB1LLPENR I2C2LPEN LL_C1_APB1_GRP1_DisableClockSleep
- APB1LLPENR I2C3LPEN LL_C1_APB1_GRP1_DisableClockSleep
- APB1LLPENR CECLPEN LL_C1_APB1_GRP1_DisableClockSleep
- APB1LLPENR DAC12LPEN LL_C1_APB1_GRP1_DisableClockSleep
- APB1LLPENR UART7LPEN LL_C1_APB1_GRP1_DisableClockSleep
- APB1LLPENR UART8LPEN LL_C1_APB1_GRP1_DisableClockSleep

LL_C1_APB1_GRP2_EnableClock

Function name

__STATIC_INLINE void LL_C1_APB1_GRP2_EnableClock (uint32_t Periphs)

Function description

Enable C1 APB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_CRS
  - LL_APB1_GRP2_PERIPH_SWPMI1
  - LL_APB1_GRP2_PERIPH_OPAMP
  - LL_APB1_GRP2_PERIPH_MDIOS
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_TIM23 (*)
  - LL_APB1_GRP2_PERIPH_TIM24 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1HENR CRSEN LL_C1_APB1_GRP2_EnableClock
- APB1HENR SWPMIEN LL_C1_APB1_GRP2_EnableClock
- APB1HENR OPAMPEN LL_C1_APB1_GRP2_EnableClock
- APB1HENR MDIOSEN LL_C1_APB1_GRP2_EnableClock
- APB1HENR FDCANEN LL_C1_APB1_GRP2_EnableClock

LL_C1_APB1_GRP2_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_C1_APB1_GRP2_IsEnabledClock (uint32_t Periphs)

Function description

Check if C1 APB1 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_CRS
  - LL_APB1_GRP2_PERIPH_SWPMI1
  - LL_APB1_GRP2_PERIPH_OPAMP
  - LL_APB1_GRP2_PERIPH_MDIOS
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_TIM23 (*)
  - LL_APB1_GRP2_PERIPH_TIM24 (*)

  (*) value not defined in all devices.

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- APB1HENR CRSEN LL_C1_APB1_GRP2_IsEnabledClock
- APB1HENR SWPMIEN LL_C1_APB1_GRP2_IsEnabledClock
- APB1HENR OPAMPEN LL_C1_APB1_GRP2_IsEnabledClock
- APB1HENR MDIOSEN LL_C1_APB1_GRP2_IsEnabledClock
- APB1HENR FDCANEN LL_C1_APB1_GRP2_IsEnabledClock

LL_C1_APB1_GRP2_DisableClock

Function name

__STATIC_INLINE void LL_C1_APB1_GRP2_DisableClock (uint32_t Periphs)

Function description

Disable C1 APB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_CRS
  - LL_APB1_GRP2_PERIPH_SWPMI1
  - LL_APB1_GRP2_PERIPH_OPAMP
  - LL_APB1_GRP2_PERIPH_MDIOS
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_TIM23 (*)
  - LL_APB1_GRP2_PERIPH_TIM24 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1HENR CRSEN LL_C1_APB1_GRP2_DisableClock
- APB1HENR SWPMIEN LL_C1_APB1_GRP2_DisableClock
- APB1HENR OPAMPEN LL_C1_APB1_GRP2_DisableClock
- APB1HENR MDIOSEN LL_C1_APB1_GRP2_DisableClock
- APB1HENR FDCANEN LL_C1_APB1_GRP2_DisableClock

LL_C1_APB1_GRP2_EnableClockSleep

Function name

__STATIC_INLINE void LL_C1_APB1_GRP2_EnableClockSleep (uint32_t Periphs)

Function description

Enable C1 APB1 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_CRS
  - LL_APB1_GRP2_PERIPH_SWPMI1
  - LL_APB1_GRP2_PERIPH_OPAMP
  - LL_APB1_GRP2_PERIPH_MDIOS
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_TIM23 (*)
  - LL_APB1_GRP2_PERIPH_TIM24 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1HLPENR CRSLPEN LL_C1_APB1_GRP2_EnableClockSleep
- APB1HLPENR SWPMILPEN LL_C1_APB1_GRP2_EnableClockSleep
- APB1HLPENR OPAMPLPEN LL_C1_APB1_GRP2_EnableClockSleep
- APB1HLPENR MDIOSLPEN LL_C1_APB1_GRP2_EnableClockSleep
- APB1HLPENR FDCANLPEN LL_C1_APB1_GRP2_EnableClockSleep

LL_C1_APB1_GRP2_DisableClockSleep

Function name

__STATIC_INLINE void LL_C1_APB1_GRP2_DisableClockSleep (uint32_t Periphs)

Function description

Disable C1 APB1 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_CRS
  - LL_APB1_GRP2_PERIPH_SWPMI1
  - LL_APB1_GRP2_PERIPH_OPAMP
  - LL_APB1_GRP2_PERIPH_MDIOS
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_TIM23 (*)
  - LL_APB1_GRP2_PERIPH_TIM24 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1HLPENR CRSLPEN LL_C1_APB1_GRP2_DisableClockSleep
- APB1HLPENR SWPMILPEN LL_C1_APB1_GRP2_DisableClockSleep
- APB1HLPENR OPAMPLPEN LL_C1_APB1_GRP2_DisableClockSleep
- APB1HLPENR MDIOSLPEN LL_C1_APB1_GRP2_DisableClockSleep
- APB1HLPENR FDCANLPEN LL_C1_APB1_GRP2_DisableClockSleep

LL_C1_APB2_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_C1_APB2_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable C1 APB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_USART6
  - LL_APB2_GRP1_PERIPH_UART9 (*)
  - LL_APB2_GRP1_PERIPH_USART10 (*)
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_SPI4
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_SPI5
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_SAI2 (*)
  - LL_APB2_GRP1_PERIPH_SAI3 (*)
  - LL_APB2_GRP1_PERIPH_DFSDM1
  - LL_APB2_GRP1_PERIPH_HRTIM (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2ENR TIM1EN LL_C1_APB2_GRP1_EnableClock
- APB2ENR TIM8EN LL_C1_APB2_GRP1_EnableClock
- APB2ENR USART1EN LL_C1_APB2_GRP1_EnableClock
- APB2ENR USART6EN LL_C1_APB2_GRP1_EnableClock
- APB2ENR UART9EN LL_C1_APB2_GRP1_EnableClock
- (*) APB2ENR USART10EN LL_C1_APB2_GRP1_EnableClock
- (*) APB2ENR SPI1EN LL_C1_APB2_GRP1_EnableClock
- APB2ENR SPI4EN LL_C1_APB2_GRP1_EnableClock
- APB2ENR TIM15EN LL_C1_APB2_GRP1_EnableClock
- APB2ENR TIM16EN LL_C1_APB2_GRP1_EnableClock
- APB2ENR TIM17EN LL_C1_APB2_GRP1_EnableClock
- APB2ENR SPI5EN LL_C1_APB2_GRP1_EnableClock
- APB2ENR SAI1EN LL_C1_APB2_GRP1_EnableClock
- APB2ENR SAI2EN LL_C1_APB2_GRP1_EnableClock
- APB2ENR SAI3EN LL_C1_APB2_GRP1_EnableClock
- (*) APB2ENR DFSDM1EN LL_C1_APB2_GRP1_EnableClock
- APB2ENR HRTIMEN LL_C1_APB2_GRP1_EnableClock (*)

LL_C1_APB2_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_C1_APB2_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if C1 APB2 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_USART6
  - LL_APB2_GRP1_PERIPH_UART9 (*)
  - LL_APB2_GRP1_PERIPH_USART10 (*)
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_SPI4
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_SPI5
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_SAI2 (*)
  - LL_APB2_GRP1_PERIPH_SAI3 (*)
  - LL_APB2_GRP1_PERIPH_DFSDM1
  - LL_APB2_GRP1_PERIPH_HRTIM (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2ENR TIM1EN LL_C1_APB2_GRP1_IsEnabledClock
- APB2ENR TIM8EN LL_C1_APB2_GRP1_IsEnabledClock
- APB2ENR USART1EN LL_C1_APB2_GRP1_IsEnabledClock
- APB2ENR USART6EN LL_C1_APB2_GRP1_IsEnabledClock
- APB2ENR UART9EN LL_C1_APB2_GRP1_IsEnabledClock
- (*) APB2ENR USART10EN LL_C1_APB2_GRP1_IsEnabledClock
- (*) APB2ENR SPI1EN LL_C1_APB2_GRP1_IsEnabledClock
- APB2ENR SPI4EN LL_C1_APB2_GRP1_IsEnabledClock
- APB2ENR TIM15EN LL_C1_APB2_GRP1_IsEnabledClock
- APB2ENR TIM16EN LL_C1_APB2_GRP1_IsEnabledClock
- APB2ENR TIM17EN LL_C1_APB2_GRP1_IsEnabledClock
- APB2ENR SPI5EN LL_C1_APB2_GRP1_IsEnabledClock
- APB2ENR SAI1EN LL_C1_APB2_GRP1_IsEnabledClock
- APB2ENR SAI2EN LL_C1_APB2_GRP1_IsEnabledClock
- APB2ENR SAI3EN LL_C1_APB2_GRP1_IsEnabledClock
- (*) APB2ENR DFSDM1EN LL_C1_APB2_GRP1_IsEnabledClock
- APB2ENR HRTIMEN LL_C1_APB2_GRP1_IsEnabledClock (*)

LL_C1_APB2_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_C1_APB2_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable C1 APB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_USART6
  - LL_APB2_GRP1_PERIPH_UART9 (*)
  - LL_APB2_GRP1_PERIPH_USART10 (*)
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_SPI4
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_SPI5
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_SAI2 (*)
  - LL_APB2_GRP1_PERIPH_SAI3 (*)
  - LL_APB2_GRP1_PERIPH_DFSDM1
  - LL_APB2_GRP1_PERIPH_HRTIM (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2ENR TIM1EN LL_C1_APB2_GRP1_DisableClock
- APB2ENR TIM8EN LL_C1_APB2_GRP1_DisableClock
- APB2ENR USART1EN LL_C1_APB2_GRP1_DisableClock
- APB2ENR USART6EN LL_C1_APB2_GRP1_DisableClock
- APB2ENR UART9EN LL_C1_APB2_GRP1_DisableClock
- (*) APB2ENR USART10EN LL_C1_APB2_GRP1_DisableClock
- (*) APB2ENR SPI1EN LL_C1_APB2_GRP1_DisableClock
- APB2ENR SPI4EN LL_C1_APB2_GRP1_DisableClock
- APB2ENR TIM15EN LL_C1_APB2_GRP1_DisableClock
- APB2ENR TIM16EN LL_C1_APB2_GRP1_DisableClock
- APB2ENR TIM17EN LL_C1_APB2_GRP1_DisableClock
- APB2ENR SPI5EN LL_C1_APB2_GRP1_DisableClock
- APB2ENR SAI1EN LL_C1_APB2_GRP1_DisableClock
- APB2ENR SAI2EN LL_C1_APB2_GRP1_DisableClock
- APB2ENR SAI3EN LL_C1_APB2_GRP1_DisableClock
- (*) APB2ENR DFSDM1EN LL_C1_APB2_GRP1_DisableClock
- APB2ENR HRTIMEN LL_C1_APB2_GRP1_DisableClock (*)

LL_C1_APB2_GRP1_EnableClockSleep

Function name

__STATIC_INLINE void LL_C1_APB2_GRP1_EnableClockSleep (uint32_t Periphs)

Function description

Enable C1 APB2 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_USART6
  - LL_APB2_GRP1_PERIPH_UART9 (*)
  - LL_APB2_GRP1_PERIPH_USART10 (*)
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_SPI4
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_SPI5
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_SAI2 (*)
  - LL_APB2_GRP1_PERIPH_SAI3 (*)
  - LL_APB2_GRP1_PERIPH_DFSDM1
  - LL_APB2_GRP1_PERIPH_HRTIM (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2LPENR TIM1LPEN LL_C1_APB2_GRP1_EnableClockSleep
- APB2LPENR TIM8LPEN LL_C1_APB2_GRP1_EnableClockSleep
- APB2LPENR USART1LPEN LL_C1_APB2_GRP1_EnableClockSleep
- APB2LPENR USART6LPEN LL_C1_APB2_GRP1_EnableClockSleep
- APB2ENR UART9EN LL_C1_APB2_GRP1_EnableClockSleep
- (*) APB2ENR USART10EN LL_C1_APB2_GRP1_EnableClockSleep
- (*) APB2LPENR SPI1LPEN LL_C1_APB2_GRP1_EnableClockSleep
- APB2LPENR SPI4LPEN LL_C1_APB2_GRP1_EnableClockSleep
- APB2LPENR TIM15LPEN LL_C1_APB2_GRP1_EnableClockSleep
- APB2LPENR TIM16LPEN LL_C1_APB2_GRP1_EnableClockSleep
- APB2LPENR TIM17LPEN LL_C1_APB2_GRP1_EnableClockSleep
- APB2LPENR SPI5LPEN LL_C1_APB2_GRP1_EnableClockSleep
- APB2LPENR SAI1LPEN LL_C1_APB2_GRP1_EnableClockSleep
- APB2LPENR SAI2LPEN LL_C1_APB2_GRP1_EnableClockSleep
- APB2LPENR SAI3LPEN LL_C1_APB2_GRP1_EnableClockSleep
- (*) APB2LPENR DFSDM1LPEN LL_C1_APB2_GRP1_EnableClockSleep
- APB2LPENR HRTIMLPEN LL_C1_APB2_GRP1_EnableClockSleep (*)

LL_C1_APB2_GRP1_DisableClockSleep

Function name

__STATIC_INLINE void LL_C1_APB2_GRP1_DisableClockSleep (uint32_t Periphs)

Function description

Disable C1 APB2 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_USART6
  - LL_APB2_GRP1_PERIPH_UART9 (*)
  - LL_APB2_GRP1_PERIPH_USART10 (*)
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_SPI4
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_SPI5
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_SAI2 (*)
  - LL_APB2_GRP1_PERIPH_SAI3 (*)
  - LL_APB2_GRP1_PERIPH_DFSDM1
  - LL_APB2_GRP1_PERIPH_HRTIM (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2LPENR TIM1LPEN LL_C1_APB2_GRP1_DisableClockSleep
- APB2LPENR TIM8LPEN LL_C1_APB2_GRP1_DisableClockSleep
- APB2LPENR USART1LPEN LL_C1_APB2_GRP1_DisableClockSleep
- APB2LPENR UART9LPEN LL_C1_APB2_GRP1_DisableClockSleep
- (*) APB2LPENR USART10LPEN LL_C1_APB2_GRP1_DisableClockSleep
- (*) APB2LPENR USART6LPEN LL_C1_APB2_GRP1_DisableClockSleep
- APB2LPENR SPI1LPEN LL_C1_APB2_GRP1_DisableClockSleep
- APB2LPENR SPI4LPEN LL_C1_APB2_GRP1_DisableClockSleep
- APB2LPENR TIM15LPEN LL_C1_APB2_GRP1_DisableClockSleep
- APB2LPENR TIM16LPEN LL_C1_APB2_GRP1_DisableClockSleep
- APB2LPENR TIM17LPEN LL_C1_APB2_GRP1_DisableClockSleep
- APB2LPENR SPI5LPEN LL_C1_APB2_GRP1_DisableClockSleep
- APB2LPENR SAI1LPEN LL_C1_APB2_GRP1_DisableClockSleep
- APB2LPENR SAI2LPEN LL_C1_APB2_GRP1_DisableClockSleep
- APB2LPENR SAI3LPEN LL_C1_APB2_GRP1_DisableClockSleep
- (*) APB2LPENR DFSDM1LPEN LL_C1_APB2_GRP1_DisableClockSleep
- APB2LPENR HRTIMLPEN LL_C1_APB2_GRP1_DisableClockSleep (*)

LL_C1_APB4_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_C1_APB4_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable C1 APB4 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP1_PERIPH_SYSCFG
  - LL_APB4_GRP1_PERIPH_LPUART1
  - LL_APB4_GRP1_PERIPH_SPI6
  - LL_APB4_GRP1_PERIPH_I2C4
  - LL_APB4_GRP1_PERIPH_LPTIM2
  - LL_APB4_GRP1_PERIPH_LPTIM3
  - LL_APB4_GRP1_PERIPH_LPTIM4 (*)
  - LL_APB4_GRP1_PERIPH_LPTIM5 (*)
  - LL_APB4_GRP1_PERIPH_COMP12
  - LL_APB4_GRP1_PERIPH_VREF
  - LL_APB4_GRP1_PERIPH_RTCAPB
  - LL_APB4_GRP1_PERIPH_SAI4 (*)
  - LL_APB4_GRP1_PERIPH_DTS (*)
  - LL_APB4_GRP1_PERIPH_DFSDM2 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB4ENR SYSCFGEN LL_C1_APB4_GRP1_EnableClock
- APB4ENR LPUART1EN LL_C1_APB4_GRP1_EnableClock
- APB4ENR SPI6EN LL_C1_APB4_GRP1_EnableClock
- APB4ENR I2C4EN LL_C1_APB4_GRP1_EnableClock
- APB4ENR LPTIM2EN LL_C1_APB4_GRP1_EnableClock
- APB4ENR LPTIM3EN LL_C1_APB4_GRP1_EnableClock
- APB4ENR LPTIM4EN LL_C1_APB4_GRP1_EnableClock
- (*) APB4ENR LPTIM5EN LL_C1_APB4_GRP1_EnableClock
- (*) APB4ENR DAC2EN LL_C1_APB4_GRP1_EnableClock
- (*) APB4ENR COMP12EN LL_C1_APB4_GRP1_EnableClock
- APB4ENR VREFEN LL_C1_APB4_GRP1_EnableClock
- APB4ENR RTCAPBEN LL_C1_APB4_GRP1_EnableClock
- APB4ENR SAI4EN LL_C1_APB4_GRP1_EnableClock
- (*) APB4ENR DTSEN LL_C1_APB4_GRP1_EnableClock
- (*) APB4ENR DFSDM2EN LL_C1_APB4_GRP1_EnableClock (*)

LL_C1_APB4_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_C1_APB4_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if C1 APB4 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP1_PERIPH_SYSCFG
  - LL_APB4_GRP1_PERIPH_LPUART1
  - LL_APB4_GRP1_PERIPH_SPI6
  - LL_APB4_GRP1_PERIPH_I2C4
  - LL_APB4_GRP1_PERIPH_LPTIM2
  - LL_APB4_GRP1_PERIPH_LPTIM3
  - LL_APB4_GRP1_PERIPH_LPTIM4 (*)
  - LL_APB4_GRP1_PERIPH_LPTIM5 (*)
  - LL_APB4_GRP1_PERIPH_COMP12
  - LL_APB4_GRP1_PERIPH_VREF
  - LL_APB4_GRP1_PERIPH_SAI4 (*)
  - LL_APB4_GRP1_PERIPH_DTS (*)
  - LL_APB4_GRP1_PERIPH_DFSDM2 (*)

  (*) value not defined in all devices.

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- APB4ENR SYSCFGEN LL_C1_APB4_GRP1_IsEnabledClock
- APB4ENR LPUART1EN LL_C1_APB4_GRP1_IsEnabledClock
- APB4ENR SPI6EN LL_C1_APB4_GRP1_IsEnabledClock
- APB4ENR I2C4EN LL_C1_APB4_GRP1_IsEnabledClock
- APB4ENR LPTIM2EN LL_C1_APB4_GRP1_IsEnabledClock
- APB4ENR LPTIM3EN LL_C1_APB4_GRP1_IsEnabledClock
- APB4ENR LPTIM4EN LL_C1_APB4_GRP1_IsEnabledClock
- (*) APB4ENR LPTIM5EN LL_C1_APB4_GRP1_IsEnabledClock
- (*) APB4ENR COMP12EN LL_C1_APB4_GRP1_IsEnabledClock
- APB4ENR VREFEN LL_C1_APB4_GRP1_IsEnabledClock
- APB4ENR RTCAPBEN LL_C1_APB4_GRP1_IsEnabledClock
- APB4ENR SAI4EN LL_C1_APB4_GRP1_IsEnabledClock
- (*) APB4ENR DTSEN LL_C1_APB4_GRP1_IsEnabledClock
- (*) APB4ENR DFSDM2EN LL_C1_APB4_GRP1_IsEnabledClock (*)

LL_C1_APB4_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_C1_APB4_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable C1 APB4 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP1_PERIPH_SYSCFG
  - LL_APB4_GRP1_PERIPH_LPUART1
  - LL_APB4_GRP1_PERIPH_SPI6
  - LL_APB4_GRP1_PERIPH_I2C4
  - LL_APB4_GRP1_PERIPH_LPTIM2
  - LL_APB4_GRP1_PERIPH_LPTIM3
  - LL_APB4_GRP1_PERIPH_LPTIM4 (*)
  - LL_APB4_GRP1_PERIPH_LPTIM5 (*)
  - LL_APB4_GRP1_PERIPH_COMP12
  - LL_APB4_GRP1_PERIPH_VREF
  - LL_APB4_GRP1_PERIPH_RTCAPB
  - LL_APB4_GRP1_PERIPH_SAI4 (*)
  - LL_APB4_GRP1_PERIPH_DTS (*)
  - LL_APB4_GRP1_PERIPH_DFSDM2 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB4ENR SYSCFGEN LL_C1_APB4_GRP1_DisableClock
- APB4ENR LPUART1EN LL_C1_APB4_GRP1_DisableClock
- APB4ENR SPI6EN LL_C1_APB4_GRP1_DisableClock
- APB4ENR I2C4EN LL_C1_APB4_GRP1_DisableClock
- APB4ENR LPTIM2EN LL_C1_APB4_GRP1_DisableClock
- APB4ENR LPTIM3EN LL_C1_APB4_GRP1_DisableClock
- APB4ENR LPTIM4EN LL_C1_APB4_GRP1_DisableClock
- (*) APB4ENR LPTIM5EN LL_C1_APB4_GRP1_DisableClock
- (*) APB4ENR COMP12EN LL_C1_APB4_GRP1_DisableClock
- APB4ENR VREFEN LL_C1_APB4_GRP1_DisableClock
- APB4ENR RTCAPBEN LL_C1_APB4_GRP1_DisableClock
- APB4ENR SAI4EN LL_C1_APB4_GRP1_DisableClock
- (*) APB4ENR DTSEN LL_C1_APB4_GRP1_DisableClock
- (*) APB4ENR DFSDM2EN LL_C1_APB4_GRP1_DisableClock (*)

LL_C1_APB4_GRP1_EnableClockSleep

Function name

__STATIC_INLINE void LL_C1_APB4_GRP1_EnableClockSleep (uint32_t Periphs)

Function description

Enable C1 APB4 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP1_PERIPH_SYSCFG
  - LL_APB4_GRP1_PERIPH_LPUART1
  - LL_APB4_GRP1_PERIPH_SPI6
  - LL_APB4_GRP1_PERIPH_I2C4
  - LL_APB4_GRP1_PERIPH_LPTIM2
  - LL_APB4_GRP1_PERIPH_LPTIM3
  - LL_APB4_GRP1_PERIPH_LPTIM4 (*)
  - LL_APB4_GRP1_PERIPH_LPTIM5 (*)
  - LL_APB4_GRP1_PERIPH_COMP12
  - LL_APB4_GRP1_PERIPH_VREF
  - LL_APB4_GRP1_PERIPH_RTCAPB
  - LL_APB4_GRP1_PERIPH_SAI4 (*)
  - LL_APB4_GRP1_PERIPH_DTS (*)
  - LL_APB4_GRP1_PERIPH_DFSDM2 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB4LPENR SYSCFGLPEN LL_C1_APB4_GRP1_EnableClockSleep
- APB4LPENR LPUART1LPEN LL_C1_APB4_GRP1_EnableClockSleep
- APB4LPENR SPI6LPEN LL_C1_APB4_GRP1_EnableClockSleep
- APB4LPENR I2C4LPEN LL_C1_APB4_GRP1_EnableClockSleep
- APB4LPENR LPTIM2LPEN LL_C1_APB4_GRP1_EnableClockSleep
- APB4LPENR LPTIM3LPEN LL_C1_APB4_GRP1_EnableClockSleep
- (*) APB4LPENR LPTIM4LPEN LL_C1_APB4_GRP1_EnableClockSleep
- (*) APB4LPENR LPTIM5LPEN LL_C1_APB4_GRP1_EnableClockSleep
- APB4LPENR COMP12LPEN LL_C1_APB4_GRP1_EnableClockSleep
- APB4LPENR VREFLPEN LL_C1_APB4_GRP1_EnableClockSleep
- APB4LPENR RTCAPBLPEN LL_C1_APB4_GRP1_EnableClockSleep
- APB4LPENR SAI4LPEN LL_C1_APB4_GRP1_EnableClockSleep
- (*) APB4ENR DTSLPEN LL_C1_APB4_GRP1_EnableClockSleep
- (*) APB4ENR DFSDM2LPEN LL_C1_APB4_GRP1_EnableClockSleep (*)

LL_C1_APB4_GRP1_DisableClockSleep

Function name

__STATIC_INLINE void LL_C1_APB4_GRP1_DisableClockSleep (uint32_t Periphs)

Function description

Disable C1 APB4 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP1_PERIPH_SYSCFG
  - LL_APB4_GRP1_PERIPH_LPUART1
  - LL_APB4_GRP1_PERIPH_SPI6
  - LL_APB4_GRP1_PERIPH_I2C4
  - LL_APB4_GRP1_PERIPH_LPTIM2
  - LL_APB4_GRP1_PERIPH_LPTIM3
  - LL_APB4_GRP1_PERIPH_LPTIM4 (*)
  - LL_APB4_GRP1_PERIPH_LPTIM5 (*)
  - LL_APB4_GRP1_PERIPH_COMP12
  - LL_APB4_GRP1_PERIPH_VREF
  - LL_APB4_GRP1_PERIPH_RTCAPB
  - LL_APB4_GRP1_PERIPH_SAI4 (*)
  - LL_APB4_GRP1_PERIPH_DTS (*)
  - LL_APB4_GRP1_PERIPH_DFSDM2 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB4LPENR SYSCFGLPEN LL_C1_APB4_GRP1_DisableClockSleep
- APB4LPENR LPUART1LPEN LL_C1_APB4_GRP1_DisableClockSleep
- APB4LPENR SPI6LPEN LL_C1_APB4_GRP1_DisableClockSleep
- APB4LPENR I2C4LPEN LL_C1_APB4_GRP1_DisableClockSleep
- APB4LPENR LPTIM2LPEN LL_C1_APB4_GRP1_DisableClockSleep
- APB4LPENR LPTIM3LPEN LL_C1_APB4_GRP1_DisableClockSleep
- APB4LPENR LPTIM4LPEN LL_C1_APB4_GRP1_DisableClockSleep
- APB4LPENR LPTIM5LPEN LL_C1_APB4_GRP1_DisableClockSleep
- APB4LPENR COMP12LPEN LL_C1_APB4_GRP1_DisableClockSleep
- APB4LPENR VREFLPEN LL_C1_APB4_GRP1_DisableClockSleep
- APB4LPENR RTCAPBLPEN LL_C1_APB4_GRP1_DisableClockSleep
- APB4LPENR SAI4LPEN LL_C1_APB4_GRP1_DisableClockSleep
- (*) APB4ENR DTSLPEN LL_C1_APB4_GRP1_DisableClockSleep
- (*) APB4ENR DFSDM2LPEN LL_C1_APB4_GRP1_DisableClockSleep (*)

LL_C2_AHB3_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_C2_AHB3_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable C2 AHB3 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_MDMA
  - LL_AHB3_GRP1_PERIPH_DMA2D
  - LL_AHB3_GRP1_PERIPH_JPGDEC (*)
  - LL_AHB3_GRP1_PERIPH_FMC
  - LL_AHB3_GRP1_PERIPH_QSPI (*)
  - LL_AHB3_GRP1_PERIPH_SDMMC1
  - LL_AHB3_GRP1_PERIPH_FLASH
  - LL_AHB3_GRP1_PERIPH_DTCM1
  - LL_AHB3_GRP1_PERIPH_DTCM2
  - LL_AHB3_GRP1_PERIPH_ITCM
  - LL_AHB3_GRP1_PERIPH_AXISRAM

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3ENR MDMAEN LL_C2_AHB3_GRP1_EnableClock
- AHB3ENR DMA2DEN LL_C2_AHB3_GRP1_EnableClock
- AHB3ENR JPGDECEN LL_C2_AHB3_GRP1_EnableClock
- AHB3ENR FMCEN LL_C2_AHB3_GRP1_EnableClock
- AHB3ENR QSPIEN LL_C2_AHB3_GRP1_EnableClock
- AHB3ENR SDMMC1EN LL_C2_AHB3_GRP1_EnableClock
- AHB3ENR FLASHEN LL_C2_AHB3_GRP1_EnableClock
- AHB3ENR DTCM1EN LL_C2_AHB3_GRP1_EnableClock
- AHB3ENR DTCM2EN LL_C2_AHB3_GRP1_EnableClock
- AHB3ENR ITCMEN LL_C2_AHB3_GRP1_EnableClock
- AHB3ENR AXISRAMEN LL_C2_AHB3_GRP1_EnableClock

LL_C2_AHB3_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_C2_AHB3_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if C2 AHB3 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_MDMA
  - LL_AHB3_GRP1_PERIPH_DMA2D
  - LL_AHB3_GRP1_PERIPH_JPGDEC (*)
  - LL_AHB3_GRP1_PERIPH_FMC
  - LL_AHB3_GRP1_PERIPH_QSPI (*)
  - LL_AHB3_GRP1_PERIPH_SDMMC1
  - LL_AHB3_GRP1_PERIPH_FLASH
  - LL_AHB3_GRP1_PERIPH_DTCM1
  - LL_AHB3_GRP1_PERIPH_DTCM2
  - LL_AHB3_GRP1_PERIPH_ITCM
  - LL_AHB3_GRP1_PERIPH_AXISRAM

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- AHB3ENR MDMAEN LL_C2_AHB3_GRP1_IsEnabledClock
- AHB3ENR DMA2DEN LL_C2_AHB3_GRP1_IsEnabledClock
- AHB3ENR JPGDECEN LL_C2_AHB3_GRP1_IsEnabledClock
- AHB3ENR FMCEN LL_C2_AHB3_GRP1_IsEnabledClock
- AHB3ENR QSPIEN LL_C2_AHB3_GRP1_IsEnabledClock
- AHB3ENR SDMMC1EN LL_C2_AHB3_GRP1_IsEnabledClock
- AHB3ENR FLASHEN LL_C2_AHB3_GRP1_IsEnabledClock
- AHB3ENR DTCM1EN LL_C2_AHB3_GRP1_IsEnabledClock
- AHB3ENR DTCM2EN LL_C2_AHB3_GRP1_IsEnabledClock
- AHB3ENR ITCMEN LL_C2_AHB3_GRP1_IsEnabledClock
- AHB3ENR AXISRAMEN LL_C2_AHB3_GRP1_IsEnabledClock

LL_C2_AHB3_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_C2_AHB3_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable C2 AHB3 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_MDMA
  - LL_AHB3_GRP1_PERIPH_DMA2D
  - LL_AHB3_GRP1_PERIPH_JPGDEC (*)
  - LL_AHB3_GRP1_PERIPH_FMC
  - LL_AHB3_GRP1_PERIPH_QSPI (*)
  - LL_AHB3_GRP1_PERIPH_SDMMC1
  - LL_AHB3_GRP1_PERIPH_FLASH
  - LL_AHB3_GRP1_PERIPH_DTCM1
  - LL_AHB3_GRP1_PERIPH_DTCM2
  - LL_AHB3_GRP1_PERIPH_ITCM
  - LL_AHB3_GRP1_PERIPH_AXISRAM

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3ENR MDMAEN LL_C2_AHB3_GRP1_DisableClock
- AHB3ENR DMA2DEN LL_C2_AHB3_GRP1_DisableClock
- AHB3ENR JPGDECEN LL_C2_AHB3_GRP1_DisableClock
- AHB3ENR FMCEN LL_C2_AHB3_GRP1_DisableClock
- AHB3ENR QSPIEN LL_C2_AHB3_GRP1_DisableClock
- AHB3ENR SDMMC1EN LL_C2_AHB3_GRP1_DisableClock
- AHB3ENR FLASHEN LL_C2_AHB3_GRP1_DisableClock
- AHB3ENR DTCM1EN LL_C2_AHB3_GRP1_DisableClock
- AHB3ENR DTCM2EN LL_C2_AHB3_GRP1_DisableClock
- AHB3ENR ITCMEN LL_C2_AHB3_GRP1_DisableClock
- AHB3ENR AXISRAMEN LL_C2_AHB3_GRP1_DisableClock

LL_C2_AHB3_GRP1_EnableClockSleep

Function name

__STATIC_INLINE void LL_C2_AHB3_GRP1_EnableClockSleep (uint32_t Periphs)

Function description

Enable C2 AHB3 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_DMA2D
  - LL_AHB3_GRP1_PERIPH_JPGDEC (*)
  - LL_AHB3_GRP1_PERIPH_FMC
  - LL_AHB3_GRP1_PERIPH_QSPI (*)
  - LL_AHB3_GRP1_PERIPH_SDMMC1
  - LL_AHB3_GRP1_PERIPH_FLASH
  - LL_AHB3_GRP1_PERIPH_DTCM1
  - LL_AHB3_GRP1_PERIPH_DTCM2
  - LL_AHB3_GRP1_PERIPH_ITCM
  - LL_AHB3_GRP1_PERIPH_AXISRAM

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3LPENR MDMALPEN LL_C2_AHB3_GRP1_EnableClockSleep
- AHB3LPENR DMA2DLPEN LL_C2_AHB3_GRP1_EnableClockSleep
- AHB3LPENR JPGDECLPEN LL_C2_AHB3_GRP1_EnableClockSleep
- AHB3LPENR FMCLPEN LL_C2_AHB3_GRP1_EnableClockSleep
- AHB3LPENR QSPILPEN LL_C2_AHB3_GRP1_EnableClockSleep
- AHB3LPENR SDMMC1LPEN LL_C2_AHB3_GRP1_EnableClockSleep
- AHB3LPENR FLASHLPEN LL_C2_AHB3_GRP1_EnableClockSleep
- AHB3LPENR DTCM1LPEN LL_C2_AHB3_GRP1_EnableClockSleep
- AHB3LPENR DTCM2LPEN LL_C2_AHB3_GRP1_EnableClockSleep
- AHB3LPENR ITCMLPEN LL_C2_AHB3_GRP1_EnableClockSleep
- AHB3LPENR AXISRAMLPEN LL_C2_AHB3_GRP1_EnableClockSleep

LL_C2_AHB3_GRP1_DisableClockSleep

Function name

__STATIC_INLINE void LL_C2_AHB3_GRP1_DisableClockSleep (uint32_t Periphs)

Function description

Disable C2 AHB3 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_DMA2D
  - LL_AHB3_GRP1_PERIPH_JPGDEC (*)
  - LL_AHB3_GRP1_PERIPH_FMC
  - LL_AHB3_GRP1_PERIPH_QSPI (*)
  - LL_AHB3_GRP1_PERIPH_SDMMC1
  - LL_AHB3_GRP1_PERIPH_FLASH
  - LL_AHB3_GRP1_PERIPH_DTCM1
  - LL_AHB3_GRP1_PERIPH_DTCM2
  - LL_AHB3_GRP1_PERIPH_ITCM
  - LL_AHB3_GRP1_PERIPH_AXISRAM

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3LPENR MDMALPEN LL_C2_AHB3_GRP1_DisableClockSleep
- AHB3LPENR DMA2DLPEN LL_C2_AHB3_GRP1_DisableClockSleep
- AHB3LPENR JPGDECLPEN LL_C2_AHB3_GRP1_DisableClockSleep
- AHB3LPENR FMCLPEN LL_C2_AHB3_GRP1_DisableClockSleep
- AHB3LPENR QSPILPEN LL_C2_AHB3_GRP1_DisableClockSleep
- AHB3LPENR SDMMC1LPEN LL_C2_AHB3_GRP1_DisableClockSleep
- AHB3LPENR FLASHLPEN LL_C2_AHB3_GRP1_DisableClockSleep
- AHB3LPENR DTCM1LPEN LL_C2_AHB3_GRP1_DisableClockSleep
- AHB3LPENR DTCM2LPEN LL_C2_AHB3_GRP1_DisableClockSleep
- AHB3LPENR ITCMLPEN LL_C2_AHB3_GRP1_DisableClockSleep
- AHB3LPENR AXISRAMLPEN LL_C2_AHB3_GRP1_DisableClockSleep

LL_C2_AHB1_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_C2_AHB1_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable C2 AHB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_DMA1
  - LL_AHB1_GRP1_PERIPH_DMA2
  - LL_AHB1_GRP1_PERIPH_ADC12
  - LL_AHB1_GRP1_PERIPH_ART (*)
  - LL_AHB1_GRP1_PERIPH_ETH1MAC (*)
  - LL_AHB1_GRP1_PERIPH_ETH1TX (*)
  - LL_AHB1_GRP1_PERIPH_ETH1RX (*)
  - LL_AHB1_GRP1_PERIPH_USB1OTGHS
  - LL_AHB1_GRP1_PERIPH_USB1OTGHSULPI
  - LL_AHB1_GRP1_PERIPH_USB2OTGHS (*)
  - LL_AHB1_GRP1_PERIPH_USB2OTGHSULPI (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1ENR DMA1EN LL_C2_AHB1_GRP1_EnableClock
- AHB1ENR DMA2EN LL_C2_AHB1_GRP1_EnableClock
- AHB1ENR ADC12EN LL_C2_AHB1_GRP1_EnableClock
- AHB1ENR ARTEN LL_C2_AHB1_GRP1_EnableClock
- AHB1ENR ETH1MACEN LL_C2_AHB1_GRP1_EnableClock
- AHB1ENR ETH1TXEN LL_C2_AHB1_GRP1_EnableClock
- AHB1ENR ETH1RXEN LL_C2_AHB1_GRP1_EnableClock
- AHB1ENR USB1OTGHSEN LL_C2_AHB1_GRP1_EnableClock
- AHB1ENR USB1OTGHSULPIEN LL_C2_AHB1_GRP1_EnableClock
- AHB1ENR USB2OTGHSEN LL_C2_AHB1_GRP1_EnableClock
- AHB1ENR USB2OTGHSULPIEN LL_C2_AHB1_GRP1_EnableClock

LL_C2_AHB1_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_C2_AHB1_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if C2 AHB1 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_DMA1
  - LL_AHB1_GRP1_PERIPH_DMA2
  - LL_AHB1_GRP1_PERIPH_ADC12
  - LL_AHB1_GRP1_PERIPH_ART (*)
  - LL_AHB1_GRP1_PERIPH_ETH1MAC (*)
  - LL_AHB1_GRP1_PERIPH_ETH1TX (*)
  - LL_AHB1_GRP1_PERIPH_ETH1RX (*)
  - LL_AHB1_GRP1_PERIPH_USB1OTGHS
  - LL_AHB1_GRP1_PERIPH_USB1OTGHSULPI
  - LL_AHB1_GRP1_PERIPH_USB2OTGHS (*)
  - LL_AHB1_GRP1_PERIPH_USB2OTGHSULPI (*)

  (*) value not defined in all devices.

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- AHB1ENR DMA1EN LL_C2_AHB1_GRP1_IsEnabledClock
- AHB1ENR DMA2EN LL_C2_AHB1_GRP1_IsEnabledClock
- AHB1ENR ADC12EN LL_C2_AHB1_GRP1_IsEnabledClock
- AHB1ENR ARTEN LL_C2_AHB1_GRP1_IsEnabledClock
- AHB1ENR ETH1MACEN LL_C2_AHB1_GRP1_IsEnabledClock
- AHB1ENR ETH1TXEN LL_C2_AHB1_GRP1_IsEnabledClock
- AHB1ENR ETH1RXEN LL_C2_AHB1_GRP1_IsEnabledClock
- AHB1ENR USB1OTGHSEN LL_C2_AHB1_GRP1_IsEnabledClock
- AHB1ENR USB1OTGHSULPIEN LL_C2_AHB1_GRP1_IsEnabledClock
- AHB1ENR USB2OTGHSEN LL_C2_AHB1_GRP1_IsEnabledClock
- AHB1ENR USB2OTGHSULPIEN LL_C2_AHB1_GRP1_IsEnabledClock

LL_C2_AHB1_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_C2_AHB1_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable C2 AHB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_DMA1
  - LL_AHB1_GRP1_PERIPH_DMA2
  - LL_AHB1_GRP1_PERIPH_ADC12
  - LL_AHB1_GRP1_PERIPH_ART (*)
  - LL_AHB1_GRP1_PERIPH_ETH1MAC (*)
  - LL_AHB1_GRP1_PERIPH_ETH1TX (*)
  - LL_AHB1_GRP1_PERIPH_ETH1RX (*)
  - LL_AHB1_GRP1_PERIPH_USB1OTGHS
  - LL_AHB1_GRP1_PERIPH_USB1OTGHSULPI
  - LL_AHB1_GRP1_PERIPH_USB2OTGHS (*)
  - LL_AHB1_GRP1_PERIPH_USB2OTGHSULPI (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1ENR DMA1EN LL_C2_AHB1_GRP1_DisableClock
- AHB1ENR DMA2EN LL_C2_AHB1_GRP1_DisableClock
- AHB1ENR ADC12EN LL_C2_AHB1_GRP1_DisableClock
- AHB1ENR ARTEN LL_C2_AHB1_GRP1_DisableClock
- AHB1ENR ETH1MACEN LL_C2_AHB1_GRP1_DisableClock
- AHB1ENR ETH1TXEN LL_C2_AHB1_GRP1_DisableClock
- AHB1ENR ETH1RXEN LL_C2_AHB1_GRP1_DisableClock
- AHB1ENR USB1OTGHSEN LL_C2_AHB1_GRP1_DisableClock
- AHB1ENR USB1OTGHSULPIEN LL_C2_AHB1_GRP1_DisableClock
- AHB1ENR USB2OTGHSEN LL_C2_AHB1_GRP1_DisableClock
- AHB1ENR USB2OTGHSULPIEN LL_C2_AHB1_GRP1_DisableClock

LL_C2_AHB1_GRP1_EnableClockSleep

Function name

__STATIC_INLINE void LL_C2_AHB1_GRP1_EnableClockSleep (uint32_t Periphs)

Function description

Enable C2 AHB1 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_DMA1
  - LL_AHB1_GRP1_PERIPH_DMA2
  - LL_AHB1_GRP1_PERIPH_ADC12
  - LL_AHB1_GRP1_PERIPH_ART (*)
  - LL_AHB1_GRP1_PERIPH_ETH1MAC (*)
  - LL_AHB1_GRP1_PERIPH_ETH1TX (*)
  - LL_AHB1_GRP1_PERIPH_ETH1RX (*)
  - LL_AHB1_GRP1_PERIPH_USB1OTGHS
  - LL_AHB1_GRP1_PERIPH_USB1OTGHSULPI
  - LL_AHB1_GRP1_PERIPH_USB2OTGHS (*)
  - LL_AHB1_GRP1_PERIPH_USB2OTGHSULPI (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1LPENR DMA1LPEN LL_C2_AHB1_GRP1_EnableClockSleep
- AHB1LPENR DMA2LPEN LL_C2_AHB1_GRP1_EnableClockSleep
- AHB1LPENR ADC12LPEN LL_C2_AHB1_GRP1_EnableClockSleep
- AHB1LPENR ARTLPEN LL_C2_AHB1_GRP1_EnableClockSleep
- AHB1LPENR ETH1MACLPEN LL_C2_AHB1_GRP1_EnableClockSleep
- AHB1LPENR ETH1TXLPEN LL_C2_AHB1_GRP1_EnableClockSleep
- AHB1LPENR ETH1RXLPEN LL_C2_AHB1_GRP1_EnableClockSleep
- AHB1LPENR USB1OTGHSLPEN LL_C2_AHB1_GRP1_EnableClockSleep
- AHB1LPENR USB1OTGHSULPILPEN LL_C2_AHB1_GRP1_EnableClockSleep
- AHB1LPENR USB2OTGHSLPEN LL_C2_AHB1_GRP1_EnableClockSleep
- AHB1LPENR USB2OTGHSULPILPEN LL_C2_AHB1_GRP1_EnableClockSleep

LL_C2_AHB1_GRP1_DisableClockSleep

Function name

__STATIC_INLINE void LL_C2_AHB1_GRP1_DisableClockSleep (uint32_t Periphs)

Function description

Disable C2 AHB1 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_DMA1
  - LL_AHB1_GRP1_PERIPH_DMA2
  - LL_AHB1_GRP1_PERIPH_ADC12
  - LL_AHB1_GRP1_PERIPH_ART (*)
  - LL_AHB1_GRP1_PERIPH_ETH1MAC (*)
  - LL_AHB1_GRP1_PERIPH_ETH1TX (*)
  - LL_AHB1_GRP1_PERIPH_ETH1RX (*)
  - LL_AHB1_GRP1_PERIPH_USB1OTGHS
  - LL_AHB1_GRP1_PERIPH_USB1OTGHSULPI
  - LL_AHB1_GRP1_PERIPH_USB2OTGHS (*)
  - LL_AHB1_GRP1_PERIPH_USB2OTGHSULPI (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1LPENR DMA1LPEN LL_C2_AHB1_GRP1_DisableClockSleep
- AHB1LPENR DMA2LPEN LL_C2_AHB1_GRP1_DisableClockSleep
- AHB1LPENR ADC12LPEN LL_C2_AHB1_GRP1_DisableClockSleep
- AHB1LPENR ARTLPEN LL_C2_AHB1_GRP1_DisableClockSleep
- AHB1LPENR ETH1MACLPEN LL_C2_AHB1_GRP1_DisableClockSleep
- AHB1LPENR ETH1TXLPEN LL_C2_AHB1_GRP1_DisableClockSleep
- AHB1LPENR ETH1RXLPEN LL_C2_AHB1_GRP1_DisableClockSleep
- AHB1LPENR USB1OTGHSLPEN LL_C2_AHB1_GRP1_DisableClockSleep
- AHB1LPENR USB1OTGHSULPILPEN LL_C2_AHB1_GRP1_DisableClockSleep
- AHB1LPENR USB2OTGHSLPEN LL_C2_AHB1_GRP1_DisableClockSleep
- AHB1LPENR USB2OTGHSULPILPEN LL_C2_AHB1_GRP1_DisableClockSleep

LL_C2_AHB2_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_C2_AHB2_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable C2 AHB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_DCMI
  - LL_AHB2_GRP1_PERIPH_CRYP (*)
  - LL_AHB2_GRP1_PERIPH_HASH (*)
  - LL_AHB2_GRP1_PERIPH_RNG
  - LL_AHB2_GRP1_PERIPH_SDMMC2

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2ENR DCMIEN LL_C2_AHB2_GRP1_EnableClock
- AHB2ENR CRYPEN LL_C2_AHB2_GRP1_EnableClock
- AHB2ENR HASHEN LL_C2_AHB2_GRP1_EnableClock
- AHB2ENR RNGEN LL_C2_AHB2_GRP1_EnableClock
- AHB2ENR SDMMC2EN LL_C2_AHB2_GRP1_EnableClock

LL_C2_AHB2_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_C2_AHB2_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if C2 AHB2 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_DCMI
  - LL_AHB2_GRP1_PERIPH_CRYP (*)
  - LL_AHB2_GRP1_PERIPH_HASH (*)
  - LL_AHB2_GRP1_PERIPH_RNG
  - LL_AHB2_GRP1_PERIPH_SDMMC2

  (*) value not defined in all devices.

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- AHB2ENR DCMIEN LL_C2_AHB2_GRP1_IsEnabledClock
- AHB2ENR CRYPEN LL_C2_AHB2_GRP1_IsEnabledClock
- AHB2ENR HASHEN LL_C2_AHB2_GRP1_IsEnabledClock
- AHB2ENR RNGEN LL_C2_AHB2_GRP1_IsEnabledClock
- AHB2ENR SDMMC2EN LL_C2_AHB2_GRP1_IsEnabledClock

LL_C2_AHB2_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_C2_AHB2_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable C2 AHB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_DCMI
  - LL_AHB2_GRP1_PERIPH_CRYP (*)
  - LL_AHB2_GRP1_PERIPH_HASH (*)
  - LL_AHB2_GRP1_PERIPH_RNG
  - LL_AHB2_GRP1_PERIPH_SDMMC2

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2ENR DCMIEN LL_C2_AHB2_GRP1_DisableClock
- AHB2ENR CRYPEN LL_C2_AHB2_GRP1_DisableClock
- AHB2ENR HASHEN LL_C2_AHB2_GRP1_DisableClock
- AHB2ENR RNGEN LL_C2_AHB2_GRP1_DisableClock
- AHB2ENR SDMMC2EN LL_C2_AHB2_GRP1_DisableClock

LL_C2_AHB2_GRP1_EnableClockSleep

Function name

__STATIC_INLINE void LL_C2_AHB2_GRP1_EnableClockSleep (uint32_t Periphs)

Function description

Enable C2 AHB2 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_DCMI
  - LL_AHB2_GRP1_PERIPH_CRYP (*)
  - LL_AHB2_GRP1_PERIPH_HASH (*)
  - LL_AHB2_GRP1_PERIPH_RNG
  - LL_AHB2_GRP1_PERIPH_SDMMC2
  - LL_AHB2_GRP1_PERIPH_D2SRAM1
  - LL_AHB2_GRP1_PERIPH_D2SRAM2
  - LL_AHB2_GRP1_PERIPH_D2SRAM3 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2LPENR DCMILPEN LL_C2_AHB2_GRP1_EnableClockSleep
- AHB2LPENR CRYPLPEN LL_C2_AHB2_GRP1_EnableClockSleep
- AHB2LPENR HASHLPEN LL_C2_AHB2_GRP1_EnableClockSleep
- AHB2LPENR RNGLPEN LL_C2_AHB2_GRP1_EnableClockSleep
- AHB2LPENR SDMMC2LPEN LL_C2_AHB2_GRP1_EnableClockSleep
- AHB2LPENR D2SRAM1LPEN LL_C2_AHB2_GRP1_EnableClockSleep
- AHB2LPENR D2SRAM2LPEN LL_C2_AHB2_GRP1_EnableClockSleep
- AHB2LPENR D2SRAM3LPEN LL_C2_AHB2_GRP1_EnableClockSleep

LL_C2_AHB2_GRP1_DisableClockSleep

Function name

__STATIC_INLINE void LL_C2_AHB2_GRP1_DisableClockSleep (uint32_t Periphs)

Function description

Disable C2 AHB2 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_DCMI
  - LL_AHB2_GRP1_PERIPH_CRYP (*)
  - LL_AHB2_GRP1_PERIPH_HASH (*)
  - LL_AHB2_GRP1_PERIPH_RNG
  - LL_AHB2_GRP1_PERIPH_SDMMC2
  - LL_AHB2_GRP1_PERIPH_D2SRAM1
  - LL_AHB2_GRP1_PERIPH_D2SRAM2
  - LL_AHB2_GRP1_PERIPH_D2SRAM3 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2LPENR DCMILPEN LL_C2_AHB2_GRP1_DisableClockSleep
- AHB2LPENR CRYPLPEN LL_C2_AHB2_GRP1_DisableClockSleep
- AHB2LPENR HASHLPEN LL_C2_AHB2_GRP1_DisableClockSleep
- AHB2LPENR RNGLPEN LL_C2_AHB2_GRP1_DisableClockSleep
- AHB2LPENR SDMMC2LPEN LL_C2_AHB2_GRP1_DisableClockSleep
- AHB2LPENR D2SRAM1LPEN LL_C2_AHB2_GRP1_DisableClockSleep
- AHB2LPENR D2SRAM2LPEN LL_C2_AHB2_GRP1_DisableClockSleep
- AHB2LPENR D2SRAM3LPEN LL_C2_AHB2_GRP1_DisableClockSleep

LL_C2_AHB4_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_C2_AHB4_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable C2 AHB4 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_GPIOA
  - LL_AHB4_GRP1_PERIPH_GPIOB
  - LL_AHB4_GRP1_PERIPH_GPIOC
  - LL_AHB4_GRP1_PERIPH_GPIOD
  - LL_AHB4_GRP1_PERIPH_GPIOE
  - LL_AHB4_GRP1_PERIPH_GPIOF
  - LL_AHB4_GRP1_PERIPH_GPIOG
  - LL_AHB4_GRP1_PERIPH_GPIOH
  - LL_AHB4_GRP1_PERIPH_GPIOI (*)
  - LL_AHB4_GRP1_PERIPH_GPIOJ
  - LL_AHB4_GRP1_PERIPH_GPIOK
  - LL_AHB4_GRP1_PERIPH_CRC (*)
  - LL_AHB4_GRP1_PERIPH_BDMA
  - LL_AHB4_GRP1_PERIPH_ADC3 (*)
  - LL_AHB4_GRP1_PERIPH_HSEM (*)
  - LL_AHB4_GRP1_PERIPH_BKPRAM
  - LL_AHB4_GRP1_PERIPH_SRAM4

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB4ENR GPIOAEN LL_C2_AHB4_GRP1_EnableClock
- AHB4ENR GPIOBEN LL_C2_AHB4_GRP1_EnableClock
- AHB4ENR GPIOCEN LL_C2_AHB4_GRP1_EnableClock
- AHB4ENR GPIODEN LL_C2_AHB4_GRP1_EnableClock
- AHB4ENR GPIOEEN LL_C2_AHB4_GRP1_EnableClock
- AHB4ENR GPIOFEN LL_C2_AHB4_GRP1_EnableClock
- AHB4ENR GPIOGEN LL_C2_AHB4_GRP1_EnableClock
- AHB4ENR GPIOHEN LL_C2_AHB4_GRP1_EnableClock
- AHB4ENR GPIOIEN LL_C2_AHB4_GRP1_EnableClock
- AHB4ENR GPIOJEN LL_C2_AHB4_GRP1_EnableClock
- AHB4ENR GPIOKEN LL_C2_AHB4_GRP1_EnableClock
- AHB4ENR CRCEN LL_C2_AHB4_GRP1_EnableClock
- AHB4ENR BDMAEN LL_C2_AHB4_GRP1_EnableClock
- AHB4ENR ADC3EN LL_C2_AHB4_GRP1_EnableClock
- AHB4ENR HSEMEN LL_C2_AHB4_GRP1_EnableClock
- AHB4ENR BKPRAMEN LL_C2_AHB4_GRP1_EnableClock
- AHB4ENR SRAM4EN LL_C2_AHB4_GRP1_EnableClock

LL_C2_AHB4_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_C2_AHB4_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if C2 AHB4 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_GPIOA
  - LL_AHB4_GRP1_PERIPH_GPIOB
  - LL_AHB4_GRP1_PERIPH_GPIOC
  - LL_AHB4_GRP1_PERIPH_GPIOD
  - LL_AHB4_GRP1_PERIPH_GPIOE
  - LL_AHB4_GRP1_PERIPH_GPIOF
  - LL_AHB4_GRP1_PERIPH_GPIOG
  - LL_AHB4_GRP1_PERIPH_GPIOH
  - LL_AHB4_GRP1_PERIPH_GPIOI (*)
  - LL_AHB4_GRP1_PERIPH_GPIOJ
  - LL_AHB4_GRP1_PERIPH_GPIOK
  - LL_AHB4_GRP1_PERIPH_CRC (*)
  - LL_AHB4_GRP1_PERIPH_BDMA
  - LL_AHB4_GRP1_PERIPH_ADC3 (*)
  - LL_AHB4_GRP1_PERIPH_HSEM (*)
  - LL_AHB4_GRP1_PERIPH_BKPRAM
  - LL_AHB4_GRP1_PERIPH_SRAM4

  (*) value not defined in all devices.

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- AHB4ENR GPIOAEN LL_C2_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOBEN LL_C2_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOCEN LL_C2_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIODEN LL_C2_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOEEN LL_C2_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOFEN LL_C2_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOGEN LL_C2_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOHEN LL_C2_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOIEN LL_C2_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOJEN LL_C2_AHB4_GRP1_IsEnabledClock
- AHB4ENR GPIOKEN LL_C2_AHB4_GRP1_IsEnabledClock
- AHB4ENR CRCEN LL_C2_AHB4_GRP1_IsEnabledClock
- AHB4ENR BDMAEN LL_C2_AHB4_GRP1_IsEnabledClock
- AHB4ENR ADC3EN LL_C2_AHB4_GRP1_IsEnabledClock
- AHB4ENR HSEMEN LL_C2_AHB4_GRP1_IsEnabledClock
- AHB4ENR BKPRAMEN LL_C2_AHB4_GRP1_IsEnabledClock
- AHB4ENR SRAM4EN LL_C2_AHB4_GRP1_IsEnabledClock

LL_C2_AHB4_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_C2_AHB4_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable C2 AHB4 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_GPIOA
  - LL_AHB4_GRP1_PERIPH_GPIOB
  - LL_AHB4_GRP1_PERIPH_GPIOC
  - LL_AHB4_GRP1_PERIPH_GPIOD
  - LL_AHB4_GRP1_PERIPH_GPIOE
  - LL_AHB4_GRP1_PERIPH_GPIOF
  - LL_AHB4_GRP1_PERIPH_GPIOG
  - LL_AHB4_GRP1_PERIPH_GPIOH
  - LL_AHB4_GRP1_PERIPH_GPIOI (*)
  - LL_AHB4_GRP1_PERIPH_GPIOJ
  - LL_AHB4_GRP1_PERIPH_GPIOK
  - LL_AHB4_GRP1_PERIPH_CRC (*)
  - LL_AHB4_GRP1_PERIPH_BDMA
  - LL_AHB4_GRP1_PERIPH_ADC3 (*)
  - LL_AHB4_GRP1_PERIPH_HSEM (*)
  - LL_AHB4_GRP1_PERIPH_BKPRAM
  - LL_AHB4_GRP1_PERIPH_SRAM4

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB4ENR GPIOAEN LL_C2_AHB4_GRP1_DisableClock
- AHB4ENR GPIOBEN LL_C2_AHB4_GRP1_DisableClock
- AHB4ENR GPIOCEN LL_C2_AHB4_GRP1_DisableClock
- AHB4ENR GPIODEN LL_C2_AHB4_GRP1_DisableClock
- AHB4ENR GPIOEEN LL_C2_AHB4_GRP1_DisableClock
- AHB4ENR GPIOFEN LL_C2_AHB4_GRP1_DisableClock
- AHB4ENR GPIOGEN LL_C2_AHB4_GRP1_DisableClock
- AHB4ENR GPIOHEN LL_C2_AHB4_GRP1_DisableClock
- AHB4ENR GPIOIEN LL_C2_AHB4_GRP1_DisableClock
- AHB4ENR GPIOJEN LL_C2_AHB4_GRP1_DisableClock
- AHB4ENR GPIOKEN LL_C2_AHB4_GRP1_DisableClock
- AHB4ENR CRCEN LL_C2_AHB4_GRP1_DisableClock
- AHB4ENR BDMAEN LL_C2_AHB4_GRP1_DisableClock
- AHB4ENR ADC3EN LL_C2_AHB4_GRP1_DisableClock
- AHB4ENR HSEMEN LL_C2_AHB4_GRP1_DisableClock
- AHB4ENR BKPRAMEN LL_C2_AHB4_GRP1_DisableClock
- AHB4ENR SRAM4EN LL_C2_AHB4_GRP1_DisableClock

LL_C2_AHB4_GRP1_EnableClockSleep

Function name

__STATIC_INLINE void LL_C2_AHB4_GRP1_EnableClockSleep (uint32_t Periphs)

Function description

Enable C2 AHB4 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_GPIOA
  - LL_AHB4_GRP1_PERIPH_GPIOB
  - LL_AHB4_GRP1_PERIPH_GPIOC
  - LL_AHB4_GRP1_PERIPH_GPIOD
  - LL_AHB4_GRP1_PERIPH_GPIOE
  - LL_AHB4_GRP1_PERIPH_GPIOF
  - LL_AHB4_GRP1_PERIPH_GPIOG
  - LL_AHB4_GRP1_PERIPH_GPIOH
  - LL_AHB4_GRP1_PERIPH_GPIOI (*)
  - LL_AHB4_GRP1_PERIPH_GPIOJ
  - LL_AHB4_GRP1_PERIPH_GPIOK
  - LL_AHB4_GRP1_PERIPH_CRC (*)
  - LL_AHB4_GRP1_PERIPH_BDMA
  - LL_AHB4_GRP1_PERIPH_ADC3 (*)
  - LL_AHB4_GRP1_PERIPH_BKPRAM
  - LL_AHB4_GRP1_PERIPH_SRAM4

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB4LPENR GPIOALPEN LL_C2_AHB4_GRP1_EnableClockSleep
- AHB4LPENR GPIOBLPEN LL_C2_AHB4_GRP1_EnableClockSleep
- AHB4LPENR GPIOCLPEN LL_C2_AHB4_GRP1_EnableClockSleep
- AHB4LPENR GPIODLPEN LL_C2_AHB4_GRP1_EnableClockSleep
- AHB4LPENR GPIOELPEN LL_C2_AHB4_GRP1_EnableClockSleep
- AHB4LPENR GPIOFLPEN LL_C2_AHB4_GRP1_EnableClockSleep
- AHB4LPENR GPIOGLPEN LL_C2_AHB4_GRP1_EnableClockSleep
- AHB4LPENR GPIOHLPEN LL_C2_AHB4_GRP1_EnableClockSleep
- AHB4LPENR GPIOILPEN LL_C2_AHB4_GRP1_EnableClockSleep
- AHB4LPENR GPIOJLPEN LL_C2_AHB4_GRP1_EnableClockSleep
- AHB4LPENR GPIOKLPEN LL_C2_AHB4_GRP1_EnableClockSleep
- AHB4LPENR CRCLPEN LL_C2_AHB4_GRP1_EnableClockSleep
- AHB4LPENR BDMALPEN LL_C2_AHB4_GRP1_EnableClockSleep
- AHB4LPENR ADC3LPEN LL_C2_AHB4_GRP1_EnableClockSleep
- AHB4LPENR BKPRAMLPEN LL_C2_AHB4_GRP1_EnableClockSleep
- AHB4LPENR SRAM4LPEN LL_C2_AHB4_GRP1_EnableClockSleep

LL_C2_AHB4_GRP1_DisableClockSleep

Function name

__STATIC_INLINE void LL_C2_AHB4_GRP1_DisableClockSleep (uint32_t Periphs)

Function description

Disable C2 AHB4 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB4_GRP1_PERIPH_GPIOA
  - LL_AHB4_GRP1_PERIPH_GPIOB
  - LL_AHB4_GRP1_PERIPH_GPIOC
  - LL_AHB4_GRP1_PERIPH_GPIOD
  - LL_AHB4_GRP1_PERIPH_GPIOE
  - LL_AHB4_GRP1_PERIPH_GPIOF
  - LL_AHB4_GRP1_PERIPH_GPIOG
  - LL_AHB4_GRP1_PERIPH_GPIOH
  - LL_AHB4_GRP1_PERIPH_GPIOI (*)
  - LL_AHB4_GRP1_PERIPH_GPIOJ
  - LL_AHB4_GRP1_PERIPH_GPIOK
  - LL_AHB4_GRP1_PERIPH_CRC (*)
  - LL_AHB4_GRP1_PERIPH_BDMA
  - LL_AHB4_GRP1_PERIPH_ADC3 (*)
  - LL_AHB4_GRP1_PERIPH_BKPRAM
  - LL_AHB4_GRP1_PERIPH_SRAM4

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB4LPENR GPIOALPEN LL_C2_AHB4_GRP1_DisableClockSleep
- AHB4LPENR GPIOBLPEN LL_C2_AHB4_GRP1_DisableClockSleep
- AHB4LPENR GPIOCLPEN LL_C2_AHB4_GRP1_DisableClockSleep
- AHB4LPENR GPIODLPEN LL_C2_AHB4_GRP1_DisableClockSleep
- AHB4LPENR GPIOELPEN LL_C2_AHB4_GRP1_DisableClockSleep
- AHB4LPENR GPIOFLPEN LL_C2_AHB4_GRP1_DisableClockSleep
- AHB4LPENR GPIOGLPEN LL_C2_AHB4_GRP1_DisableClockSleep
- AHB4LPENR GPIOHLPEN LL_C2_AHB4_GRP1_DisableClockSleep
- AHB4LPENR GPIOILPEN LL_C2_AHB4_GRP1_DisableClockSleep
- AHB4LPENR GPIOJLPEN LL_C2_AHB4_GRP1_DisableClockSleep
- AHB4LPENR GPIOKLPEN LL_C2_AHB4_GRP1_DisableClockSleep
- AHB4LPENR CRCLPEN LL_C2_AHB4_GRP1_DisableClockSleep
- AHB4LPENR BDMALPEN LL_C2_AHB4_GRP1_DisableClockSleep
- AHB4LPENR ADC3LPEN LL_C2_AHB4_GRP1_DisableClockSleep
- AHB4LPENR BKPRAMLPEN LL_C2_AHB4_GRP1_DisableClockSleep
- AHB4LPENR SRAM4LPEN LL_C2_AHB4_GRP1_DisableClockSleep

LL_C2_APB3_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_C2_APB3_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable C2 APB3 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB3_GRP1_PERIPH_LTDC (*)
  - LL_APB3_GRP1_PERIPH_DSI (*)
  - LL_APB3_GRP1_PERIPH_WWDG1

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB3ENR LTDCEN LL_C2_APB3_GRP1_EnableClock
- APB3ENR DSIEN LL_C2_APB3_GRP1_EnableClock
- APB3ENR WWDG1EN LL_C2_APB3_GRP1_EnableClock

LL_C2_APB3_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_C2_APB3_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if C2 APB3 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB3_GRP1_PERIPH_LTDC (*)
  - LL_APB3_GRP1_PERIPH_DSI (*)
  - LL_APB3_GRP1_PERIPH_WWDG1

  (*) value not defined in all devices.

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- APB3ENR LTDCEN LL_C2_APB3_GRP1_IsEnabledClock
- APB3ENR DSIEN LL_C2_APB3_GRP1_IsEnabledClock
- APB3ENR WWDG1EN LL_C2_APB3_GRP1_IsEnabledClock

LL_C2_APB3_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_C2_APB3_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable C2 APB3 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB3_GRP1_PERIPH_LTDC (*)
  - LL_APB3_GRP1_PERIPH_DSI (*)
  - LL_APB3_GRP1_PERIPH_WWDG1

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB3ENR LTDCEN LL_C2_APB3_GRP1_DisableClock
- APB3ENR DSIEN LL_C2_APB3_GRP1_DisableClock
- APB3ENR WWDG1EN LL_C2_APB3_GRP1_DisableClock

LL_C2_APB3_GRP1_EnableClockSleep

Function name

__STATIC_INLINE void LL_C2_APB3_GRP1_EnableClockSleep (uint32_t Periphs)

Function description

Enable C2 APB3 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB3_GRP1_PERIPH_LTDC (*)
  - LL_APB3_GRP1_PERIPH_DSI (*)
  - LL_APB3_GRP1_PERIPH_WWDG1

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB3LPENR LTDCLPEN LL_C2_APB3_GRP1_EnableClockSleep
- APB3LPENR DSILPEN LL_C2_APB3_GRP1_EnableClockSleep
- APB3LPENR WWDG1LPEN LL_C2_APB3_GRP1_EnableClockSleep

LL_C2_APB3_GRP1_DisableClockSleep

Function name

__STATIC_INLINE void LL_C2_APB3_GRP1_DisableClockSleep (uint32_t Periphs)

Function description

Disable C2 APB3 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB3_GRP1_PERIPH_LTDC (*)
  - LL_APB3_GRP1_PERIPH_DSI (*)
  - LL_APB3_GRP1_PERIPH_WWDG1

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB3LPENR LTDCLPEN LL_C2_APB3_GRP1_DisableClockSleep
- APB3LPENR DSILPEN LL_C2_APB3_GRP1_DisableClockSleep
- APB3LPENR WWDG1LPEN LL_C2_APB3_GRP1_DisableClockSleep

LL_C2_APB1_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_C2_APB1_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable C2 APB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM12
  - LL_APB1_GRP1_PERIPH_TIM13
  - LL_APB1_GRP1_PERIPH_TIM14
  - LL_APB1_GRP1_PERIPH_LPTIM1
  - LL_APB1_GRP1_PERIPH_WWDG2 (*)
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_SPDIFRX
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4
  - LL_APB1_GRP1_PERIPH_UART5
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_CEC
  - LL_APB1_GRP1_PERIPH_DAC12
  - LL_APB1_GRP1_PERIPH_UART7
  - LL_APB1_GRP1_PERIPH_UART8

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1LENR TIM2EN LL_C2_APB1_GRP1_EnableClock
- APB1LENR TIM3EN LL_C2_APB1_GRP1_EnableClock
- APB1LENR TIM4EN LL_C2_APB1_GRP1_EnableClock
- APB1LENR TIM5EN LL_C2_APB1_GRP1_EnableClock
- APB1LENR TIM6EN LL_C2_APB1_GRP1_EnableClock
- APB1LENR TIM7EN LL_C2_APB1_GRP1_EnableClock
- APB1LENR TIM12EN LL_C2_APB1_GRP1_EnableClock
- APB1LENR TIM13EN LL_C2_APB1_GRP1_EnableClock
- APB1LENR TIM14EN LL_C2_APB1_GRP1_EnableClock
- APB1LENR LPTIM1EN LL_C2_APB1_GRP1_EnableClock
- APB1LENR WWDG2EN LL_C2_APB1_GRP1_EnableClock
- APB1LENR SPI2EN LL_C2_APB1_GRP1_EnableClock
- APB1LENR SPI3EN LL_C2_APB1_GRP1_EnableClock
- APB1LENR SPDIFRXEN LL_C2_APB1_GRP1_EnableClock
- APB1LENR USART2EN LL_C2_APB1_GRP1_EnableClock
- APB1LENR USART3EN LL_C2_APB1_GRP1_EnableClock
- APB1LENR UART4EN LL_C2_APB1_GRP1_EnableClock
- APB1LENR UART5EN LL_C2_APB1_GRP1_EnableClock
- APB1LENR I2C1EN LL_C2_APB1_GRP1_EnableClock
- APB1LENR I2C2EN LL_C2_APB1_GRP1_EnableClock
- APB1LENR I2C3EN LL_C2_APB1_GRP1_EnableClock
- APB1LENR CECEN LL_C2_APB1_GRP1_EnableClock
- APB1LENR DAC12EN LL_C2_APB1_GRP1_EnableClock
- APB1LENR UART7EN LL_C2_APB1_GRP1_EnableClock
- APB1LENR UART8EN LL_C2_APB1_GRP1_EnableClock

LL_C2_APB1_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_C2_APB1_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if C2 APB1 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM12
  - LL_APB1_GRP1_PERIPH_TIM13
  - LL_APB1_GRP1_PERIPH_TIM14
  - LL_APB1_GRP1_PERIPH_LPTIM1
  - LL_APB1_GRP1_PERIPH_WWDG2 (*)
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_SPDIFRX
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4
  - LL_APB1_GRP1_PERIPH_UART5
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_CEC
  - LL_APB1_GRP1_PERIPH_DAC12
  - LL_APB1_GRP1_PERIPH_UART7
  - LL_APB1_GRP1_PERIPH_UART8

  (*) value not defined in all devices.

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- APB1LENR TIM2EN LL_C2_APB1_GRP1_IsEnabledClock
- APB1LENR TIM3EN LL_C2_APB1_GRP1_IsEnabledClock
- APB1LENR TIM4EN LL_C2_APB1_GRP1_IsEnabledClock
- APB1LENR TIM5EN LL_C2_APB1_GRP1_IsEnabledClock
- APB1LENR TIM6EN LL_C2_APB1_GRP1_IsEnabledClock
- APB1LENR TIM7EN LL_C2_APB1_GRP1_IsEnabledClock
- APB1LENR TIM12EN LL_C2_APB1_GRP1_IsEnabledClock
- APB1LENR TIM13EN LL_C2_APB1_GRP1_IsEnabledClock
- APB1LENR TIM14EN LL_C2_APB1_GRP1_IsEnabledClock
- APB1LENR LPTIM1EN LL_C2_APB1_GRP1_IsEnabledClock
- APB1LENR WWDG2EN LL_C2_APB1_GRP1_IsEnabledClock
- APB1LENR SPI2EN LL_C2_APB1_GRP1_IsEnabledClock
- APB1LENR SPI3EN LL_C2_APB1_GRP1_IsEnabledClock
- APB1LENR SPDIFRXEN LL_C2_APB1_GRP1_IsEnabledClock
- APB1LENR USART2EN LL_C2_APB1_GRP1_IsEnabledClock
- APB1LENR USART3EN LL_C2_APB1_GRP1_IsEnabledClock
- APB1LENR UART4EN LL_C2_APB1_GRP1_IsEnabledClock
- APB1LENR UART5EN LL_C2_APB1_GRP1_IsEnabledClock
- APB1LENR I2C1EN LL_C2_APB1_GRP1_IsEnabledClock
- APB1LENR I2C2EN LL_C2_APB1_GRP1_IsEnabledClock
- APB1LENR I2C3EN LL_C2_APB1_GRP1_IsEnabledClock
- APB1LENR CECEN LL_C2_APB1_GRP1_IsEnabledClock
- APB1LENR DAC12EN LL_C2_APB1_GRP1_IsEnabledClock
- APB1LENR UART7EN LL_C2_APB1_GRP1_IsEnabledClock
- APB1LENR UART8EN LL_C2_APB1_GRP1_IsEnabledClock

LL_C2_APB1_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_C2_APB1_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable C2 APB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM12
  - LL_APB1_GRP1_PERIPH_TIM13
  - LL_APB1_GRP1_PERIPH_TIM14
  - LL_APB1_GRP1_PERIPH_LPTIM1
  - LL_APB1_GRP1_PERIPH_WWDG2 (*)
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_SPDIFRX
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4
  - LL_APB1_GRP1_PERIPH_UART5
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_CEC
  - LL_APB1_GRP1_PERIPH_DAC12
  - LL_APB1_GRP1_PERIPH_UART7
  - LL_APB1_GRP1_PERIPH_UART8

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1LENR TIM2EN LL_C2_APB1_GRP1_DisableClock
- APB1LENR TIM3EN LL_C2_APB1_GRP1_DisableClock
- APB1LENR TIM4EN LL_C2_APB1_GRP1_DisableClock
- APB1LENR TIM5EN LL_C2_APB1_GRP1_DisableClock
- APB1LENR TIM6EN LL_C2_APB1_GRP1_DisableClock
- APB1LENR TIM7EN LL_C2_APB1_GRP1_DisableClock
- APB1LENR TIM12EN LL_C2_APB1_GRP1_DisableClock
- APB1LENR TIM13EN LL_C2_APB1_GRP1_DisableClock
- APB1LENR TIM14EN LL_C2_APB1_GRP1_DisableClock
- APB1LENR LPTIM1EN LL_C2_APB1_GRP1_DisableClock
- APB1LENR WWDG2EN LL_C2_APB1_GRP1_DisableClock
- APB1LENR SPI2EN LL_C2_APB1_GRP1_DisableClock
- APB1LENR SPI3EN LL_C2_APB1_GRP1_DisableClock
- APB1LENR SPDIFRXEN LL_C2_APB1_GRP1_DisableClock
- APB1LENR USART2EN LL_C2_APB1_GRP1_DisableClock
- APB1LENR USART3EN LL_C2_APB1_GRP1_DisableClock
- APB1LENR UART4EN LL_C2_APB1_GRP1_DisableClock
- APB1LENR UART5EN LL_C2_APB1_GRP1_DisableClock
- APB1LENR I2C1EN LL_C2_APB1_GRP1_DisableClock
- APB1LENR I2C2EN LL_C2_APB1_GRP1_DisableClock
- APB1LENR I2C3EN LL_C2_APB1_GRP1_DisableClock
- APB1LENR CECEN LL_C2_APB1_GRP1_DisableClock
- APB1LENR DAC12EN LL_C2_APB1_GRP1_DisableClock
- APB1LENR UART7EN LL_C2_APB1_GRP1_DisableClock
- APB1LENR UART8EN LL_C2_APB1_GRP1_DisableClock

LL_C2_APB1_GRP1_EnableClockSleep

Function name

__STATIC_INLINE void LL_C2_APB1_GRP1_EnableClockSleep (uint32_t Periphs)

Function description

Enable C2 APB1 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM12
  - LL_APB1_GRP1_PERIPH_TIM13
  - LL_APB1_GRP1_PERIPH_TIM14
  - LL_APB1_GRP1_PERIPH_LPTIM1
  - LL_APB1_GRP1_PERIPH_WWDG2 (*)
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_SPDIFRX
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4
  - LL_APB1_GRP1_PERIPH_UART5
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_CEC
  - LL_APB1_GRP1_PERIPH_DAC12
  - LL_APB1_GRP1_PERIPH_UART7
  - LL_APB1_GRP1_PERIPH_UART8

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1LLPENR TIM2LPEN LL_C2_APB1_GRP1_EnableClockSleep
- APB1LLPENR TIM3LPEN LL_C2_APB1_GRP1_EnableClockSleep
- APB1LLPENR TIM4LPEN LL_C2_APB1_GRP1_EnableClockSleep
- APB1LLPENR TIM5LPEN LL_C2_APB1_GRP1_EnableClockSleep
- APB1LLPENR TIM6LPEN LL_C2_APB1_GRP1_EnableClockSleep
- APB1LLPENR TIM7LPEN LL_C2_APB1_GRP1_EnableClockSleep
- APB1LLPENR TIM12LPEN LL_C2_APB1_GRP1_EnableClockSleep
- APB1LLPENR TIM13LPEN LL_C2_APB1_GRP1_EnableClockSleep
- APB1LLPENR TIM14LPEN LL_C2_APB1_GRP1_EnableClockSleep
- APB1LLPENR LPTIM1LPEN LL_C2_APB1_GRP1_EnableClockSleep
- APB1LLPENR WWDG2LPEN LL_C2_APB1_GRP1_EnableClockSleep
- APB1LLPENR SPI2LPEN LL_C2_APB1_GRP1_EnableClockSleep
- APB1LLPENR SPI3LPEN LL_C2_APB1_GRP1_EnableClockSleep
- APB1LLPENR SPDIFRXLPEN LL_C2_APB1_GRP1_EnableClockSleep
- APB1LLPENR USART2LPEN LL_C2_APB1_GRP1_EnableClockSleep
- APB1LLPENR USART3LPEN LL_C2_APB1_GRP1_EnableClockSleep
- APB1LLPENR UART4LPEN LL_C2_APB1_GRP1_EnableClockSleep
- APB1LLPENR UART5LPEN LL_C2_APB1_GRP1_EnableClockSleep
- APB1LLPENR I2C1LPEN LL_C2_APB1_GRP1_EnableClockSleep
- APB1LLPENR I2C2LPEN LL_C2_APB1_GRP1_EnableClockSleep
- APB1LLPENR I2C3LPEN LL_C2_APB1_GRP1_EnableClockSleep
- APB1LLPENR CECLPEN LL_C2_APB1_GRP1_EnableClockSleep
- APB1LLPENR DAC12LPEN LL_C2_APB1_GRP1_EnableClockSleep
- APB1LLPENR UART7LPEN LL_C2_APB1_GRP1_EnableClockSleep
- APB1LLPENR UART8LPEN LL_C2_APB1_GRP1_EnableClockSleep

LL_C2_APB1_GRP1_DisableClockSleep

Function name

__STATIC_INLINE void LL_C2_APB1_GRP1_DisableClockSleep (uint32_t Periphs)

Function description

Disable C2 APB1 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_TIM12
  - LL_APB1_GRP1_PERIPH_TIM13
  - LL_APB1_GRP1_PERIPH_TIM14
  - LL_APB1_GRP1_PERIPH_LPTIM1
  - LL_APB1_GRP1_PERIPH_WWDG2 (*)
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_SPDIFRX
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4
  - LL_APB1_GRP1_PERIPH_UART5
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_I2C3
  - LL_APB1_GRP1_PERIPH_CEC
  - LL_APB1_GRP1_PERIPH_DAC12
  - LL_APB1_GRP1_PERIPH_UART7
  - LL_APB1_GRP1_PERIPH_UART8

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1LLPENR TIM2LPEN LL_C2_APB1_GRP1_DisableClockSleep
- APB1LLPENR TIM3LPEN LL_C2_APB1_GRP1_DisableClockSleep
- APB1LLPENR TIM4LPEN LL_C2_APB1_GRP1_DisableClockSleep
- APB1LLPENR TIM5LPEN LL_C2_APB1_GRP1_DisableClockSleep
- APB1LLPENR TIM6LPEN LL_C2_APB1_GRP1_DisableClockSleep
- APB1LLPENR TIM7LPEN LL_C2_APB1_GRP1_DisableClockSleep
- APB1LLPENR TIM12LPEN LL_C2_APB1_GRP1_DisableClockSleep
- APB1LLPENR TIM13LPEN LL_C2_APB1_GRP1_DisableClockSleep
- APB1LLPENR TIM14LPEN LL_C2_APB1_GRP1_DisableClockSleep
- APB1LLPENR LPTIM1LPEN LL_C2_APB1_GRP1_DisableClockSleep
- APB1LLPENR WWDG2LPEN LL_C2_APB1_GRP1_DisableClockSleep
- APB1LLPENR SPI2LPEN LL_C2_APB1_GRP1_DisableClockSleep
- APB1LLPENR SPI3LPEN LL_C2_APB1_GRP1_DisableClockSleep
- APB1LLPENR SPDIFRXLPEN LL_C2_APB1_GRP1_DisableClockSleep
- APB1LLPENR USART2LPEN LL_C2_APB1_GRP1_DisableClockSleep
- APB1LLPENR USART3LPEN LL_C2_APB1_GRP1_DisableClockSleep
- APB1LLPENR UART4LPEN LL_C2_APB1_GRP1_DisableClockSleep
- APB1LLPENR UART5LPEN LL_C2_APB1_GRP1_DisableClockSleep
- APB1LLPENR I2C1LPEN LL_C2_APB1_GRP1_DisableClockSleep
- APB1LLPENR I2C2LPEN LL_C2_APB1_GRP1_DisableClockSleep
- APB1LLPENR I2C3LPEN LL_C2_APB1_GRP1_DisableClockSleep
- APB1LLPENR CECLPEN LL_C2_APB1_GRP1_DisableClockSleep
- APB1LLPENR DAC12LPEN LL_C2_APB1_GRP1_DisableClockSleep
- APB1LLPENR UART7LPEN LL_C2_APB1_GRP1_DisableClockSleep
- APB1LLPENR UART8LPEN LL_C2_APB1_GRP1_DisableClockSleep

LL_C2_APB1_GRP2_EnableClock

Function name

__STATIC_INLINE void LL_C2_APB1_GRP2_EnableClock (uint32_t Periphs)

Function description

Enable C2 APB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_CRS
  - LL_APB1_GRP2_PERIPH_SWPMI1
  - LL_APB1_GRP2_PERIPH_OPAMP
  - LL_APB1_GRP2_PERIPH_MDIOS
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_TIM23 (*)
  - LL_APB1_GRP2_PERIPH_TIM24 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1HENR CRSEN LL_C2_APB1_GRP2_EnableClock
- APB1HENR SWPMIEN LL_C2_APB1_GRP2_EnableClock
- APB1HENR OPAMPEN LL_C2_APB1_GRP2_EnableClock
- APB1HENR MDIOSEN LL_C2_APB1_GRP2_EnableClock
- APB1HENR FDCANEN LL_C2_APB1_GRP2_EnableClock

LL_C2_APB1_GRP2_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_C2_APB1_GRP2_IsEnabledClock (uint32_t Periphs)

Function description

Check if C2 APB1 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_CRS
  - LL_APB1_GRP2_PERIPH_SWPMI1
  - LL_APB1_GRP2_PERIPH_OPAMP
  - LL_APB1_GRP2_PERIPH_MDIOS
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_TIM23 (*)
  - LL_APB1_GRP2_PERIPH_TIM24 (*)

  (*) value not defined in all devices.

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- APB1HENR CRSEN LL_C2_APB1_GRP2_IsEnabledClock
- APB1HENR SWPMIEN LL_C2_APB1_GRP2_IsEnabledClock
- APB1HENR OPAMPEN LL_C2_APB1_GRP2_IsEnabledClock
- APB1HENR MDIOSEN LL_C2_APB1_GRP2_IsEnabledClock
- APB1HENR FDCANEN LL_C2_APB1_GRP2_IsEnabledClock

LL_C2_APB1_GRP2_DisableClock

Function name

__STATIC_INLINE void LL_C2_APB1_GRP2_DisableClock (uint32_t Periphs)

Function description

Disable C2 APB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_CRS
  - LL_APB1_GRP2_PERIPH_SWPMI1
  - LL_APB1_GRP2_PERIPH_OPAMP
  - LL_APB1_GRP2_PERIPH_MDIOS
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_TIM23 (*)
  - LL_APB1_GRP2_PERIPH_TIM24 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1HENR CRSEN LL_C2_APB1_GRP2_DisableClock
- APB1HENR SWPMIEN LL_C2_APB1_GRP2_DisableClock
- APB1HENR OPAMPEN LL_C2_APB1_GRP2_DisableClock
- APB1HENR MDIOSEN LL_C2_APB1_GRP2_DisableClock
- APB1HENR FDCANEN LL_C2_APB1_GRP2_DisableClock

LL_C2_APB1_GRP2_EnableClockSleep

Function name

__STATIC_INLINE void LL_C2_APB1_GRP2_EnableClockSleep (uint32_t Periphs)

Function description

Enable C2 APB1 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_CRS
  - LL_APB1_GRP2_PERIPH_SWPMI1
  - LL_APB1_GRP2_PERIPH_OPAMP
  - LL_APB1_GRP2_PERIPH_MDIOS
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_TIM23 (*)
  - LL_APB1_GRP2_PERIPH_TIM24 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1HLPENR CRSLPEN LL_C2_APB1_GRP2_EnableClockSleep
- APB1HLPENR SWPMILPEN LL_C2_APB1_GRP2_EnableClockSleep
- APB1HLPENR OPAMPLPEN LL_C2_APB1_GRP2_EnableClockSleep
- APB1HLPENR MDIOSLPEN LL_C2_APB1_GRP2_EnableClockSleep
- APB1HLPENR FDCANLPEN LL_C2_APB1_GRP2_EnableClockSleep

LL_C2_APB1_GRP2_DisableClockSleep

Function name

__STATIC_INLINE void LL_C2_APB1_GRP2_DisableClockSleep (uint32_t Periphs)

Function description

Disable C2 APB1 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_CRS
  - LL_APB1_GRP2_PERIPH_SWPMI1
  - LL_APB1_GRP2_PERIPH_OPAMP
  - LL_APB1_GRP2_PERIPH_MDIOS
  - LL_APB1_GRP2_PERIPH_FDCAN
  - LL_APB1_GRP2_PERIPH_TIM23 (*)
  - LL_APB1_GRP2_PERIPH_TIM24 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1HLPENR CRSLPEN LL_C2_APB1_GRP2_DisableClockSleep
- APB1HLPENR SWPMILPEN LL_C2_APB1_GRP2_DisableClockSleep
- APB1HLPENR OPAMPLPEN LL_C2_APB1_GRP2_DisableClockSleep
- APB1HLPENR MDIOSLPEN LL_C2_APB1_GRP2_DisableClockSleep
- APB1HLPENR FDCANLPEN LL_C2_APB1_GRP2_DisableClockSleep

LL_C2_APB2_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_C2_APB2_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable C2 APB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_USART6
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_SPI4
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_SPI5
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_SAI2 (*)
  - LL_APB2_GRP1_PERIPH_SAI3 (*)
  - LL_APB2_GRP1_PERIPH_DFSDM1
  - LL_APB2_GRP1_PERIPH_HRTIM (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2ENR TIM1EN LL_C2_APB2_GRP1_EnableClock
- APB2ENR TIM8EN LL_C2_APB2_GRP1_EnableClock
- APB2ENR USART1EN LL_C2_APB2_GRP1_EnableClock
- APB2ENR USART6EN LL_C2_APB2_GRP1_EnableClock
- APB2ENR SPI1EN LL_C2_APB2_GRP1_EnableClock
- APB2ENR SPI4EN LL_C2_APB2_GRP1_EnableClock
- APB2ENR TIM15EN LL_C2_APB2_GRP1_EnableClock
- APB2ENR TIM16EN LL_C2_APB2_GRP1_EnableClock
- APB2ENR TIM17EN LL_C2_APB2_GRP1_EnableClock
- APB2ENR SPI5EN LL_C2_APB2_GRP1_EnableClock
- APB2ENR SAI1EN LL_C2_APB2_GRP1_EnableClock
- APB2ENR SAI2EN LL_C2_APB2_GRP1_EnableClock
- APB2ENR SAI3EN LL_C2_APB2_GRP1_EnableClock
- APB2ENR DFSDM1EN LL_C2_APB2_GRP1_EnableClock
- APB2ENR HRTIMEN LL_C2_APB2_GRP1_EnableClock

LL_C2_APB2_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_C2_APB2_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if C2 APB2 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_USART6
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_SPI4
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_SPI5
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_SAI2 (*)
  - LL_APB2_GRP1_PERIPH_SAI3 (*)
  - LL_APB2_GRP1_PERIPH_DFSDM1
  - LL_APB2_GRP1_PERIPH_HRTIM (*)

  (*) value not defined in all devices.

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- APB2ENR TIM1EN LL_C2_APB2_GRP1_IsEnabledClock
- APB2ENR TIM8EN LL_C2_APB2_GRP1_IsEnabledClock
- APB2ENR USART1EN LL_C2_APB2_GRP1_IsEnabledClock
- APB2ENR USART6EN LL_C2_APB2_GRP1_IsEnabledClock
- APB2ENR SPI1EN LL_C2_APB2_GRP1_IsEnabledClock
- APB2ENR SPI4EN LL_C2_APB2_GRP1_IsEnabledClock
- APB2ENR TIM15EN LL_C2_APB2_GRP1_IsEnabledClock
- APB2ENR TIM16EN LL_C2_APB2_GRP1_IsEnabledClock
- APB2ENR TIM17EN LL_C2_APB2_GRP1_IsEnabledClock
- APB2ENR SPI5EN LL_C2_APB2_GRP1_IsEnabledClock
- APB2ENR SAI1EN LL_C2_APB2_GRP1_IsEnabledClock
- APB2ENR SAI2EN LL_C2_APB2_GRP1_IsEnabledClock
- APB2ENR SAI3EN LL_C2_APB2_GRP1_IsEnabledClock
- APB2ENR DFSDM1EN LL_C2_APB2_GRP1_IsEnabledClock
- APB2ENR HRTIMEN LL_C2_APB2_GRP1_IsEnabledClock

LL_C2_APB2_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_C2_APB2_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable C2 APB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_USART6
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_SPI4
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_SPI5
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_SAI2 (*)
  - LL_APB2_GRP1_PERIPH_SAI3 (*)
  - LL_APB2_GRP1_PERIPH_DFSDM1
  - LL_APB2_GRP1_PERIPH_HRTIM (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2ENR TIM1EN LL_C2_APB2_GRP1_DisableClock
- APB2ENR TIM8EN LL_C2_APB2_GRP1_DisableClock
- APB2ENR USART1EN LL_C2_APB2_GRP1_DisableClock
- APB2ENR USART6EN LL_C2_APB2_GRP1_DisableClock
- APB2ENR SPI1EN LL_C2_APB2_GRP1_DisableClock
- APB2ENR SPI4EN LL_C2_APB2_GRP1_DisableClock
- APB2ENR TIM15EN LL_C2_APB2_GRP1_DisableClock
- APB2ENR TIM16EN LL_C2_APB2_GRP1_DisableClock
- APB2ENR TIM17EN LL_C2_APB2_GRP1_DisableClock
- APB2ENR SPI5EN LL_C2_APB2_GRP1_DisableClock
- APB2ENR SAI1EN LL_C2_APB2_GRP1_DisableClock
- APB2ENR SAI2EN LL_C2_APB2_GRP1_DisableClock
- APB2ENR SAI3EN LL_C2_APB2_GRP1_DisableClock
- APB2ENR DFSDM1EN LL_C2_APB2_GRP1_DisableClock
- APB2ENR HRTIMEN LL_C2_APB2_GRP1_DisableClock

LL_C2_APB2_GRP1_EnableClockSleep

Function name

__STATIC_INLINE void LL_C2_APB2_GRP1_EnableClockSleep (uint32_t Periphs)

Function description

Enable C2 APB2 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_USART6
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_SPI4
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_SPI5
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_SAI2 (*)
  - LL_APB2_GRP1_PERIPH_SAI3 (*)
  - LL_APB2_GRP1_PERIPH_DFSDM1
  - LL_APB2_GRP1_PERIPH_HRTIM (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2LPENR TIM1LPEN LL_C2_APB2_GRP1_EnableClockSleep
- APB2LPENR TIM8LPEN LL_C2_APB2_GRP1_EnableClockSleep
- APB2LPENR USART1LPEN LL_C2_APB2_GRP1_EnableClockSleep
- APB2LPENR USART6LPEN LL_C2_APB2_GRP1_EnableClockSleep
- APB2LPENR SPI1LPEN LL_C2_APB2_GRP1_EnableClockSleep
- APB2LPENR SPI4LPEN LL_C2_APB2_GRP1_EnableClockSleep
- APB2LPENR TIM15LPEN LL_C2_APB2_GRP1_EnableClockSleep
- APB2LPENR TIM16LPEN LL_C2_APB2_GRP1_EnableClockSleep
- APB2LPENR TIM17LPEN LL_C2_APB2_GRP1_EnableClockSleep
- APB2LPENR SPI5LPEN LL_C2_APB2_GRP1_EnableClockSleep
- APB2LPENR SAI1LPEN LL_C2_APB2_GRP1_EnableClockSleep
- APB2LPENR SAI2LPEN LL_C2_APB2_GRP1_EnableClockSleep
- APB2LPENR SAI3LPEN LL_C2_APB2_GRP1_EnableClockSleep
- APB2LPENR DFSDM1LPEN LL_C2_APB2_GRP1_EnableClockSleep
- APB2LPENR HRTIMLPEN LL_C2_APB2_GRP1_EnableClockSleep

LL_C2_APB2_GRP1_DisableClockSleep

Function name

__STATIC_INLINE void LL_C2_APB2_GRP1_DisableClockSleep (uint32_t Periphs)

Function description

Disable C2 APB2 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_USART6
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_SPI4
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_SPI5
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_SAI2 (*)
  - LL_APB2_GRP1_PERIPH_SAI3 (*)
  - LL_APB2_GRP1_PERIPH_DFSDM1
  - LL_APB2_GRP1_PERIPH_HRTIM (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2LPENR TIM1LPEN LL_C2_APB2_GRP1_DisableClockSleep
- APB2LPENR TIM8LPEN LL_C2_APB2_GRP1_DisableClockSleep
- APB2LPENR USART1LPEN LL_C2_APB2_GRP1_DisableClockSleep
- APB2LPENR USART6LPEN LL_C2_APB2_GRP1_DisableClockSleep
- APB2LPENR SPI1LPEN LL_C2_APB2_GRP1_DisableClockSleep
- APB2LPENR SPI4LPEN LL_C2_APB2_GRP1_DisableClockSleep
- APB2LPENR TIM15LPEN LL_C2_APB2_GRP1_DisableClockSleep
- APB2LPENR TIM16LPEN LL_C2_APB2_GRP1_DisableClockSleep
- APB2LPENR TIM17LPEN LL_C2_APB2_GRP1_DisableClockSleep
- APB2LPENR SPI5LPEN LL_C2_APB2_GRP1_DisableClockSleep
- APB2LPENR SAI1LPEN LL_C2_APB2_GRP1_DisableClockSleep
- APB2LPENR SAI2LPEN LL_C2_APB2_GRP1_DisableClockSleep
- APB2LPENR SAI3LPEN LL_C2_APB2_GRP1_DisableClockSleep
- APB2LPENR DFSDM1LPEN LL_C2_APB2_GRP1_DisableClockSleep
- APB2LPENR HRTIMLPEN LL_C2_APB2_GRP1_DisableClockSleep

LL_C2_APB4_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_C2_APB4_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable C2 APB4 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP1_PERIPH_SYSCFG
  - LL_APB4_GRP1_PERIPH_LPUART1
  - LL_APB4_GRP1_PERIPH_SPI6
  - LL_APB4_GRP1_PERIPH_I2C4
  - LL_APB4_GRP1_PERIPH_LPTIM2
  - LL_APB4_GRP1_PERIPH_LPTIM3
  - LL_APB4_GRP1_PERIPH_LPTIM4 (*)
  - LL_APB4_GRP1_PERIPH_LPTIM5 (*)
  - LL_APB4_GRP1_PERIPH_COMP12
  - LL_APB4_GRP1_PERIPH_VREF
  - LL_APB4_GRP1_PERIPH_RTCAPB
  - LL_APB4_GRP1_PERIPH_SAI4 (*)

  (*) value not defined in all devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB4ENR SYSCFGEN LL_C2_APB4_GRP1_EnableClock
- APB4ENR LPUART1EN LL_C2_APB4_GRP1_EnableClock
- APB4ENR SPI6EN LL_C2_APB4_GRP1_EnableClock
- APB4ENR I2C4EN LL_C2_APB4_GRP1_EnableClock
- APB4ENR LPTIM2EN LL_C2_APB4_GRP1_EnableClock
- APB4ENR LPTIM3EN LL_C2_APB4_GRP1_EnableClock
- APB4ENR LPTIM4EN LL_C2_APB4_GRP1_EnableClock
- APB4ENR LPTIM5EN LL_C2_APB4_GRP1_EnableClock
- APB4ENR COMP12EN LL_C2_APB4_GRP1_EnableClock
- APB4ENR VREFEN LL_C2_APB4_GRP1_EnableClock
- APB4ENR RTCAPBEN LL_C2_APB4_GRP1_EnableClock
- APB4ENR SAI4EN LL_C2_APB4_GRP1_EnableClock

LL_C2_APB4_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_C2_APB4_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if C2 APB4 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP1_PERIPH_SYSCFG
  - LL_APB4_GRP1_PERIPH_LPUART1
  - LL_APB4_GRP1_PERIPH_SPI6
  - LL_APB4_GRP1_PERIPH_I2C4
  - LL_APB4_GRP1_PERIPH_LPTIM2
  - LL_APB4_GRP1_PERIPH_LPTIM3
  - LL_APB4_GRP1_PERIPH_LPTIM4 (*)
  - LL_APB4_GRP1_PERIPH_LPTIM5 (*)
  - LL_APB4_GRP1_PERIPH_COMP12
  - LL_APB4_GRP1_PERIPH_VREF
  - LL_APB4_GRP1_PERIPH_RTCAPB
  - LL_APB4_GRP1_PERIPH_SAI4 (*)

  (*) value not defined in all devices

Return values

- **uint32_t:**

Reference Manual to LL API cross reference:

- APB4ENR SYSCFGEN LL_C2_APB4_GRP1_IsEnabledClock
- APB4ENR LPUART1EN LL_C2_APB4_GRP1_IsEnabledClock
- APB4ENR SPI6EN LL_C2_APB4_GRP1_IsEnabledClock
- APB4ENR I2C4EN LL_C2_APB4_GRP1_IsEnabledClock
- APB4ENR LPTIM2EN LL_C2_APB4_GRP1_IsEnabledClock
- APB4ENR LPTIM3EN LL_C2_APB4_GRP1_IsEnabledClock
- APB4ENR LPTIM4EN LL_C2_APB4_GRP1_IsEnabledClock
- APB4ENR LPTIM5EN LL_C2_APB4_GRP1_IsEnabledClock
- APB4ENR COMP12EN LL_C2_APB4_GRP1_IsEnabledClock
- APB4ENR VREFEN LL_C2_APB4_GRP1_IsEnabledClock
- APB4ENR RTCAPBEN LL_C2_APB4_GRP1_IsEnabledClock
- APB4ENR SAI4EN LL_C2_APB4_GRP1_IsEnabledClock

LL_C2_APB4_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_C2_APB4_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable C2 APB4 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP1_PERIPH_SYSCFG
  - LL_APB4_GRP1_PERIPH_LPUART1
  - LL_APB4_GRP1_PERIPH_SPI6
  - LL_APB4_GRP1_PERIPH_I2C4
  - LL_APB4_GRP1_PERIPH_LPTIM2
  - LL_APB4_GRP1_PERIPH_LPTIM3
  - LL_APB4_GRP1_PERIPH_LPTIM4 (*)
  - LL_APB4_GRP1_PERIPH_LPTIM5 (*)
  - LL_APB4_GRP1_PERIPH_COMP12
  - LL_APB4_GRP1_PERIPH_VREF
  - LL_APB4_GRP1_PERIPH_RTCAPB
  - LL_APB4_GRP1_PERIPH_SAI4 (*)

  (*) value not defined in all devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB4ENR SYSCFGEN LL_C2_APB4_GRP1_DisableClock
- APB4ENR LPUART1EN LL_C2_APB4_GRP1_DisableClock
- APB4ENR SPI6EN LL_C2_APB4_GRP1_DisableClock
- APB4ENR I2C4EN LL_C2_APB4_GRP1_DisableClock
- APB4ENR LPTIM2EN LL_C2_APB4_GRP1_DisableClock
- APB4ENR LPTIM3EN LL_C2_APB4_GRP1_DisableClock
- APB4ENR LPTIM4EN LL_C2_APB4_GRP1_DisableClock
- APB4ENR LPTIM5EN LL_C2_APB4_GRP1_DisableClock
- APB4ENR COMP12EN LL_C2_APB4_GRP1_DisableClock
- APB4ENR VREFEN LL_C2_APB4_GRP1_DisableClock
- APB4ENR RTCAPBEN LL_C2_APB4_GRP1_DisableClock
- APB4ENR SAI4EN LL_C2_APB4_GRP1_DisableClock

LL_C2_APB4_GRP1_EnableClockSleep

Function name

__STATIC_INLINE void LL_C2_APB4_GRP1_EnableClockSleep (uint32_t Periphs)

Function description

Enable C2 APB4 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP1_PERIPH_SYSCFG
  - LL_APB4_GRP1_PERIPH_LPUART1
  - LL_APB4_GRP1_PERIPH_SPI6
  - LL_APB4_GRP1_PERIPH_I2C4
  - LL_APB4_GRP1_PERIPH_LPTIM2
  - LL_APB4_GRP1_PERIPH_LPTIM3
  - LL_APB4_GRP1_PERIPH_LPTIM4 (*)
  - LL_APB4_GRP1_PERIPH_LPTIM5 (*)
  - LL_APB4_GRP1_PERIPH_COMP12
  - LL_APB4_GRP1_PERIPH_VREF
  - LL_APB4_GRP1_PERIPH_RTCAPB
  - LL_APB4_GRP1_PERIPH_SAI4 (*)

  (*) value not defined in all devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB4LPENR SYSCFGLPEN LL_C2_APB4_GRP1_EnableClockSleep
- APB4LPENR LPUART1LPEN LL_C2_APB4_GRP1_EnableClockSleep
- APB4LPENR SPI6LPEN LL_C2_APB4_GRP1_EnableClockSleep
- APB4LPENR I2C4LPEN LL_C2_APB4_GRP1_EnableClockSleep
- APB4LPENR LPTIM2LPEN LL_C2_APB4_GRP1_EnableClockSleep
- APB4LPENR LPTIM3LPEN LL_C2_APB4_GRP1_EnableClockSleep
- APB4LPENR LPTIM4LPEN LL_C2_APB4_GRP1_EnableClockSleep
- APB4LPENR LPTIM5LPEN LL_C2_APB4_GRP1_EnableClockSleep
- APB4LPENR COMP12LPEN LL_C2_APB4_GRP1_EnableClockSleep
- APB4LPENR VREFLPEN LL_C2_APB4_GRP1_EnableClockSleep
- APB4LPENR RTCAPBLPEN LL_C2_APB4_GRP1_EnableClockSleep
- APB4LPENR SAI4LPEN LL_C2_APB4_GRP1_EnableClockSleep

LL_C2_APB4_GRP1_DisableClockSleep

Function name

__STATIC_INLINE void LL_C2_APB4_GRP1_DisableClockSleep (uint32_t Periphs)

Function description

Disable C2 APB4 peripherals clock during Low Power (Sleep) mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB4_GRP1_PERIPH_SYSCFG
  - LL_APB4_GRP1_PERIPH_LPUART1
  - LL_APB4_GRP1_PERIPH_SPI6
  - LL_APB4_GRP1_PERIPH_I2C4
  - LL_APB4_GRP1_PERIPH_LPTIM2
  - LL_APB4_GRP1_PERIPH_LPTIM3
  - LL_APB4_GRP1_PERIPH_LPTIM4 (*)
  - LL_APB4_GRP1_PERIPH_LPTIM5 (*)
  - LL_APB4_GRP1_PERIPH_COMP12
  - LL_APB4_GRP1_PERIPH_VREF
  - LL_APB4_GRP1_PERIPH_RTCAPB
  - LL_APB4_GRP1_PERIPH_SAI4 (*)

  (*) value not defined in all devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB4LPENR SYSCFGLPEN LL_C2_APB4_GRP1_DisableClockSleep
- APB4LPENR LPUART1LPEN LL_C2_APB4_GRP1_DisableClockSleep
- APB4LPENR SPI6LPEN LL_C2_APB4_GRP1_DisableClockSleep
- APB4LPENR I2C4LPEN LL_C2_APB4_GRP1_DisableClockSleep
- APB4LPENR LPTIM2LPEN LL_C2_APB4_GRP1_DisableClockSleep
- APB4LPENR LPTIM3LPEN LL_C2_APB4_GRP1_DisableClockSleep
- APB4LPENR LPTIM4LPEN LL_C2_APB4_GRP1_DisableClockSleep
- APB4LPENR LPTIM5LPEN LL_C2_APB4_GRP1_DisableClockSleep
- APB4LPENR COMP12LPEN LL_C2_APB4_GRP1_DisableClockSleep
- APB4LPENR VREFLPEN LL_C2_APB4_GRP1_DisableClockSleep
- APB4LPENR RTCAPBLPEN LL_C2_APB4_GRP1_DisableClockSleep
- APB4LPENR SAI4LPEN LL_C2_APB4_GRP1_DisableClockSleep

97.2 BUS Firmware driver defines

The following section lists the various define and macros of the module.

97.2.1 BUS

BUS

AHB1 GRP1 PERIPH

LL_AHB1_GRP1_PERIPH_DMA1

LL_AHB1_GRP1_PERIPH_DMA2

LL_AHB1_GRP1_PERIPH_ADC12

LL_AHB1_GRP1_PERIPH_ART

LL_AHB1_GRP1_PERIPH_ETH1MAC

LL_AHB1_GRP1_PERIPH_ETH1TX

LL_AHB1_GRP1_PERIPH_ETH1RX

LL_AHB1_GRP1_PERIPH_USB1OTGHS

LL_AHB1_GRP1_PERIPH_USB1OTGHSULPI

LL_AHB1_GRP1_PERIPH_USB2OTGHS

LL_AHB1_GRP1_PERIPH_USB2OTGHSULPI

AHB2 GRP1 PERIPH

LL_AHB2_GRP1_PERIPH_DCMI

LL_AHB2_GRP1_PERIPH_CRYP

LL_AHB2_GRP1_PERIPH_HASH

LL_AHB2_GRP1_PERIPH_RNG

LL_AHB2_GRP1_PERIPH_SDMMC2

LL_AHB2_GRP1_PERIPH_D2SRAM1

LL_AHB2_GRP1_PERIPH_D2SRAM2

LL_AHB2_GRP1_PERIPH_D2SRAM3

AHB3 GRP1 PERIPH

LL_AHB3_GRP1_PERIPH_MDMA

LL_AHB3_GRP1_PERIPH_DMA2D

LL_AHB3_GRP1_PERIPH_JPGDEC

LL_AHB3_GRP1_PERIPH_FMC

LL_AHB3_GRP1_PERIPH_QSPI

LL_AHB3_GRP1_PERIPH_SDMMC1

LL_AHB3_GRP1_PERIPH_FLASH

LL_AHB3_GRP1_PERIPH_DTCM1

LL_AHB3_GRP1_PERIPH_DTCM2

LL_AHB3_GRP1_PERIPH_ITCM

LL_AHB3_GRP1_PERIPH_AXISRAM

AHB4 GRP1 PERIPH

LL_AHB4_GRP1_PERIPH_GPIOA

LL_AHB4_GRP1_PERIPH_GPIOB

LL_AHB4_GRP1_PERIPH_GPIOC

LL_AHB4_GRP1_PERIPH_GPIOD

LL_AHB4_GRP1_PERIPH_GPIOE

LL_AHB4_GRP1_PERIPH_GPIOF

LL_AHB4_GRP1_PERIPH_GPIOG

LL_AHB4_GRP1_PERIPH_GPIOH

LL_AHB4_GRP1_PERIPH_GPIOI

LL_AHB4_GRP1_PERIPH_GPIOJ

LL_AHB4_GRP1_PERIPH_GPIOK

LL_AHB4_GRP1_PERIPH_CRC

LL_AHB4_GRP1_PERIPH_BDMA

LL_AHB4_GRP1_PERIPH_ADC3

LL_AHB4_GRP1_PERIPH_HSEM

LL_AHB4_GRP1_PERIPH_BKPRAM

LL_AHB4_GRP1_PERIPH_SRAM4

LL_AHB4_GRP1_PERIPH_D3SRAM1

APB1 GRP1 PERIPH

LL_APB1_GRP1_PERIPH_TIM2

LL_APB1_GRP1_PERIPH_TIM3

LL_APB1_GRP1_PERIPH_TIM4

LL_APB1_GRP1_PERIPH_TIM5

LL_APB1_GRP1_PERIPH_TIM6

LL_APB1_GRP1_PERIPH_TIM7

LL_APB1_GRP1_PERIPH_TIM12

LL_APB1_GRP1_PERIPH_TIM13

LL_APB1_GRP1_PERIPH_TIM14

LL_APB1_GRP1_PERIPH_LPTIM1

LL_APB1_GRP1_PERIPH_WWDG2

LL_APB1_GRP1_PERIPH_SPI2

LL_APB1_GRP1_PERIPH_SPI3

LL_APB1_GRP1_PERIPH_SPDIFRX

LL_APB1_GRP1_PERIPH_USART2

LL_APB1_GRP1_PERIPH_USART3

LL_APB1_GRP1_PERIPH_UART4

LL_APB1_GRP1_PERIPH_UART5

LL_APB1_GRP1_PERIPH_I2C1

LL_APB1_GRP1_PERIPH_I2C2

LL_APB1_GRP1_PERIPH_I2C3

LL_APB1_GRP1_PERIPH_CEC

LL_APB1_GRP1_PERIPH_DAC12

LL_APB1_GRP1_PERIPH_UART7

LL_APB1_GRP1_PERIPH_UART8

APB1 GRP2 PERIPH

LL_APB1_GRP2_PERIPH_CRS

LL_APB1_GRP2_PERIPH_SWPMI1

LL_APB1_GRP2_PERIPH_OPAMP

LL_APB1_GRP2_PERIPH_MDIOS

LL_APB1_GRP2_PERIPH_FDCAN

APB2 GRP1 PERIPH

LL_APB2_GRP1_PERIPH_TIM1

LL_APB2_GRP1_PERIPH_TIM8

LL_APB2_GRP1_PERIPH_USART1

LL_APB2_GRP1_PERIPH_USART6

LL_APB2_GRP1_PERIPH_SPI1

LL_APB2_GRP1_PERIPH_SPI4

LL_APB2_GRP1_PERIPH_TIM15

LL_APB2_GRP1_PERIPH_TIM16

LL_APB2_GRP1_PERIPH_TIM17

LL_APB2_GRP1_PERIPH_SPI5

LL_APB2_GRP1_PERIPH_SAI1

LL_APB2_GRP1_PERIPH_SAI2

LL_APB2_GRP1_PERIPH_SAI3

LL_APB2_GRP1_PERIPH_DFSDM1

LL_APB2_GRP1_PERIPH_HRTIM

APB3 GRP1 PERIPH

LL_APB3_GRP1_PERIPH_LTDC

LL_APB3_GRP1_PERIPH_WWDG1

APB4 GRP1 PERIPH

LL_APB4_GRP1_PERIPH_SYSCFG

LL_APB4_GRP1_PERIPH_LPUART1

LL_APB4_GRP1_PERIPH_SPI6

LL_APB4_GRP1_PERIPH_I2C4

LL_APB4_GRP1_PERIPH_LPTIM2

LL_APB4_GRP1_PERIPH_LPTIM3

LL_APB4_GRP1_PERIPH_LPTIM4

LL_APB4_GRP1_PERIPH_LPTIM5

LL_APB4_GRP1_PERIPH_COMP12

LL_APB4_GRP1_PERIPH_VREF

LL_APB4_GRP1_PERIPH_RTCAPB

LL_APB4_GRP1_PERIPH_SAI4

CLKAM PERIPH

LL_CLKAM_PERIPH_BDMA

LL_CLKAM_PERIPH_LPUART1

LL_CLKAM_PERIPH_SPI6

LL_CLKAM_PERIPH_I2C4

LL_CLKAM_PERIPH_LPTIM2

LL_CLKAM_PERIPH_LPTIM3

LL_CLKAM_PERIPH_LPTIM4

LL_CLKAM_PERIPH_LPTIM5

LL_CLKAM_PERIPH_COMP12

LL_CLKAM_PERIPH_VREF

LL_CLKAM_PERIPH_RTC

LL_CLKAM_PERIPH_CRC

LL_CLKAM_PERIPH_SAI4

LL_CLKAM_PERIPH_ADC3

LL_CLKAM_PERIPH_BKPRAM

LL_CLKAM_PERIPH_SRAM4

98 LL COMP Generic Driver

98.1 COMP Firmware driver registers structures

98.1.1 LL_COMP_InitTypeDef

LL_COMP_InitTypeDef is defined in the stm32h7xx_ll_comp.h

Data Fields

- ***uint32_t PowerMode***
- ***uint32_t InputPlus***
- ***uint32_t InputMinus***
- ***uint32_t InputHysteresis***
- ***uint32_t OutputPolarity***
- ***uint32_t OutputBlankingSource***

Field Documentation

- ***uint32_t LL_COMP_InitTypeDef::PowerMode***

  Set comparator operating mode to adjust power and speed. This parameter can be a value of ***COMP_LL_EC_POWERMODE***This feature can be modified afterwards using unitary function **LL_COMP_SetPowerMode()**.
- ***uint32_t LL_COMP_InitTypeDef::InputPlus***

  Set comparator input plus (non-inverting input). This parameter can be a value of ***COMP_LL_EC_INPUT_PLUS***This feature can be modified afterwards using unitary function **LL_COMP_SetInputPlus()**.
- ***uint32_t LL_COMP_InitTypeDef::InputMinus***

  Set comparator input minus (inverting input). This parameter can be a value of ***COMP_LL_EC_INPUT_MINUS***This feature can be modified afterwards using unitary function **LL_COMP_SetInputMinus()**.
- ***uint32_t LL_COMP_InitTypeDef::InputHysteresis***

  Set comparator hysteresis mode of the input minus. This parameter can be a value of ***COMP_LL_EC_INPUT_HYSTERESIS***This feature can be modified afterwards using unitary function **LL_COMP_SetInputHysteresis()**.
- ***uint32_t LL_COMP_InitTypeDef::OutputPolarity***

  Set comparator output polarity. This parameter can be a value of ***COMP_LL_EC_OUTPUT_POLARITY***This feature can be modified afterwards using unitary function **LL_COMP_SetOutputPolarity()**.
- ***uint32_t LL_COMP_InitTypeDef::OutputBlankingSource***

  Set comparator blanking source. This parameter can be a value of ***COMP_LL_EC_OUTPUT_BLANKING_SOURCE***This feature can be modified afterwards using unitary function **LL_COMP_SetOutputBlankingSource()**.

98.2 COMP Firmware driver API description

The following section lists the various functions of the COMP library.

98.2.1 Detailed description of functions

LL_COMP_SetCommonWindowMode

Function name

__STATIC_INLINE void LL_COMP_SetCommonWindowMode (COMP_Common_TypeDef * COMPxy_COMMON, uint32_t WindowMode)

Function description

Set window mode of a pair of comparators instances (2 consecutive COMP instances odd and even COMP<x> and COMP<x+1>).

Parameters

- **COMPxy_COMMON:** Comparator common instance (can be set directly from CMSIS definition or by using helper macro __LL_COMP_COMMON_INSTANCE())
- **WindowMode:** This parameter can be one of the following values:
  - LL_COMP_WINDOWMODE_DISABLE
  - LL_COMP_WINDOWMODE_COMP1_INPUT_PLUS_COMMON

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGRx WINMODE LL_COMP_SetCommonWindowMode

LL_COMP_GetCommonWindowMode

Function name

__STATIC_INLINE uint32_t LL_COMP_GetCommonWindowMode (COMP_Common_TypeDef * COMPxy_COMMON)

Function description

Get window mode of a pair of comparators instances (2 consecutive COMP instances odd and even COMP<x> and COMP<x+1>).

Parameters

- **COMPxy_COMMON:** Comparator common instance (can be set directly from CMSIS definition or by using helper macro __LL_COMP_COMMON_INSTANCE())

Return values

- **Returned:** value can be one of the following values:
  - LL_COMP_WINDOWMODE_DISABLE
  - LL_COMP_WINDOWMODE_COMP1_INPUT_PLUS_COMMON

Reference Manual to LL API cross reference:

- CFGRx WINMODE LL_COMP_GetCommonWindowMode

LL_COMP_SetPowerMode

Function name

__STATIC_INLINE void LL_COMP_SetPowerMode (COMP_TypeDef * COMPx, uint32_t PowerMode)

Function description

Set comparator instance operating mode to adjust power and speed.

Parameters

- **COMPx:** Comparator instance
- **PowerMode:** This parameter can be one of the following values:
  - LL_COMP_POWERMODE_HIGHSPEED
  - LL_COMP_POWERMODE_MEDIUMSPEED
  - LL_COMP_POWERMODE_ULTRALOWPOWER

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGRx PWRMODE LL_COMP_SetPowerMode

LL_COMP_GetPowerMode

Function name

__STATIC_INLINE uint32_t LL_COMP_GetPowerMode (COMP_TypeDef * COMPx)

Function description

Get comparator instance operating mode to adjust power and speed.

Parameters

- **COMPx:** Comparator instance

Return values

- **Returned:** value can be one of the following values:
  - LL_COMP_POWERMODE_HIGHSPEED
  - LL_COMP_POWERMODE_MEDIUMSPEED
  - LL_COMP_POWERMODE_ULTRALOWPOWER

Reference Manual to LL API cross reference:

- CFGRx PWRMODE LL_COMP_GetPowerMode

LL_COMP_ConfigInputs

Function name

__STATIC_INLINE void LL_COMP_ConfigInputs (COMP_TypeDef * COMPx, uint32_t InputMinus, uint32_t InputPlus)

Function description

Set comparator inputs minus (inverting) and plus (non-inverting).

Parameters

- **COMPx:** Comparator instance
- **InputMinus:** This parameter can be one of the following values:
  - LL_COMP_INPUT_MINUS_1_4VREFINT
  - LL_COMP_INPUT_MINUS_1_2VREFINT
  - LL_COMP_INPUT_MINUS_3_4VREFINT
  - LL_COMP_INPUT_MINUS_VREFINT
  - LL_COMP_INPUT_MINUS_DAC1_CH1
  - LL_COMP_INPUT_MINUS_DAC1_CH2
  - LL_COMP_INPUT_MINUS_IO1
  - LL_COMP_INPUT_MINUS_IO2
  - LL_COMP_INPUT_MINUS_TPSENS_DAC2CH1
  - LL_COMP_INPUT_MINUS_VBAT_VDDAP
- **InputPlus:** This parameter can be one of the following values:
  - LL_COMP_INPUT_PLUS_IO1
  - LL_COMP_INPUT_PLUS_IO2
  - LL_COMP_INPUT_PLUS_DAC2_CH1

Return values

- **None:**

Notes

- In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.
- On this STM32 series, scaler bridge is configurable: to optimize power consumption, this function enables the voltage scaler bridge only when required (when selecting comparator input based on VrefInt: VrefInt or subdivision of VrefInt). For scaler bridge power consumption values, refer to device datasheet, parameter "IDDA(SCALER)".Voltage scaler requires a delay for voltage stabilization. Refer to device datasheet, parameter "tSTART_SCALER".Scaler bridge is common for all comparator instances, therefore if at least one of the comparator instance is requiring the scaler bridge, it remains enabled.

Reference Manual to LL API cross reference:

- CFGRx INMSEL LL_COMP_ConfigInputs
- CFGRx INPSEL LL_COMP_ConfigInputs
- CFGRx BRGEN LL_COMP_ConfigInputs
- CFGRx SCALEN LL_COMP_ConfigInputs

LL_COMP_SetInputPlus

Function name

__STATIC_INLINE void LL_COMP_SetInputPlus (COMP_TypeDef * COMPx, uint32_t InputPlus)

Function description

Set comparator input plus (non-inverting).

Parameters

- **COMPx:** Comparator instance
- **InputPlus:** This parameter can be one of the following values:
  - LL_COMP_INPUT_PLUS_IO1
  - LL_COMP_INPUT_PLUS_IO2
  - LL_COMP_INPUT_PLUS_DAC2_CH1

Return values

- **None:**

Notes

- In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.

Reference Manual to LL API cross reference:

- CFGRx INPSEL LL_COMP_SetInputPlus

LL_COMP_GetInputPlus

Function name

__STATIC_INLINE uint32_t LL_COMP_GetInputPlus (COMP_TypeDef * COMPx)

Function description

Get comparator input plus (non-inverting).

Parameters

- **COMPx:** Comparator instance

Return values

- **Returned:** value can be one of the following values:
  - LL_COMP_INPUT_PLUS_IO1
  - LL_COMP_INPUT_PLUS_IO2
  - LL_COMP_INPUT_PLUS_DAC2_CH1

Notes

- In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.

Reference Manual to LL API cross reference:

- CFGRx INPSEL LL_COMP_GetInputPlus

LL_COMP_SetInputMinus

Function name

__STATIC_INLINE void LL_COMP_SetInputMinus (COMP_TypeDef * COMPx, uint32_t InputMinus)

Function description

Set comparator input minus (inverting).

Parameters

- **COMPx:** Comparator instance
- **InputMinus:** This parameter can be one of the following values:
  - LL_COMP_INPUT_MINUS_1_4VREFINT
  - LL_COMP_INPUT_MINUS_1_2VREFINT
  - LL_COMP_INPUT_MINUS_3_4VREFINT
  - LL_COMP_INPUT_MINUS_VREFINT
  - LL_COMP_INPUT_MINUS_DAC1_CH1
  - LL_COMP_INPUT_MINUS_DAC1_CH2
  - LL_COMP_INPUT_MINUS_IO1
  - LL_COMP_INPUT_MINUS_IO2
  - LL_COMP_INPUT_MINUS_TPSENS_DAC2CH1
  - LL_COMP_INPUT_MINUS_VBAT_VDDAP

Return values

- **None:**

Notes

- In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.
- On this STM32 series, scaler bridge is configurable: to optimize power consumption, this function enables the voltage scaler bridge only when required (when selecting comparator input based on VrefInt: VrefInt or subdivision of VrefInt). For scaler bridge power consumption values, refer to device datasheet, parameter "IDDA(SCALER)".Voltage scaler requires a delay for voltage stabilization. Refer to device datasheet, parameter "tSTART_SCALER".Scaler bridge is common for all comparator instances, therefore if at least one of the comparator instance is requiring the scaler bridge, it remains enabled.

Reference Manual to LL API cross reference:

- CFGRx INMSEL LL_COMP_SetInputMinus
- CFGRx BRGEN LL_COMP_SetInputMinus
- CFGRx SCALEN LL_COMP_SetInputMinus

LL_COMP_GetInputMinus

Function name

__STATIC_INLINE uint32_t LL_COMP_GetInputMinus (COMP_TypeDef * COMPx)

Function description

Get comparator input minus (inverting).

Parameters

- **COMPx:** Comparator instance

Return values

- **Returned:** value can be one of the following values:
  - LL_COMP_INPUT_MINUS_1_4VREFINT
  - LL_COMP_INPUT_MINUS_1_2VREFINT
  - LL_COMP_INPUT_MINUS_3_4VREFINT
  - LL_COMP_INPUT_MINUS_VREFINT
  - LL_COMP_INPUT_MINUS_DAC1_CH1
  - LL_COMP_INPUT_MINUS_DAC1_CH2
  - LL_COMP_INPUT_MINUS_IO1
  - LL_COMP_INPUT_MINUS_IO2
  - LL_COMP_INPUT_MINUS_TPSENS_DAC2CH1
  - LL_COMP_INPUT_MINUS_VBAT_VDDAP

Notes

- In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.

Reference Manual to LL API cross reference:

- CFGRx INMSEL LL_COMP_GetInputMinus
- CFGRx BRGEN LL_COMP_GetInputMinus
- CFGRx SCALEN LL_COMP_GetInputMinus

LL_COMP_SetInputHysteresis

Function name

__STATIC_INLINE void LL_COMP_SetInputHysteresis (COMP_TypeDef * COMPx, uint32_t InputHysteresis)

Function description

Set comparator instance hysteresis mode of the input minus (inverting input).

Parameters

- **COMPx:** Comparator instance
- **InputHysteresis:** This parameter can be one of the following values:
  - LL_COMP_HYSTERESIS_NONE
  - LL_COMP_HYSTERESIS_LOW
  - LL_COMP_HYSTERESIS_MEDIUM
  - LL_COMP_HYSTERESIS_HIGH

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGRx HYST LL_COMP_SetInputHysteresis

LL_COMP_GetInputHysteresis

Function name

__STATIC_INLINE uint32_t LL_COMP_GetInputHysteresis (COMP_TypeDef * COMPx)

Function description

Get comparator instance hysteresis mode of the minus (inverting) input.

Parameters

- **COMPx:** Comparator instance

Return values

- **Returned:** value can be one of the following values:
  - LL_COMP_HYSTERESIS_NONE
  - LL_COMP_HYSTERESIS_LOW
  - LL_COMP_HYSTERESIS_MEDIUM
  - LL_COMP_HYSTERESIS_HIGH

Reference Manual to LL API cross reference:

- CSR HYST LL_COMP_GetInputHysteresis

LL_COMP_SetOutputPolarity

Function name

__STATIC_INLINE void LL_COMP_SetOutputPolarity (COMP_TypeDef * COMPx, uint32_t OutputPolarity)

Function description

Set comparator instance output polarity.

Parameters

- **COMPx:** Comparator instance
- **OutputPolarity:** This parameter can be one of the following values:
  - LL_COMP_OUTPUTPOL_NONINVERTED
  - LL_COMP_OUTPUTPOL_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGRx POLARITY LL_COMP_SetOutputPolarity

LL_COMP_GetOutputPolarity

Function name

__STATIC_INLINE uint32_t LL_COMP_GetOutputPolarity (COMP_TypeDef * COMPx)

Function description

Get comparator instance output polarity.

Parameters

- **COMPx:** Comparator instance

Return values

- **Returned:** value can be one of the following values:
  - LL_COMP_OUTPUTPOL_NONINVERTED
  - LL_COMP_OUTPUTPOL_INVERTED

Reference Manual to LL API cross reference:

- CFGRx POLARITY LL_COMP_GetOutputPolarity

LL_COMP_SetOutputBlankingSource

Function name

__STATIC_INLINE void LL_COMP_SetOutputBlankingSource (COMP_TypeDef * COMPx, uint32_t BlankingSource)

Function description

Set comparator instance blanking source.

Parameters

- **COMPx:** Comparator instance
- **BlankingSource:** This parameter can be one of the following values:
  - LL_COMP_BLANKINGSRC_NONE
  - LL_COMP_BLANKINGSRC_TIM1_OC5
  - LL_COMP_BLANKINGSRC_TIM2_OC3
  - LL_COMP_BLANKINGSRC_TIM3_OC3
  - LL_COMP_BLANKINGSRC_TIM3_OC4
  - LL_COMP_BLANKINGSRC_TIM8_OC5
  - LL_COMP_BLANKINGSRC_TIM15_OC1

Return values

- **None:**

Notes

- Blanking source may be specific to each comparator instance. Refer to description of parameters or to reference manual.
- Availability of parameters of blanking source from timer depends on timers availability on the selected device.

Reference Manual to LL API cross reference:

- CFGR BLANKING LL_COMP_SetOutputBlankingSource

LL_COMP_GetOutputBlankingSource

Function name

__STATIC_INLINE uint32_t LL_COMP_GetOutputBlankingSource (COMP_TypeDef * COMPx)

Function description

Get comparator instance blanking source.

Parameters

- **COMPx:** Comparator instance

Return values

- **Returned:** value can be one of the following values:
  - LL_COMP_BLANKINGSRC_NONE
  - LL_COMP_BLANKINGSRC_TIM1_OC5
  - LL_COMP_BLANKINGSRC_TIM2_OC3
  - LL_COMP_BLANKINGSRC_TIM3_OC3
  - LL_COMP_BLANKINGSRC_TIM3_OC4
  - LL_COMP_BLANKINGSRC_TIM8_OC5
  - LL_COMP_BLANKINGSRC_TIM15_OC1

Notes

- Availability of parameters of blanking source from timer depends on timers availability on the selected device.
- Blanking source may be specific to each comparator instance. Refer to description of parameters or to reference manual.

Reference Manual to LL API cross reference:

- CFGR BLANKING LL_COMP_GetOutputBlankingSource

LL_COMP_SetOutputAlternateFunction

Function name

__STATIC_INLINE void LL_COMP_SetOutputAlternateFunction (COMP_TypeDef * COMPx, uint32_t CompAFx)

Function description

Set the output alternate function in the Option register in order to be used with the alternate function of the timer break input.

Parameters

- **COMPx:** specifies the instance.
- **CompAFx:** specifies the Alternate Function source selection. This parameter can be one of the following values:
  - LL_COMP_AF_PA6
  - LL_COMP_AF_PA8
  - LL_COMP_AF_PB12
  - LL_COMP_AF_PE6
  - LL_COMP_AF_PE15
  - LL_COMP_AF_PG2
  - LL_COMP_AF_PG3
  - LL_COMP_AF_PG4
  - LL_COMP_AF_PI1
  - LL_COMP_AF_PI4
  - LL_COMP_AF_PK2

Return values

- **None:**

Reference Manual to LL API cross reference:

- OR AFOP COMP_LL_EC_OUTPUT_BKIN_TIMER

LL_COMP_GetOutputAlternateFunction

Function name

__STATIC_INLINE uint32_t LL_COMP_GetOutputAlternateFunction (COMP_TypeDef * COMPx)

Function description

Get the output alternate function from the Option register.

Parameters

- **COMPx:** specifies the Comparator instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_COMP_AF_PA6
  - LL_COMP_AF_PA8
  - LL_COMP_AF_PB12
  - LL_COMP_AF_PE6
  - LL_COMP_AF_PE15
  - LL_COMP_AF_PG2
  - LL_COMP_AF_PG3
  - LL_COMP_AF_PG4
  - LL_COMP_AF_PI1
  - LL_COMP_AF_PI4
  - LL_COMP_AF_PK2

Reference Manual to LL API cross reference:

- OR AFOP COMP_LL_EC_OUTPUT_BKIN_TIMER

LL_COMP_Enable

Function name

__STATIC_INLINE void LL_COMP_Enable (COMP_TypeDef * COMPx)

Function description

Enable comparator instance.

Parameters

- **COMPx:** Comparator instance

Return values

- **None:**

Notes

- After enable from off state, comparator requires a delay to reach reach propagation delay specification. Refer to device datasheet, parameter "tSTART".

Reference Manual to LL API cross reference:

- CFGR EN LL_COMP_Enable

LL_COMP_Disable

Function name

__STATIC_INLINE void LL_COMP_Disable (COMP_TypeDef * COMPx)

Function description

Disable comparator instance.

Parameters

- **COMPx:** Comparator instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR EN LL_COMP_Disable

LL_COMP_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_COMP_IsEnabled (COMP_TypeDef * COMPx)

Function description

Get comparator enable state (0: COMP is disabled, 1: COMP is enabled)

Parameters

- **COMPx:** Comparator instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR EN LL_COMP_IsEnabled

LL_COMP_Lock

Function name

__STATIC_INLINE void LL_COMP_Lock (COMP_TypeDef * COMPx)

Function description

Lock comparator instance.

Parameters

- **COMPx:** Comparator instance

Return values

- **None:**

Notes

- Once locked, comparator configuration can be accessed in read-only.
- The only way to unlock the comparator is a device hardware reset.

Reference Manual to LL API cross reference:

- CFGR LOCK LL_COMP_Lock

LL_COMP_IsLocked

Function name

__STATIC_INLINE uint32_t LL_COMP_IsLocked (COMP_TypeDef * COMPx)

Function description

Get comparator lock state (0: COMP is unlocked, 1: COMP is locked).

Parameters

- **COMPx:** Comparator instance

Return values

- **State:** of bit (1 or 0).

Notes

- Once locked, comparator configuration can be accessed in read-only.
- The only way to unlock the comparator is a device hardware reset.

Reference Manual to LL API cross reference:

- CFGR LOCK LL_COMP_IsLocked

LL_COMP_ReadOutputLevel

Function name

__STATIC_INLINE uint32_t LL_COMP_ReadOutputLevel (COMP_TypeDef * COMPx)

Function description

Read comparator instance output level.

Parameters

- **COMPx:** Comparator instance

Return values

- **Returned:** value can be one of the following values:
  - LL_COMP_OUTPUT_LEVEL_LOW
  - LL_COMP_OUTPUT_LEVEL_HIGH

Notes

- The comparator output level depends on the selected polarity (Refer to function LL_COMP_SetOutputPolarity()). If the comparator polarity is not inverted: Comparator output is low when the input plus is at a lower voltage than the input minusComparator output is high when the input plus is at a higher voltage than the input minus If the comparator polarity is inverted:Comparator output is high when the input plus is at a lower voltage than the input minusComparator output is low when the input plus is at a higher voltage than the input minus

Reference Manual to LL API cross reference:

- CFGR VALUE LL_COMP_ReadOutputLevel

LL_COMP_DeInit

Function name

ErrorStatus LL_COMP_DeInit (COMP_TypeDef * COMPx)

Function description

De-initialize registers of the selected COMP instance to their default reset values.

Parameters

- **COMPx:** COMP instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: COMP registers are de-initialized
  - ERROR: COMP registers are not de-initialized

Notes

- If comparator is locked, de-initialization by software is not possible. The only way to unlock the comparator is a device hardware reset.

LL_COMP_Init

Function name

ErrorStatus LL_COMP_Init (COMP_TypeDef * COMPx, LL_COMP_InitTypeDef * COMP_InitStruct)

Function description

Initialize some features of COMP instance.

Parameters

- **COMPx:** COMP instance
- **COMP_InitStruct:** Pointer to a LL_COMP_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: COMP registers are initialized
  - ERROR: COMP registers are not initialized

Notes

- This function configures features of the selected COMP instance. Some features are also available at scope COMP common instance (common to several COMP instances). Refer to functions having argument "COMPxy_COMMON" as parameter.

LL_COMP_StructInit

Function name

void LL_COMP_StructInit (LL_COMP_InitTypeDef * COMP_InitStruct)

Function description

Set each LL_COMP_InitTypeDef field to default value.

Parameters

- **COMP_InitStruct:** Pointer to a LL_COMP_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

98.3 COMP Firmware driver defines

The following section lists the various define and macros of the module.

98.3.1 COMP

COMP

Comparator common modes - Window mode

LL_COMP_WINDOWMODE_DISABLE

Window mode disable: Comparators 1 and 2 are independent

LL_COMP_WINDOWMODE_COMP1_INPUT_PLUS_COMMON

Window mode enable: Comparators instances pair COMP1 and COMP2 have their input plus connected together. The common input is COMP1 input plus (COMP2 input plus is no more accessible).

Definitions of COMP hardware constraints delays

LL_COMP_DELAY_STARTUP_US

Delay for COMP startup time

LL_COMP_DELAY_VOLTAGE_SCALER_STAB_US

Delay for COMP voltage scaler stabilization time

Comparator input - Hysteresis

LL_COMP_HYSTERESIS_NONE

No hysteresis

LL_COMP_HYSTERESIS_LOW

Hysteresis level low

LL_COMP_HYSTERESIS_MEDIUM

Hysteresis level medium

LL_COMP_HYSTERESIS_HIGH

Hysteresis level high

Comparator inputs - Input minus (input inverting) selection

LL_COMP_INPUT_MINUS_1_4VREFINT

Comparator input minus connected to 1/4 VrefInt

LL_COMP_INPUT_MINUS_1_2VREFINT

Comparator input minus connected to 1/2 VrefInt

LL_COMP_INPUT_MINUS_3_4VREFINT

Comparator input minus connected to 3/4 VrefInt

LL_COMP_INPUT_MINUS_VREFINT

Comparator input minus connected to VrefInt

LL_COMP_INPUT_MINUS_DAC1_CH1

Comparator input minus connected to DAC1 channel 1 (DAC_OUT1)

LL_COMP_INPUT_MINUS_DAC1_CH2

Comparator input minus connected to DAC1 channel 2 (DAC_OUT2)

LL_COMP_INPUT_MINUS_IO1

Comparator input minus connected to IO1 (pin PB1 for COMP1, pin PE10 for COMP2)

LL_COMP_INPUT_MINUS_IO2

Comparator input minus connected to IO2 (pin PC4 for COMP1, pin PE7 for COMP2)

Comparator inputs - Input plus (input non-inverting) selection

LL_COMP_INPUT_PLUS_IO1

Comparator input plus connected to IO1 (pin PB0 for COMP1, pin PE9 for COMP2)

LL_COMP_INPUT_PLUS_IO2

Comparator input plus connected to IO2 (pin PB2 for COMP1, pin PE11 for COMP2)

Comparator output - Output to BKIN timer

LL_COMP_AF_PA6

Comparator Alternate Function PA6 source selected to timer BKIN input

LL_COMP_AF_PA8

Comparator Alternate Function PA8 source selected to timer BKIN input

LL_COMP_AF_PB12

Comparator Alternate Function PB12 source selected to timer BKIN input

LL_COMP_AF_PE6

Comparator Alternate Function PE6 source selected to timer BKIN input

LL_COMP_AF_PE15

Comparator Alternate Function PE15 source selected to timer BKIN input

LL_COMP_AF_PG2

Comparator Alternate Function PG2 source selected to timer BKIN input

LL_COMP_AF_PG3

Comparator Alternate Function PG3 source selected to timer BKIN input

LL_COMP_AF_PG4

Comparator Alternate Function PG4 source selected to timer BKIN input

LL_COMP_AF_PI1

Comparator Alternate Function PI1 source selected to timer BKIN input

LL_COMP_AF_PI4

Comparator Alternate Function PI4 source selected to timer BKIN input

LL_COMP_AF_PK2

Comparator Alternate Function PK2 source selected to timer BKIN input

Comparator output - Blanking source

LL_COMP_BLANKINGSRC_NONE

Comparator output without blanking

LL_COMP_BLANKINGSRC_TIM1_OC5

Comparator output blanking source TIM1 OC5 (common to all COMP instances: COMP1, COMP2)

LL_COMP_BLANKINGSRC_TIM2_OC3

Comparator output blanking source TIM2 OC3 (common to all COMP instances: COMP1, COMP2)

LL_COMP_BLANKINGSRC_TIM3_OC3

Comparator output blanking source TIM3 OC3 (common to all COMP instances: COMP1, COMP2)

LL_COMP_BLANKINGSRC_TIM3_OC4

Comparator output blanking source TIM3 OC4 (common to all COMP instances: COMP1, COMP2)

LL_COMP_BLANKINGSRC_TIM8_OC5

Comparator output blanking source TIM8 OC5 (common to all COMP instances: COMP1, COMP2)

LL_COMP_BLANKINGSRC_TIM15_OC1

Comparator output blanking source TIM15 OC1 (common to all COMP instances: COMP1, COMP2)

Comparator output - Output level

LL_COMP_OUTPUT_LEVEL_LOW

Comparator output level low (if the polarity is not inverted, otherwise to be complemented)

LL_COMP_OUTPUT_LEVEL_HIGH

Comparator output level high (if the polarity is not inverted, otherwise to be complemented)

Comparator output - Output polarity

LL_COMP_OUTPUTPOL_NONINVERTED

COMP output polarity is not inverted: comparator output is high when the plus (non-inverting) input is at a higher voltage than the minus (inverting) input

LL_COMP_OUTPUTPOL_INVERTED

COMP output polarity is inverted: comparator output is low when the plus (non-inverting) input is at a lower voltage than the minus (inverting) input

Comparator modes - Power mode

LL_COMP_POWERMODE_HIGHSPEED

COMP power mode to high speed

LL_COMP_POWERMODE_MEDIUMSPEED

COMP power mode to medium speed

LL_COMP_POWERMODE_ULTRALOWPOWER

COMP power mode to ultra-low power

COMP helper macro

__LL_COMP_COMMON_INSTANCE

Description:

- Helper macro to select the COMP common instance to which is belonging the selected COMP instance.

Parameters:

- __COMPx__: COMP instance

Return value:

- COMP: common instance or value "0" if there is no COMP common instance.

Notes:

- COMP common register instance can be used to set parameters common to several COMP instances. Refer to functions having argument "COMPxy_COMMON" as parameter.

Common write and read registers macro

LL_COMP_WriteReg

Description:

- Write a value in COMP register.

Parameters:

- __INSTANCE__: comparator instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_COMP_ReadReg

Description:

- Read a value in COMP register.

Parameters:

- __INSTANCE__: comparator instance
- __REG__: Register to be read

Return value:

- Register: value

99 LL CORDIC Generic Driver

99.1 CORDIC Firmware driver API description

The following section lists the various functions of the CORDIC library.

99.1.1 Detailed description of functions

LL_CORDIC_Config

Function name

__STATIC_INLINE void LL_CORDIC_Config (CORDIC_TypeDef * CORDICx, uint32_t Function, uint32_t Precision, uint32_t Scale, uint32_t NbWrite, uint32_t NbRead, uint32_t InSize, uint32_t OutSize)

Function description

Configure the CORDIC processing.

Parameters

- **CORDICx:** CORDIC instance
- **Function:** parameter can be one of the following values:
  - LL_CORDIC_FUNCTION_COSINE
  - LL_CORDIC_FUNCTION_SINE
  - LL_CORDIC_FUNCTION_PHASE
  - LL_CORDIC_FUNCTION_MODULUS
  - LL_CORDIC_FUNCTION_ARCTANGENT
  - LL_CORDIC_FUNCTION_HCOSINE
  - LL_CORDIC_FUNCTION_HSINE
  - LL_CORDIC_FUNCTION_HARCTANGENT
  - LL_CORDIC_FUNCTION_NATURALLOG
  - LL_CORDIC_FUNCTION_SQUAREROOT
- **Precision:** parameter can be one of the following values:
  - LL_CORDIC_PRECISION_1CYCLE
  - LL_CORDIC_PRECISION_2CYCLES
  - LL_CORDIC_PRECISION_3CYCLES
  - LL_CORDIC_PRECISION_4CYCLES
  - LL_CORDIC_PRECISION_5CYCLES
  - LL_CORDIC_PRECISION_6CYCLES
  - LL_CORDIC_PRECISION_7CYCLES
  - LL_CORDIC_PRECISION_8CYCLES
  - LL_CORDIC_PRECISION_9CYCLES
  - LL_CORDIC_PRECISION_10CYCLES
  - LL_CORDIC_PRECISION_11CYCLES
  - LL_CORDIC_PRECISION_12CYCLES
  - LL_CORDIC_PRECISION_13CYCLES
  - LL_CORDIC_PRECISION_14CYCLES
  - LL_CORDIC_PRECISION_15CYCLES
- **Scale:** parameter can be one of the following values:
  - LL_CORDIC_SCALE_0
  - LL_CORDIC_SCALE_1
  - LL_CORDIC_SCALE_2
  - LL_CORDIC_SCALE_3
  - LL_CORDIC_SCALE_4
  - LL_CORDIC_SCALE_5
  - LL_CORDIC_SCALE_6
  - LL_CORDIC_SCALE_7
- **NbWrite:** parameter can be one of the following values:
  - LL_CORDIC_NBWRITE_1
  - LL_CORDIC_NBWRITE_2
- **NbRead:** parameter can be one of the following values:
  - LL_CORDIC_NBREAD_1
  - LL_CORDIC_NBREAD_2
- **InSize:** parameter can be one of the following values:
  - LL_CORDIC_INSIZE_32BITS
  - LL_CORDIC_INSIZE_16BITS

- **OutSize:** parameter can be one of the following values:
  - LL_CORDIC_OUTSIZE_32BITS
  - LL_CORDIC_OUTSIZE_16BITS

Return values

- **None:**

Notes

- This function set all parameters of CORDIC processing. These parameters can also be set individually using dedicated functions: LL_CORDIC_SetFunction()LL_CORDIC_SetPrecision()LL_CORDIC_SetScale()LL_CORDIC_SetNbWrite( )LL_CORDIC_SetNbRead()LL_CORDIC_SetInSize()LL_CORDIC_SetOutSize()

Reference Manual to LL API cross reference:

- CSR FUNC LL_CORDIC_Config
- CSR PRECISION LL_CORDIC_Config
- CSR SCALE LL_CORDIC_Config
- CSR NARGS LL_CORDIC_Config
- CSR NRES LL_CORDIC_Config
- CSR ARGSIZE LL_CORDIC_Config
- CSR RESIZE LL_CORDIC_Config

LL_CORDIC_SetFunction

Function name

__STATIC_INLINE void LL_CORDIC_SetFunction (CORDIC_TypeDef * CORDICx, uint32_t Function)

Function description

Configure function.

Parameters

- **CORDICx:** CORDIC Instance
- **Function:** parameter can be one of the following values:
  - LL_CORDIC_FUNCTION_COSINE
  - LL_CORDIC_FUNCTION_SINE
  - LL_CORDIC_FUNCTION_PHASE
  - LL_CORDIC_FUNCTION_MODULUS
  - LL_CORDIC_FUNCTION_ARCTANGENT
  - LL_CORDIC_FUNCTION_HCOSINE
  - LL_CORDIC_FUNCTION_HSINE
  - LL_CORDIC_FUNCTION_HARCTANGENT
  - LL_CORDIC_FUNCTION_NATURALLOG
  - LL_CORDIC_FUNCTION_SQUAREROOT

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR FUNC LL_CORDIC_SetFunction

LL_CORDIC_GetFunction

Function name

__STATIC_INLINE uint32_t LL_CORDIC_GetFunction (const CORDIC_TypeDef * CORDICx)

Function description

Return function.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CORDIC_FUNCTION_COSINE
  - LL_CORDIC_FUNCTION_SINE
  - LL_CORDIC_FUNCTION_PHASE
  - LL_CORDIC_FUNCTION_MODULUS
  - LL_CORDIC_FUNCTION_ARCTANGENT
  - LL_CORDIC_FUNCTION_HCOSINE
  - LL_CORDIC_FUNCTION_HSINE
  - LL_CORDIC_FUNCTION_HARCTANGENT
  - LL_CORDIC_FUNCTION_NATURALLOG
  - LL_CORDIC_FUNCTION_SQUAREROOT

Reference Manual to LL API cross reference:

- CSR FUNC LL_CORDIC_GetFunction

LL_CORDIC_SetPrecision

Function name

__STATIC_INLINE void LL_CORDIC_SetPrecision (CORDIC_TypeDef * CORDICx, uint32_t Precision)

Function description

Configure precision in cycles number.

Parameters

- **CORDICx:** CORDIC Instance
- **Precision:** parameter can be one of the following values:
  - LL_CORDIC_PRECISION_1CYCLE
  - LL_CORDIC_PRECISION_2CYCLES
  - LL_CORDIC_PRECISION_3CYCLES
  - LL_CORDIC_PRECISION_4CYCLES
  - LL_CORDIC_PRECISION_5CYCLES
  - LL_CORDIC_PRECISION_6CYCLES
  - LL_CORDIC_PRECISION_7CYCLES
  - LL_CORDIC_PRECISION_8CYCLES
  - LL_CORDIC_PRECISION_9CYCLES
  - LL_CORDIC_PRECISION_10CYCLES
  - LL_CORDIC_PRECISION_11CYCLES
  - LL_CORDIC_PRECISION_12CYCLES
  - LL_CORDIC_PRECISION_13CYCLES
  - LL_CORDIC_PRECISION_14CYCLES
  - LL_CORDIC_PRECISION_15CYCLES

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR PRECISION LL_CORDIC_SetPrecision

LL_CORDIC_GetPrecision

Function name

__STATIC_INLINE uint32_t LL_CORDIC_GetPrecision (const CORDIC_TypeDef * CORDICx)

Function description

Return precision in cycles number.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CORDIC_PRECISION_1CYCLE
  - LL_CORDIC_PRECISION_2CYCLES
  - LL_CORDIC_PRECISION_3CYCLES
  - LL_CORDIC_PRECISION_4CYCLES
  - LL_CORDIC_PRECISION_5CYCLES
  - LL_CORDIC_PRECISION_6CYCLES
  - LL_CORDIC_PRECISION_7CYCLES
  - LL_CORDIC_PRECISION_8CYCLES
  - LL_CORDIC_PRECISION_9CYCLES
  - LL_CORDIC_PRECISION_10CYCLES
  - LL_CORDIC_PRECISION_11CYCLES
  - LL_CORDIC_PRECISION_12CYCLES
  - LL_CORDIC_PRECISION_13CYCLES
  - LL_CORDIC_PRECISION_14CYCLES
  - LL_CORDIC_PRECISION_15CYCLES

Reference Manual to LL API cross reference:

- CSR PRECISION LL_CORDIC_GetPrecision

LL_CORDIC_SetScale

Function name

__STATIC_INLINE void LL_CORDIC_SetScale (CORDIC_TypeDef * CORDICx, uint32_t Scale)

Function description

Configure scaling factor.

Parameters

- **CORDICx:** CORDIC Instance
- **Scale:** parameter can be one of the following values:
  - LL_CORDIC_SCALE_0
  - LL_CORDIC_SCALE_1
  - LL_CORDIC_SCALE_2
  - LL_CORDIC_SCALE_3
  - LL_CORDIC_SCALE_4
  - LL_CORDIC_SCALE_5
  - LL_CORDIC_SCALE_6
  - LL_CORDIC_SCALE_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR SCALE LL_CORDIC_SetScale

LL_CORDIC_GetScale

Function name

__STATIC_INLINE uint32_t LL_CORDIC_GetScale (const CORDIC_TypeDef * CORDICx)

Function description

Return scaling factor.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CORDIC_SCALE_0
  - LL_CORDIC_SCALE_1
  - LL_CORDIC_SCALE_2
  - LL_CORDIC_SCALE_3
  - LL_CORDIC_SCALE_4
  - LL_CORDIC_SCALE_5
  - LL_CORDIC_SCALE_6
  - LL_CORDIC_SCALE_7

Reference Manual to LL API cross reference:

- CSR SCALE LL_CORDIC_GetScale

LL_CORDIC_SetNbWrite

Function name

__STATIC_INLINE void LL_CORDIC_SetNbWrite (CORDIC_TypeDef * CORDICx, uint32_t NbWrite)

Function description

Configure number of 32-bit write expected for one calculation.

Parameters

- **CORDICx:** CORDIC Instance
- **NbWrite:** parameter can be one of the following values:
  - LL_CORDIC_NBWRITE_1
  - LL_CORDIC_NBWRITE_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR NARGS LL_CORDIC_SetNbWrite

LL_CORDIC_GetNbWrite

Function name

__STATIC_INLINE uint32_t LL_CORDIC_GetNbWrite (const CORDIC_TypeDef * CORDICx)

Function description

Return number of 32-bit write expected for one calculation.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CORDIC_NBWRITE_1
  - LL_CORDIC_NBWRITE_2

Reference Manual to LL API cross reference:

- CSR NARGS LL_CORDIC_GetNbWrite

LL_CORDIC_SetNbRead

Function name

__STATIC_INLINE void LL_CORDIC_SetNbRead (CORDIC_TypeDef * CORDICx, uint32_t NbRead)

Function description

Configure number of 32-bit read expected after one calculation.

Parameters

- **CORDICx:** CORDIC Instance
- **NbRead:** parameter can be one of the following values:
  - LL_CORDIC_NBREAD_1
  - LL_CORDIC_NBREAD_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR NRES LL_CORDIC_SetNbRead

LL_CORDIC_GetNbRead

Function name

__STATIC_INLINE uint32_t LL_CORDIC_GetNbRead (const CORDIC_TypeDef * CORDICx)

Function description

Return number of 32-bit read expected after one calculation.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CORDIC_NBREAD_1
  - LL_CORDIC_NBREAD_2

Reference Manual to LL API cross reference:

- CSR NRES LL_CORDIC_GetNbRead

LL_CORDIC_SetInSize

Function name

__STATIC_INLINE void LL_CORDIC_SetInSize (CORDIC_TypeDef * CORDICx, uint32_t InSize)

Function description

Configure width of input data.

Parameters

- **CORDICx:** CORDIC Instance
- **InSize:** parameter can be one of the following values:
  - LL_CORDIC_INSIZE_32BITS
  - LL_CORDIC_INSIZE_16BITS

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR ARGSIZE LL_CORDIC_SetInSize

LL_CORDIC_GetInSize

Function name

__STATIC_INLINE uint32_t LL_CORDIC_GetInSize (const CORDIC_TypeDef * CORDICx)

Function description

Return width of input data.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CORDIC_INSIZE_32BITS
  - LL_CORDIC_INSIZE_16BITS

Reference Manual to LL API cross reference:

- CSR ARGSIZE LL_CORDIC_GetInSize

LL_CORDIC_SetOutSize

Function name

__STATIC_INLINE void LL_CORDIC_SetOutSize (CORDIC_TypeDef * CORDICx, uint32_t OutSize)

Function description

Configure width of output data.

Parameters

- **CORDICx:** CORDIC Instance
- **OutSize:** parameter can be one of the following values:
  - LL_CORDIC_OUTSIZE_32BITS
  - LL_CORDIC_OUTSIZE_16BITS

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR RESIZE LL_CORDIC_SetOutSize

LL_CORDIC_GetOutSize

Function name

__STATIC_INLINE uint32_t LL_CORDIC_GetOutSize (const CORDIC_TypeDef * CORDICx)

Function description

Return width of output data.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CORDIC_OUTSIZE_32BITS
  - LL_CORDIC_OUTSIZE_16BITS

Reference Manual to LL API cross reference:

- CSR RESIZE LL_CORDIC_GetOutSize

LL_CORDIC_EnableIT

Function name

__STATIC_INLINE void LL_CORDIC_EnableIT (CORDIC_TypeDef * CORDICx)

Function description

Enable CORDIC result ready interrupt.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR IEN LL_CORDIC_EnableIT

LL_CORDIC_DisableIT

Function name

__STATIC_INLINE void LL_CORDIC_DisableIT (CORDIC_TypeDef * CORDICx)

Function description

Disable CORDIC result ready interrupt.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR IEN LL_CORDIC_DisableIT

LL_CORDIC_IsEnabledIT

Function name

__STATIC_INLINE uint32_t LL_CORDIC_IsEnabledIT (const CORDIC_TypeDef * CORDICx)

Function description

Check CORDIC result ready interrupt state.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR IEN LL_CORDIC_IsEnabledIT

LL_CORDIC_EnableDMAReq_RD

Function name

__STATIC_INLINE void LL_CORDIC_EnableDMAReq_RD (CORDIC_TypeDef * CORDICx)

Function description

Enable CORDIC DMA read channel request.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR DMAREN LL_CORDIC_EnableDMAReq_RD

LL_CORDIC_DisableDMAReq_RD

Function name

__STATIC_INLINE void LL_CORDIC_DisableDMAReq_RD (CORDIC_TypeDef * CORDICx)

Function description

Disable CORDIC DMA read channel request.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR DMAREN LL_CORDIC_DisableDMAReq_RD

LL_CORDIC_IsEnabledDMAReq_RD

Function name

__STATIC_INLINE uint32_t LL_CORDIC_IsEnabledDMAReq_RD (const CORDIC_TypeDef * CORDICx)

Function description

Check CORDIC DMA read channel request state.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR DMAREN LL_CORDIC_IsEnabledDMAReq_RD

LL_CORDIC_EnableDMAReq_WR

Function name

__STATIC_INLINE void LL_CORDIC_EnableDMAReq_WR (CORDIC_TypeDef * CORDICx)

Function description

Enable CORDIC DMA write channel request.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR DMAWEN LL_CORDIC_EnableDMAReq_WR

LL_CORDIC_DisableDMAReq_WR

Function name

__STATIC_INLINE void LL_CORDIC_DisableDMAReq_WR (CORDIC_TypeDef * CORDICx)

Function description

Disable CORDIC DMA write channel request.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR DMAWEN LL_CORDIC_DisableDMAReq_WR

LL_CORDIC_IsEnabledDMAReq_WR

Function name

__STATIC_INLINE uint32_t LL_CORDIC_IsEnabledDMAReq_WR (const CORDIC_TypeDef * CORDICx)

Function description

Check CORDIC DMA write channel request state.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR DMAWEN LL_CORDIC_IsEnabledDMAReq_WR

LL_CORDIC_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_CORDIC_DMA_GetRegAddr (const CORDIC_TypeDef * CORDICx, uint32_t Direction)

Function description

Get the CORDIC data register address used for DMA transfer.

Parameters

- **CORDICx:** CORDIC Instance
- **Direction:** parameter can be one of the following values:
  - LL_CORDIC_DMA_REG_DATA_IN
  - LL_CORDIC_DMA_REG_DATA_OUT

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- RDATA RES LL_CORDIC_DMA_GetRegAddr
-
- WDATA ARG LL_CORDIC_DMA_GetRegAddr

LL_CORDIC_IsActiveFlag_RRDY

Function name

__STATIC_INLINE uint32_t LL_CORDIC_IsActiveFlag_RRDY (const CORDIC_TypeDef * CORDICx)

Function description

Check CORDIC result ready flag state.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR RRDY LL_CORDIC_IsActiveFlag_RRDY

LL_CORDIC_WriteData

Function name

__STATIC_INLINE void LL_CORDIC_WriteData (CORDIC_TypeDef * CORDICx, uint32_t InData)

Function description

Write 32-bit input data for the CORDIC processing.

Parameters

- **CORDICx:** CORDIC Instance
- **InData:** 0 .. 0xFFFFFFFF : 32-bit value to be provided as input data for CORDIC processing.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WDATA ARG LL_CORDIC_WriteData

LL_CORDIC_ReadData

Function name

__STATIC_INLINE uint32_t LL_CORDIC_ReadData (const CORDIC_TypeDef * CORDICx)

Function description

Return 32-bit output data of CORDIC processing.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **32-bit:** output data of CORDIC processing.

Reference Manual to LL API cross reference:

- RDATA RES LL_CORDIC_ReadData

LL_CORDIC_DeInit

Function name

ErrorStatus LL_CORDIC_DeInit (const CORDIC_TypeDef * CORDICx)

Function description

De-Initialize CORDIC peripheral registers to their default reset values.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: CORDIC registers are de-initialized
  - ERROR: CORDIC registers are not de-initialized

99.2 CORDIC Firmware driver defines

The following section lists the various define and macros of the module.

99.2.1 CORDIC

CORDIC

DMA register data

LL_CORDIC_DMA_REG_DATA_IN

Get address of input data register

LL_CORDIC_DMA_REG_DATA_OUT

Get address of output data register

FUNCTION

LL_CORDIC_FUNCTION_COSINE

Cosine

LL_CORDIC_FUNCTION_SINE

Sine

LL_CORDIC_FUNCTION_PHASE

Phase

LL_CORDIC_FUNCTION_MODULUS

Modulus

LL_CORDIC_FUNCTION_ARCTANGENT

Arctangent

LL_CORDIC_FUNCTION_HCOSINE

Hyperbolic Cosine

LL_CORDIC_FUNCTION_HSINE

Hyperbolic Sine

LL_CORDIC_FUNCTION_HARCTANGENT

Hyperbolic Arctangent

LL_CORDIC_FUNCTION_NATURALLOG

Natural Logarithm

LL_CORDIC_FUNCTION_SQUAREROOT

Square Root

Get Flags Defines

LL_CORDIC_FLAG_RRDY

INSIZE

LL_CORDIC_INSIZE_32BITS

32 bits input data size (Q1.31 format)

LL_CORDIC_INSIZE_16BITS

16 bits input data size (Q1.15 format)

IT Defines

LL_CORDIC_IT_IEN

Result Ready interrupt enable

NBREAD

LL_CORDIC_NBREAD_1

One 32-bits read containing either only one 32-bit data output (Q1.31 format), or two 16-bit data output (Q1.15 format) packed in one 32 bits Data

LL_CORDIC_NBREAD_2

Two 32-bit Data containing two 32-bits data output (Q1.31 format)

NBWRITE

LL_CORDIC_NBWRITE_1

One 32-bits write containing either only one 32-bit data input (Q1.31 format), or two 16-bit data input (Q1.15 format) packed in one 32 bits Data

LL_CORDIC_NBWRITE_2

Two 32-bit write containing two 32-bits data input (Q1.31 format)

OUTSIZE

LL_CORDIC_OUTSIZE_32BITS

32 bits output data size (Q1.31 format)

LL_CORDIC_OUTSIZE_16BITS

16 bits output data size (Q1.15 format)

PRECISION

LL_CORDIC_PRECISION_1CYCLE

LL_CORDIC_PRECISION_2CYCLES

LL_CORDIC_PRECISION_3CYCLES

LL_CORDIC_PRECISION_4CYCLES

LL_CORDIC_PRECISION_5CYCLES

LL_CORDIC_PRECISION_6CYCLES

LL_CORDIC_PRECISION_7CYCLES

LL_CORDIC_PRECISION_8CYCLES

LL_CORDIC_PRECISION_9CYCLES

LL_CORDIC_PRECISION_10CYCLES

LL_CORDIC_PRECISION_11CYCLES

LL_CORDIC_PRECISION_12CYCLES

LL_CORDIC_PRECISION_13CYCLES

LL_CORDIC_PRECISION_14CYCLES

LL_CORDIC_PRECISION_15CYCLES

SCALE

LL_CORDIC_SCALE_0

LL_CORDIC_SCALE_1

LL_CORDIC_SCALE_2

LL_CORDIC_SCALE_3

LL_CORDIC_SCALE_4

LL_CORDIC_SCALE_5

LL_CORDIC_SCALE_6

LL_CORDIC_SCALE_7

Common Write and read registers Macros

LL_CORDIC_WriteReg

Description:

- Write a value in CORDIC register.

Parameters:

- __INSTANCE__: CORDIC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_CORDIC_ReadReg

Description:

- Read a value in CORDIC register.

Parameters:

- __INSTANCE__: CORDIC Instance
- __REG__: Register to be read

Return value:

- Register: value

100 LL CORTEX Generic Driver

100.1 CORTEX Firmware driver API description

The following section lists the various functions of the CORTEX library.

100.1.1 Detailed description of functions

LL_SYSTICK_IsActiveCounterFlag

Function name

__STATIC_INLINE uint32_t LL_SYSTICK_IsActiveCounterFlag (void)

Function description

This function checks if the Systick counter flag is active or not.

Return values

- **State:** of bit (1 or 0).

Notes

- It can be used in timeout function on application side.

Reference Manual to LL API cross reference:

- STK_CTRL COUNTFLAG LL_SYSTICK_IsActiveCounterFlag

LL_SYSTICK_SetClkSource

Function name

__STATIC_INLINE void LL_SYSTICK_SetClkSource (uint32_t Source)

Function description

Configures the SysTick clock source.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_SYSTICK_CLKSOURCE_HCLK_DIV8
  - LL_SYSTICK_CLKSOURCE_HCLK

Return values

- **None:**

Reference Manual to LL API cross reference:

- STK_CTRL CLKSOURCE LL_SYSTICK_SetClkSource

LL_SYSTICK_GetClkSource

Function name

__STATIC_INLINE uint32_t LL_SYSTICK_GetClkSource (void)

Function description

Get the SysTick clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_SYSTICK_CLKSOURCE_HCLK_DIV8
  - LL_SYSTICK_CLKSOURCE_HCLK

Reference Manual to LL API cross reference:

- STK_CTRL CLKSOURCE LL_SYSTICK_GetClkSource

LL_SYSTICK_EnableIT

Function name

__STATIC_INLINE void LL_SYSTICK_EnableIT (void)

Function description

Enable SysTick exception request.

Return values

- **None:**

Reference Manual to LL API cross reference:

- STK_CTRL TICKINT LL_SYSTICK_EnableIT

LL_SYSTICK_DisableIT

Function name

__STATIC_INLINE void LL_SYSTICK_DisableIT (void)

Function description

Disable SysTick exception request.

Return values

- **None:**

Reference Manual to LL API cross reference:

- STK_CTRL TICKINT LL_SYSTICK_DisableIT

LL_SYSTICK_IsEnabledIT

Function name

__STATIC_INLINE uint32_t LL_SYSTICK_IsEnabledIT (void)

Function description

Checks if the SYSTICK interrupt is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- STK_CTRL TICKINT LL_SYSTICK_IsEnabledIT

LL_LPM_EnableSleep

Function name

__STATIC_INLINE void LL_LPM_EnableSleep (void)

Function description

Processor uses sleep as its low power mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SCR SLEEPDEEP LL_LPM_EnableSleep

LL_LPM_EnableDeepSleep

Function name

__STATIC_INLINE void LL_LPM_EnableDeepSleep (void)

Function description

Processor uses deep sleep as its low power mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SCR SLEEPDEEP LL_LPM_EnableDeepSleep

LL_LPM_EnableSleepOnExit

Function name

__STATIC_INLINE void LL_LPM_EnableSleepOnExit (void)

Function description

Configures sleep-on-exit when returning from Handler mode to Thread mode.

Return values

- **None:**

Notes

- Setting this bit to 1 enables an interrupt-driven application to avoid returning to an empty main application.

Reference Manual to LL API cross reference:

- SCB_SCR SLEEPONEXIT LL_LPM_EnableSleepOnExit

LL_LPM_DisableSleepOnExit

Function name

__STATIC_INLINE void LL_LPM_DisableSleepOnExit (void)

Function description

Do not sleep when returning to Thread mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SCR SLEEPONEXIT LL_LPM_DisableSleepOnExit

LL_LPM_EnableEventOnPend

Function name

__STATIC_INLINE void LL_LPM_EnableEventOnPend (void)

Function description

Enabled events and all interrupts, including disabled interrupts, can wakeup the processor.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SCR SEVEONPEND LL_LPM_EnableEventOnPend

LL_LPM_DisableEventOnPend

Function name

__STATIC_INLINE void LL_LPM_DisableEventOnPend (void)

Function description

Only enabled interrupts or events can wakeup the processor, disabled interrupts are excluded.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SCR SEVEONPEND LL_LPM_DisableEventOnPend

LL_HANDLER_EnableFault

Function name

__STATIC_INLINE void LL_HANDLER_EnableFault (uint32_t Fault)

Function description

Enable a fault in System handler control register (SHCSR)

Parameters

- **Fault:** This parameter can be a combination of the following values:
  - LL_HANDLER_FAULT_USG
  - LL_HANDLER_FAULT_BUS
  - LL_HANDLER_FAULT_MEM

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SHCSR MEMFAULTENA LL_HANDLER_EnableFault

LL_HANDLER_DisableFault

Function name

__STATIC_INLINE void LL_HANDLER_DisableFault (uint32_t Fault)

Function description

Disable a fault in System handler control register (SHCSR)

Parameters

- **Fault:** This parameter can be a combination of the following values:
  - LL_HANDLER_FAULT_USG
  - LL_HANDLER_FAULT_BUS
  - LL_HANDLER_FAULT_MEM

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SHCSR MEMFAULTENA LL_HANDLER_DisableFault

LL_CPUID_GetImplementer

Function name

__STATIC_INLINE uint32_t LL_CPUID_GetImplementer (void)

Function description

Get Implementer code.

Return values

- **Value:** should be equal to 0x41 for ARM

Reference Manual to LL API cross reference:

- SCB_CPUID IMPLEMENTER LL_CPUID_GetImplementer

LL_CPUID_GetVariant

Function name

__STATIC_INLINE uint32_t LL_CPUID_GetVariant (void)

Function description

Get Variant number (The r value in the rnpn product revision identifier)

Return values

- **Value:** between 0 and 255 (0x0: revision 0)

Reference Manual to LL API cross reference:

- SCB_CPUID VARIANT LL_CPUID_GetVariant

LL_CPUID_GetConstant

Function name

__STATIC_INLINE uint32_t LL_CPUID_GetConstant (void)

Function description

Get Constant number.

Return values

- **Value:** should be equal to 0xF for Cortex-M7 and Cortex-M4 devices

Reference Manual to LL API cross reference:

- SCB_CPUID ARCHITECTURE LL_CPUID_GetConstant

LL_CPUID_GetParNo

Function name

__STATIC_INLINE uint32_t LL_CPUID_GetParNo (void)

Function description

Get Part number.

Return values

- **Value:** should be equal to 0xC27 for Cortex-M7 and equal to 0xC24 for Cortex-M4

Reference Manual to LL API cross reference:

- SCB_CPUID PARTNO LL_CPUID_GetParNo

LL_CPUID_GetRevision

Function name

__STATIC_INLINE uint32_t LL_CPUID_GetRevision (void)

Function description

Get Revision number (The p value in the rnpn product revision identifier, indicates patch release)

Return values

- **Value:** between 0 and 255 (0x1: patch 1)

Reference Manual to LL API cross reference:

- SCB_CPUID REVISION LL_CPUID_GetRevision

100.2 CORTEX Firmware driver defines

The following section lists the various define and macros of the module.

100.2.1 CORTEX

CORTEX

SYSTICK Clock Source

LL_SYSTICK_CLKSOURCE_HCLK_DIV8

AHB clock divided by 8 selected as SysTick clock source.

LL_SYSTICK_CLKSOURCE_HCLK

AHB clock selected as SysTick clock source.

Handler Fault type

LL_HANDLER_FAULT_USG

Usage fault

LL_HANDLER_FAULT_BUS

Bus fault

LL_HANDLER_FAULT_MEM

Memory management fault

101 LL CRC Generic Driver

101.1 CRC Firmware driver API description

The following section lists the various functions of the CRC library.

101.1.1 Detailed description of functions

LL_CRC_ResetCRCCalculationUnit

Function name

__STATIC_INLINE void LL_CRC_ResetCRCCalculationUnit (CRC_TypeDef * CRCx)

Function description

Reset the CRC calculation unit.

Parameters

- **CRCx:** CRC Instance

Return values

- **None:**

Notes

- If Programmable Initial CRC value feature is available, also set the Data Register to the value stored in the CRC_INIT register, otherwise, reset Data Register to its default value.

Reference Manual to LL API cross reference:

- CR RESET LL_CRC_ResetCRCCalculationUnit

LL_CRC_SetPolynomialSize

Function name

__STATIC_INLINE void LL_CRC_SetPolynomialSize (CRC_TypeDef * CRCx, uint32_t PolySize)

Function description

Configure size of the polynomial.

Parameters

- **CRCx:** CRC Instance
- **PolySize:** This parameter can be one of the following values:
  - LL_CRC_POLYLENGTH_32B
  - LL_CRC_POLYLENGTH_16B
  - LL_CRC_POLYLENGTH_8B
  - LL_CRC_POLYLENGTH_7B

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR POLYSIZE LL_CRC_SetPolynomialSize

LL_CRC_GetPolynomialSize

Function name

__STATIC_INLINE uint32_t LL_CRC_GetPolynomialSize (CRC_TypeDef * CRCx)

Function description

Return size of the polynomial.

Parameters

- **CRCx:** CRC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CRC_POLYLENGTH_32B
  - LL_CRC_POLYLENGTH_16B
  - LL_CRC_POLYLENGTH_8B
  - LL_CRC_POLYLENGTH_7B

Reference Manual to LL API cross reference:

- CR POLYSIZE LL_CRC_GetPolynomialSize

LL_CRC_SetInputDataReverseMode

Function name

__STATIC_INLINE void LL_CRC_SetInputDataReverseMode (CRC_TypeDef * CRCx, uint32_t ReverseMode)

Function description

Configure the reversal of the bit order of the input data.

Parameters

- **CRCx:** CRC Instance
- **ReverseMode:** This parameter can be one of the following values:
  - LL_CRC_INDATA_REVERSE_NONE
  - LL_CRC_INDATA_REVERSE_BYTE
  - LL_CRC_INDATA_REVERSE_HALFWORD
  - LL_CRC_INDATA_REVERSE_WORD

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR REV_IN LL_CRC_SetInputDataReverseMode

LL_CRC_GetInputDataReverseMode

Function name

__STATIC_INLINE uint32_t LL_CRC_GetInputDataReverseMode (CRC_TypeDef * CRCx)

Function description

Return type of reversal for input data bit order.

Parameters

- **CRCx:** CRC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CRC_INDATA_REVERSE_NONE
  - LL_CRC_INDATA_REVERSE_BYTE
  - LL_CRC_INDATA_REVERSE_HALFWORD
  - LL_CRC_INDATA_REVERSE_WORD

Reference Manual to LL API cross reference:

- CR REV_IN LL_CRC_GetInputDataReverseMode

LL_CRC_SetOutputDataReverseMode

Function name

__STATIC_INLINE void LL_CRC_SetOutputDataReverseMode (CRC_TypeDef * CRCx, uint32_t ReverseMode)

Function description

Configure the reversal of the bit order of the Output data.

Parameters

- **CRCx:** CRC Instance
- **ReverseMode:** This parameter can be one of the following values:
  - LL_CRC_OUTDATA_REVERSE_NONE
  - LL_CRC_OUTDATA_REVERSE_BIT

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR REV_OUT LL_CRC_SetOutputDataReverseMode

LL_CRC_GetOutputDataReverseMode

Function name

__STATIC_INLINE uint32_t LL_CRC_GetOutputDataReverseMode (CRC_TypeDef * CRCx)

Function description

Return type of reversal of the bit order of the Output data.

Parameters

- **CRCx:** CRC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CRC_OUTDATA_REVERSE_NONE
  - LL_CRC_OUTDATA_REVERSE_BIT

Reference Manual to LL API cross reference:

- CR REV_OUT LL_CRC_GetOutputDataReverseMode

LL_CRC_SetInitialData

Function name

__STATIC_INLINE void LL_CRC_SetInitialData (CRC_TypeDef * CRCx, uint32_t InitCrc)

Function description

Initialize the Programmable initial CRC value.

Parameters

- **CRCx:** CRC Instance
- **InitCrc:** Value to be programmed in Programmable initial CRC value register

Return values

- **None:**

Notes

- If the CRC size is less than 32 bits, the least significant bits are used to write the correct value
- LL_CRC_DEFAULT_CRC_INITVALUE could be used as value for InitCrc parameter.

Reference Manual to LL API cross reference:

- INIT INIT LL_CRC_SetInitialData

LL_CRC_GetInitialData

Function name

__STATIC_INLINE uint32_t LL_CRC_GetInitialData (CRC_TypeDef * CRCx)

Function description

Return current Initial CRC value.

Parameters

- **CRCx:** CRC Instance

Return values

- **Value:** programmed in Programmable initial CRC value register

Notes

- If the CRC size is less than 32 bits, the least significant bits are used to read the correct value

Reference Manual to LL API cross reference:

- INIT INIT LL_CRC_GetInitialData

LL_CRC_SetPolynomialCoef

Function name

__STATIC_INLINE void LL_CRC_SetPolynomialCoef (CRC_TypeDef * CRCx, uint32_t PolynomCoef)

Function description

Initialize the Programmable polynomial value (coefficients of the polynomial to be used for CRC calculation).

Parameters

- **CRCx:** CRC Instance
- **PolynomCoef:** Value to be programmed in Programmable Polynomial value register

Return values

- **None:**

Notes

- LL_CRC_DEFAULT_CRC32_POLY could be used as value for PolynomCoef parameter.
- Please check Reference Manual and existing Errata Sheets, regarding possible limitations for Polynomial values usage. For example, for a polynomial of degree 7, X^7 + X^6 + X^5 + X^2 + 1 is written 0x65

Reference Manual to LL API cross reference:

- POL POL LL_CRC_SetPolynomialCoef

LL_CRC_GetPolynomialCoef

Function name

__STATIC_INLINE uint32_t LL_CRC_GetPolynomialCoef (CRC_TypeDef * CRCx)

Function description

Return current Programmable polynomial value.

Parameters

- **CRCx:** CRC Instance

Return values

- **Value:** programmed in Programmable Polynomial value register

Notes

- Please check Reference Manual and existing Errata Sheets, regarding possible limitations for Polynomial values usage. For example, for a polynomial of degree 7, X^7 + X^6 + X^5 + X^2 + 1 is written 0x65

Reference Manual to LL API cross reference:

- POL POL LL_CRC_GetPolynomialCoef

LL_CRC_FeedData32

Function name

__STATIC_INLINE void LL_CRC_FeedData32 (CRC_TypeDef * CRCx, uint32_t InData)

Function description

Write given 32-bit data to the CRC calculator.

Parameters

- **CRCx:** CRC Instance
- **InData:** value to be provided to CRC calculator between between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DR DR LL_CRC_FeedData32

LL_CRC_FeedData16

Function name

__STATIC_INLINE void LL_CRC_FeedData16 (CRC_TypeDef * CRCx, uint16_t InData)

Function description

Write given 16-bit data to the CRC calculator.

Parameters

- **CRCx:** CRC Instance
- **InData:** 16 bit value to be provided to CRC calculator between between Min_Data=0 and Max_Data=0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DR DR LL_CRC_FeedData16

LL_CRC_FeedData8

Function name

__STATIC_INLINE void LL_CRC_FeedData8 (CRC_TypeDef * CRCx, uint8_t InData)

Function description

Write given 8-bit data to the CRC calculator.

Parameters

- **CRCx:** CRC Instance
- **InData:** 8 bit value to be provided to CRC calculator between between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DR DR LL_CRC_FeedData8

LL_CRC_ReadData32

Function name

__STATIC_INLINE uint32_t LL_CRC_ReadData32 (CRC_TypeDef * CRCx)

Function description

Return current CRC calculation result.

Parameters

- **CRCx:** CRC Instance

Return values

- **Current:** CRC calculation result as stored in CRC_DR register (32 bits).

Reference Manual to LL API cross reference:

- DR DR LL_CRC_ReadData32

LL_CRC_ReadData16

Function name

__STATIC_INLINE uint16_t LL_CRC_ReadData16 (CRC_TypeDef * CRCx)

Function description

Return current CRC calculation result.

Parameters

- **CRCx:** CRC Instance

Return values

- **Current:** CRC calculation result as stored in CRC_DR register (16 bits).

Notes

- This function is expected to be used in a 16 bits CRC polynomial size context.

Reference Manual to LL API cross reference:

- DR DR LL_CRC_ReadData16

LL_CRC_ReadData8

Function name

__STATIC_INLINE uint8_t LL_CRC_ReadData8 (CRC_TypeDef * CRCx)

Function description

Return current CRC calculation result.

Parameters

- **CRCx:** CRC Instance

Return values

- **Current:** CRC calculation result as stored in CRC_DR register (8 bits).

Notes

- This function is expected to be used in a 8 bits CRC polynomial size context.

Reference Manual to LL API cross reference:

- DR DR LL_CRC_ReadData8

LL_CRC_ReadData7

Function name

__STATIC_INLINE uint8_t LL_CRC_ReadData7 (CRC_TypeDef * CRCx)

Function description

Return current CRC calculation result.

Parameters

- **CRCx:** CRC Instance

Return values

- **Current:** CRC calculation result as stored in CRC_DR register (7 bits).

Notes

- This function is expected to be used in a 7 bits CRC polynomial size context.

Reference Manual to LL API cross reference:

- DR DR LL_CRC_ReadData7

LL_CRC_Read_IDR

Function name

__STATIC_INLINE uint32_t LL_CRC_Read_IDR (CRC_TypeDef * CRCx)

Function description

Return data stored in the Independent Data(IDR) register.

Parameters

- **CRCx:** CRC Instance

Return values

- **Value:** stored in CRC_IDR register (General-purpose 32-bit data register).

Notes

- This register can be used as a temporary storage location for one 32-bit long data.

Reference Manual to LL API cross reference:

- IDR IDR LL_CRC_Read_IDR

LL_CRC_Write_IDR

Function name

__STATIC_INLINE void LL_CRC_Write_IDR (CRC_TypeDef * CRCx, uint32_t InData)

Function description

Store data in the Independent Data(IDR) register.

Parameters

- **CRCx:** CRC Instance
- **InData:** value to be stored in CRC_IDR register (32-bit) between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Notes

- This register can be used as a temporary storage location for one 32-bit long data.

Reference Manual to LL API cross reference:

- IDR IDR LL_CRC_Write_IDR

LL_CRC_DeInit

Function name

ErrorStatus LL_CRC_DeInit (CRC_TypeDef * CRCx)

Function description

De-initialize CRC registers (Registers restored to their default values).

Parameters

- **CRCx:** CRC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: CRC registers are de-initialized
  - ERROR: CRC registers are not de-initialized

101.2 CRC Firmware driver defines

The following section lists the various define and macros of the module.

101.2.1 CRC

CRC

Default CRC computation initialization value

LL_CRC_DEFAULT_CRC_INITVALUE

Default CRC computation initialization value

Default CRC generating polynomial value

LL_CRC_DEFAULT_CRC32_POLY

Default CRC generating polynomial value

Input Data Reverse

LL_CRC_INDATA_REVERSE_NONE

Input Data bit order not affected

LL_CRC_INDATA_REVERSE_BYTE

Input Data bit reversal done by byte

LL_CRC_INDATA_REVERSE_HALFWORD

Input Data bit reversal done by half-word

LL_CRC_INDATA_REVERSE_WORD

Input Data bit reversal done by word

Output Data Reverse

LL_CRC_OUTDATA_REVERSE_NONE

Output Data bit order not affected

LL_CRC_OUTDATA_REVERSE_BIT

Output Data bit reversal done by bit

Polynomial length

LL_CRC_POLYLENGTH_32B

32 bits Polynomial size

LL_CRC_POLYLENGTH_16B

16 bits Polynomial size

LL_CRC_POLYLENGTH_8B

8 bits Polynomial size

LL_CRC_POLYLENGTH_7B

7 bits Polynomial size

Common Write and read registers Macros

LL_CRC_WriteReg

Description:

- Write a value in CRC register.

Parameters:

- __INSTANCE__: CRC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_CRC_ReadReg

Description:

- Read a value in CRC register.

Parameters:

- __INSTANCE__: CRC Instance
- __REG__: Register to be read

Return value:

- Register: value

102 LL CRS Generic Driver

102.1 CRS Firmware driver API description

The following section lists the various functions of the CRS library.

102.1.1 Detailed description of functions

LL_CRS_EnableFreqErrorCounter

Function name

__STATIC_INLINE void LL_CRS_EnableFreqErrorCounter (void)

Function description

Enable Frequency error counter.

Return values

- **None:**

Notes

- When this bit is set, the CRS_CFGR register is write-protected and cannot be modified

Reference Manual to LL API cross reference:

- CR CEN LL_CRS_EnableFreqErrorCounter

LL_CRS_DisableFreqErrorCounter

Function name

__STATIC_INLINE void LL_CRS_DisableFreqErrorCounter (void)

Function description

Disable Frequency error counter.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CEN LL_CRS_DisableFreqErrorCounter

LL_CRS_IsEnabledFreqErrorCounter

Function name

__STATIC_INLINE uint32_t LL_CRS_IsEnabledFreqErrorCounter (void)

Function description

Check if Frequency error counter is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CEN LL_CRS_IsEnabledFreqErrorCounter

LL_CRS_EnableAutoTrimming

Function name

__STATIC_INLINE void LL_CRS_EnableAutoTrimming (void)

Function description

Enable Automatic trimming counter.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR AUTOTRIMEN LL_CRS_EnableAutoTrimming

LL_CRS_DisableAutoTrimming

Function name

__STATIC_INLINE void LL_CRS_DisableAutoTrimming (void)

Function description

Disable Automatic trimming counter.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR AUTOTRIMEN LL_CRS_DisableAutoTrimming

LL_CRS_IsEnabledAutoTrimming

Function name

__STATIC_INLINE uint32_t LL_CRS_IsEnabledAutoTrimming (void)

Function description

Check if Automatic trimming is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR AUTOTRIMEN LL_CRS_IsEnabledAutoTrimming

LL_CRS_SetHSI48SmoothTrimming

Function name

__STATIC_INLINE void LL_CRS_SetHSI48SmoothTrimming (uint32_t Value)

Function description

Set HSI48 oscillator smooth trimming.

Parameters

- **Value:** a number between Min_Data = 0 and Max_Data = 127

Return values

- **None:**

Notes

- When the AUTOTRIMEN bit is set, this field is controlled by hardware and is read-only
- Default value can be set thanks to LL_CRS_HSI48CALIBRATION_DEFAULT

Reference Manual to LL API cross reference:

- CR TRIM LL_CRS_SetHSI48SmoothTrimming

LL_CRS_GetHSI48SmoothTrimming

Function name

__STATIC_INLINE uint32_t LL_CRS_GetHSI48SmoothTrimming (void)

Function description

Get HSI48 oscillator smooth trimming.

Return values

- **a:** number between Min_Data = 0 and Max_Data = 127

Reference Manual to LL API cross reference:

- CR TRIM LL_CRS_GetHSI48SmoothTrimming

LL_CRS_SetReloadCounter

Function name

__STATIC_INLINE void LL_CRS_SetReloadCounter (uint32_t Value)

Function description

Set counter reload value.

Parameters

- **Value:** a number between Min_Data = 0 and Max_Data = 0xFFFF

Return values

- **None:**

Notes

- Default value can be set thanks to LL_CRS_RELOADVALUE_DEFAULT Otherwise it can be calculated in using macro __LL_CRS_CALC_CALCULATE_RELOADVALUE (_FTARGET_, _FSYNC_)

Reference Manual to LL API cross reference:

- CFGR RELOAD LL_CRS_SetReloadCounter

LL_CRS_GetReloadCounter

Function name

__STATIC_INLINE uint32_t LL_CRS_GetReloadCounter (void)

Function description

Get counter reload value.

Return values

- **a:** number between Min_Data = 0 and Max_Data = 0xFFFF

Reference Manual to LL API cross reference:

- CFGR RELOAD LL_CRS_GetReloadCounter

LL_CRS_SetFreqErrorLimit

Function name

__STATIC_INLINE void LL_CRS_SetFreqErrorLimit (uint32_t Value)

Function description

Set frequency error limit.

Parameters

- **Value:** a number between Min_Data = 0 and Max_Data = 255

Return values

- **None:**

Notes

- Default value can be set thanks to LL_CRS_ERRORLIMIT_DEFAULT

Reference Manual to LL API cross reference:

- CFGR FELIM LL_CRS_SetFreqErrorLimit

LL_CRS_GetFreqErrorLimit

Function name

__STATIC_INLINE uint32_t LL_CRS_GetFreqErrorLimit (void)

Function description

Get frequency error limit.

Return values

- **A:** number between Min_Data = 0 and Max_Data = 255

Reference Manual to LL API cross reference:

- CFGR FELIM LL_CRS_GetFreqErrorLimit

LL_CRS_SetSyncDivider

Function name

__STATIC_INLINE void LL_CRS_SetSyncDivider (uint32_t Divider)

Function description

Set division factor for SYNC signal.

Parameters

- **Divider:** This parameter can be one of the following values:
  - LL_CRS_SYNC_DIV_1
  - LL_CRS_SYNC_DIV_2
  - LL_CRS_SYNC_DIV_4
  - LL_CRS_SYNC_DIV_8
  - LL_CRS_SYNC_DIV_16
  - LL_CRS_SYNC_DIV_32
  - LL_CRS_SYNC_DIV_64
  - LL_CRS_SYNC_DIV_128

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR SYNCDIV LL_CRS_SetSyncDivider

LL_CRS_GetSyncDivider

Function name

__STATIC_INLINE uint32_t LL_CRS_GetSyncDivider (void)

Function description

Get division factor for SYNC signal.

Return values

- **Returned:** value can be one of the following values:
  - LL_CRS_SYNC_DIV_1
  - LL_CRS_SYNC_DIV_2
  - LL_CRS_SYNC_DIV_4
  - LL_CRS_SYNC_DIV_8
  - LL_CRS_SYNC_DIV_16
  - LL_CRS_SYNC_DIV_32
  - LL_CRS_SYNC_DIV_64
  - LL_CRS_SYNC_DIV_128

Reference Manual to LL API cross reference:

- CFGR SYNCDIV LL_CRS_GetSyncDivider

LL_CRS_SetSyncSignalSource

Function name

__STATIC_INLINE void LL_CRS_SetSyncSignalSource (uint32_t Source)

Function description

Set SYNC signal source.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_CRS_SYNC_SOURCE_GPIO
  - LL_CRS_SYNC_SOURCE_LSE
  - LL_CRS_SYNC_SOURCE_USB

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR SYNCSRC LL_CRS_SetSyncSignalSource

LL_CRS_GetSyncSignalSource

Function name

__STATIC_INLINE uint32_t LL_CRS_GetSyncSignalSource (void)

Function description

Get SYNC signal source.

Return values

- **Returned:** value can be one of the following values:
  - LL_CRS_SYNC_SOURCE_GPIO
  - LL_CRS_SYNC_SOURCE_LSE
  - LL_CRS_SYNC_SOURCE_USB

Reference Manual to LL API cross reference:

- CFGR SYNCSRC LL_CRS_GetSyncSignalSource

LL_CRS_SetSyncPolarity

Function name

__STATIC_INLINE void LL_CRS_SetSyncPolarity (uint32_t Polarity)

Function description

Set input polarity for the SYNC signal source.

Parameters

- **Polarity:** This parameter can be one of the following values:
  - LL_CRS_SYNC_POLARITY_RISING
  - LL_CRS_SYNC_POLARITY_FALLING

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR SYNCPOL LL_CRS_SetSyncPolarity

LL_CRS_GetSyncPolarity

Function name

__STATIC_INLINE uint32_t LL_CRS_GetSyncPolarity (void)

Function description

Get input polarity for the SYNC signal source.

Return values

- **Returned:** value can be one of the following values:
  - LL_CRS_SYNC_POLARITY_RISING
  - LL_CRS_SYNC_POLARITY_FALLING

Reference Manual to LL API cross reference:

- CFGR SYNCPOL LL_CRS_GetSyncPolarity

LL_CRS_ConfigSynchronization

Function name

__STATIC_INLINE void LL_CRS_ConfigSynchronization (uint32_t HSI48CalibrationValue, uint32_t ErrorLimitValue, uint32_t ReloadValue, uint32_t Settings)

Function description

Configure CRS for the synchronization.

Parameters

- **HSI48CalibrationValue:** a number between Min_Data = 0 and Max_Data = 63
- **ErrorLimitValue:** a number between Min_Data = 0 and Max_Data = 0xFFFF
- **ReloadValue:** a number between Min_Data = 0 and Max_Data = 255
- **Settings:** This parameter can be a combination of the following values:
  - LL_CRS_SYNC_DIV_1 or LL_CRS_SYNC_DIV_2 or LL_CRS_SYNC_DIV_4 or LL_CRS_SYNC_DIV_8 or LL_CRS_SYNC_DIV_16 or LL_CRS_SYNC_DIV_32 or LL_CRS_SYNC_DIV_64 or LL_CRS_SYNC_DIV_128
  - LL_CRS_SYNC_SOURCE_GPIO or LL_CRS_SYNC_SOURCE_LSE or LL_CRS_SYNC_SOURCE_USB
  - LL_CRS_SYNC_POLARITY_RISING or LL_CRS_SYNC_POLARITY_FALLING

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TRIM LL_CRS_ConfigSynchronization
- CFGR RELOAD LL_CRS_ConfigSynchronization
- CFGR FELIM LL_CRS_ConfigSynchronization
- CFGR SYNCDIV LL_CRS_ConfigSynchronization
- CFGR SYNCSRC LL_CRS_ConfigSynchronization
- CFGR SYNCPOL LL_CRS_ConfigSynchronization

LL_CRS_GenerateEvent_SWSYNC

Function name

__STATIC_INLINE void LL_CRS_GenerateEvent_SWSYNC (void)

Function description

Generate software SYNC event.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SWSYNC LL_CRS_GenerateEvent_SWSYNC

LL_CRS_GetFreqErrorDirection

Function name

__STATIC_INLINE uint32_t LL_CRS_GetFreqErrorDirection (void)

Function description

Get the frequency error direction latched in the time of the last SYNC event.

Return values

- **Returned:** value can be one of the following values:
  - LL_CRS_FREQ_ERROR_DIR_UP
  - LL_CRS_FREQ_ERROR_DIR_DOWN

Reference Manual to LL API cross reference:

- ISR FEDIR LL_CRS_GetFreqErrorDirection

LL_CRS_GetFreqErrorCapture

Function name

__STATIC_INLINE uint32_t LL_CRS_GetFreqErrorCapture (void)

Function description

Get the frequency error counter value latched in the time of the last SYNC event.

Return values

- **A:** number between Min_Data = 0x0000 and Max_Data = 0xFFFF

Reference Manual to LL API cross reference:

- ISR FECAP LL_CRS_GetFreqErrorCapture

LL_CRS_IsActiveFlag_SYNCOK

Function name

__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCOK (void)

Function description

Check if SYNC event OK signal occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR SYNCOKF LL_CRS_IsActiveFlag_SYNCOK

LL_CRS_IsActiveFlag_SYNCWARN

Function name

__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCWARN (void)

Function description

Check if SYNC warning signal occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR SYNCWARNF LL_CRS_IsActiveFlag_SYNCWARN

LL_CRS_IsActiveFlag_ERR

Function name

__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_ERR (void)

Function description

Check if Synchronization or trimming error signal occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ERRF LL_CRS_IsActiveFlag_ERR

LL_CRS_IsActiveFlag_ESYNC

Function name

__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_ESYNC (void)

Function description

Check if Expected SYNC signal occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ESYNCF LL_CRS_IsActiveFlag_ESYNC

LL_CRS_IsActiveFlag_SYNCERR

Function name

__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCERR (void)

Function description

Check if SYNC error signal occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR SYNCERR LL_CRS_IsActiveFlag_SYNCERR

LL_CRS_IsActiveFlag_SYNCMISS

Function name

__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCMISS (void)

Function description

Check if SYNC missed error signal occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR SYNCMISS LL_CRS_IsActiveFlag_SYNCMISS

LL_CRS_IsActiveFlag_TRIMOVF

Function name

__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_TRIMOVF (void)

Function description

Check if Trimming overflow or underflow occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TRIMOVF LL_CRS_IsActiveFlag_TRIMOVF

LL_CRS_ClearFlag_SYNCOK

Function name

__STATIC_INLINE void LL_CRS_ClearFlag_SYNCOK (void)

Function description

Clear the SYNC event OK flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR SYNCOKC LL_CRS_ClearFlag_SYNCOK

LL_CRS_ClearFlag_SYNCWARN

Function name

__STATIC_INLINE void LL_CRS_ClearFlag_SYNCWARN (void)

Function description

Clear the SYNC warning flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR SYNCWARNC LL_CRS_ClearFlag_SYNCWARN

LL_CRS_ClearFlag_ERR

Function name

__STATIC_INLINE void LL_CRS_ClearFlag_ERR (void)

Function description

Clear TRIMOVF, SYNCMISS and SYNCERR bits and consequently also the ERR flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ERRC LL_CRS_ClearFlag_ERR

LL_CRS_ClearFlag_ESYNC

Function name

__STATIC_INLINE void LL_CRS_ClearFlag_ESYNC (void)

Function description

Clear Expected SYNC flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ESYNCC LL_CRS_ClearFlag_ESYNC

LL_CRS_EnableIT_SYNCOK

Function name

__STATIC_INLINE void LL_CRS_EnableIT_SYNCOK (void)

Function description

Enable SYNC event OK interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SYNCOKIE LL_CRS_EnableIT_SYNCOK

LL_CRS_DisableIT_SYNCOK

Function name

__STATIC_INLINE void LL_CRS_DisableIT_SYNCOK (void)

Function description

Disable SYNC event OK interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SYNCOKIE LL_CRS_DisableIT_SYNCOK

LL_CRS_IsEnabledIT_SYNCOK

Function name

__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_SYNCOK (void)

Function description

Check if SYNC event OK interrupt is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR SYNCOKIE LL_CRS_IsEnabledIT_SYNCOK

LL_CRS_EnableIT_SYNCWARN

Function name

__STATIC_INLINE void LL_CRS_EnableIT_SYNCWARN (void)

Function description

Enable SYNC warning interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SYNCWARNIE LL_CRS_EnableIT_SYNCWARN

LL_CRS_DisableIT_SYNCWARN

Function name

__STATIC_INLINE void LL_CRS_DisableIT_SYNCWARN (void)

Function description

Disable SYNC warning interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SYNCWARNIE LL_CRS_DisableIT_SYNCWARN

LL_CRS_IsEnabledIT_SYNCWARN

Function name

__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_SYNCWARN (void)

Function description

Check if SYNC warning interrupt is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR SYNCWARNIE LL_CRS_IsEnabledIT_SYNCWARN

LL_CRS_EnableIT_ERR

Function name

__STATIC_INLINE void LL_CRS_EnableIT_ERR (void)

Function description

Enable Synchronization or trimming error interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ERRIE LL_CRS_EnableIT_ERR

LL_CRS_DisableIT_ERR

Function name

__STATIC_INLINE void LL_CRS_DisableIT_ERR (void)

Function description

Disable Synchronization or trimming error interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ERRIE LL_CRS_DisableIT_ERR

LL_CRS_IsEnabledIT_ERR

Function name

__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_ERR (void)

Function description

Check if Synchronization or trimming error interrupt is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR ERRIE LL_CRS_IsEnabledIT_ERR

LL_CRS_EnableIT_ESYNC

Function name

__STATIC_INLINE void LL_CRS_EnableIT_ESYNC (void)

Function description

Enable Expected SYNC interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ESYNCIE LL_CRS_EnableIT_ESYNC

LL_CRS_DisableIT_ESYNC

Function name

__STATIC_INLINE void LL_CRS_DisableIT_ESYNC (void)

Function description

Disable Expected SYNC interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ESYNCIE LL_CRS_DisableIT_ESYNC

LL_CRS_IsEnabledIT_ESYNC

Function name

__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_ESYNC (void)

Function description

Check if Expected SYNC interrupt is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR ESYNCIE LL_CRS_IsEnabledIT_ESYNC

LL_CRS_DeInit

Function name

ErrorStatus LL_CRS_DeInit (void)

Function description

De-Initializes CRS peripheral registers to their default reset values.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: CRS registers are de-initialized
  - ERROR: not applicable

102.2 CRS Firmware driver defines

The following section lists the various define and macros of the module.

102.2.1 CRS

CRS

Default Values

LL_CRS_RELOADVALUE_DEFAULT

Notes:

- The reset value of the RELOAD field corresponds to a target frequency of 48 MHz and a synchronization signal frequency of 1 kHz (SOF signal from USB)

LL_CRS_ERRORLIMIT_DEFAULT

LL_CRS_HSI48CALIBRATION_DEFAULT

Notes:

- The default value is 64, which corresponds to the middle of the trimming interval. The trimming step is specified in the product datasheet. A higher TRIM value corresponds to a higher output frequency.

Frequency Error Direction

LL_CRS_FREQ_ERROR_DIR_UP

Upcounting direction, the actual frequency is above the target

LL_CRS_FREQ_ERROR_DIR_DOWN

Downcounting direction, the actual frequency is below the target

Get Flags Defines

LL_CRS_ISR_SYNCOKF

LL_CRS_ISR_SYNCWARNF

LL_CRS_ISR_ERRF

LL_CRS_ISR_ESYNCF

LL_CRS_ISR_SYNCERR

LL_CRS_ISR_SYNCMISS

LL_CRS_ISR_TRIMOVF

IT Defines

LL_CRS_CR_SYNCOKIE

LL_CRS_CR_SYNCWARNIE

LL_CRS_CR_ERRIE

LL_CRS_CR_ESYNCIE

Synchronization Signal Divider

LL_CRS_SYNC_DIV_1

Synchro Signal not divided (default)

LL_CRS_SYNC_DIV_2

Synchro Signal divided by 2

LL_CRS_SYNC_DIV_4

Synchro Signal divided by 4

LL_CRS_SYNC_DIV_8

Synchro Signal divided by 8

LL_CRS_SYNC_DIV_16

Synchro Signal divided by 16

LL_CRS_SYNC_DIV_32

Synchro Signal divided by 32

LL_CRS_SYNC_DIV_64

Synchro Signal divided by 64

LL_CRS_SYNC_DIV_128

Synchro Signal divided by 128

Synchronization Signal Polarity

LL_CRS_SYNC_POLARITY_RISING

Synchro Active on rising edge (default)

LL_CRS_SYNC_POLARITY_FALLING

Synchro Active on falling edge

Synchronization Signal Source

LL_CRS_SYNC_SOURCE_GPIO

Synchro Signal source GPIO

LL_CRS_SYNC_SOURCE_LSE

Synchro Signal source LSE

LL_CRS_SYNC_SOURCE_USB

Synchro Signal source USB SOF (default)

Exported_Macros_Calculate_Reload

__LL_CRS_CALC_CALCULATE_RELOADVALUE

Description:

- Macro to calculate reload value to be set in CRS register according to target and sync frequencies.

Parameters:

- __FTARGET__: Target frequency (value in Hz)
- __FSYNC__: Synchronization signal frequency (value in Hz)

Return value:

- Reload: value (in Hz)

Notes:

- The RELOAD value should be selected according to the ratio between the target frequency and the frequency of the synchronization source after prescaling. It is then decreased by one in order to reach the expected synchronization on the zero value. The formula is the following: RELOAD = (fTARGET / fSYNC) -1

Common Write and read registers Macros

LL_CRS_WriteReg

Description:

- Write a value in CRS register.

Parameters:

- __INSTANCE__: CRS Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_CRS_ReadReg

Description:

- Read a value in CRS register.

Parameters:

- __INSTANCE__: CRS Instance
- __REG__: Register to be read

Return value:

- Register: value

103 LL DAC Generic Driver

103.1 DAC Firmware driver registers structures

103.1.1 LL_DAC_InitTypeDef

LL_DAC_InitTypeDef is defined in the stm32h7xx_ll_dac.h

Data Fields

- ***uint32_t TriggerSource***
- ***uint32_t WaveAutoGeneration***
- ***uint32_t WaveAutoGenerationConfig***
- ***uint32_t OutputBuffer***
- ***uint32_t OutputConnection***
- ***uint32_t OutputMode***

Field Documentation

- ***uint32_t LL_DAC_InitTypeDef::TriggerSource***

  Set the conversion trigger source for the selected DAC channel: internal (SW start) or from external peripheral (timer event, external interrupt line). This parameter can be a value of ***DAC_LL_EC_TRIGGER_SOURCE***This feature can be modified afterwards using unitary function **LL_DAC_SetTriggerSource()**.
- ***uint32_t LL_DAC_InitTypeDef::WaveAutoGeneration***

  Set the waveform automatic generation mode for the selected DAC channel. This parameter can be a value of ***DAC_LL_EC_WAVE_AUTO_GENERATION_MODE***This feature can be modified afterwards using unitary function **LL_DAC_SetWaveAutoGeneration()**.
- ***uint32_t LL_DAC_InitTypeDef::WaveAutoGenerationConfig***

  Set the waveform automatic generation mode for the selected DAC channel. If waveform automatic generation mode is set to noise, this parameter can be a value of ***DAC_LL_EC_WAVE_NOISE_LFSR_UNMASK_BITS*** If waveform automatic generation mode is set to triangle, this parameter can be a value of ***DAC_LL_EC_WAVE_TRIANGLE_AMPLITUDE***

  Note:

  – If waveform automatic generation mode is disabled, this parameter is discarded.

  This feature can be modified afterwards using unitary function **LL_DAC_SetWaveNoiseLFSR()**, **LL_DAC_SetWaveTriangleAmplitude()** depending on the wave automatic generation selected.
- ***uint32_t LL_DAC_InitTypeDef::OutputBuffer***

  Set the output buffer for the selected DAC channel. This parameter can be a value of ***DAC_LL_EC_OUTPUT_BUFFER***This feature can be modified afterwards using unitary function **LL_DAC_SetOutputBuffer()**.
- ***uint32_t LL_DAC_InitTypeDef::OutputConnection***

  Set the output connection for the selected DAC channel. This parameter can be a value of ***DAC_LL_EC_OUTPUT_CONNECTION***This feature can be modified afterwards using unitary function **LL_DAC_SetOutputConnection()**.
- ***uint32_t LL_DAC_InitTypeDef::OutputMode***

  Set the output mode normal or sample-and-hold for the selected DAC channel. This parameter can be a value of ***DAC_LL_EC_OUTPUT_MODE***This feature can be modified afterwards using unitary function **LL_DAC_SetOutputMode()**.

103.2 DAC Firmware driver API description

The following section lists the various functions of the DAC library.

103.2.1 Detailed description of functions

LL_DAC_SetMode

Function name

__STATIC_INLINE void LL_DAC_SetMode (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t ChannelMode)

Function description

Set the operating mode for the selected DAC channel: calibration or normal operating mode.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **ChannelMode:** This parameter can be one of the following values:
  - LL_DAC_MODE_NORMAL_OPERATION
  - LL_DAC_MODE_CALIBRATION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CEN1 LL_DAC_SetMode
- CR CEN2 LL_DAC_SetMode

LL_DAC_GetMode

Function name

__STATIC_INLINE uint32_t LL_DAC_GetMode (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the operating mode for the selected DAC channel: calibration or normal operating mode.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_MODE_NORMAL_OPERATION
  - LL_DAC_MODE_CALIBRATION

Reference Manual to LL API cross reference:

- CR CEN1 LL_DAC_GetMode
- CR CEN2 LL_DAC_GetMode

LL_DAC_SetTrimmingValue

Function name

__STATIC_INLINE void LL_DAC_SetTrimmingValue (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t TrimmingValue)

Function description

Set the offset trimming value for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **TrimmingValue:** Value between Min_Data=0x00 and Max_Data=0x1F

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR OTRIM1 LL_DAC_SetTrimmingValue
- CCR OTRIM2 LL_DAC_SetTrimmingValue

LL_DAC_GetTrimmingValue

Function name

__STATIC_INLINE uint32_t LL_DAC_GetTrimmingValue (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the offset trimming value for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **TrimmingValue:** Value between Min_Data=0x00 and Max_Data=0x1F

Reference Manual to LL API cross reference:

- CCR OTRIM1 LL_DAC_GetTrimmingValue
- CCR OTRIM2 LL_DAC_GetTrimmingValue

LL_DAC_SetTriggerSource

Function name

__STATIC_INLINE void LL_DAC_SetTriggerSource (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t TriggerSource)

Function description

Set the conversion trigger source for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **TriggerSource:** This parameter can be one of the following values:
  - LL_DAC_TRIG_SOFTWARE
  - LL_DAC_TRIG_EXT_TIM1_TRGO
  - LL_DAC_TRIG_EXT_TIM2_TRGO
  - LL_DAC_TRIG_EXT_TIM4_TRGO
  - LL_DAC_TRIG_EXT_TIM5_TRGO
  - LL_DAC_TRIG_EXT_TIM6_TRGO
  - LL_DAC_TRIG_EXT_TIM7_TRGO
  - LL_DAC_TRIG_EXT_TIM8_TRGO
  - LL_DAC_TRIG_EXT_TIM15_TRGO
  - LL_DAC_TRIG_EXT_HRTIM_TRGO1 (1)
  - LL_DAC_TRIG_EXT_HRTIM_TRGO2 (1)
  - LL_DAC_TRIG_EXT_LPTIM1_OUT
  - LL_DAC_TRIG_EXT_LPTIM2_OUT
  - LL_DAC_TRIG_EXT_LPTIM3_OUT (2)
  - LL_DAC_TRIG_EXT_EXTI_LINE9
  - LL_DAC_TRIG_EXT_TIM23_TRGO (3)
  - LL_DAC_TRIG_EXT_TIM24_TRGO (4)

  (1) On this STM32 series, parameter not available on all devices. Only available if HRTIM feature is supported (refer to device datasheet for supported features list) (2) On this STM32 series, parameter only available on DAC2. (3) On this STM32 series, parameter not available on all devices. Only available if TIM23 feature is supported (refer to device datasheet for supported features list) (4) On this STM32 series, parameter not available on all devices. Only available if TIM24 feature is supported (refer to device datasheet for supported features list)

Return values

- **None:**

Notes

- For conversion trigger source to be effective, DAC trigger must be enabled using function LL_DAC_EnableTrigger().
- To set conversion trigger source, DAC channel must be disabled. Otherwise, the setting is discarded.
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.

Reference Manual to LL API cross reference:

- CR TSEL1 LL_DAC_SetTriggerSource
- CR TSEL2 LL_DAC_SetTriggerSource

LL_DAC_GetTriggerSource

Function name

__STATIC_INLINE uint32_t LL_DAC_GetTriggerSource (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the conversion trigger source for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_TRIG_SOFTWARE
  - LL_DAC_TRIG_EXT_TIM1_TRGO
  - LL_DAC_TRIG_EXT_TIM2_TRGO
  - LL_DAC_TRIG_EXT_TIM4_TRGO
  - LL_DAC_TRIG_EXT_TIM5_TRGO
  - LL_DAC_TRIG_EXT_TIM6_TRGO
  - LL_DAC_TRIG_EXT_TIM7_TRGO
  - LL_DAC_TRIG_EXT_TIM8_TRGO
  - LL_DAC_TRIG_EXT_TIM15_TRGO
  - LL_DAC_TRIG_EXT_HRTIM_TRGO1 (1)
  - LL_DAC_TRIG_EXT_HRTIM_TRGO2 (1)
  - LL_DAC_TRIG_EXT_LPTIM1_OUT
  - LL_DAC_TRIG_EXT_LPTIM2_OUT
  - LL_DAC_TRIG_EXT_LPTIM3_OUT (2)
  - LL_DAC_TRIG_EXT_EXTI_LINE9
  - LL_DAC_TRIG_EXT_TIM23_TRGO (3)
  - LL_DAC_TRIG_EXT_TIM24_TRGO (4)

  (1) On this STM32 series, parameter not available on all devices. Only available if HRTIM feature is supported (refer to device datasheet for supported features list) (2) On this STM32 series, parameter only available on DAC2. (3) On this STM32 series, parameter not available on all devices. Only available if TIM23 feature is supported (refer to device datasheet for supported features list) (4) On this STM32 series, parameter not available on all devices. Only available if TIM24 feature is supported (refer to device datasheet for supported features list)

Notes

- For conversion trigger source to be effective, DAC trigger must be enabled using function LL_DAC_EnableTrigger().
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.

Reference Manual to LL API cross reference:

- CR TSEL1 LL_DAC_GetTriggerSource
- CR TSEL2 LL_DAC_GetTriggerSource

LL_DAC_SetWaveAutoGeneration

Function name

__STATIC_INLINE void LL_DAC_SetWaveAutoGeneration (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t WaveAutoGeneration)

Function description

Set the waveform automatic generation mode for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **WaveAutoGeneration:** This parameter can be one of the following values:
  - LL_DAC_WAVE_AUTO_GENERATION_NONE
  - LL_DAC_WAVE_AUTO_GENERATION_NOISE
  - LL_DAC_WAVE_AUTO_GENERATION_TRIANGLE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR WAVE1 LL_DAC_SetWaveAutoGeneration
- CR WAVE2 LL_DAC_SetWaveAutoGeneration

LL_DAC_GetWaveAutoGeneration

Function name

__STATIC_INLINE uint32_t LL_DAC_GetWaveAutoGeneration (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the waveform automatic generation mode for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_WAVE_AUTO_GENERATION_NONE
  - LL_DAC_WAVE_AUTO_GENERATION_NOISE
  - LL_DAC_WAVE_AUTO_GENERATION_TRIANGLE

Reference Manual to LL API cross reference:

- CR WAVE1 LL_DAC_GetWaveAutoGeneration
- CR WAVE2 LL_DAC_GetWaveAutoGeneration

LL_DAC_SetWaveNoiseLFSR

Function name

__STATIC_INLINE void LL_DAC_SetWaveNoiseLFSR (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t NoiseLFSRMask)

Function description

Set the noise waveform generation for the selected DAC channel: Noise mode and parameters LFSR (linear feedback shift register).

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **NoiseLFSRMask:** This parameter can be one of the following values:
  - LL_DAC_NOISE_LFSR_UNMASK_BIT0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS1_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS2_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS3_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS4_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS5_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS6_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS7_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS8_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS9_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS10_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS11_0

Return values

- **None:**

Notes

- For wave generation to be effective, DAC channel wave generation mode must be enabled using function LL_DAC_SetWaveAutoGeneration().
- This setting can be set when the selected DAC channel is disabled (otherwise, the setting operation is ignored).

Reference Manual to LL API cross reference:

- CR MAMP1 LL_DAC_SetWaveNoiseLFSR
- CR MAMP2 LL_DAC_SetWaveNoiseLFSR

LL_DAC_GetWaveNoiseLFSR

Function name

__STATIC_INLINE uint32_t LL_DAC_GetWaveNoiseLFSR (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the noise waveform generation for the selected DAC channel: Noise mode and parameters LFSR (linear feedback shift register).

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_NOISE_LFSR_UNMASK_BIT0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS1_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS2_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS3_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS4_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS5_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS6_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS7_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS8_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS9_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS10_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS11_0

Reference Manual to LL API cross reference:

- CR MAMP1 LL_DAC_GetWaveNoiseLFSR
- CR MAMP2 LL_DAC_GetWaveNoiseLFSR

LL_DAC_SetWaveTriangleAmplitude

Function name

__STATIC_INLINE void LL_DAC_SetWaveTriangleAmplitude (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t TriangleAmplitude)

Function description

Set the triangle waveform generation for the selected DAC channel: triangle mode and amplitude.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **TriangleAmplitude:** This parameter can be one of the following values:
  - LL_DAC_TRIANGLE_AMPLITUDE_1
  - LL_DAC_TRIANGLE_AMPLITUDE_3
  - LL_DAC_TRIANGLE_AMPLITUDE_7
  - LL_DAC_TRIANGLE_AMPLITUDE_15
  - LL_DAC_TRIANGLE_AMPLITUDE_31
  - LL_DAC_TRIANGLE_AMPLITUDE_63
  - LL_DAC_TRIANGLE_AMPLITUDE_127
  - LL_DAC_TRIANGLE_AMPLITUDE_255
  - LL_DAC_TRIANGLE_AMPLITUDE_511
  - LL_DAC_TRIANGLE_AMPLITUDE_1023
  - LL_DAC_TRIANGLE_AMPLITUDE_2047
  - LL_DAC_TRIANGLE_AMPLITUDE_4095

Return values

- **None:**

Notes

- For wave generation to be effective, DAC channel wave generation mode must be enabled using function LL_DAC_SetWaveAutoGeneration().
- This setting can be set when the selected DAC channel is disabled (otherwise, the setting operation is ignored).

Reference Manual to LL API cross reference:

- CR MAMP1 LL_DAC_SetWaveTriangleAmplitude
- CR MAMP2 LL_DAC_SetWaveTriangleAmplitude

LL_DAC_GetWaveTriangleAmplitude

Function name

__STATIC_INLINE uint32_t LL_DAC_GetWaveTriangleAmplitude (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the triangle waveform generation for the selected DAC channel: triangle mode and amplitude.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_TRIANGLE_AMPLITUDE_1
  - LL_DAC_TRIANGLE_AMPLITUDE_3
  - LL_DAC_TRIANGLE_AMPLITUDE_7
  - LL_DAC_TRIANGLE_AMPLITUDE_15
  - LL_DAC_TRIANGLE_AMPLITUDE_31
  - LL_DAC_TRIANGLE_AMPLITUDE_63
  - LL_DAC_TRIANGLE_AMPLITUDE_127
  - LL_DAC_TRIANGLE_AMPLITUDE_255
  - LL_DAC_TRIANGLE_AMPLITUDE_511
  - LL_DAC_TRIANGLE_AMPLITUDE_1023
  - LL_DAC_TRIANGLE_AMPLITUDE_2047
  - LL_DAC_TRIANGLE_AMPLITUDE_4095

Reference Manual to LL API cross reference:

- CR MAMP1 LL_DAC_GetWaveTriangleAmplitude
- CR MAMP2 LL_DAC_GetWaveTriangleAmplitude

LL_DAC_ConfigOutput

Function name

__STATIC_INLINE void LL_DAC_ConfigOutput (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t OutputMode, uint32_t OutputBuffer, uint32_t OutputConnection)

Function description

Set the output for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **OutputMode:** This parameter can be one of the following values:
  - LL_DAC_OUTPUT_MODE_NORMAL
  - LL_DAC_OUTPUT_MODE_SAMPLE_AND_HOLD
- **OutputBuffer:** This parameter can be one of the following values:
  - LL_DAC_OUTPUT_BUFFER_ENABLE
  - LL_DAC_OUTPUT_BUFFER_DISABLE
- **OutputConnection:** This parameter can be one of the following values:
  - LL_DAC_OUTPUT_CONNECT_GPIO
  - LL_DAC_OUTPUT_CONNECT_INTERNAL

Return values

- **None:**

Notes

- This function set several features: mode normal or sample-and-holdbufferconnection to GPIO or internal path. These features can also be set individually using dedicated functions:LL_DAC_SetOutputBuffer()LL_DAC_SetOutputMode()LL_DAC_SetOutputConnection()
- On this STM32 series, output connection depends on output mode (normal or sample and hold) and output buffer state. if output connection is set to internal path and output buffer is enabled (whatever output mode): output connection is also connected to GPIO pin (both connections to GPIO pin and internal path).if output connection is set to GPIO pin, output buffer is disabled, output mode set to sample and hold: output connection is also connected to internal path (both connections to GPIO pin and internal path).
- Mode sample-and-hold requires an external capacitor to be connected between DAC channel output and ground. Capacitor value depends on load on DAC channel output and sample-and-hold timings configured. As indication, capacitor typical value is 100nF (refer to device datasheet, parameter "CSH").

Reference Manual to LL API cross reference:

- CR MODE1 LL_DAC_ConfigOutput
- CR MODE2 LL_DAC_ConfigOutput

LL_DAC_SetOutputMode

Function name

__STATIC_INLINE void LL_DAC_SetOutputMode (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t OutputMode)

Function description

Set the output mode normal or sample-and-hold for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **OutputMode:** This parameter can be one of the following values:
  - LL_DAC_OUTPUT_MODE_NORMAL
  - LL_DAC_OUTPUT_MODE_SAMPLE_AND_HOLD

Return values

- **None:**

Notes

- Mode sample-and-hold requires an external capacitor to be connected between DAC channel output and ground. Capacitor value depends on load on DAC channel output and sample-and-hold timings configured. As indication, capacitor typical value is 100nF (refer to device datasheet, parameter "CSH").

Reference Manual to LL API cross reference:

- CR MODE1 LL_DAC_SetOutputMode
- CR MODE2 LL_DAC_SetOutputMode

LL_DAC_GetOutputMode

Function name

__STATIC_INLINE uint32_t LL_DAC_GetOutputMode (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the output mode normal or sample-and-hold for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_OUTPUT_MODE_NORMAL
  - LL_DAC_OUTPUT_MODE_SAMPLE_AND_HOLD

Reference Manual to LL API cross reference:

- CR MODE1 LL_DAC_GetOutputMode
- CR MODE2 LL_DAC_GetOutputMode

LL_DAC_SetOutputBuffer

Function name

__STATIC_INLINE void LL_DAC_SetOutputBuffer (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t OutputBuffer)

Function description

Set the output buffer for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **OutputBuffer:** This parameter can be one of the following values:
  - LL_DAC_OUTPUT_BUFFER_ENABLE
  - LL_DAC_OUTPUT_BUFFER_DISABLE

Return values

- **None:**

Notes

- On this STM32 series, when buffer is enabled, its offset can be trimmed: factory calibration default values can be replaced by user trimming values, using function LL_DAC_SetTrimmingValue().

Reference Manual to LL API cross reference:

- CR MODE1 LL_DAC_SetOutputBuffer
- CR MODE2 LL_DAC_SetOutputBuffer

LL_DAC_GetOutputBuffer

Function name

__STATIC_INLINE uint32_t LL_DAC_GetOutputBuffer (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the output buffer state for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_OUTPUT_BUFFER_ENABLE
  - LL_DAC_OUTPUT_BUFFER_DISABLE

Reference Manual to LL API cross reference:

- CR MODE1 LL_DAC_GetOutputBuffer
- CR MODE2 LL_DAC_GetOutputBuffer

LL_DAC_SetOutputConnection

Function name

__STATIC_INLINE void LL_DAC_SetOutputConnection (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t OutputConnection)

Function description

Set the output connection for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **OutputConnection:** This parameter can be one of the following values:
  - LL_DAC_OUTPUT_CONNECT_GPIO
  - LL_DAC_OUTPUT_CONNECT_INTERNAL

Return values

- **None:**

Notes

- On this STM32 series, output connection depends on output mode (normal or sample and hold) and output buffer state. if output connection is set to internal path and output buffer is enabled (whatever output mode): output connection is also connected to GPIO pin (both connections to GPIO pin and internal path).if output connection is set to GPIO pin, output buffer is disabled, output mode set to sample and hold: output connection is also connected to internal path (both connections to GPIO pin and internal path).

Reference Manual to LL API cross reference:

- CR MODE1 LL_DAC_SetOutputConnection
- CR MODE2 LL_DAC_SetOutputConnection

LL_DAC_GetOutputConnection

Function name

__STATIC_INLINE uint32_t LL_DAC_GetOutputConnection (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the output connection for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_OUTPUT_CONNECT_GPIO
  - LL_DAC_OUTPUT_CONNECT_INTERNAL

Notes

- On this STM32 series, output connection depends on output mode (normal or sample and hold) and output buffer state. if output connection is set to internal path and output buffer is enabled (whatever output mode): output connection is also connected to GPIO pin (both connections to GPIO pin and internal path).if output connection is set to GPIO pin, output buffer is disabled, output mode set to sample and hold: output connection is also connected to internal path (both connections to GPIO pin and internal path).

Reference Manual to LL API cross reference:

- CR MODE1 LL_DAC_GetOutputConnection
- CR MODE2 LL_DAC_GetOutputConnection

LL_DAC_SetSampleAndHoldSampleTime

Function name

__STATIC_INLINE void LL_DAC_SetSampleAndHoldSampleTime (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t SampleTime)

Function description

Set the sample-and-hold timing for the selected DAC channel: sample time.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **SampleTime:** Value between Min_Data=0x000 and Max_Data=0x3FF

Return values

- **None:**

Notes

- Sample time must be set when DAC channel is disabled or during DAC operation when DAC channel flag BWSTx is reset, otherwise the setting is ignored. Check BWSTx flag state using function "LL_DAC_IsActiveFlag_BWSTx()".

Reference Manual to LL API cross reference:

- SHSR1 TSAMPLE1 LL_DAC_SetSampleAndHoldSampleTime
- SHSR2 TSAMPLE2 LL_DAC_SetSampleAndHoldSampleTime

LL_DAC_GetSampleAndHoldSampleTime

Function name

__STATIC_INLINE uint32_t LL_DAC_GetSampleAndHoldSampleTime (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the sample-and-hold timing for the selected DAC channel: sample time.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0x3FF

Reference Manual to LL API cross reference:

- SHSR1 TSAMPLE1 LL_DAC_GetSampleAndHoldSampleTime
- SHSR2 TSAMPLE2 LL_DAC_GetSampleAndHoldSampleTime

LL_DAC_SetSampleAndHoldHoldTime

Function name

__STATIC_INLINE void LL_DAC_SetSampleAndHoldHoldTime (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t HoldTime)

Function description

Set the sample-and-hold timing for the selected DAC channel: hold time.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **HoldTime:** Value between Min_Data=0x000 and Max_Data=0x3FF

Return values

- **None:**

Reference Manual to LL API cross reference:

- SHHR THOLD1 LL_DAC_SetSampleAndHoldHoldTime
- SHHR THOLD2 LL_DAC_SetSampleAndHoldHoldTime

LL_DAC_GetSampleAndHoldHoldTime

Function name

__STATIC_INLINE uint32_t LL_DAC_GetSampleAndHoldHoldTime (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the sample-and-hold timing for the selected DAC channel: hold time.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0x3FF

Reference Manual to LL API cross reference:

- SHHR THOLD1 LL_DAC_GetSampleAndHoldHoldTime
- SHHR THOLD2 LL_DAC_GetSampleAndHoldHoldTime

LL_DAC_SetSampleAndHoldRefreshTime

Function name

__STATIC_INLINE void LL_DAC_SetSampleAndHoldRefreshTime (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t RefreshTime)

Function description

Set the sample-and-hold timing for the selected DAC channel: refresh time.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **RefreshTime:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- SHRR TREFRESH1 LL_DAC_SetSampleAndHoldRefreshTime
- SHRR TREFRESH2 LL_DAC_SetSampleAndHoldRefreshTime

LL_DAC_GetSampleAndHoldRefreshTime

Function name

__STATIC_INLINE uint32_t LL_DAC_GetSampleAndHoldRefreshTime (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the sample-and-hold timing for the selected DAC channel: refresh time.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- SHRR TREFRESH1 LL_DAC_GetSampleAndHoldRefreshTime
- SHRR TREFRESH2 LL_DAC_GetSampleAndHoldRefreshTime

LL_DAC_EnableDMAReq

Function name

__STATIC_INLINE void LL_DAC_EnableDMAReq (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Enable DAC DMA transfer request of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **None:**

Notes

- To configure DMA source address (peripheral address), use function LL_DAC_DMA_GetRegAddr().

Reference Manual to LL API cross reference:

- CR DMAEN1 LL_DAC_EnableDMAReq
- CR DMAEN2 LL_DAC_EnableDMAReq

LL_DAC_DisableDMAReq

Function name

__STATIC_INLINE void LL_DAC_DisableDMAReq (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Disable DAC DMA transfer request of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **None:**

Notes

- To configure DMA source address (peripheral address), use function LL_DAC_DMA_GetRegAddr().

Reference Manual to LL API cross reference:

- CR DMAEN1 LL_DAC_DisableDMAReq
- CR DMAEN2 LL_DAC_DisableDMAReq

LL_DAC_IsDMAReqEnabled

Function name

__STATIC_INLINE uint32_t LL_DAC_IsDMAReqEnabled (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get DAC DMA transfer request state of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR DMAEN1 LL_DAC_IsDMAReqEnabled
- CR DMAEN2 LL_DAC_IsDMAReqEnabled

LL_DAC_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_DAC_DMA_GetRegAddr (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Register)

Function description

Function to help to configure DMA transfer to DAC: retrieve the DAC register address from DAC instance and a list of DAC registers intended to be used (most commonly) with DMA transfer.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **Register:** This parameter can be one of the following values:
  - LL_DAC_DMA_REG_DATA_12BITS_RIGHT_ALIGNED
  - LL_DAC_DMA_REG_DATA_12BITS_LEFT_ALIGNED
  - LL_DAC_DMA_REG_DATA_8BITS_RIGHT_ALIGNED

Return values

- **DAC:** register address

Notes

- These DAC registers are data holding registers: when DAC conversion is requested, DAC generates a DMA transfer request to have data available in DAC data holding registers.
- This macro is intended to be used with LL DMA driver, refer to function "LL_DMA_ConfigAddresses()". Example: LL_DMA_ConfigAddresses(DMA1, LL_DMA_CHANNEL_1, (uint32_t)&< array or variable >, LL_DAC_DMA_GetRegAddr(DAC1, LL_DAC_CHANNEL_1, LL_DAC_DMA_REG_DATA_12BITS_RIGHT_ALIGNED), LL_DMA_DIRECTION_MEMORY_TO_PERIPH);

Reference Manual to LL API cross reference:

- DHR12R1 DACC1DHR LL_DAC_DMA_GetRegAddr
- DHR12L1 DACC1DHR LL_DAC_DMA_GetRegAddr
- DHR8R1 DACC1DHR LL_DAC_DMA_GetRegAddr
- DHR12R2 DACC2DHR LL_DAC_DMA_GetRegAddr
- DHR12L2 DACC2DHR LL_DAC_DMA_GetRegAddr
- DHR8R2 DACC2DHR LL_DAC_DMA_GetRegAddr

LL_DAC_Enable

Function name

__STATIC_INLINE void LL_DAC_Enable (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Enable DAC selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **None:**

Notes

- After enable from off state, DAC channel requires a delay for output voltage to reach accuracy +/- 1 LSB. Refer to device datasheet, parameter "tWAKEUP".

Reference Manual to LL API cross reference:

- CR EN1 LL_DAC_Enable
- CR EN2 LL_DAC_Enable

LL_DAC_Disable

Function name

__STATIC_INLINE void LL_DAC_Disable (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Disable DAC selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR EN1 LL_DAC_Disable
- CR EN2 LL_DAC_Disable

LL_DAC_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_DAC_IsEnabled (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get DAC enable state of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR EN1 LL_DAC_IsEnabled
- CR EN2 LL_DAC_IsEnabled

LL_DAC_EnableTrigger

Function name

__STATIC_INLINE void LL_DAC_EnableTrigger (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Enable DAC trigger of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **None:**

Notes

- - If DAC trigger is disabled, DAC conversion is performed automatically once the data holding register is updated, using functions "LL_DAC_ConvertData{8; 12}{Right; Left} Aligned()": LL_DAC_ConvertData12RightAligned(), ... If DAC trigger is enabled, DAC conversion is performed only when a hardware of software trigger event is occurring. Select trigger source using function LL_DAC_SetTriggerSource().

Reference Manual to LL API cross reference:

- CR TEN1 LL_DAC_EnableTrigger
- CR TEN2 LL_DAC_EnableTrigger

LL_DAC_DisableTrigger

Function name

__STATIC_INLINE void LL_DAC_DisableTrigger (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Disable DAC trigger of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TEN1 LL_DAC_DisableTrigger
- CR TEN2 LL_DAC_DisableTrigger

LL_DAC_IsTriggerEnabled

Function name

__STATIC_INLINE uint32_t LL_DAC_IsTriggerEnabled (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get DAC trigger state of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR TEN1 LL_DAC_IsTriggerEnabled
- CR TEN2 LL_DAC_IsTriggerEnabled

LL_DAC_TrigSWConversion

Function name

__STATIC_INLINE void LL_DAC_TrigSWConversion (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Trig DAC conversion by software for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can a combination of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **None:**

Notes

- Preliminarily, DAC trigger must be set to software trigger using function LL_DAC_Init() LL_DAC_SetTriggerSource() with parameter "LL_DAC_TRIGGER_SOFTWARE". and DAC trigger must be enabled using function LL_DAC_EnableTrigger().
- For devices featuring DAC with 2 channels: this function can perform a SW start of both DAC channels simultaneously. Two channels can be selected as parameter. Example: (LL_DAC_CHANNEL_1 | LL_DAC_CHANNEL_2)

Reference Manual to LL API cross reference:

- SWTRIGR SWTRIG1 LL_DAC_TrigSWConversion
- SWTRIGR SWTRIG2 LL_DAC_TrigSWConversion

LL_DAC_ConvertData12RightAligned

Function name

__STATIC_INLINE void LL_DAC_ConvertData12RightAligned (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Data)

Function description

Set the data to be loaded in the data holding register in format 12 bits left alignment (LSB aligned on bit 0), for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **Data:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DHR12R1 DACC1DHR LL_DAC_ConvertData12RightAligned
- DHR12R2 DACC2DHR LL_DAC_ConvertData12RightAligned

LL_DAC_ConvertData12LeftAligned

Function name

__STATIC_INLINE void LL_DAC_ConvertData12LeftAligned (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Data)

Function description

Set the data to be loaded in the data holding register in format 12 bits left alignment (MSB aligned on bit 15), for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **Data:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DHR12L1 DACC1DHR LL_DAC_ConvertData12LeftAligned
- DHR12L2 DACC2DHR LL_DAC_ConvertData12LeftAligned

LL_DAC_ConvertData8RightAligned

Function name

__STATIC_INLINE void LL_DAC_ConvertData8RightAligned (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Data)

Function description

Set the data to be loaded in the data holding register in format 8 bits left alignment (LSB aligned on bit 0), for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **Data:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DHR8R1 DACC1DHR LL_DAC_ConvertData8RightAligned
- DHR8R2 DACC2DHR LL_DAC_ConvertData8RightAligned

LL_DAC_ConvertDualData12RightAligned

Function name

__STATIC_INLINE void LL_DAC_ConvertDualData12RightAligned (DAC_TypeDef * DACx, uint32_t DataChannel1, uint32_t DataChannel2)

Function description

Set the data to be loaded in the data holding register in format 12 bits left alignment (LSB aligned on bit 0), for both DAC channels.

Parameters

- **DACx:** DAC instance
- **DataChannel1:** Value between Min_Data=0x000 and Max_Data=0xFFF
- **DataChannel2:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DHR12RD DACC1DHR LL_DAC_ConvertDualData12RightAligned
- DHR12RD DACC2DHR LL_DAC_ConvertDualData12RightAligned

LL_DAC_ConvertDualData12LeftAligned

Function name

__STATIC_INLINE void LL_DAC_ConvertDualData12LeftAligned (DAC_TypeDef * DACx, uint32_t DataChannel1, uint32_t DataChannel2)

Function description

Set the data to be loaded in the data holding register in format 12 bits left alignment (MSB aligned on bit 15), for both DAC channels.

Parameters

- **DACx:** DAC instance
- **DataChannel1:** Value between Min_Data=0x000 and Max_Data=0xFFF
- **DataChannel2:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DHR12LD DACC1DHR LL_DAC_ConvertDualData12LeftAligned
- DHR12LD DACC2DHR LL_DAC_ConvertDualData12LeftAligned

LL_DAC_ConvertDualData8RightAligned

Function name

__STATIC_INLINE void LL_DAC_ConvertDualData8RightAligned (DAC_TypeDef * DACx, uint32_t DataChannel1, uint32_t DataChannel2)

Function description

Set the data to be loaded in the data holding register in format 8 bits left alignment (LSB aligned on bit 0), for both DAC channels.

Parameters

- **DACx:** DAC instance
- **DataChannel1:** Value between Min_Data=0x00 and Max_Data=0xFF
- **DataChannel2:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DHR8RD DACC1DHR LL_DAC_ConvertDualData8RightAligned
- DHR8RD DACC2DHR LL_DAC_ConvertDualData8RightAligned

LL_DAC_RetrieveOutputData

Function name

__STATIC_INLINE uint32_t LL_DAC_RetrieveOutputData (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Retrieve output data currently generated for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0xFFF

Notes

- Whatever alignment and resolution settings (using functions "LL_DAC_ConvertData{8; 12}{Right; Left} Aligned()": LL_DAC_ConvertData12RightAligned(), ...), output data format is 12 bits right aligned (LSB aligned on bit 0).

Reference Manual to LL API cross reference:

- DOR1 DACC1DOR LL_DAC_RetrieveOutputData
- DOR2 DACC2DOR LL_DAC_RetrieveOutputData

LL_DAC_IsActiveFlag_CAL1

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_CAL1 (DAC_TypeDef * DACx)

Function description

Get DAC calibration offset flag for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CAL_FLAG1 LL_DAC_IsActiveFlag_CAL1

LL_DAC_IsActiveFlag_CAL2

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_CAL2 (DAC_TypeDef * DACx)

Function description

Get DAC calibration offset flag for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CAL_FLAG2 LL_DAC_IsActiveFlag_CAL2

LL_DAC_IsActiveFlag_BWST1

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_BWST1 (DAC_TypeDef * DACx)

Function description

Get DAC busy writing sample time flag for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BWST1 LL_DAC_IsActiveFlag_BWST1

LL_DAC_IsActiveFlag_BWST2

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_BWST2 (DAC_TypeDef * DACx)

Function description

Get DAC busy writing sample time flag for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BWST2 LL_DAC_IsActiveFlag_BWST2

LL_DAC_IsActiveFlag_DMAUDR1

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_DMAUDR1 (DAC_TypeDef * DACx)

Function description

Get DAC underrun flag for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR DMAUDR1 LL_DAC_IsActiveFlag_DMAUDR1

LL_DAC_IsActiveFlag_DMAUDR2

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_DMAUDR2 (DAC_TypeDef * DACx)

Function description

Get DAC underrun flag for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR DMAUDR2 LL_DAC_IsActiveFlag_DMAUDR2

LL_DAC_ClearFlag_DMAUDR1

Function name

__STATIC_INLINE void LL_DAC_ClearFlag_DMAUDR1 (DAC_TypeDef * DACx)

Function description

Clear DAC underrun flag for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR DMAUDR1 LL_DAC_ClearFlag_DMAUDR1

LL_DAC_ClearFlag_DMAUDR2

Function name

__STATIC_INLINE void LL_DAC_ClearFlag_DMAUDR2 (DAC_TypeDef * DACx)

Function description

Clear DAC underrun flag for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR DMAUDR2 LL_DAC_ClearFlag_DMAUDR2

LL_DAC_EnableIT_DMAUDR1

Function name

__STATIC_INLINE void LL_DAC_EnableIT_DMAUDR1 (DAC_TypeDef * DACx)

Function description

Enable DMA underrun interrupt for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAUDRIE1 LL_DAC_EnableIT_DMAUDR1

LL_DAC_EnableIT_DMAUDR2

Function name

__STATIC_INLINE void LL_DAC_EnableIT_DMAUDR2 (DAC_TypeDef * DACx)

Function description

Enable DMA underrun interrupt for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAUDRIE2 LL_DAC_EnableIT_DMAUDR2

LL_DAC_DisableIT_DMAUDR1

Function name

__STATIC_INLINE void LL_DAC_DisableIT_DMAUDR1 (DAC_TypeDef * DACx)

Function description

Disable DMA underrun interrupt for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAUDRIE1 LL_DAC_DisableIT_DMAUDR1

LL_DAC_DisableIT_DMAUDR2

Function name

__STATIC_INLINE void LL_DAC_DisableIT_DMAUDR2 (DAC_TypeDef * DACx)

Function description

Disable DMA underrun interrupt for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAUDRIE2 LL_DAC_DisableIT_DMAUDR2

LL_DAC_IsEnabledIT_DMAUDR1

Function name

__STATIC_INLINE uint32_t LL_DAC_IsEnabledIT_DMAUDR1 (DAC_TypeDef * DACx)

Function description

Get DMA underrun interrupt for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR DMAUDRIE1 LL_DAC_IsEnabledIT_DMAUDR1

LL_DAC_IsEnabledIT_DMAUDR2

Function name

__STATIC_INLINE uint32_t LL_DAC_IsEnabledIT_DMAUDR2 (DAC_TypeDef * DACx)

Function description

Get DMA underrun interrupt for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR DMAUDRIE2 LL_DAC_IsEnabledIT_DMAUDR2

LL_DAC_DeInit

Function name

ErrorStatus LL_DAC_DeInit (DAC_TypeDef * DACx)

Function description

De-initialize registers of the selected DAC instance to their default reset values.

Parameters

- **DACx:** DAC instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: DAC registers are de-initialized
  - ERROR: not applicable

LL_DAC_Init

Function name

ErrorStatus LL_DAC_Init (DAC_TypeDef * DACx, uint32_t DAC_Channel, LL_DAC_InitTypeDef * DAC_InitStruct)

Function description

Initialize some features of DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **DAC_InitStruct:** Pointer to a LL_DAC_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: DAC registers are initialized
  - ERROR: DAC registers are not initialized

Notes

- LL_DAC_Init() aims to ease basic configuration of a DAC channel. Leaving it ready to be enabled and output: a level by calling one of LL_DAC_ConvertData12RightAligned LL_DAC_ConvertData12LeftAligned LL_DAC_ConvertData8RightAligned or one of the supported autogenerated wave.
- This function allows configuration of: Output modeTriggerWave generation
- The setting of these parameters by function LL_DAC_Init() is conditioned to DAC state: DAC channel must be disabled.

LL_DAC_StructInit

Function name

void LL_DAC_StructInit (LL_DAC_InitTypeDef * DAC_InitStruct)

Function description

Set each LL_DAC_InitTypeDef field to default value.

Parameters

- **DAC_InitStruct:** pointer to a LL_DAC_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

103.3 DAC Firmware driver defines

The following section lists the various define and macros of the module.

103.3.1 DAC

DAC

DAC channels

LL_DAC_CHANNEL_1

DAC channel 1

LL_DAC_CHANNEL_2

DAC channel 2

DAC flags

LL_DAC_FLAG_DMAUDR1

DAC channel 1 flag DMA underrun

LL_DAC_FLAG_CAL1

DAC channel 1 flag offset calibration status

LL_DAC_FLAG_BWST1

DAC channel 1 flag busy writing sample time

LL_DAC_FLAG_DMAUDR2

DAC channel 2 flag DMA underrun

LL_DAC_FLAG_CAL2

DAC channel 2 flag offset calibration status

LL_DAC_FLAG_BWST2

DAC channel 2 flag busy writing sample time

Definitions of DAC hardware constraints delays

LL_DAC_DELAY_STARTUP_VOLTAGE_SETTLING_US

Delay for DAC channel voltage settling time from DAC channel startup (transition from disable to enable)

LL_DAC_DELAY_VOLTAGE_SETTLING_US

Delay for DAC channel voltage settling time

DAC interruptions

LL_DAC_IT_DMAUDRIE1

DAC channel 1 interruption DMA underrun

LL_DAC_IT_DMAUDRIE2

DAC channel 2 interruption DMA underrun

DAC literals legacy naming

LL_DAC_TRIGGER_SOFTWARE

LL_DAC_TRIGGER_TIM2_TRGO

LL_DAC_TRIGGER_TIM4_TRGO

LL_DAC_TRIGGER_TIM6_TRGO

LL_DAC_TRIGGER_TIM7_TRGO

LL_DAC_TRIGGER_TIM8_TRGO

LL_DAC_TRIGGER_EXT_IT9

LL_DAC_WAVEGENERATION_NONE

LL_DAC_WAVEGENERATION_NOISE

LL_DAC_WAVEGENERATION_TRIANGLE

LL_DAC_CONNECT_GPIO

LL_DAC_CONNECT_INTERNAL

DAC operating mode

LL_DAC_MODE_NORMAL_OPERATION

DAC channel in mode normal operation

LL_DAC_MODE_CALIBRATION

DAC channel in mode calibration

DAC channel output buffer

LL_DAC_OUTPUT_BUFFER_ENABLE

The selected DAC channel output is buffered: higher drive current capability, but also higher current consumption

LL_DAC_OUTPUT_BUFFER_DISABLE

The selected DAC channel output is not buffered: lower drive current capability, but also lower current consumption

DAC channel output connection

LL_DAC_OUTPUT_CONNECT_GPIO

The selected DAC channel output is connected to external pin

LL_DAC_OUTPUT_CONNECT_INTERNAL

The selected DAC channel output is connected to on-chip peripherals via internal paths. On this STM32 series, output connection depends on output mode (normal or sample and hold) and output buffer state. Refer to comments of function

DAC channel output mode

LL_DAC_OUTPUT_MODE_NORMAL

The selected DAC channel output is on mode normal.

LL_DAC_OUTPUT_MODE_SAMPLE_AND_HOLD

The selected DAC channel output is on mode sample-and-hold. Mode sample-and-hold requires an external capacitor, refer to description of function

DAC registers compliant with specific purpose

LL_DAC_DMA_REG_DATA_12BITS_RIGHT_ALIGNED

DAC channel data holding register 12 bits right aligned

LL_DAC_DMA_REG_DATA_12BITS_LEFT_ALIGNED

DAC channel data holding register 12 bits left aligned

LL_DAC_DMA_REG_DATA_8BITS_RIGHT_ALIGNED

DAC channel data holding register 8 bits right aligned

DAC channel output resolution

LL_DAC_RESOLUTION_12B

DAC channel resolution 12 bits

LL_DAC_RESOLUTION_8B

DAC channel resolution 8 bits

DAC trigger source

LL_DAC_TRIG_SOFTWARE

DAC channel conversion trigger internal (SW start)

LL_DAC_TRIG_EXT_TIM1_TRGO

DAC channel conversion trigger from external peripheral: TIM1 TRGO.

LL_DAC_TRIG_EXT_TIM2_TRGO

DAC channel conversion trigger from external peripheral: TIM2 TRGO.

LL_DAC_TRIG_EXT_TIM4_TRGO

DAC channel conversion trigger from external peripheral: TIM4 TRGO.

LL_DAC_TRIG_EXT_TIM5_TRGO

DAC channel conversion trigger from external peripheral: TIM5 TRGO.

LL_DAC_TRIG_EXT_TIM6_TRGO

DAC channel conversion trigger from external peripheral: TIM6 TRGO.

LL_DAC_TRIG_EXT_TIM7_TRGO

DAC channel conversion trigger from external peripheral: TIM7 TRGO.

LL_DAC_TRIG_EXT_TIM8_TRGO

DAC channel conversion trigger from external peripheral: TIM8 TRGO.

LL_DAC_TRIG_EXT_TIM15_TRGO

DAC channel conversion trigger from external peripheral: TIM15 TRGO.

LL_DAC_TRIG_EXT_HRTIM_TRGO1

HR1 TRGO1 selected as external conversion trigger for DAC channel 1

LL_DAC_TRIG_EXT_HRTIM_TRGO2

HR1 TRGO2 selected as external conversion trigger for DAC channel 2

LL_DAC_TRIG_EXT_LPTIM1_OUT

DAC channel conversion trigger from external peripheral: LPTIM1 TRGO.

LL_DAC_TRIG_EXT_LPTIM2_OUT

DAC channel conversion trigger from external peripheral: LPTIM2 TRGO.

LL_DAC_TRIG_EXT_EXTI_LINE9

DAC channel conversion trigger from external peripheral: external interrupt line 9.

DAC waveform automatic generation mode

LL_DAC_WAVE_AUTO_GENERATION_NONE

DAC channel wave auto generation mode disabled.

LL_DAC_WAVE_AUTO_GENERATION_NOISE

DAC channel wave auto generation mode enabled, set generated noise waveform.

LL_DAC_WAVE_AUTO_GENERATION_TRIANGLE

DAC channel wave auto generation mode enabled, set generated triangle waveform.

DAC wave generation - Noise LFSR unmask bits

LL_DAC_NOISE_LFSR_UNMASK_BIT0

Noise wave generation, unmask LFSR bit0, for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS1_0

Noise wave generation, unmask LFSR bits[1:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS2_0

Noise wave generation, unmask LFSR bits[2:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS3_0

Noise wave generation, unmask LFSR bits[3:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS4_0

Noise wave generation, unmask LFSR bits[4:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS5_0

Noise wave generation, unmask LFSR bits[5:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS6_0

Noise wave generation, unmask LFSR bits[6:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS7_0

Noise wave generation, unmask LFSR bits[7:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS8_0

Noise wave generation, unmask LFSR bits[8:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS9_0

Noise wave generation, unmask LFSR bits[9:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS10_0

Noise wave generation, unmask LFSR bits[10:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS11_0

Noise wave generation, unmask LFSR bits[11:0], for the selected DAC channel

DAC wave generation - Triangle amplitude

LL_DAC_TRIANGLE_AMPLITUDE_1

Triangle wave generation, amplitude of 1 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_3

Triangle wave generation, amplitude of 3 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_7

Triangle wave generation, amplitude of 7 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_15

Triangle wave generation, amplitude of 15 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_31

Triangle wave generation, amplitude of 31 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_63

Triangle wave generation, amplitude of 63 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_127

Triangle wave generation, amplitude of 127 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_255

Triangle wave generation, amplitude of 255 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_511

Triangle wave generation, amplitude of 512 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_1023

Triangle wave generation, amplitude of 1023 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_2047

Triangle wave generation, amplitude of 2047 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_4095

Triangle wave generation, amplitude of 4095 LSB of DAC output range, for the selected DAC channel

DAC helper macro

__LL_DAC_CHANNEL_TO_DECIMAL_NB

Description:

- Helper macro to get DAC channel number in decimal format from literals LL_DAC_CHANNEL_x.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return value:

- 1...2

Notes:

- The input can be a value from functions where a channel number is returned.

__LL_DAC_DECIMAL_NB_TO_CHANNEL

Description:

- Helper macro to get DAC channel in literal format LL_DAC_CHANNEL_x from number in decimal format.

Parameters:

- __DECIMAL_NB__: 1...2

Return value:

- Returned: value can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Notes:

- If the input parameter does not correspond to a DAC channel, this macro returns value '0'.

__LL_DAC_DIGITAL_SCALE

Description:

- Helper macro to define the DAC conversion data full-scale digital value corresponding to the selected DAC resolution.

Parameters:

- __DAC_RESOLUTION__: This parameter can be one of the following values:
  - LL_DAC_RESOLUTION_12B
  - LL_DAC_RESOLUTION_8B

Return value:

- ADC: conversion data equivalent voltage value (unit: mVolt)

Notes:

- DAC conversion data full-scale corresponds to voltage range determined by analog voltage references Vref+ and Vref- (refer to reference manual).

__LL_DAC_CALC_VOLTAGE_TO_DATA

Description:

- Helper macro to calculate the DAC conversion data (unit: digital value) corresponding to a voltage (unit: mVolt).

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __DAC_VOLTAGE__: Voltage to be generated by DAC channel (unit: mVolt).
- __DAC_RESOLUTION__: This parameter can be one of the following values:
  - LL_DAC_RESOLUTION_12B
  - LL_DAC_RESOLUTION_8B

Return value:

- DAC: conversion data (unit: digital value)

Notes:

- This helper macro is intended to provide input data in voltage rather than digital value, to be used with LL DAC functions such as LL_DAC_ConvertData12RightAligned(). Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE().

Common write and read registers macros

LL_DAC_WriteReg

Description:

- Write a value in DAC register.

Parameters:

- __INSTANCE__: DAC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_DAC_ReadReg

Description:

- Read a value in DAC register.

Parameters:

- __INSTANCE__: DAC Instance
- __REG__: Register to be read

Return value:

- Register: value

104 LL DELAYBLOCK Generic Driver

104.1 DELAYBLOCK Firmware driver API description

The following section lists the various functions of the DELAYBLOCK library.

104.1.1 DelayBlock peripheral features

The Delay block is used to generate an Output clock which is de-phased from the Input clock. The phase of the Output clock is programmed by FW. The Output clock is then used to clock the receive data in i.e. a SDMMC or QSPI interface. The delay is Voltage and Temperature dependent, which may require FW to do re-tuning and recenter the Output clock phase to the receive data.

The Delay Block features include the following:

- Input clock frequency range 25MHz to 208MHz.
- Up to 12 oversampling phases.

104.1.2 How to use this driver

This driver is a considered as a driver of service for external devices drivers that interfaces with the DELAY peripheral. The DelayBlock_Enable() function, enables the DelayBlock instance, configure the delay line length and configure the Output clock phase. The DelayBlock_Disable() function, disables the DelayBlock instance by setting DEN flag to 0.

104.1.3 Initialization and de-initialization functions

This section provides functions allowing to:

This section contains the following APIs:

- ***DelayBlock_Enable()***
- ***DelayBlock_Disable()***
- ***DelayBlock_Configure()***

104.1.4 Detailed description of functions

DelayBlock_Enable

Function name

HAL_StatusTypeDef DelayBlock_Enable (DLYB_TypeDef * DLYBx)

Function description

Enable the Delay Block instance.

Parameters

- **DLYBx:** Pointer to DLYB instance.

Return values

- **HAL:** status

DelayBlock_Disable

Function name

HAL_StatusTypeDef DelayBlock_Disable (DLYB_TypeDef * DLYBx)

Function description

Disable the Delay Block instance.

Parameters

- **DLYBx:** Pointer to DLYB instance.

Return values

- **HAL:** status

DelayBlock_Configure

Function name

HAL_StatusTypeDef DelayBlock_Configure (DLYB_TypeDef * DLYBx, uint32_t PhaseSel, uint32_t Units)

Function description

Configure the Delay Block instance.

Parameters

- **DLYBx:** Pointer to DLYB instance.
- **PhaseSel:** Phase selection [0..11].
- **Units:** Delay units[0..127].

Return values

- **HAL:** status

104.2 DELAYBLOCK Firmware driver defines

The following section lists the various define and macros of the module.

104.2.1 DELAYBLOCK

DELAYBLOCK

105 LL DMA2D Generic Driver

105.1 DMA2D Firmware driver registers structures

105.1.1 LL_DMA2D_InitTypeDef

LL_DMA2D_InitTypeDef is defined in the stm32h7xx_ll_dma2d.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t ColorMode***
- ***uint32_t OutputBlue***
- ***uint32_t OutputGreen***
- ***uint32_t OutputRed***
- ***uint32_t OutputAlpha***
- ***uint32_t OutputMemoryAddress***
- ***uint32_t OutputSwapMode***
- ***uint32_t LineOffsetMode***
- ***uint32_t LineOffset***
- ***uint32_t NbrOfLines***
- ***uint32_t NbrOfPixelsPerLines***
- ***uint32_t AlphaInversionMode***
- ***uint32_t RBSwapMode***

Field Documentation

- ***uint32_t LL_DMA2D_InitTypeDef::Mode***

  Specifies the DMA2D transfer mode.
  - This parameter can be one value of ***DMA2D_LL_EC_MODE***.

  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetMode()**.
- ***uint32_t LL_DMA2D_InitTypeDef::ColorMode***

  Specifies the color format of the output image.
  - This parameter can be one value of ***DMA2D_LL_EC_OUTPUT_COLOR_MODE***.

  This parameter can be modified afterwards using, unitary function **LL_DMA2D_SetOutputColorMode()**.
- ***uint32_t LL_DMA2D_InitTypeDef::OutputBlue***

  Specifies the Blue value of the output image.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if RGB888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x1F if RGB565 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x1F if ARGB1555 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected.

  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.

- ***uint32_t LL_DMA2D_InitTypeDef::OutputGreen***

  Specifies the Green value of the output image.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if RGB888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x3F if RGB565 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x1F if ARGB1555 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected.

  This parameter can be modified afterwards using unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.
- ***uint32_t LL_DMA2D_InitTypeDef::OutputRed***

  Specifies the Red value of the output image.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if RGB888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x1F if RGB565 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x1F if ARGB1555 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected.

  This parameter can be modified afterwards using unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.
- ***uint32_t LL_DMA2D_InitTypeDef::OutputAlpha***

  Specifies the Alpha channel of the output image.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x01 if ARGB1555 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected.
  - This parameter is not considered if RGB888 or RGB565 color mode is selected.

  This parameter can be modified afterwards using, unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.
- ***uint32_t LL_DMA2D_InitTypeDef::OutputMemoryAddress***

  Specifies the memory address.
  - This parameter must be a number between: Min_Data = 0x0000 and Max_Data = 0xFFFFFFFF.

  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetOutputMemAddr()**.
- ***uint32_t LL_DMA2D_InitTypeDef::OutputSwapMode***

  Specifies the output swap mode color format of the output image.
  - This parameter can be one value of ***DMA2D_LL_EC_OUTPUT_SWAP_MODE***.

  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetOutputSwapMode()**.
- ***uint32_t LL_DMA2D_InitTypeDef::LineOffsetMode***

  Specifies the output line offset mode.
  - This parameter can be one value of ***DMA2D_LL_EC_LINE_OFFSET_MODE***.

  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetLineOffsetMode()**.

- ***uint32_t LL_DMA2D_InitTypeDef::LineOffset***
  Specifies the output line offset value.
  – This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.
  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetLineOffset()**.
- ***uint32_t LL_DMA2D_InitTypeDef::NbrOfLines***
  Specifies the number of lines of the area to be transferred.
  – This parameter must be a number between: Min_Data = 0x0000 and Max_Data = 0xFFFF.
  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetNbrOfLines()**.
- ***uint32_t LL_DMA2D_InitTypeDef::NbrOfPixelsPerLines***
  Specifies the number of pixels per lines of the area to be transferred.
  – This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x3FFF.
  This parameter can be modified afterwards using, unitary function **LL_DMA2D_SetNbrOfPixelsPerLines()**.
- ***uint32_t LL_DMA2D_InitTypeDef::AlphaInversionMode***
  Specifies the output alpha inversion mode.
  – This parameter can be one value of ***DMA2D_LL_EC_ALPHA_INVERSION***.
  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetOutputAlphaInvMode()**.
- ***uint32_t LL_DMA2D_InitTypeDef::RBSwapMode***
  Specifies the output Red Blue swap mode.
  – This parameter can be one value of ***DMA2D_LL_EC_RED_BLUE_SWAP***.
  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetOutputRBSwapMode()**.

105.1.2 LL_DMA2D_LayerCfgTypeDef

LL_DMA2D_LayerCfgTypeDef is defined in the stm32h7xx_ll_dma2d.h

Data Fields

- ***uint32_t MemoryAddress***
- ***uint32_t LineOffset***
- ***uint32_t ColorMode***
- ***uint32_t CLUTColorMode***
- ***uint32_t CLUTSize***
- ***uint32_t AlphaMode***
- ***uint32_t Alpha***
- ***uint32_t Blue***
- ***uint32_t Green***
- ***uint32_t Red***
- ***uint32_t CLUTMemoryAddress***
- ***uint32_t AlphaInversionMode***
- ***uint32_t RBSwapMode***
- ***uint32_t ChromaSubSampling***

Field Documentation

- ***uint32_t LL_DMA2D_LayerCfgTypeDef::MemoryAddress***
  Specifies the foreground or background memory address.
  – This parameter must be a number between: Min_Data = 0x0000 and Max_Data = 0xFFFFFFFF.
  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetMemAddr()** for foreground layer,
  – **LL_DMA2D_BGND_SetMemAddr()** for background layer.

- ***uint32_t LL_DMA2D_LayerCfgTypeDef::LineOffset***
  Specifies the foreground or background line offset value.
  – This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x3FFF.

  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetLineOffset()** for foreground layer,
  – **LL_DMA2D_BGND_SetLineOffset()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::ColorMode***
  Specifies the foreground or background color mode.
  – This parameter can be one value of ***DMA2D_LL_EC_INPUT_COLOR_MODE***.

  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetColorMode()** for foreground layer,
  – **LL_DMA2D_BGND_SetColorMode()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::CLUTColorMode***
  Specifies the foreground or background CLUT color mode.
  – This parameter can be one value of ***DMA2D_LL_EC_CLUT_COLOR_MODE***.

  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetCLUTColorMode()** for foreground layer,
  – **LL_DMA2D_BGND_SetCLUTColorMode()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::CLUTSize***
  Specifies the foreground or background CLUT size.
  – This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.

  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetCLUTSize()** for foreground layer,
  – **LL_DMA2D_BGND_SetCLUTSize()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::AlphaMode***
  Specifies the foreground or background alpha mode.
  – This parameter can be one value of ***DMA2D_LL_EC_ALPHA_MODE***.

  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetAlphaMode()** for foreground layer,
  – **LL_DMA2D_BGND_SetAlphaMode()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::Alpha***
  Specifies the foreground or background Alpha value.
  – This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.

  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetAlpha()** for foreground layer,
  – **LL_DMA2D_BGND_SetAlpha()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::Blue***
  Specifies the foreground or background Blue color value.
  – This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.

  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetBlueColor()** for foreground layer,
  – **LL_DMA2D_BGND_SetBlueColor()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::Green***
  Specifies the foreground or background Green color value.
  – This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.

  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetGreenColor()** for foreground layer,
  – **LL_DMA2D_BGND_SetGreenColor()** for background layer.

- ***uint32_t LL_DMA2D_LayerCfgTypeDef::Red***

  Specifies the foreground or background Red color value.
  - This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.

  This parameter can be modified afterwards using unitary functions
  - **LL_DMA2D_FGND_SetRedColor()** for foreground layer,
  - **LL_DMA2D_BGND_SetRedColor()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::CLUTMemoryAddress***

  Specifies the foreground or background CLUT memory address.
  - This parameter must be a number between: Min_Data = 0x0000 and Max_Data = 0xFFFFFFFF.

  This parameter can be modified afterwards using unitary functions
  - **LL_DMA2D_FGND_SetCLUTMemAddr()** for foreground layer,
  - **LL_DMA2D_BGND_SetCLUTMemAddr()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::AlphaInversionMode***

  Specifies the foreground or background alpha inversion mode.
  - This parameter can be one value of ***DMA2D_LL_EC_ALPHA_INVERSION***.

  This parameter can be modified afterwards using unitary functions
  - **LL_DMA2D_FGND_SetAlphaInvMode()** for foreground layer,
  - **LL_DMA2D_BGND_SetAlphaInvMode()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::RBSwapMode***

  Specifies the foreground or background Red Blue swap mode. This parameter can be one value of ***DMA2D_LL_EC_RED_BLUE_SWAP*** .This parameter can be modified afterwards using unitary functions
  - **LL_DMA2D_FGND_SetRBSwapMode()** for foreground layer,
  - **LL_DMA2D_BGND_SetRBSwapMode()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::ChromaSubSampling***

  Configure the chroma sub-sampling mode for the YCbCr color mode This parameter is applicable for foreground layer only. This parameter can be one value of ***DMA2D_LL_CHROMA_SUB_SAMPLING***This parameter can be modified afterwards using unitary functions
  - **LL_DMA2D_FGND_SetChrSubSampling()** for foreground layer.

105.1.3 LL_DMA2D_ColorTypeDef

LL_DMA2D_ColorTypeDef is defined in the stm32h7xx_ll_dma2d.h

Data Fields

- ***uint32_t ColorMode***
- ***uint32_t OutputBlue***
- ***uint32_t OutputGreen***
- ***uint32_t OutputRed***
- ***uint32_t OutputAlpha***

Field Documentation

- ***uint32_t LL_DMA2D_ColorTypeDef::ColorMode***

  Specifies the color format of the output image.
  - This parameter can be one value of ***DMA2D_LL_EC_OUTPUT_COLOR_MODE***.

  This parameter can be modified afterwards using unitary function **LL_DMA2D_SetOutputColorMode()**.

- ***uint32_t LL_DMA2D_ColorTypeDef::OutputBlue***

  Specifies the Blue value of the output image.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if RGB888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x1F if RGB565 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x1F if ARGB1555 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected.

  This parameter can be modified afterwards using, unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.
- ***uint32_t LL_DMA2D_ColorTypeDef::OutputGreen***

  Specifies the Green value of the output image.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.
  - This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF if RGB888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x3F if RGB565 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x1F if ARGB1555 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected.

  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.
- ***uint32_t LL_DMA2D_ColorTypeDef::OutputRed***

  Specifies the Red value of the output image.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if RGB888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x1F if RGB565 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x1F if ARGB1555 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected.

  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.
- ***uint32_t LL_DMA2D_ColorTypeDef::OutputAlpha***

  Specifies the Alpha channel of the output image.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x01 if ARGB1555 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected.
  - This parameter is not considered if RGB888 or RGB565 color mode is selected.

  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.

105.2 DMA2D Firmware driver API description

The following section lists the various functions of the DMA2D library.

105.2.1 Detailed description of functions

LL_DMA2D_Start

Function name

__STATIC_INLINE void LL_DMA2D_Start (DMA2D_TypeDef * DMA2Dx)

Function description

Start a DMA2D transfer.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR START LL_DMA2D_Start

LL_DMA2D_IsTransferOngoing

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsTransferOngoing (DMA2D_TypeDef * DMA2Dx)

Function description

Indicate if a DMA2D transfer is ongoing.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR START LL_DMA2D_IsTransferOngoing

LL_DMA2D_Suspend

Function name

__STATIC_INLINE void LL_DMA2D_Suspend (DMA2D_TypeDef * DMA2Dx)

Function description

Suspend DMA2D transfer.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Notes

- This API can be used to suspend automatic foreground or background CLUT loading.

Reference Manual to LL API cross reference:

- CR SUSP LL_DMA2D_Suspend

LL_DMA2D_Resume

Function name

__STATIC_INLINE void LL_DMA2D_Resume (DMA2D_TypeDef * DMA2Dx)

Function description

Resume DMA2D transfer.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Notes

- This API can be used to resume automatic foreground or background CLUT loading.

Reference Manual to LL API cross reference:

- CR SUSP LL_DMA2D_Resume

LL_DMA2D_IsSuspended

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsSuspended (DMA2D_TypeDef * DMA2Dx)

Function description

Indicate if DMA2D transfer is suspended.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Notes

- This API can be used to indicate whether or not automatic foreground or background CLUT loading is suspended.

Reference Manual to LL API cross reference:

- CR SUSP LL_DMA2D_IsSuspended

LL_DMA2D_Abort

Function name

__STATIC_INLINE void LL_DMA2D_Abort (DMA2D_TypeDef * DMA2Dx)

Function description

Abort DMA2D transfer.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Notes

- This API can be used to abort automatic foreground or background CLUT loading.

Reference Manual to LL API cross reference:

- CR ABORT LL_DMA2D_Abort

LL_DMA2D_IsAborted

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsAborted (DMA2D_TypeDef * DMA2Dx)

Function description

Indicate if DMA2D transfer is aborted.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Notes

- This API can be used to indicate whether or not automatic foreground or background CLUT loading is aborted.

Reference Manual to LL API cross reference:

- CR ABORT LL_DMA2D_IsAborted

LL_DMA2D_SetMode

Function name

__STATIC_INLINE void LL_DMA2D_SetMode (DMA2D_TypeDef * DMA2Dx, uint32_t Mode)

Function description

Set DMA2D mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **Mode:** This parameter can be one of the following values:
  - LL_DMA2D_MODE_M2M
  - LL_DMA2D_MODE_M2M_PFC
  - LL_DMA2D_MODE_M2M_BLEND
  - LL_DMA2D_MODE_R2M
  - LL_DMA2D_MODE_M2M_BLEND_FIXED_COLOR_FG
  - LL_DMA2D_MODE_M2M_BLEND_FIXED_COLOR_BG

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR MODE LL_DMA2D_SetMode

LL_DMA2D_GetMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetMode (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_MODE_M2M
  - LL_DMA2D_MODE_M2M_PFC
  - LL_DMA2D_MODE_M2M_BLEND
  - LL_DMA2D_MODE_R2M
  - LL_DMA2D_MODE_M2M_BLEND_FIXED_COLOR_FG
  - LL_DMA2D_MODE_M2M_BLEND_FIXED_COLOR_BG

Reference Manual to LL API cross reference:

- CR MODE LL_DMA2D_GetMode

LL_DMA2D_SetOutputColorMode

Function name

__STATIC_INLINE void LL_DMA2D_SetOutputColorMode (DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)

Function description

Set DMA2D output color mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **ColorMode:** This parameter can be one of the following values:
  - LL_DMA2D_OUTPUT_MODE_ARGB8888
  - LL_DMA2D_OUTPUT_MODE_RGB888
  - LL_DMA2D_OUTPUT_MODE_RGB565
  - LL_DMA2D_OUTPUT_MODE_ARGB1555
  - LL_DMA2D_OUTPUT_MODE_ARGB4444

Return values

- **None:**

Reference Manual to LL API cross reference:

- OPFCCR CM LL_DMA2D_SetOutputColorMode

LL_DMA2D_GetOutputColorMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetOutputColorMode (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D output color mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_OUTPUT_MODE_ARGB8888
  - LL_DMA2D_OUTPUT_MODE_RGB888
  - LL_DMA2D_OUTPUT_MODE_RGB565
  - LL_DMA2D_OUTPUT_MODE_ARGB1555
  - LL_DMA2D_OUTPUT_MODE_ARGB4444

Reference Manual to LL API cross reference:

- OPFCCR CM LL_DMA2D_GetOutputColorMode

LL_DMA2D_SetOutputRBSwapMode

Function name

__STATIC_INLINE void LL_DMA2D_SetOutputRBSwapMode (DMA2D_TypeDef * DMA2Dx, uint32_t RBSwapMode)

Function description

Set DMA2D output Red Blue swap mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **RBSwapMode:** This parameter can be one of the following values:
  - LL_DMA2D_RB_MODE_REGULAR
  - LL_DMA2D_RB_MODE_SWAP

Return values

- **None:**

Reference Manual to LL API cross reference:

- OPFCCR RBS LL_DMA2D_SetOutputRBSwapMode

LL_DMA2D_GetOutputRBSwapMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetOutputRBSwapMode (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D output Red Blue swap mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_RB_MODE_REGULAR
  - LL_DMA2D_RB_MODE_SWAP

Reference Manual to LL API cross reference:

- OPFCCR RBS LL_DMA2D_GetOutputRBSwapMode

LL_DMA2D_SetOutputAlphaInvMode

Function name

__STATIC_INLINE void LL_DMA2D_SetOutputAlphaInvMode (DMA2D_TypeDef * DMA2Dx, uint32_t AlphaInversionMode)

Function description

Set DMA2D output alpha inversion mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **AlphaInversionMode:** This parameter can be one of the following values:
  - LL_DMA2D_ALPHA_REGULAR
  - LL_DMA2D_ALPHA_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- OPFCCR AI LL_DMA2D_SetOutputAlphaInvMode

LL_DMA2D_GetOutputAlphaInvMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetOutputAlphaInvMode (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D output alpha inversion mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_ALPHA_REGULAR
  - LL_DMA2D_ALPHA_INVERTED

Reference Manual to LL API cross reference:

- OPFCCR AI LL_DMA2D_GetOutputAlphaInvMode

LL_DMA2D_SetOutputSwapMode

Function name

__STATIC_INLINE void LL_DMA2D_SetOutputSwapMode (DMA2D_TypeDef * DMA2Dx, uint32_t OutputSwapMode)

Function description

Set DMA2D output swap mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **OutputSwapMode:** This parameter can be one of the following values:
  - LL_DMA2D_SWAP_MODE_REGULAR
  - LL_DMA2D_SWAP_MODE_TWO_BY_TWO

Return values

- **None:**

Reference Manual to LL API cross reference:

- OPFCCR SB LL_DMA2D_SetOutputSwapMode

LL_DMA2D_GetOutputSwapMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetOutputSwapMode (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D output swap mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_SWAP_MODE_REGULAR
  - LL_DMA2D_SWAP_MODE_TWO_BY_TWO

Reference Manual to LL API cross reference:

- OPFCCR SB LL_DMA2D_GetOutputSwapMode

LL_DMA2D_SetLineOffsetMode

Function name

__STATIC_INLINE void LL_DMA2D_SetLineOffsetMode (DMA2D_TypeDef * DMA2Dx, uint32_t LineOffsetMode)

Function description

Set DMA2D line offset mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **LineOffsetMode:** This parameter can be one of the following values:
  - LL_DMA2D_LINE_OFFSET_PIXELS
  - LL_DMA2D_LINE_OFFSET_BYTES

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR LOM LL_DMA2D_SetLineOffsetMode

LL_DMA2D_GetLineOffsetMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetLineOffsetMode (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D line offset mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_LINE_OFFSET_PIXELS
  - LL_DMA2D_LINE_OFFSET_BYTES

Reference Manual to LL API cross reference:

- CR LOM LL_DMA2D_GetLineOffsetMode

LL_DMA2D_SetLineOffset

Function name

__STATIC_INLINE void LL_DMA2D_SetLineOffset (DMA2D_TypeDef * DMA2Dx, uint32_t LineOffset)

Function description

Set DMA2D line offset, expressed on 14 bits ([13:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **LineOffset:** Value between Min_Data=0 and Max_Data=0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- OOR LO LL_DMA2D_SetLineOffset

LL_DMA2D_GetLineOffset

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetLineOffset (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D line offset, expressed on 14 bits ([13:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Line:** offset value between Min_Data=0 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- OOR LO LL_DMA2D_GetLineOffset

LL_DMA2D_SetNbrOfPixelsPerLines

Function name

__STATIC_INLINE void LL_DMA2D_SetNbrOfPixelsPerLines (DMA2D_TypeDef * DMA2Dx, uint32_t NbrOfPixelsPerLines)

Function description

Set DMA2D number of pixels per lines, expressed on 14 bits ([13:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **NbrOfPixelsPerLines:** Value between Min_Data=0 and Max_Data=0x3FFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- NLR PL LL_DMA2D_SetNbrOfPixelsPerLines

LL_DMA2D_GetNbrOfPixelsPerLines

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetNbrOfPixelsPerLines (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D number of pixels per lines, expressed on 14 bits ([13:0] bits)

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Number:** of pixels per lines value between Min_Data=0 and Max_Data=0x3FFF

Reference Manual to LL API cross reference:

- NLR PL LL_DMA2D_GetNbrOfPixelsPerLines

LL_DMA2D_SetNbrOfLines

Function name

__STATIC_INLINE void LL_DMA2D_SetNbrOfLines (DMA2D_TypeDef * DMA2Dx, uint32_t NbrOfLines)

Function description

Set DMA2D number of lines, expressed on 16 bits ([15:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **NbrOfLines:** Value between Min_Data=0 and Max_Data=0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- NLR NL LL_DMA2D_SetNbrOfLines

LL_DMA2D_GetNbrOfLines

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetNbrOfLines (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D number of lines, expressed on 16 bits ([15:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Number:** of lines value between Min_Data=0 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- NLR NL LL_DMA2D_GetNbrOfLines

LL_DMA2D_SetOutputMemAddr

Function name

__STATIC_INLINE void LL_DMA2D_SetOutputMemAddr (DMA2D_TypeDef * DMA2Dx, uint32_t OutputMemoryAddress)

Function description

Set DMA2D output memory address, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **OutputMemoryAddress:** Value between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- OMAR MA LL_DMA2D_SetOutputMemAddr

LL_DMA2D_GetOutputMemAddr

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetOutputMemAddr (DMA2D_TypeDef * DMA2Dx)

Function description

Get DMA2D output memory address, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Output:** memory address value between Min_Data=0 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- OMAR MA LL_DMA2D_GetOutputMemAddr

LL_DMA2D_SetOutputColor

Function name

__STATIC_INLINE void LL_DMA2D_SetOutputColor (DMA2D_TypeDef * DMA2Dx, uint32_t OutputColor)

Function description

Set DMA2D output color, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **OutputColor:** Value between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Notes

- Output color format depends on output color mode, ARGB8888, RGB888, RGB565, ARGB1555 or ARGB4444.
- LL_DMA2D_ConfigOutputColor() API may be used instead if colors values formatting with respect to color mode is not done by the user code.

Reference Manual to LL API cross reference:

- OCOLR BLUE LL_DMA2D_SetOutputColor
- OCOLR GREEN LL_DMA2D_SetOutputColor
- OCOLR RED LL_DMA2D_SetOutputColor
- OCOLR ALPHA LL_DMA2D_SetOutputColor

LL_DMA2D_GetOutputColor

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetOutputColor (DMA2D_TypeDef * DMA2Dx)

Function description

Get DMA2D output color, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Output:** color value between Min_Data=0 and Max_Data=0xFFFFFFFF

Notes

- Alpha channel and red, green, blue color values must be retrieved from the returned value based on the output color mode (ARGB8888, RGB888, RGB565, ARGB1555 or ARGB4444) as set by LL_DMA2D_SetOutputColorMode.

Reference Manual to LL API cross reference:

- OCOLR BLUE LL_DMA2D_GetOutputColor
- OCOLR GREEN LL_DMA2D_GetOutputColor
- OCOLR RED LL_DMA2D_GetOutputColor
- OCOLR ALPHA LL_DMA2D_GetOutputColor

LL_DMA2D_SetLineWatermark

Function name

__STATIC_INLINE void LL_DMA2D_SetLineWatermark (DMA2D_TypeDef * DMA2Dx, uint32_t LineWatermark)

Function description

Set DMA2D line watermark, expressed on 16 bits ([15:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **LineWatermark:** Value between Min_Data=0 and Max_Data=0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- LWR LW LL_DMA2D_SetLineWatermark

LL_DMA2D_GetLineWatermark

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetLineWatermark (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D line watermark, expressed on 16 bits ([15:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Line:** watermark value between Min_Data=0 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- LWR LW LL_DMA2D_GetLineWatermark

LL_DMA2D_SetDeadTime

Function name

__STATIC_INLINE void LL_DMA2D_SetDeadTime (DMA2D_TypeDef * DMA2Dx, uint32_t DeadTime)

Function description

Set DMA2D dead time, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **DeadTime:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- AMTCR DT LL_DMA2D_SetDeadTime

LL_DMA2D_GetDeadTime

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetDeadTime (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D dead time, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Dead:** time value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- AMTCR DT LL_DMA2D_GetDeadTime

LL_DMA2D_EnableDeadTime

Function name

__STATIC_INLINE void LL_DMA2D_EnableDeadTime (DMA2D_TypeDef * DMA2Dx)

Function description

Enable DMA2D dead time functionality.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- AMTCR EN LL_DMA2D_EnableDeadTime

LL_DMA2D_DisableDeadTime

Function name

__STATIC_INLINE void LL_DMA2D_DisableDeadTime (DMA2D_TypeDef * DMA2Dx)

Function description

Disable DMA2D dead time functionality.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- AMTCR EN LL_DMA2D_DisableDeadTime

LL_DMA2D_IsEnabledDeadTime

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledDeadTime (DMA2D_TypeDef * DMA2Dx)

Function description

Indicate if DMA2D dead time functionality is enabled.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- AMTCR EN LL_DMA2D_IsEnabledDeadTime

LL_DMA2D_FGND_SetMemAddr

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetMemAddr (DMA2D_TypeDef * DMA2Dx, uint32_t MemoryAddress)

Function description

Set DMA2D foreground memory address, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **MemoryAddress:** Value between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGMAR MA LL_DMA2D_FGND_SetMemAddr

LL_DMA2D_FGND_GetMemAddr

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetMemAddr (DMA2D_TypeDef * DMA2Dx)

Function description

Get DMA2D foreground memory address, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Foreground:** memory address value between Min_Data=0 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- FGMAR MA LL_DMA2D_FGND_GetMemAddr

LL_DMA2D_FGND_EnableCLUTLoad

Function name

__STATIC_INLINE void LL_DMA2D_FGND_EnableCLUTLoad (DMA2D_TypeDef * DMA2Dx)

Function description

Enable DMA2D foreground CLUT loading.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGPFCCR START LL_DMA2D_FGND_EnableCLUTLoad

LL_DMA2D_FGND_IsEnabledCLUTLoad

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_IsEnabledCLUTLoad (DMA2D_TypeDef * DMA2Dx)

Function description

Indicate if DMA2D foreground CLUT loading is enabled.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- FGPFCCR START LL_DMA2D_FGND_IsEnabledCLUTLoad

LL_DMA2D_FGND_SetColorMode

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetColorMode (DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)

Function description

Set DMA2D foreground color mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **ColorMode:** This parameter can be one of the following values:
  - LL_DMA2D_INPUT_MODE_ARGB8888
  - LL_DMA2D_INPUT_MODE_RGB888
  - LL_DMA2D_INPUT_MODE_RGB565
  - LL_DMA2D_INPUT_MODE_ARGB1555
  - LL_DMA2D_INPUT_MODE_ARGB4444
  - LL_DMA2D_INPUT_MODE_L8
  - LL_DMA2D_INPUT_MODE_AL44
  - LL_DMA2D_INPUT_MODE_AL88
  - LL_DMA2D_INPUT_MODE_L4
  - LL_DMA2D_INPUT_MODE_A8
  - LL_DMA2D_INPUT_MODE_A4

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGPFCCR CM LL_DMA2D_FGND_SetColorMode

LL_DMA2D_FGND_GetColorMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetColorMode (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground color mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_INPUT_MODE_ARGB8888
  - LL_DMA2D_INPUT_MODE_RGB888
  - LL_DMA2D_INPUT_MODE_RGB565
  - LL_DMA2D_INPUT_MODE_ARGB1555
  - LL_DMA2D_INPUT_MODE_ARGB4444
  - LL_DMA2D_INPUT_MODE_L8
  - LL_DMA2D_INPUT_MODE_AL44
  - LL_DMA2D_INPUT_MODE_AL88
  - LL_DMA2D_INPUT_MODE_L4
  - LL_DMA2D_INPUT_MODE_A8
  - LL_DMA2D_INPUT_MODE_A4

Reference Manual to LL API cross reference:

- FGPFCCR CM LL_DMA2D_FGND_GetColorMode

LL_DMA2D_FGND_SetAlphaMode

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetAlphaMode (DMA2D_TypeDef * DMA2Dx, uint32_t AphaMode)

Function description

Set DMA2D foreground alpha mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **AphaMode:** This parameter can be one of the following values:
  - LL_DMA2D_ALPHA_MODE_NO_MODIF
  - LL_DMA2D_ALPHA_MODE_REPLACE
  - LL_DMA2D_ALPHA_MODE_COMBINE

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGPFCCR AM LL_DMA2D_FGND_SetAlphaMode

LL_DMA2D_FGND_GetAlphaMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetAlphaMode (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground alpha mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_ALPHA_MODE_NO_MODIF
  - LL_DMA2D_ALPHA_MODE_REPLACE
  - LL_DMA2D_ALPHA_MODE_COMBINE

Reference Manual to LL API cross reference:

- FGPFCCR AM LL_DMA2D_FGND_GetAlphaMode

LL_DMA2D_FGND_SetAlpha

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetAlpha (DMA2D_TypeDef * DMA2Dx, uint32_t Alpha)

Function description

Set DMA2D foreground alpha value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **Alpha:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGPFCCR ALPHA LL_DMA2D_FGND_SetAlpha

LL_DMA2D_FGND_GetAlpha

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetAlpha (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground alpha value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Alpha:** value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- FGPFCCR ALPHA LL_DMA2D_FGND_GetAlpha

LL_DMA2D_FGND_SetRBSwapMode

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetRBSwapMode (DMA2D_TypeDef * DMA2Dx, uint32_t RBSwapMode)

Function description

Set DMA2D foreground Red Blue swap mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **RBSwapMode:** This parameter can be one of the following values:
  - LL_DMA2D_RB_MODE_REGULAR
  - LL_DMA2D_RB_MODE_SWAP

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGPFCCR RBS LL_DMA2D_FGND_SetRBSwapMode

LL_DMA2D_FGND_GetRBSwapMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetRBSwapMode (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground Red Blue swap mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_RB_MODE_REGULAR
  - LL_DMA2D_RB_MODE_SWAP

Reference Manual to LL API cross reference:

- FGPFCCR RBS LL_DMA2D_FGND_GetRBSwapMode

LL_DMA2D_FGND_SetAlphaInvMode

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetAlphaInvMode (DMA2D_TypeDef * DMA2Dx, uint32_t AlphaInversionMode)

Function description

Set DMA2D foreground alpha inversion mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **AlphaInversionMode:** This parameter can be one of the following values:
  - LL_DMA2D_ALPHA_REGULAR
  - LL_DMA2D_ALPHA_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGPFCCR AI LL_DMA2D_FGND_SetAlphaInvMode

LL_DMA2D_FGND_GetAlphaInvMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetAlphaInvMode (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground alpha inversion mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_ALPHA_REGULAR
  - LL_DMA2D_ALPHA_INVERTED

Reference Manual to LL API cross reference:

- FGPFCCR AI LL_DMA2D_FGND_GetAlphaInvMode

LL_DMA2D_FGND_SetLineOffset

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetLineOffset (DMA2D_TypeDef * DMA2Dx, uint32_t LineOffset)

Function description

Set DMA2D foreground line offset, expressed on 14 bits ([13:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **LineOffset:** Value between Min_Data=0 and Max_Data=0x3FF

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGOR LO LL_DMA2D_FGND_SetLineOffset

LL_DMA2D_FGND_GetLineOffset

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetLineOffset (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground line offset, expressed on 14 bits ([13:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Foreground:** line offset value between Min_Data=0 and Max_Data=0x3FF

Reference Manual to LL API cross reference:

- FGOR LO LL_DMA2D_FGND_GetLineOffset

LL_DMA2D_FGND_SetColor

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetColor (DMA2D_TypeDef * DMA2Dx, uint32_t Red, uint32_t Green, uint32_t Blue)

Function description

Set DMA2D foreground color values, expressed on 24 bits ([23:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **Red:** Value between Min_Data=0 and Max_Data=0xFF
- **Green:** Value between Min_Data=0 and Max_Data=0xFF
- **Blue:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGCOLR RED LL_DMA2D_FGND_SetColor
- FGCOLR GREEN LL_DMA2D_FGND_SetColor
- FGCOLR BLUE LL_DMA2D_FGND_SetColor

LL_DMA2D_FGND_SetRedColor

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetRedColor (DMA2D_TypeDef * DMA2Dx, uint32_t Red)

Function description

Set DMA2D foreground red color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **Red:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGCOLR RED LL_DMA2D_FGND_SetRedColor

LL_DMA2D_FGND_GetRedColor

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetRedColor (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground red color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Red:** color value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- FGCOLR RED LL_DMA2D_FGND_GetRedColor

LL_DMA2D_FGND_SetGreenColor

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetGreenColor (DMA2D_TypeDef * DMA2Dx, uint32_t Green)

Function description

Set DMA2D foreground green color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **Green:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGCOLR GREEN LL_DMA2D_FGND_SetGreenColor

LL_DMA2D_FGND_GetGreenColor

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetGreenColor (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground green color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Green:** color value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- FGCOLR GREEN LL_DMA2D_FGND_GetGreenColor

LL_DMA2D_FGND_SetBlueColor

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetBlueColor (DMA2D_TypeDef * DMA2Dx, uint32_t Blue)

Function description

Set DMA2D foreground blue color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **Blue:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGCOLR BLUE LL_DMA2D_FGND_SetBlueColor

LL_DMA2D_FGND_GetBlueColor

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetBlueColor (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground blue color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Blue:** color value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- FGCOLR BLUE LL_DMA2D_FGND_GetBlueColor

LL_DMA2D_FGND_SetCLUTMemAddr

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetCLUTMemAddr (DMA2D_TypeDef * DMA2Dx, uint32_t CLUTMemoryAddress)

Function description

Set DMA2D foreground CLUT memory address, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **CLUTMemoryAddress:** Value between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGCMAR MA LL_DMA2D_FGND_SetCLUTMemAddr

LL_DMA2D_FGND_GetCLUTMemAddr

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetCLUTMemAddr (DMA2D_TypeDef * DMA2Dx)

Function description

Get DMA2D foreground CLUT memory address, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Foreground:** CLUT memory address value between Min_Data=0 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- FGCMAR MA LL_DMA2D_FGND_GetCLUTMemAddr

LL_DMA2D_FGND_SetCLUTSize

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetCLUTSize (DMA2D_TypeDef * DMA2Dx, uint32_t CLUTSize)

Function description

Set DMA2D foreground CLUT size, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **CLUTSize:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGPFCCR CS LL_DMA2D_FGND_SetCLUTSize

LL_DMA2D_FGND_GetCLUTSize

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetCLUTSize (DMA2D_TypeDef * DMA2Dx)

Function description

Get DMA2D foreground CLUT size, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Foreground:** CLUT size value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- FGPFCCR CS LL_DMA2D_FGND_GetCLUTSize

LL_DMA2D_FGND_SetCLUTColorMode

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetCLUTColorMode (DMA2D_TypeDef * DMA2Dx, uint32_t CLUTColorMode)

Function description

Set DMA2D foreground CLUT color mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **CLUTColorMode:** This parameter can be one of the following values:
  - LL_DMA2D_CLUT_COLOR_MODE_ARGB8888
  - LL_DMA2D_CLUT_COLOR_MODE_RGB888

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGPFCCR CCM LL_DMA2D_FGND_SetCLUTColorMode

LL_DMA2D_FGND_GetCLUTColorMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetCLUTColorMode (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground CLUT color mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_CLUT_COLOR_MODE_ARGB8888
  - LL_DMA2D_CLUT_COLOR_MODE_RGB888

Reference Manual to LL API cross reference:

- FGPFCCR CCM LL_DMA2D_FGND_GetCLUTColorMode

LL_DMA2D_FGND_SetChrSubSampling

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetChrSubSampling (DMA2D_TypeDef * DMA2Dx, uint32_t ChromaSubSampling)

Function description

Set DMA2D foreground Chroma Sub Sampling (for YCbCr input color mode).

Parameters

- **DMA2Dx:** DMA2D Instance
- **ChromaSubSampling:** This parameter can be one of the following values:
  - LL_DMA2D_CSS_444
  - LL_DMA2D_CSS_422
  - LL_DMA2D_CSS_420

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGPFCCR CSS LL_DMA2D_FGND_SetChrSubSampling

LL_DMA2D_FGND_GetChrSubSampling

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetChrSubSampling (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground Chroma Sub Sampling (for YCbCr input color mode).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_CSS_444
  - LL_DMA2D_CSS_422
  - LL_DMA2D_CSS_420

Reference Manual to LL API cross reference:

- FGPFCCR CSS LL_DMA2D_FGND_GetChrSubSampling

LL_DMA2D_BGND_SetMemAddr

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetMemAddr (DMA2D_TypeDef * DMA2Dx, uint32_t MemoryAddress)

Function description

Set DMA2D background memory address, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **MemoryAddress:** Value between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGMAR MA LL_DMA2D_BGND_SetMemAddr

LL_DMA2D_BGND_GetMemAddr

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetMemAddr (DMA2D_TypeDef * DMA2Dx)

Function description

Get DMA2D background memory address, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Background:** memory address value between Min_Data=0 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- BGMAR MA LL_DMA2D_BGND_GetMemAddr

LL_DMA2D_BGND_EnableCLUTLoad

Function name

__STATIC_INLINE void LL_DMA2D_BGND_EnableCLUTLoad (DMA2D_TypeDef * DMA2Dx)

Function description

Enable DMA2D background CLUT loading.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGPFCCR START LL_DMA2D_BGND_EnableCLUTLoad

LL_DMA2D_BGND_IsEnabledCLUTLoad

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_IsEnabledCLUTLoad (DMA2D_TypeDef * DMA2Dx)

Function description

Indicate if DMA2D background CLUT loading is enabled.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BGPFCCR START LL_DMA2D_BGND_IsEnabledCLUTLoad

LL_DMA2D_BGND_SetColorMode

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetColorMode (DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)

Function description

Set DMA2D background color mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **ColorMode:** This parameter can be one of the following values:
  - LL_DMA2D_INPUT_MODE_ARGB8888
  - LL_DMA2D_INPUT_MODE_RGB888
  - LL_DMA2D_INPUT_MODE_RGB565
  - LL_DMA2D_INPUT_MODE_ARGB1555
  - LL_DMA2D_INPUT_MODE_ARGB4444
  - LL_DMA2D_INPUT_MODE_L8
  - LL_DMA2D_INPUT_MODE_AL44
  - LL_DMA2D_INPUT_MODE_AL88
  - LL_DMA2D_INPUT_MODE_L4
  - LL_DMA2D_INPUT_MODE_A8
  - LL_DMA2D_INPUT_MODE_A4

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGPFCCR CM LL_DMA2D_BGND_SetColorMode

LL_DMA2D_BGND_GetColorMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetColorMode (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D background color mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_INPUT_MODE_ARGB8888
  - LL_DMA2D_INPUT_MODE_RGB888
  - LL_DMA2D_INPUT_MODE_RGB565
  - LL_DMA2D_INPUT_MODE_ARGB1555
  - LL_DMA2D_INPUT_MODE_ARGB4444
  - LL_DMA2D_INPUT_MODE_L8
  - LL_DMA2D_INPUT_MODE_AL44
  - LL_DMA2D_INPUT_MODE_AL88
  - LL_DMA2D_INPUT_MODE_L4
  - LL_DMA2D_INPUT_MODE_A8
  - LL_DMA2D_INPUT_MODE_A4

Reference Manual to LL API cross reference:

- BGPFCCR CM LL_DMA2D_BGND_GetColorMode

LL_DMA2D_BGND_SetAlphaMode

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetAlphaMode (DMA2D_TypeDef * DMA2Dx, uint32_t AphaMode)

Function description

Set DMA2D background alpha mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **AphaMode:** This parameter can be one of the following values:
  - LL_DMA2D_ALPHA_MODE_NO_MODIF
  - LL_DMA2D_ALPHA_MODE_REPLACE
  - LL_DMA2D_ALPHA_MODE_COMBINE

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGPFCCR AM LL_DMA2D_BGND_SetAlphaMode

LL_DMA2D_BGND_GetAlphaMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetAlphaMode (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D background alpha mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_ALPHA_MODE_NO_MODIF
  - LL_DMA2D_ALPHA_MODE_REPLACE
  - LL_DMA2D_ALPHA_MODE_COMBINE

Reference Manual to LL API cross reference:

- BGPFCCR AM LL_DMA2D_BGND_GetAlphaMode

LL_DMA2D_BGND_SetAlpha

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetAlpha (DMA2D_TypeDef * DMA2Dx, uint32_t Alpha)

Function description

Set DMA2D background alpha value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **Alpha:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGPFCCR ALPHA LL_DMA2D_BGND_SetAlpha

LL_DMA2D_BGND_GetAlpha

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetAlpha (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D background alpha value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Alpha:** value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- BGPFCCR ALPHA LL_DMA2D_BGND_GetAlpha

LL_DMA2D_BGND_SetRBSwapMode

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetRBSwapMode (DMA2D_TypeDef * DMA2Dx, uint32_t RBSwapMode)

Function description

Set DMA2D background Red Blue swap mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **RBSwapMode:** This parameter can be one of the following values:
  - LL_DMA2D_RB_MODE_REGULAR
  - LL_DMA2D_RB_MODE_SWAP

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGPFCCR RBS LL_DMA2D_BGND_SetRBSwapMode

LL_DMA2D_BGND_GetRBSwapMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetRBSwapMode (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D background Red Blue swap mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_RB_MODE_REGULAR
  - LL_DMA2D_RB_MODE_SWAP

Reference Manual to LL API cross reference:

- BGPFCCR RBS LL_DMA2D_BGND_GetRBSwapMode

LL_DMA2D_BGND_SetAlphaInvMode

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetAlphaInvMode (DMA2D_TypeDef * DMA2Dx, uint32_t AlphaInversionMode)

Function description

Set DMA2D background alpha inversion mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **AlphaInversionMode:** This parameter can be one of the following values:
  - LL_DMA2D_ALPHA_REGULAR
  - LL_DMA2D_ALPHA_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGPFCCR AI LL_DMA2D_BGND_SetAlphaInvMode

LL_DMA2D_BGND_GetAlphaInvMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetAlphaInvMode (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D background alpha inversion mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_ALPHA_REGULAR
  - LL_DMA2D_ALPHA_INVERTED

Reference Manual to LL API cross reference:

- BGPFCCR AI LL_DMA2D_BGND_GetAlphaInvMode

LL_DMA2D_BGND_SetLineOffset

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetLineOffset (DMA2D_TypeDef * DMA2Dx, uint32_t LineOffset)

Function description

Set DMA2D background line offset, expressed on 14 bits ([13:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **LineOffset:** Value between Min_Data=0 and Max_Data=0x3FF

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGOR LO LL_DMA2D_BGND_SetLineOffset

LL_DMA2D_BGND_GetLineOffset

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetLineOffset (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D background line offset, expressed on 14 bits ([13:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Background:** line offset value between Min_Data=0 and Max_Data=0x3FF

Reference Manual to LL API cross reference:

- BGOR LO LL_DMA2D_BGND_GetLineOffset

LL_DMA2D_BGND_SetColor

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetColor (DMA2D_TypeDef * DMA2Dx, uint32_t Red, uint32_t Green, uint32_t Blue)

Function description

Set DMA2D background color values, expressed on 24 bits ([23:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **Red:** Value between Min_Data=0 and Max_Data=0xFF
- **Green:** Value between Min_Data=0 and Max_Data=0xFF
- **Blue:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGCOLR RED LL_DMA2D_BGND_SetColor
- BGCOLR GREEN LL_DMA2D_BGND_SetColor
- BGCOLR BLUE LL_DMA2D_BGND_SetColor

LL_DMA2D_BGND_SetRedColor

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetRedColor (DMA2D_TypeDef * DMA2Dx, uint32_t Red)

Function description

Set DMA2D background red color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **Red:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGCOLR RED LL_DMA2D_BGND_SetRedColor

LL_DMA2D_BGND_GetRedColor

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetRedColor (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D background red color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Red:** color value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- BGCOLR RED LL_DMA2D_BGND_GetRedColor

LL_DMA2D_BGND_SetGreenColor

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetGreenColor (DMA2D_TypeDef * DMA2Dx, uint32_t Green)

Function description

Set DMA2D background green color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **Green:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGCOLR GREEN LL_DMA2D_BGND_SetGreenColor

LL_DMA2D_BGND_GetGreenColor

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetGreenColor (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D background green color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Green:** color value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- BGCOLR GREEN LL_DMA2D_BGND_GetGreenColor

LL_DMA2D_BGND_SetBlueColor

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetBlueColor (DMA2D_TypeDef * DMA2Dx, uint32_t Blue)

Function description

Set DMA2D background blue color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **Blue:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGCOLR BLUE LL_DMA2D_BGND_SetBlueColor

LL_DMA2D_BGND_GetBlueColor

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetBlueColor (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D background blue color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Blue:** color value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- BGCOLR BLUE LL_DMA2D_BGND_GetBlueColor

LL_DMA2D_BGND_SetCLUTMemAddr

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetCLUTMemAddr (DMA2D_TypeDef * DMA2Dx, uint32_t CLUTMemoryAddress)

Function description

Set DMA2D background CLUT memory address, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **CLUTMemoryAddress:** Value between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGCMAR MA LL_DMA2D_BGND_SetCLUTMemAddr

LL_DMA2D_BGND_GetCLUTMemAddr

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetCLUTMemAddr (DMA2D_TypeDef * DMA2Dx)

Function description

Get DMA2D background CLUT memory address, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Background:** CLUT memory address value between Min_Data=0 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- BGCMAR MA LL_DMA2D_BGND_GetCLUTMemAddr

LL_DMA2D_BGND_SetCLUTSize

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetCLUTSize (DMA2D_TypeDef * DMA2Dx, uint32_t CLUTSize)

Function description

Set DMA2D background CLUT size, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **CLUTSize:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGPFCCR CS LL_DMA2D_BGND_SetCLUTSize

LL_DMA2D_BGND_GetCLUTSize

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetCLUTSize (DMA2D_TypeDef * DMA2Dx)

Function description

Get DMA2D background CLUT size, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Background:** CLUT size value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- BGPFCCR CS LL_DMA2D_BGND_GetCLUTSize

LL_DMA2D_BGND_SetCLUTColorMode

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetCLUTColorMode (DMA2D_TypeDef * DMA2Dx, uint32_t CLUTColorMode)

Function description

Set DMA2D background CLUT color mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **CLUTColorMode:** This parameter can be one of the following values:
  - LL_DMA2D_CLUT_COLOR_MODE_ARGB8888
  - LL_DMA2D_CLUT_COLOR_MODE_RGB888

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGPFCCR CCM LL_DMA2D_BGND_SetCLUTColorMode

LL_DMA2D_BGND_GetCLUTColorMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetCLUTColorMode (DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D background CLUT color mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_CLUT_COLOR_MODE_ARGB8888
  - LL_DMA2D_CLUT_COLOR_MODE_RGB888

Reference Manual to LL API cross reference:

- BGPFCCR CCM LL_DMA2D_BGND_GetCLUTColorMode

LL_DMA2D_IsActiveFlag_CE

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsActiveFlag_CE (DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D Configuration Error Interrupt Flag is set or not.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CEIF LL_DMA2D_IsActiveFlag_CE

LL_DMA2D_IsActiveFlag_CTC

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsActiveFlag_CTC (DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D CLUT Transfer Complete Interrupt Flag is set or not.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CTCIF LL_DMA2D_IsActiveFlag_CTC

LL_DMA2D_IsActiveFlag_CAE

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsActiveFlag_CAE (DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D CLUT Access Error Interrupt Flag is set or not.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CAEIF LL_DMA2D_IsActiveFlag_CAE

LL_DMA2D_IsActiveFlag_TW

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsActiveFlag_TW (DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D Transfer Watermark Interrupt Flag is set or not.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TWIF LL_DMA2D_IsActiveFlag_TW

LL_DMA2D_IsActiveFlag_TC

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsActiveFlag_TC (DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D Transfer Complete Interrupt Flag is set or not.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCIF LL_DMA2D_IsActiveFlag_TC

LL_DMA2D_IsActiveFlag_TE

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsActiveFlag_TE (DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D Transfer Error Interrupt Flag is set or not.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEIF LL_DMA2D_IsActiveFlag_TE

LL_DMA2D_ClearFlag_CE

Function name

__STATIC_INLINE void LL_DMA2D_ClearFlag_CE (DMA2D_TypeDef * DMA2Dx)

Function description

Clear DMA2D Configuration Error Interrupt Flag.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CCEIF LL_DMA2D_ClearFlag_CE

LL_DMA2D_ClearFlag_CTC

Function name

__STATIC_INLINE void LL_DMA2D_ClearFlag_CTC (DMA2D_TypeDef * DMA2Dx)

Function description

Clear DMA2D CLUT Transfer Complete Interrupt Flag.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CCTCIF LL_DMA2D_ClearFlag_CTC

LL_DMA2D_ClearFlag_CAE

Function name

__STATIC_INLINE void LL_DMA2D_ClearFlag_CAE (DMA2D_TypeDef * DMA2Dx)

Function description

Clear DMA2D CLUT Access Error Interrupt Flag.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CAECIF LL_DMA2D_ClearFlag_CAE

LL_DMA2D_ClearFlag_TW

Function name

__STATIC_INLINE void LL_DMA2D_ClearFlag_TW (DMA2D_TypeDef * DMA2Dx)

Function description

Clear DMA2D Transfer Watermark Interrupt Flag.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTWIF LL_DMA2D_ClearFlag_TW

LL_DMA2D_ClearFlag_TC

Function name

__STATIC_INLINE void LL_DMA2D_ClearFlag_TC (DMA2D_TypeDef * DMA2Dx)

Function description

Clear DMA2D Transfer Complete Interrupt Flag.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTCIF LL_DMA2D_ClearFlag_TC

LL_DMA2D_ClearFlag_TE

Function name

__STATIC_INLINE void LL_DMA2D_ClearFlag_TE (DMA2D_TypeDef * DMA2Dx)

Function description

Clear DMA2D Transfer Error Interrupt Flag.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTEIF LL_DMA2D_ClearFlag_TE

LL_DMA2D_EnableIT_CE

Function name

__STATIC_INLINE void LL_DMA2D_EnableIT_CE (DMA2D_TypeDef * DMA2Dx)

Function description

Enable Configuration Error Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CEIE LL_DMA2D_EnableIT_CE

LL_DMA2D_EnableIT_CTC

Function name

__STATIC_INLINE void LL_DMA2D_EnableIT_CTC (DMA2D_TypeDef * DMA2Dx)

Function description

Enable CLUT Transfer Complete Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CTCIE LL_DMA2D_EnableIT_CTC

LL_DMA2D_EnableIT_CAE

Function name

__STATIC_INLINE void LL_DMA2D_EnableIT_CAE (DMA2D_TypeDef * DMA2Dx)

Function description

Enable CLUT Access Error Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CAEIE LL_DMA2D_EnableIT_CAE

LL_DMA2D_EnableIT_TW

Function name

__STATIC_INLINE void LL_DMA2D_EnableIT_TW (DMA2D_TypeDef * DMA2Dx)

Function description

Enable Transfer Watermark Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TWIE LL_DMA2D_EnableIT_TW

LL_DMA2D_EnableIT_TC

Function name

__STATIC_INLINE void LL_DMA2D_EnableIT_TC (DMA2D_TypeDef * DMA2Dx)

Function description

Enable Transfer Complete Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TCIE LL_DMA2D_EnableIT_TC

LL_DMA2D_EnableIT_TE

Function name

__STATIC_INLINE void LL_DMA2D_EnableIT_TE (DMA2D_TypeDef * DMA2Dx)

Function description

Enable Transfer Error Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TEIE LL_DMA2D_EnableIT_TE

LL_DMA2D_DisableIT_CE

Function name

__STATIC_INLINE void LL_DMA2D_DisableIT_CE (DMA2D_TypeDef * DMA2Dx)

Function description

Disable Configuration Error Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CEIE LL_DMA2D_DisableIT_CE

LL_DMA2D_DisableIT_CTC

Function name

__STATIC_INLINE void LL_DMA2D_DisableIT_CTC (DMA2D_TypeDef * DMA2Dx)

Function description

Disable CLUT Transfer Complete Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CTCIE LL_DMA2D_DisableIT_CTC

LL_DMA2D_DisableIT_CAE

Function name

__STATIC_INLINE void LL_DMA2D_DisableIT_CAE (DMA2D_TypeDef * DMA2Dx)

Function description

Disable CLUT Access Error Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CAEIE LL_DMA2D_DisableIT_CAE

LL_DMA2D_DisableIT_TW

Function name

__STATIC_INLINE void LL_DMA2D_DisableIT_TW (DMA2D_TypeDef * DMA2Dx)

Function description

Disable Transfer Watermark Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TWIE LL_DMA2D_DisableIT_TW

LL_DMA2D_DisableIT_TC

Function name

__STATIC_INLINE void LL_DMA2D_DisableIT_TC (DMA2D_TypeDef * DMA2Dx)

Function description

Disable Transfer Complete Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TCIE LL_DMA2D_DisableIT_TC

LL_DMA2D_DisableIT_TE

Function name

__STATIC_INLINE void LL_DMA2D_DisableIT_TE (DMA2D_TypeDef * DMA2Dx)

Function description

Disable Transfer Error Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TEIE LL_DMA2D_DisableIT_TE

LL_DMA2D_IsEnabledIT_CE

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledIT_CE (DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D Configuration Error interrupt source is enabled or disabled.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CEIE LL_DMA2D_IsEnabledIT_CE

LL_DMA2D_IsEnabledIT_CTC

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledIT_CTC (DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D CLUT Transfer Complete interrupt source is enabled or disabled.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CTCIE LL_DMA2D_IsEnabledIT_CTC

LL_DMA2D_IsEnabledIT_CAE

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledIT_CAE (DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D CLUT Access Error interrupt source is enabled or disabled.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CAEIE LL_DMA2D_IsEnabledIT_CAE

LL_DMA2D_IsEnabledIT_TW

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledIT_TW (DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D Transfer Watermark interrupt source is enabled or disabled.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR TWIE LL_DMA2D_IsEnabledIT_TW

LL_DMA2D_IsEnabledIT_TC

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledIT_TC (DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D Transfer Complete interrupt source is enabled or disabled.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR TCIE LL_DMA2D_IsEnabledIT_TC

LL_DMA2D_IsEnabledIT_TE

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledIT_TE (DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D Transfer Error interrupt source is enabled or disabled.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR TEIE LL_DMA2D_IsEnabledIT_TE

LL_DMA2D_DeInit

Function name

ErrorStatus LL_DMA2D_DeInit (DMA2D_TypeDef * DMA2Dx)

Function description

De-initialize DMA2D registers (registers restored to their default values).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: DMA2D registers are de-initialized
  - ERROR: DMA2D registers are not de-initialized

LL_DMA2D_Init

Function name

ErrorStatus LL_DMA2D_Init (DMA2D_TypeDef * DMA2Dx, LL_DMA2D_InitTypeDef * DMA2D_InitStruct)

Function description

Initialize DMA2D registers according to the specified parameters in DMA2D_InitStruct.

Parameters

- **DMA2Dx:** DMA2D Instance
- **DMA2D_InitStruct:** pointer to a LL_DMA2D_InitTypeDef structure that contains the configuration information for the specified DMA2D peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: DMA2D registers are initialized according to DMA2D_InitStruct content
  - ERROR: Issue occurred during DMA2D registers initialization

Notes

- DMA2D transfers must be disabled to set initialization bits in configuration registers, otherwise ERROR result is returned.

LL_DMA2D_StructInit

Function name

void LL_DMA2D_StructInit (LL_DMA2D_InitTypeDef * DMA2D_InitStruct)

Function description

Set each LL_DMA2D_InitTypeDef field to default value.

Parameters

- **DMA2D_InitStruct:** pointer to a LL_DMA2D_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_DMA2D_ConfigLayer

Function name

void LL_DMA2D_ConfigLayer (DMA2D_TypeDef * DMA2Dx, LL_DMA2D_LayerCfgTypeDef * DMA2D_LayerCfg, uint32_t LayerIdx)

Function description

Configure the foreground or background according to the specified parameters in the LL_DMA2D_LayerCfgTypeDef structure.

Parameters

- **DMA2Dx:** DMA2D Instance
- **DMA2D_LayerCfg:** pointer to a LL_DMA2D_LayerCfgTypeDef structure that contains the configuration information for the specified layer.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: 0(background) / 1(foreground)

Return values

- **None:**

LL_DMA2D_LayerCfgStructInit

Function name

void LL_DMA2D_LayerCfgStructInit (LL_DMA2D_LayerCfgTypeDef * DMA2D_LayerCfg)

Function description

Set each LL_DMA2D_LayerCfgTypeDef field to default value.

Parameters

- **DMA2D_LayerCfg:** pointer to a LL_DMA2D_LayerCfgTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_DMA2D_ConfigOutputColor

Function name

void LL_DMA2D_ConfigOutputColor (DMA2D_TypeDef * DMA2Dx, LL_DMA2D_ColorTypeDef * DMA2D_ColorStruct)

Function description

Initialize DMA2D output color register according to the specified parameters in DMA2D_ColorStruct.

Parameters

- **DMA2Dx:** DMA2D Instance
- **DMA2D_ColorStruct:** pointer to a LL_DMA2D_ColorTypeDef structure that contains the color configuration information for the specified DMA2D peripheral.

Return values

- **None:**

LL_DMA2D_GetOutputBlueColor

Function name

uint32_t LL_DMA2D_GetOutputBlueColor (DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)

Function description

Return DMA2D output Blue color.

Parameters

- **DMA2Dx:** DMA2D Instance.
- **ColorMode:** This parameter can be one of the following values:
  - LL_DMA2D_OUTPUT_MODE_ARGB8888
  - LL_DMA2D_OUTPUT_MODE_RGB888
  - LL_DMA2D_OUTPUT_MODE_RGB565
  - LL_DMA2D_OUTPUT_MODE_ARGB1555
  - LL_DMA2D_OUTPUT_MODE_ARGB4444

Return values

- **Output:** Blue color value between Min_Data=0 and Max_Data=0xFF

LL_DMA2D_GetOutputGreenColor

Function name

uint32_t LL_DMA2D_GetOutputGreenColor (DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)

Function description

Return DMA2D output Green color.

Parameters

- **DMA2Dx:** DMA2D Instance.
- **ColorMode:** This parameter can be one of the following values:
  - LL_DMA2D_OUTPUT_MODE_ARGB8888
  - LL_DMA2D_OUTPUT_MODE_RGB888
  - LL_DMA2D_OUTPUT_MODE_RGB565
  - LL_DMA2D_OUTPUT_MODE_ARGB1555
  - LL_DMA2D_OUTPUT_MODE_ARGB4444

Return values

- **Output:** Green color value between Min_Data=0 and Max_Data=0xFF

LL_DMA2D_GetOutputRedColor

Function name

uint32_t LL_DMA2D_GetOutputRedColor (DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)

Function description

Return DMA2D output Red color.

Parameters

- **DMA2Dx:** DMA2D Instance.
- **ColorMode:** This parameter can be one of the following values:
  - LL_DMA2D_OUTPUT_MODE_ARGB8888
  - LL_DMA2D_OUTPUT_MODE_RGB888
  - LL_DMA2D_OUTPUT_MODE_RGB565
  - LL_DMA2D_OUTPUT_MODE_ARGB1555
  - LL_DMA2D_OUTPUT_MODE_ARGB4444

Return values

- **Output:** Red color value between Min_Data=0 and Max_Data=0xFF

LL_DMA2D_GetOutputAlphaColor

Function name

uint32_t LL_DMA2D_GetOutputAlphaColor (DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)

Function description

Return DMA2D output Alpha color.

Parameters

- **DMA2Dx:** DMA2D Instance.
- **ColorMode:** This parameter can be one of the following values:
  - LL_DMA2D_OUTPUT_MODE_ARGB8888
  - LL_DMA2D_OUTPUT_MODE_RGB888
  - LL_DMA2D_OUTPUT_MODE_RGB565
  - LL_DMA2D_OUTPUT_MODE_ARGB1555
  - LL_DMA2D_OUTPUT_MODE_ARGB4444

Return values

- **Output:** Alpha color value between Min_Data=0 and Max_Data=0xFF

LL_DMA2D_ConfigSize

Function name

void LL_DMA2D_ConfigSize (DMA2D_TypeDef * DMA2Dx, uint32_t NbrOfLines, uint32_t NbrOfPixelsPerLines)

Function description

Configure DMA2D transfer size.

Parameters

- **DMA2Dx:** DMA2D Instance
- **NbrOfLines:** Value between Min_Data=0 and Max_Data=0xFFFF
- **NbrOfPixelsPerLines:** Value between Min_Data=0 and Max_Data=0x3FFF

Return values

- **None:**

105.3 DMA2D Firmware driver defines

The following section lists the various define and macros of the module.

105.3.1 DMA2D

DMA2D

Chroma Sub Sampling

LL_DMA2D_CSS_444

No chroma sub-sampling 4:4:4

LL_DMA2D_CSS_422

chroma sub-sampling 4:2:2

LL_DMA2D_CSS_420

chroma sub-sampling 4:2:0

Alpha Inversion

LL_DMA2D_ALPHA_REGULAR

Regular alpha

LL_DMA2D_ALPHA_INVERTED

Inverted alpha

Alpha Mode

LL_DMA2D_ALPHA_MODE_NO_MODIF

No modification of the alpha channel value

LL_DMA2D_ALPHA_MODE_REPLACE

Replace original alpha channel value by programmed alpha value

LL_DMA2D_ALPHA_MODE_COMBINE

Replace original alpha channel value by programmed alpha value with, original alpha channel value

CLUT Color Mode

LL_DMA2D_CLUT_COLOR_MODE_ARGB8888

ARGB8888

LL_DMA2D_CLUT_COLOR_MODE_RGB888

RGB888

Get Flags Defines

LL_DMA2D_FLAG_CEIF

Configuration Error Interrupt Flag

LL_DMA2D_FLAG_CTCIF

CLUT Transfer Complete Interrupt Flag

LL_DMA2D_FLAG_CAEIF

CLUT Access Error Interrupt Flag

LL_DMA2D_FLAG_TWIF

Transfer Watermark Interrupt Flag

LL_DMA2D_FLAG_TCIF

Transfer Complete Interrupt Flag

LL_DMA2D_FLAG_TEIF

Transfer Error Interrupt Flag

Input Color Mode

LL_DMA2D_INPUT_MODE_ARGB8888

ARGB8888

LL_DMA2D_INPUT_MODE_RGB888

RGB888

LL_DMA2D_INPUT_MODE_RGB565

RGB565

LL_DMA2D_INPUT_MODE_ARGB1555

ARGB1555

LL_DMA2D_INPUT_MODE_ARGB4444

ARGB4444

LL_DMA2D_INPUT_MODE_L8

L8

LL_DMA2D_INPUT_MODE_AL44

AL44

LL_DMA2D_INPUT_MODE_AL88

AL88

LL_DMA2D_INPUT_MODE_L4

L4

LL_DMA2D_INPUT_MODE_A8

A8

LL_DMA2D_INPUT_MODE_A4

A4

LL_DMA2D_INPUT_MODE_YCBCR

YCbCr

IT Defines

LL_DMA2D_IT_CEIE

Configuration Error Interrupt

LL_DMA2D_IT_CTCIE

CLUT Transfer Complete Interrupt

LL_DMA2D_IT_CAEIE

CLUT Access Error Interrupt

LL_DMA2D_IT_TWIE

Transfer Watermark Interrupt

LL_DMA2D_IT_TCIE

Transfer Complete Interrupt

LL_DMA2D_IT_TEIE

Transfer Error Interrupt

Line Offset Mode

LL_DMA2D_LINE_OFFSET_PIXELS

Line offsets are expressed in pixels

LL_DMA2D_LINE_OFFSET_BYTES

Line offsets are expressed in bytes

Mode

LL_DMA2D_MODE_M2M

DMA2D memory to memory transfer mode

LL_DMA2D_MODE_M2M_PFC

DMA2D memory to memory with pixel format conversion transfer mode

LL_DMA2D_MODE_M2M_BLEND

DMA2D memory to memory with blending transfer mode

LL_DMA2D_MODE_R2M

DMA2D register to memory transfer mode

LL_DMA2D_MODE_M2M_BLEND_FIXED_COLOR_FG

DMA2D memory to memory with blending transfer mode and fixed color foreground

LL_DMA2D_MODE_M2M_BLEND_FIXED_COLOR_BG

DMA2D memory to memory with blending transfer mode and fixed color background

Output Color Mode

LL_DMA2D_OUTPUT_MODE_ARGB8888

ARGB8888

LL_DMA2D_OUTPUT_MODE_RGB888

RGB888

LL_DMA2D_OUTPUT_MODE_RGB565

RGB565

LL_DMA2D_OUTPUT_MODE_ARGB1555

ARGB1555

LL_DMA2D_OUTPUT_MODE_ARGB4444

ARGB4444

Swap Mode

LL_DMA2D_SWAP_MODE_REGULAR

Regular order

LL_DMA2D_SWAP_MODE_TWO_BY_TWO

Bytes swapped two by two

Red Blue Swap

LL_DMA2D_RB_MODE_REGULAR

RGB or ARGB

LL_DMA2D_RB_MODE_SWAP

BGR or ABGR

Common Write and read registers Macros

LL_DMA2D_WriteReg

Description:

- Write a value in DMA2D register.

Parameters:

- __INSTANCE__: DMA2D Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_DMA2D_ReadReg

Description:

- Read a value in DMA2D register.

Parameters:

- __INSTANCE__: DMA2D Instance
- __REG__: Register to be read

Return value:

- Register: value

106 LL DMAMUX Generic Driver

106.1 DMAMUX Firmware driver API description

The following section lists the various functions of the DMAMUX library.

106.1.1 Detailed description of functions

LL_DMAMUX_SetRequestID

Function name

__STATIC_INLINE void LL_DMAMUX_SetRequestID (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel, uint32_t Request)

Function description

Set DMAMUX request ID for DMAMUX Channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15

- **Request:** This parameter can be one of the following values:
  - LL_DMAMUX1_REQ_MEM2MEM
  - LL_DMAMUX1_REQ_GENERATOR0
  - LL_DMAMUX1_REQ_GENERATOR1
  - LL_DMAMUX1_REQ_GENERATOR2
  - LL_DMAMUX1_REQ_GENERATOR3
  - LL_DMAMUX1_REQ_GENERATOR4
  - LL_DMAMUX1_REQ_GENERATOR5
  - LL_DMAMUX1_REQ_GENERATOR6
  - LL_DMAMUX1_REQ_GENERATOR7
  - LL_DMAMUX1_REQ_ADC1
  - LL_DMAMUX1_REQ_ADC2
  - LL_DMAMUX1_REQ_TIM1_CH1
  - LL_DMAMUX1_REQ_TIM1_CH2
  - LL_DMAMUX1_REQ_TIM1_CH3
  - LL_DMAMUX1_REQ_TIM1_CH4
  - LL_DMAMUX1_REQ_TIM1_UP
  - LL_DMAMUX1_REQ_TIM1_TRIG
  - LL_DMAMUX1_REQ_TIM1_COM
  - LL_DMAMUX1_REQ_TIM2_CH1
  - LL_DMAMUX1_REQ_TIM2_CH2
  - LL_DMAMUX1_REQ_TIM2_CH3
  - LL_DMAMUX1_REQ_TIM2_CH4
  - LL_DMAMUX1_REQ_TIM2_UP
  - LL_DMAMUX1_REQ_TIM3_CH1
  - LL_DMAMUX1_REQ_TIM3_CH2
  - LL_DMAMUX1_REQ_TIM3_CH3
  - LL_DMAMUX1_REQ_TIM3_CH4
  - LL_DMAMUX1_REQ_TIM3_UP
  - LL_DMAMUX1_REQ_TIM3_TRIG
  - LL_DMAMUX1_REQ_TIM4_CH1
  - LL_DMAMUX1_REQ_TIM4_CH2
  - LL_DMAMUX1_REQ_TIM4_CH3
  - LL_DMAMUX1_REQ_TIM4_UP
  - LL_DMAMUX1_REQ_I2C1_RX
  - LL_DMAMUX1_REQ_I2C1_TX
  - LL_DMAMUX1_REQ_I2C2_RX
  - LL_DMAMUX1_REQ_I2C2_TX
  - LL_DMAMUX1_REQ_SPI1_RX
  - LL_DMAMUX1_REQ_SPI1_TX
  - LL_DMAMUX1_REQ_SPI2_RX
  - LL_DMAMUX1_REQ_SPI2_TX
  - LL_DMAMUX1_REQ_USART1_RX
  - LL_DMAMUX1_REQ_USART1_TX
  - LL_DMAMUX1_REQ_USART2_RX
  - LL_DMAMUX1_REQ_USART2_TX
  - LL_DMAMUX1_REQ_USART3_RX
  - LL_DMAMUX1_REQ_USART3_TX
  - LL_DMAMUX1_REQ_TIM8_CH1
  - LL_DMAMUX1_REQ_TIM8_CH2
  - LL_DMAMUX1_REQ_TIM8_CH3
  - LL_DMAMUX1_REQ_TIM8_CH4
  - LL_DMAMUX1_REQ_TIM8_UP

Return values

- **None:**

Notes

- DMAMUX1 channel 0 to 7 are mapped to DMA1 channel 0 to 7. DMAMUX1 channel 8 to 15 are mapped to DMA2 channel 0 to 7. DMAMUX2 channel 0 to 7 are mapped to BDMA channel 0 to 7.
- (*) Availability depends on devices.

Reference Manual to LL API cross reference:

- CxCR DMAREQ_ID LL_DMAMUX_SetRequestID

LL_DMAMUX_GetRequestID

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_GetRequestID (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Get DMAMUX request ID for DMAMUX Channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMAMUX1_REQ_MEM2MEM
  - LL_DMAMUX1_REQ_GENERATOR0
  - LL_DMAMUX1_REQ_GENERATOR1
  - LL_DMAMUX1_REQ_GENERATOR2
  - LL_DMAMUX1_REQ_GENERATOR3
  - LL_DMAMUX1_REQ_GENERATOR4
  - LL_DMAMUX1_REQ_GENERATOR5
  - LL_DMAMUX1_REQ_GENERATOR6
  - LL_DMAMUX1_REQ_GENERATOR7
  - LL_DMAMUX1_REQ_ADC1
  - LL_DMAMUX1_REQ_ADC2
  - LL_DMAMUX1_REQ_TIM1_CH1
  - LL_DMAMUX1_REQ_TIM1_CH2
  - LL_DMAMUX1_REQ_TIM1_CH3
  - LL_DMAMUX1_REQ_TIM1_CH4
  - LL_DMAMUX1_REQ_TIM1_UP
  - LL_DMAMUX1_REQ_TIM1_TRIG
  - LL_DMAMUX1_REQ_TIM1_COM
  - LL_DMAMUX1_REQ_TIM2_CH1
  - LL_DMAMUX1_REQ_TIM2_CH2
  - LL_DMAMUX1_REQ_TIM2_CH3
  - LL_DMAMUX1_REQ_TIM2_CH4
  - LL_DMAMUX1_REQ_TIM2_UP
  - LL_DMAMUX1_REQ_TIM3_CH1
  - LL_DMAMUX1_REQ_TIM3_CH2
  - LL_DMAMUX1_REQ_TIM3_CH3
  - LL_DMAMUX1_REQ_TIM3_CH4
  - LL_DMAMUX1_REQ_TIM3_UP
  - LL_DMAMUX1_REQ_TIM3_TRIG
  - LL_DMAMUX1_REQ_TIM4_CH1
  - LL_DMAMUX1_REQ_TIM4_CH2
  - LL_DMAMUX1_REQ_TIM4_CH3
  - LL_DMAMUX1_REQ_TIM4_UP
  - LL_DMAMUX1_REQ_I2C1_RX
  - LL_DMAMUX1_REQ_I2C1_TX
  - LL_DMAMUX1_REQ_I2C2_RX
  - LL_DMAMUX1_REQ_I2C2_TX
  - LL_DMAMUX1_REQ_SPI1_RX
  - LL_DMAMUX1_REQ_SPI1_TX
  - LL_DMAMUX1_REQ_SPI2_RX
  - LL_DMAMUX1_REQ_SPI2_TX
  - LL_DMAMUX1_REQ_USART1_RX
  - LL_DMAMUX1_REQ_USART1_TX
  - LL_DMAMUX1_REQ_USART2_RX
  - LL_DMAMUX1_REQ_USART2_TX
  - LL_DMAMUX1_REQ_USART3_RX
  - LL_DMAMUX1_REQ_USART3_TX
  - LL_DMAMUX1_REQ_TIM8_CH1
  - LL_DMAMUX1_REQ_TIM8_CH2
  - LL_DMAMUX1_REQ_TIM8_CH3

- **None:**

Notes

- DMAMUX1 channel 0 to 7 are mapped to DMA1 channel 0 to 7. DMAMUX1 channel 8 to 15 are mapped to DMA2 channel 0 to 7. DMAMUX2 channel 0 to 7 are mapped to BDMA channel 0 to 7.
- (*) Availability depends on devices.

Reference Manual to LL API cross reference:

- CxCR DMAREQ_ID LL_DMAMUX_GetRequestID

LL_DMAMUX_SetSyncRequestNb

Function name

__STATIC_INLINE void LL_DMAMUX_SetSyncRequestNb (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel, uint32_t RequestNb)

Function description

Set the number of DMA request that will be autorized after a synchronization event and/or the number of DMA request needed to generate an event.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15
- **RequestNb:** This parameter must be a value between Min_Data = 1 and Max_Data = 32.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CxCR NBREQ LL_DMAMUX_SetSyncRequestNb

LL_DMAMUX_GetSyncRequestNb

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_GetSyncRequestNb (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Get the number of DMA request that will be autorized after a synchronization event and/or the number of DMA request needed to generate an event.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15

Return values

- **Between:** Min_Data = 1 and Max_Data = 32

Reference Manual to LL API cross reference:

- CxCR NBREQ LL_DMAMUX_GetSyncRequestNb

LL_DMAMUX_SetSyncPolarity

Function name

__STATIC_INLINE void LL_DMAMUX_SetSyncPolarity (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel, uint32_t Polarity)

Function description

Set the polarity of the signal on which the DMA request is synchronized.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15
- **Polarity:** This parameter can be one of the following values:
  - LL_DMAMUX_SYNC_NO_EVENT
  - LL_DMAMUX_SYNC_POL_RISING
  - LL_DMAMUX_SYNC_POL_FALLING
  - LL_DMAMUX_SYNC_POL_RISING_FALLING

Return values

- **None:**

Reference Manual to LL API cross reference:

- CxCR SPOL LL_DMAMUX_SetSyncPolarity

LL_DMAMUX_GetSyncPolarity

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_GetSyncPolarity (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Get the polarity of the signal on which the DMA request is synchronized.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMAMUX_SYNC_NO_EVENT
  - LL_DMAMUX_SYNC_POL_RISING
  - LL_DMAMUX_SYNC_POL_FALLING
  - LL_DMAMUX_SYNC_POL_RISING_FALLING

Reference Manual to LL API cross reference:

- CxCR SPOL LL_DMAMUX_GetSyncPolarity

LL_DMAMUX_EnableEventGeneration

Function name

__STATIC_INLINE void LL_DMAMUX_EnableEventGeneration (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Enable the Event Generation on DMAMUX channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CxCR EGE LL_DMAMUX_EnableEventGeneration

LL_DMAMUX_DisableEventGeneration

Function name

__STATIC_INLINE void LL_DMAMUX_DisableEventGeneration (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Disable the Event Generation on DMAMUX channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CxCR EGE LL_DMAMUX_DisableEventGeneration

LL_DMAMUX_IsEnabledEventGeneration

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsEnabledEventGeneration (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Check if the Event Generation on DMAMUX channel x is enabled or disabled.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CxCR EGE LL_DMAMUX_IsEnabledEventGeneration

LL_DMAMUX_EnableSync

Function name

__STATIC_INLINE void LL_DMAMUX_EnableSync (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Enable the synchronization mode.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CxCR SE LL_DMAMUX_EnableSync

LL_DMAMUX_DisableSync

Function name

__STATIC_INLINE void LL_DMAMUX_DisableSync (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Disable the synchronization mode.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CxCR SE LL_DMAMUX_DisableSync

LL_DMAMUX_IsEnabledSync

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsEnabledSync (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Check if the synchronization mode is enabled or disabled.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CxCR SE LL_DMAMUX_IsEnabledSync

LL_DMAMUX_SetSyncID

Function name

__STATIC_INLINE void LL_DMAMUX_SetSyncID (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel, uint32_t SyncID)

Function description

Set DMAMUX synchronization ID on DMAMUX Channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15
- **SyncID:** This parameter can be one of the following values:
  - LL_DMAMUX1_SYNC_DMAMUX1_CH0_EVT
  - LL_DMAMUX1_SYNC_DMAMUX1_CH1_EVT
  - LL_DMAMUX1_SYNC_DMAMUX1_CH2_EVT
  - LL_DMAMUX1_SYNC_LPTIM1_OUT
  - LL_DMAMUX1_SYNC_LPTIM2_OUT
  - LL_DMAMUX1_SYNC_LPTIM3_OUT
  - LL_DMAMUX1_SYNC_EXTI0
  - LL_DMAMUX1_SYNC_TIM12_TRGO
  - LL_DMAMUX2_SYNC_DMAMUX2_CH0_EVT
  - LL_DMAMUX2_SYNC_DMAMUX2_CH1_EVT
  - LL_DMAMUX2_SYNC_DMAMUX2_CH2_EVT
  - LL_DMAMUX2_SYNC_DMAMUX2_CH3_EVT
  - LL_DMAMUX2_SYNC_DMAMUX2_CH4_EVT
  - LL_DMAMUX2_SYNC_DMAMUX2_CH5_EVT
  - LL_DMAMUX2_SYNC_LPUART1_RX_WKUP
  - LL_DMAMUX2_SYNC_LPUART1_TX_WKUP
  - LL_DMAMUX2_SYNC_LPTIM2_OUT
  - LL_DMAMUX2_SYNC_LPTIM3_OUT
  - LL_DMAMUX2_SYNC_I2C4_WKUP
  - LL_DMAMUX2_SYNC_SPI6_WKUP
  - LL_DMAMUX2_SYNC_COMP1_OUT
  - LL_DMAMUX2_SYNC_RTC_WKUP
  - LL_DMAMUX2_SYNC_EXTI0
  - LL_DMAMUX2_SYNC_EXTI2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CxCR SYNC_ID LL_DMAMUX_SetSyncID

LL_DMAMUX_GetSyncID

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_GetSyncID (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Get DMAMUX synchronization ID on DMAMUX Channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMAMUX1_SYNC_DMAMUX1_CH0_EVT
  - LL_DMAMUX1_SYNC_DMAMUX1_CH1_EVT
  - LL_DMAMUX1_SYNC_DMAMUX1_CH2_EVT
  - LL_DMAMUX1_SYNC_LPTIM1_OUT
  - LL_DMAMUX1_SYNC_LPTIM2_OUT
  - LL_DMAMUX1_SYNC_LPTIM3_OUT
  - LL_DMAMUX1_SYNC_EXTI0
  - LL_DMAMUX1_SYNC_TIM12_TRGO
  - LL_DMAMUX2_SYNC_DMAMUX2_CH0_EVT
  - LL_DMAMUX2_SYNC_DMAMUX2_CH1_EVT
  - LL_DMAMUX2_SYNC_DMAMUX2_CH2_EVT
  - LL_DMAMUX2_SYNC_DMAMUX2_CH3_EVT
  - LL_DMAMUX2_SYNC_DMAMUX2_CH4_EVT
  - LL_DMAMUX2_SYNC_DMAMUX2_CH5_EVT
  - LL_DMAMUX2_SYNC_LPUART1_RX_WKUP
  - LL_DMAMUX2_SYNC_LPUART1_TX_WKUP
  - LL_DMAMUX2_SYNC_LPTIM2_OUT
  - LL_DMAMUX2_SYNC_LPTIM3_OUT
  - LL_DMAMUX2_SYNC_I2C4_WKUP
  - LL_DMAMUX2_SYNC_SPI6_WKUP
  - LL_DMAMUX2_SYNC_COMP1_OUT
  - LL_DMAMUX2_SYNC_RTC_WKUP
  - LL_DMAMUX2_SYNC_EXTI0
  - LL_DMAMUX2_SYNC_EXTI2

Reference Manual to LL API cross reference:

- CxCR SYNC_ID LL_DMAMUX_GetSyncID

LL_DMAMUX_EnableRequestGen

Function name

__STATIC_INLINE void LL_DMAMUX_EnableRequestGen (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)

Function description

Enable the Request Generator.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3
  - LL_DMAMUX_REQ_GEN_4
  - LL_DMAMUX_REQ_GEN_5
  - LL_DMAMUX_REQ_GEN_6
  - LL_DMAMUX_REQ_GEN_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGxCR GE LL_DMAMUX_EnableRequestGen

LL_DMAMUX_DisableRequestGen

Function name

__STATIC_INLINE void LL_DMAMUX_DisableRequestGen (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)

Function description

Disable the Request Generator.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGxCR GE LL_DMAMUX_DisableRequestGen

LL_DMAMUX_IsEnabledRequestGen

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsEnabledRequestGen (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)

Function description

Check if the Request Generator is enabled or disabled.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3
  - LL_DMAMUX_REQ_GEN_4
  - LL_DMAMUX_REQ_GEN_5
  - LL_DMAMUX_REQ_GEN_6
  - LL_DMAMUX_REQ_GEN_7

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RGxCR GE LL_DMAMUX_IsEnabledRequestGen

LL_DMAMUX_SetRequestGenPolarity

Function name

__STATIC_INLINE void LL_DMAMUX_SetRequestGenPolarity (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel, uint32_t Polarity)

Function description

Set the polarity of the signal on which the DMA request is generated.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3
  - LL_DMAMUX_REQ_GEN_4
  - LL_DMAMUX_REQ_GEN_5
  - LL_DMAMUX_REQ_GEN_6
  - LL_DMAMUX_REQ_GEN_7
- **Polarity:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_NO_EVENT
  - LL_DMAMUX_REQ_GEN_POL_RISING
  - LL_DMAMUX_REQ_GEN_POL_FALLING
  - LL_DMAMUX_REQ_GEN_POL_RISING_FALLING

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGxCR GPOL LL_DMAMUX_SetRequestGenPolarity

LL_DMAMUX_GetRequestGenPolarity

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_GetRequestGenPolarity (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)

Function description

Get the polarity of the signal on which the DMA request is generated.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3
  - LL_DMAMUX_REQ_GEN_4
  - LL_DMAMUX_REQ_GEN_5
  - LL_DMAMUX_REQ_GEN_6
  - LL_DMAMUX_REQ_GEN_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMAMUX_REQ_GEN_NO_EVENT
  - LL_DMAMUX_REQ_GEN_POL_RISING
  - LL_DMAMUX_REQ_GEN_POL_FALLING
  - LL_DMAMUX_REQ_GEN_POL_RISING_FALLING

Reference Manual to LL API cross reference:

- RGxCR GPOL LL_DMAMUX_GetRequestGenPolarity

LL_DMAMUX_SetGenRequestNb

Function name

__STATIC_INLINE void LL_DMAMUX_SetGenRequestNb (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel, uint32_t RequestNb)

Function description

Set the number of DMA request that will be autorized after a generation event.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3
  - LL_DMAMUX_REQ_GEN_4
  - LL_DMAMUX_REQ_GEN_5
  - LL_DMAMUX_REQ_GEN_6
  - LL_DMAMUX_REQ_GEN_7
- **RequestNb:** This parameter must be a value between Min_Data = 1 and Max_Data = 32.

Return values

- **None:**

Notes

- This field can only be written when Generator is disabled.

Reference Manual to LL API cross reference:

- RGxCR GNBREQ LL_DMAMUX_SetGenRequestNb

LL_DMAMUX_GetGenRequestNb

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_GetGenRequestNb (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)

Function description

Get the number of DMA request that will be autorized after a generation event.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3
  - LL_DMAMUX_REQ_GEN_4
  - LL_DMAMUX_REQ_GEN_5
  - LL_DMAMUX_REQ_GEN_6
  - LL_DMAMUX_REQ_GEN_7

Return values

- **Between:** Min_Data = 1 and Max_Data = 32

Reference Manual to LL API cross reference:

- RGxCR GNBREQ LL_DMAMUX_GetGenRequestNb

LL_DMAMUX_SetRequestSignalID

Function name

__STATIC_INLINE void LL_DMAMUX_SetRequestSignalID (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel, uint32_t RequestSignalID)

Function description

Set DMAMUX external Request Signal ID on DMAMUX Request Generation Trigger Event Channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3
  - LL_DMAMUX_REQ_GEN_4
  - LL_DMAMUX_REQ_GEN_5
  - LL_DMAMUX_REQ_GEN_6
  - LL_DMAMUX_REQ_GEN_7

- **RequestSignalID:** This parameter can be one of the following values:
  - LL_DMAMUX1_REQ_GEN_DMAMUX1_CH0_EVT
  - LL_DMAMUX1_REQ_GEN_DMAMUX1_CH1_EVT
  - LL_DMAMUX1_REQ_GEN_DMAMUX1_CH2_EVT
  - LL_DMAMUX1_REQ_GEN_LPTIM1_OUT
  - LL_DMAMUX1_REQ_GEN_LPTIM2_OUT
  - LL_DMAMUX1_REQ_GEN_LPTIM3_OUT
  - LL_DMAMUX1_REQ_GEN_EXTI0
  - LL_DMAMUX1_REQ_GEN_TIM12_TRGO
  - LL_DMAMUX2_REQ_GEN_DMAMUX2_CH0_EVT
  - LL_DMAMUX2_REQ_GEN_DMAMUX2_CH1_EVT
  - LL_DMAMUX2_REQ_GEN_DMAMUX2_CH2_EVT
  - LL_DMAMUX2_REQ_GEN_DMAMUX2_CH3_EVT
  - LL_DMAMUX2_REQ_GEN_DMAMUX2_CH4_EVT
  - LL_DMAMUX2_REQ_GEN_DMAMUX2_CH5_EVT
  - LL_DMAMUX2_REQ_GEN_DMAMUX2_CH6_EVT
  - LL_DMAMUX2_REQ_GEN_LPUART1_RX_WKUP
  - LL_DMAMUX2_REQ_GEN_LPUART1_TX_WKUP
  - LL_DMAMUX2_REQ_GEN_LPTIM2_WKUP
  - LL_DMAMUX2_REQ_GEN_LPTIM2_OUT
  - LL_DMAMUX2_REQ_GEN_LPTIM3_WKUP
  - LL_DMAMUX2_REQ_GEN_LPTIM3_OUT
  - LL_DMAMUX2_REQ_GEN_LPTIM4_WKUP (*)
  - LL_DMAMUX2_REQ_GEN_LPTIM5_WKUP (*)
  - LL_DMAMUX2_REQ_GEN_I2C4_WKUP
  - LL_DMAMUX2_REQ_GEN_SPI6_WKUP
  - LL_DMAMUX2_REQ_GEN_COMP1_OUT
  - LL_DMAMUX2_REQ_GEN_COMP2_OUT
  - LL_DMAMUX2_REQ_GEN_RTC_WKUP
  - LL_DMAMUX2_REQ_GEN_EXTI0
  - LL_DMAMUX2_REQ_GEN_EXTI2
  - LL_DMAMUX2_REQ_GEN_I2C4_IT_EVT
  - LL_DMAMUX2_REQ_GEN_SPI6_IT
  - LL_DMAMUX2_REQ_GEN_LPUART1_TX_IT
  - LL_DMAMUX2_REQ_GEN_LPUART1_RX_IT
  - LL_DMAMUX2_REQ_GEN_ADC3_IT (*)
  - LL_DMAMUX2_REQ_GEN_ADC3_AWD1_OUT (*)
  - LL_DMAMUX2_REQ_GEN_BDMA_CH0_IT
  - LL_DMAMUX2_REQ_GEN_BDMA_CH1_IT

Return values

- **None:**

Notes

- (*) Availability depends on devices.

Reference Manual to LL API cross reference:

- RGxCR SIG_ID LL_DMAMUX_SetRequestSignalID

LL_DMAMUX_GetRequestSignalID

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_GetRequestSignalID (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)

Function description

Get DMAMUX external Request Signal ID set on DMAMUX Channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3
  - LL_DMAMUX_REQ_GEN_4
  - LL_DMAMUX_REQ_GEN_5
  - LL_DMAMUX_REQ_GEN_6
  - LL_DMAMUX_REQ_GEN_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMAMUX1_SYNC_DMAMUX1_CH0_EVT
  - LL_DMAMUX1_SYNC_DMAMUX1_CH1_EVT
  - LL_DMAMUX1_SYNC_DMAMUX1_CH2_EVT
  - LL_DMAMUX1_SYNC_LPTIM1_OUT
  - LL_DMAMUX1_SYNC_LPTIM2_OUT
  - LL_DMAMUX1_SYNC_LPTIM3_OUT
  - LL_DMAMUX1_SYNC_EXTI0
  - LL_DMAMUX1_SYNC_TIM12_TRGO
  - LL_DMAMUX2_SYNC_DMAMUX2_CH0_EVT
  - LL_DMAMUX2_SYNC_DMAMUX2_CH1_EVT
  - LL_DMAMUX2_SYNC_DMAMUX2_CH2_EVT
  - LL_DMAMUX2_SYNC_DMAMUX2_CH3_EVT
  - LL_DMAMUX2_SYNC_DMAMUX2_CH4_EVT
  - LL_DMAMUX2_SYNC_DMAMUX2_CH5_EVT
  - LL_DMAMUX2_SYNC_LPUART1_RX_WKUP
  - LL_DMAMUX2_SYNC_LPUART1_TX_WKUP
  - LL_DMAMUX2_SYNC_LPTIM2_OUT
  - LL_DMAMUX2_SYNC_LPTIM3_OUT
  - LL_DMAMUX2_SYNC_I2C4_WKUP
  - LL_DMAMUX2_SYNC_SPI6_WKUP
  - LL_DMAMUX2_SYNC_COMP1_OUT
  - LL_DMAMUX2_SYNC_RTC_WKUP
  - LL_DMAMUX2_SYNC_EXTI0
  - LL_DMAMUX2_SYNC_EXTI2

Reference Manual to LL API cross reference:

- RGxCR SIG_ID LL_DMAMUX_GetRequestSignalID

LL_DMAMUX_IsActiveFlag_SO0

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO0 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 0.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF0 LL_DMAMUX_IsActiveFlag_SO0

LL_DMAMUX_IsActiveFlag_SO1

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO1 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 1.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF1 LL_DMAMUX_IsActiveFlag_SO1

LL_DMAMUX_IsActiveFlag_SO2

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO2 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 2.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF2 LL_DMAMUX_IsActiveFlag_SO2

LL_DMAMUX_IsActiveFlag_SO3

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO3 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 3.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF3 LL_DMAMUX_IsActiveFlag_SO3

LL_DMAMUX_IsActiveFlag_SO4

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO4 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 4.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF4 LL_DMAMUX_IsActiveFlag_SO4

LL_DMAMUX_IsActiveFlag_SO5

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO5 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 5.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF5 LL_DMAMUX_IsActiveFlag_SO5

LL_DMAMUX_IsActiveFlag_SO6

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO6 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 6.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF6 LL_DMAMUX_IsActiveFlag_SO6

LL_DMAMUX_IsActiveFlag_SO7

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO7 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 7.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF7 LL_DMAMUX_IsActiveFlag_SO7

LL_DMAMUX_IsActiveFlag_SO8

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO8 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 8.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF8 LL_DMAMUX_IsActiveFlag_SO8

LL_DMAMUX_IsActiveFlag_SO9

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO9 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 9.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF9 LL_DMAMUX_IsActiveFlag_SO9

LL_DMAMUX_IsActiveFlag_SO10

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO10 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 10.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF10 LL_DMAMUX_IsActiveFlag_SO10

LL_DMAMUX_IsActiveFlag_SO11

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO11 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 11.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF11 LL_DMAMUX_IsActiveFlag_SO11

LL_DMAMUX_IsActiveFlag_SO12

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO12 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 12.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF12 LL_DMAMUX_IsActiveFlag_SO12

LL_DMAMUX_IsActiveFlag_SO13

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO13 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 13.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF13 LL_DMAMUX_IsActiveFlag_SO13

LL_DMAMUX_IsActiveFlag_SO14

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO14 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 14.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF14 LL_DMAMUX_IsActiveFlag_SO14

LL_DMAMUX_IsActiveFlag_SO15

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO15 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 15.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF15 LL_DMAMUX_IsActiveFlag_SO15

LL_DMAMUX_IsActiveFlag_RGO0

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_RGO0 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Request Generator 0 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RGSR OF0 LL_DMAMUX_IsActiveFlag_RGO0

LL_DMAMUX_IsActiveFlag_RGO1

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_RGO1 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Request Generator 1 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RGSR OF1 LL_DMAMUX_IsActiveFlag_RGO1

LL_DMAMUX_IsActiveFlag_RGO2

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_RGO2 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Request Generator 2 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RGSR OF2 LL_DMAMUX_IsActiveFlag_RGO2

LL_DMAMUX_IsActiveFlag_RGO3

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_RGO3 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Request Generator 3 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RGSR OF3 LL_DMAMUX_IsActiveFlag_RGO3

LL_DMAMUX_IsActiveFlag_RGO4

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_RGO4 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Request Generator 4 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RGSR OF4 LL_DMAMUX_IsActiveFlag_RGO4

LL_DMAMUX_IsActiveFlag_RGO5

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_RGO5 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Request Generator 5 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RGSR OF5 LL_DMAMUX_IsActiveFlag_RGO5

LL_DMAMUX_IsActiveFlag_RGO6

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_RGO6 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Request Generator 6 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RGSR OF6 LL_DMAMUX_IsActiveFlag_RGO6

LL_DMAMUX_IsActiveFlag_RGO7

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_RGO7 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Request Generator 7 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RGSR OF7 LL_DMAMUX_IsActiveFlag_RGO7

LL_DMAMUX_ClearFlag_SO0

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO0 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 0.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF0 LL_DMAMUX_ClearFlag_SO0

LL_DMAMUX_ClearFlag_SO1

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO1 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 1.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF1 LL_DMAMUX_ClearFlag_SO1

LL_DMAMUX_ClearFlag_SO2

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO2 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 2.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF2 LL_DMAMUX_ClearFlag_SO2

LL_DMAMUX_ClearFlag_SO3

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO3 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 3.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF3 LL_DMAMUX_ClearFlag_SO3

LL_DMAMUX_ClearFlag_SO4

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO4 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 4.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF4 LL_DMAMUX_ClearFlag_SO4

LL_DMAMUX_ClearFlag_SO5

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO5 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 5.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF5 LL_DMAMUX_ClearFlag_SO5

LL_DMAMUX_ClearFlag_SO6

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO6 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 6.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF6 LL_DMAMUX_ClearFlag_SO6

LL_DMAMUX_ClearFlag_SO7

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO7 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 7.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF7 LL_DMAMUX_ClearFlag_SO7

LL_DMAMUX_ClearFlag_SO8

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO8 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 8.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF8 LL_DMAMUX_ClearFlag_SO8

LL_DMAMUX_ClearFlag_SO9

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO9 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 9.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF9 LL_DMAMUX_ClearFlag_SO9

LL_DMAMUX_ClearFlag_SO10

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO10 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 10.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF10 LL_DMAMUX_ClearFlag_SO10

LL_DMAMUX_ClearFlag_SO11

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO11 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 11.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF11 LL_DMAMUX_ClearFlag_SO11

LL_DMAMUX_ClearFlag_SO12

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO12 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 12.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF12 LL_DMAMUX_ClearFlag_SO12

LL_DMAMUX_ClearFlag_SO13

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO13 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 13.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF13 LL_DMAMUX_ClearFlag_SO13

LL_DMAMUX_ClearFlag_SO14

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO14 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 14.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF14 LL_DMAMUX_ClearFlag_SO14

LL_DMAMUX_ClearFlag_SO15

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO15 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 15.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF15 LL_DMAMUX_ClearFlag_SO15

LL_DMAMUX_ClearFlag_RGO0

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_RGO0 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Request Generator 0 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGCFR COF0 LL_DMAMUX_ClearFlag_RGO0

LL_DMAMUX_ClearFlag_RGO1

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_RGO1 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Request Generator 1 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGCFR COF1 LL_DMAMUX_ClearFlag_RGO1

LL_DMAMUX_ClearFlag_RGO2

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_RGO2 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Request Generator 2 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGCFR COF2 LL_DMAMUX_ClearFlag_RGO2

LL_DMAMUX_ClearFlag_RGO3

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_RGO3 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Request Generator 3 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGCFR COF3 LL_DMAMUX_ClearFlag_RGO3

LL_DMAMUX_ClearFlag_RGO4

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_RGO4 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Request Generator 4 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGCFR COF4 LL_DMAMUX_ClearFlag_RGO4

LL_DMAMUX_ClearFlag_RGO5

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_RGO5 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Request Generator 5 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGCFR COF5 LL_DMAMUX_ClearFlag_RGO5

LL_DMAMUX_ClearFlag_RGO6

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_RGO6 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Request Generator 6 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGCFR COF6 LL_DMAMUX_ClearFlag_RGO6

LL_DMAMUX_ClearFlag_RGO7

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_RGO7 (DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Request Generator 7 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGCFR COF7 LL_DMAMUX_ClearFlag_RGO7

LL_DMAMUX_EnableIT_SO

Function name

__STATIC_INLINE void LL_DMAMUX_EnableIT_SO (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Enable the Synchronization Event Overrun Interrupt on DMAMUX channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CxCR SOIE LL_DMAMUX_EnableIT_SO

LL_DMAMUX_DisableIT_SO

Function name

__STATIC_INLINE void LL_DMAMUX_DisableIT_SO (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Disable the Synchronization Event Overrun Interrupt on DMAMUX channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CxCR SOIE LL_DMAMUX_DisableIT_SO

LL_DMAMUX_IsEnabledIT_SO

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsEnabledIT_SO (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Check if the Synchronization Event Overrun Interrupt on DMAMUX channel x is enabled or disabled.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3
  - LL_DMAMUX_CHANNEL_4
  - LL_DMAMUX_CHANNEL_5
  - LL_DMAMUX_CHANNEL_6
  - LL_DMAMUX_CHANNEL_7
  - LL_DMAMUX_CHANNEL_8
  - LL_DMAMUX_CHANNEL_9
  - LL_DMAMUX_CHANNEL_10
  - LL_DMAMUX_CHANNEL_11
  - LL_DMAMUX_CHANNEL_12
  - LL_DMAMUX_CHANNEL_13
  - LL_DMAMUX_CHANNEL_14
  - LL_DMAMUX_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CxCR SOIE LL_DMAMUX_IsEnabledIT_SO

LL_DMAMUX_EnableIT_RGO

Function name

__STATIC_INLINE void LL_DMAMUX_EnableIT_RGO (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)

Function description

Enable the Request Generation Trigger Event Overrun Interrupt on DMAMUX channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3
  - LL_DMAMUX_REQ_GEN_4
  - LL_DMAMUX_REQ_GEN_5
  - LL_DMAMUX_REQ_GEN_6
  - LL_DMAMUX_REQ_GEN_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGxCR OIE LL_DMAMUX_EnableIT_RGO

LL_DMAMUX_DisableIT_RGO

Function name

__STATIC_INLINE void LL_DMAMUX_DisableIT_RGO (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)

Function description

Disable the Request Generation Trigger Event Overrun Interrupt on DMAMUX channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3
  - LL_DMAMUX_REQ_GEN_4
  - LL_DMAMUX_REQ_GEN_5
  - LL_DMAMUX_REQ_GEN_6
  - LL_DMAMUX_REQ_GEN_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGxCR OIE LL_DMAMUX_DisableIT_RGO

LL_DMAMUX_IsEnabledIT_RGO

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsEnabledIT_RGO (DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)

Function description

Check if the Request Generation Trigger Event Overrun Interrupt on DMAMUX channel x is enabled or disabled.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3
  - LL_DMAMUX_REQ_GEN_4
  - LL_DMAMUX_REQ_GEN_5
  - LL_DMAMUX_REQ_GEN_6
  - LL_DMAMUX_REQ_GEN_7

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RGxCR OIE LL_DMAMUX_IsEnabledIT_RGO

106.2 DMAMUX Firmware driver defines

The following section lists the various define and macros of the module.

106.2.1 DMAMUX

DMAMUX

DMAMUX Channel

LL_DMAMUX_CHANNEL_0

DMAMUX1 Channel 0 connected to DMA1 Channel 0 , DMAMUX2 Channel 0 connected to BDMA Channel 0

LL_DMAMUX_CHANNEL_1

DMAMUX1 Channel 1 connected to DMA1 Channel 1 , DMAMUX2 Channel 1 connected to BDMA Channel 1

LL_DMAMUX_CHANNEL_2

DMAMUX1 Channel 2 connected to DMA1 Channel 2 , DMAMUX2 Channel 2 connected to BDMA Channel 2

LL_DMAMUX_CHANNEL_3

DMAMUX1 Channel 3 connected to DMA1 Channel 3 , DMAMUX2 Channel 3 connected to BDMA Channel 3

LL_DMAMUX_CHANNEL_4

DMAMUX1 Channel 4 connected to DMA1 Channel 4 , DMAMUX2 Channel 4 connected to BDMA Channel 4

LL_DMAMUX_CHANNEL_5

DMAMUX1 Channel 5 connected to DMA1 Channel 5 , DMAMUX2 Channel 5 connected to BDMA Channel 5

LL_DMAMUX_CHANNEL_6

DMAMUX1 Channel 6 connected to DMA1 Channel 6 , DMAMUX2 Channel 6 connected to BDMA Channel 6

LL_DMAMUX_CHANNEL_7

DMAMUX1 Channel 7 connected to DMA1 Channel 7 , DMAMUX2 Channel 7 connected to BDMA Channel 7

LL_DMAMUX_CHANNEL_8

DMAMUX1 Channel 8 connected to DMA2 Channel 0

LL_DMAMUX_CHANNEL_9

DMAMUX1 Channel 9 connected to DMA2 Channel 1

LL_DMAMUX_CHANNEL_10

DMAMUX1 Channel 10 connected to DMA2 Channel 2

LL_DMAMUX_CHANNEL_11

DMAMUX1 Channel 11 connected to DMA2 Channel 3

LL_DMAMUX_CHANNEL_12

DMAMUX1 Channel 12 connected to DMA2 Channel 4

LL_DMAMUX_CHANNEL_13

DMAMUX1 Channel 13 connected to DMA2 Channel 5

LL_DMAMUX_CHANNEL_14

DMAMUX1 Channel 14 connected to DMA2 Channel 6

LL_DMAMUX_CHANNEL_15

DMAMUX1 Channel 15 connected to DMA2 Channel 7

Clear Flags Defines

LL_DMAMUX_CFR_CSOF0

Synchronization Event Overrun Flag Channel 0

LL_DMAMUX_CFR_CSOF1

Synchronization Event Overrun Flag Channel 1

LL_DMAMUX_CFR_CSOF2

Synchronization Event Overrun Flag Channel 2

LL_DMAMUX_CFR_CSOF3

Synchronization Event Overrun Flag Channel 3

LL_DMAMUX_CFR_CSOF4

Synchronization Event Overrun Flag Channel 4

LL_DMAMUX_CFR_CSOF5

Synchronization Event Overrun Flag Channel 5

LL_DMAMUX_CFR_CSOF6

Synchronization Event Overrun Flag Channel 6

LL_DMAMUX_CFR_CSOF7

Synchronization Event Overrun Flag Channel 7

LL_DMAMUX_CFR_CSOF8

Synchronization Event Overrun Flag Channel 8

LL_DMAMUX_CFR_CSOF9

Synchronization Event Overrun Flag Channel 9

LL_DMAMUX_CFR_CSOF10

Synchronization Event Overrun Flag Channel 10

LL_DMAMUX_CFR_CSOF11

Synchronization Event Overrun Flag Channel 11

LL_DMAMUX_CFR_CSOF12

Synchronization Event Overrun Flag Channel 12

LL_DMAMUX_CFR_CSOF13

Synchronization Event Overrun Flag Channel 13

LL_DMAMUX_CFR_CSOF14

Synchronization Event Overrun Flag Channel 14

LL_DMAMUX_CFR_CSOF15

Synchronization Event Overrun Flag Channel 15

LL_DMAMUX_RGCFR_RGCOF0

Request Generator 0 Trigger Event Overrun Flag

LL_DMAMUX_RGCFR_RGCOF1

Request Generator 1 Trigger Event Overrun Flag

LL_DMAMUX_RGCFR_RGCOF2

Request Generator 2 Trigger Event Overrun Flag

LL_DMAMUX_RGCFR_RGCOF3

Request Generator 3 Trigger Event Overrun Flag

LL_DMAMUX_RGCFR_RGCOF4

Request Generator 4 Trigger Event Overrun Flag

LL_DMAMUX_RGCFR_RGCOF5

Request Generator 5 Trigger Event Overrun Flag

LL_DMAMUX_RGCFR_RGCOF6

Request Generator 6 Trigger Event Overrun Flag

LL_DMAMUX_RGCFR_RGCOF7

Request Generator 7 Trigger Event Overrun Flag

Get Flags Defines

LL_DMAMUX_CSR_SOF0

Synchronization Event Overrun Flag Channel 0

LL_DMAMUX_CSR_SOF1

Synchronization Event Overrun Flag Channel 1

LL_DMAMUX_CSR_SOF2

Synchronization Event Overrun Flag Channel 2

LL_DMAMUX_CSR_SOF3

Synchronization Event Overrun Flag Channel 3

LL_DMAMUX_CSR_SOF4

Synchronization Event Overrun Flag Channel 4

LL_DMAMUX_CSR_SOF5

Synchronization Event Overrun Flag Channel 5

LL_DMAMUX_CSR_SOF6

Synchronization Event Overrun Flag Channel 6

LL_DMAMUX_CSR_SOF7

Synchronization Event Overrun Flag Channel 7

LL_DMAMUX_CSR_SOF8

Synchronization Event Overrun Flag Channel 8

LL_DMAMUX_CSR_SOF9

Synchronization Event Overrun Flag Channel 9

LL_DMAMUX_CSR_SOF10

Synchronization Event Overrun Flag Channel 10

LL_DMAMUX_CSR_SOF11

Synchronization Event Overrun Flag Channel 11

LL_DMAMUX_CSR_SOF12

Synchronization Event Overrun Flag Channel 12

LL_DMAMUX_CSR_SOF13

Synchronization Event Overrun Flag Channel 13

LL_DMAMUX_CSR_SOF14

Synchronization Event Overrun Flag Channel 14

LL_DMAMUX_CSR_SOF15

Synchronization Event Overrun Flag Channel 15

LL_DMAMUX_RGSR_RGOF0

Request Generator 0 Trigger Event Overrun Flag

LL_DMAMUX_RGSR_RGOF1

Request Generator 1 Trigger Event Overrun Flag

LL_DMAMUX_RGSR_RGOF2

Request Generator 2 Trigger Event Overrun Flag

LL_DMAMUX_RGSR_RGOF3

Request Generator 3 Trigger Event Overrun Flag

LL_DMAMUX_RGSR_RGOF4

Request Generator 4 Trigger Event Overrun Flag

LL_DMAMUX_RGSR_RGOF5

Request Generator 5 Trigger Event Overrun Flag

LL_DMAMUX_RGSR_RGOF6

Request Generator 6 Trigger Event Overrun Flag

LL_DMAMUX_RGSR_RGOF7

Request Generator 7 Trigger Event Overrun Flag

IT Defines

LL_DMAMUX_CCR_SOIE

Synchronization Event Overrun Interrupt

LL_DMAMUX_RGCR_RGOIE

Request Generation Trigger Event Overrun Interrupt

External Request Signal Generation

LL_DMAMUX1_REQ_GEN_DMAMUX1_CH0_EVT

DMAMUX1 Request generator Signal is DMAMUX1 Channel0 Event

LL_DMAMUX1_REQ_GEN_DMAMUX1_CH1_EVT

DMAMUX1 Request generator Signal is DMAMUX1 Channel1 Event

LL_DMAMUX1_REQ_GEN_DMAMUX1_CH2_EVT

DMAMUX1 Request generator Signal is DMAMUX1 Channel2 Event

LL_DMAMUX1_REQ_GEN_LPTIM1_OUT

DMAMUX1 Request generator Signal is LPTIM1 OUT

LL_DMAMUX1_REQ_GEN_LPTIM2_OUT

DMAMUX1 Request generator Signal is LPTIM2 OUT

LL_DMAMUX1_REQ_GEN_LPTIM3_OUT

DMAMUX1 Request generator Signal is LPTIM3 OUT

LL_DMAMUX1_REQ_GEN_EXTI0

DMAMUX1 Request generator Signal is EXTI0 IT

LL_DMAMUX1_REQ_GEN_TIM12_TRGO

DMAMUX1 Request generator Signal is TIM12 TRGO

LL_DMAMUX2_REQ_GEN_DMAMUX2_CH0_EVT

DMAMUX2 Request generator Signal is DMAMUX2 Channel0 Event

LL_DMAMUX2_REQ_GEN_DMAMUX2_CH1_EVT

DMAMUX2 Request generator Signal is DMAMUX2 Channel1 Event

LL_DMAMUX2_REQ_GEN_DMAMUX2_CH2_EVT

DMAMUX2 Request generator Signal is DMAMUX2 Channel2 Event

LL_DMAMUX2_REQ_GEN_DMAMUX2_CH3_EVT

DMAMUX2 Request generator Signal is DMAMUX2 Channel3 Event

LL_DMAMUX2_REQ_GEN_DMAMUX2_CH4_EVT

DMAMUX2 Request generator Signal is DMAMUX2 Channel4 Event

LL_DMAMUX2_REQ_GEN_DMAMUX2_CH5_EVT

DMAMUX2 Request generator Signal is DMAMUX2 Channel5 Event

LL_DMAMUX2_REQ_GEN_DMAMUX2_CH6_EVT

DMAMUX2 Request generator Signal is DMAMUX2 Channel6 Event

LL_DMAMUX2_REQ_GEN_LPUART1_RX_WKUP

DMAMUX2 Request generator Signal is LPUART1 RX Wakeup

LL_DMAMUX2_REQ_GEN_LPUART1_TX_WKUP

DMAMUX2 Request generator Signal is LPUART1 TX Wakeup

LL_DMAMUX2_REQ_GEN_LPTIM2_WKUP

DMAMUX2 Request generator Signal is LPTIM2 Wakeup

LL_DMAMUX2_REQ_GEN_LPTIM2_OUT

DMAMUX2 Request generator Signal is LPTIM2 OUT

LL_DMAMUX2_REQ_GEN_LPTIM3_WKUP

DMAMUX2 Request generator Signal is LPTIM3 Wakeup

LL_DMAMUX2_REQ_GEN_LPTIM3_OUT

DMAMUX2 Request generator Signal is LPTIM3 OUT

LL_DMAMUX2_REQ_GEN_LPTIM4_WKUP

DMAMUX2 Request generator Signal is LPTIM4 Wakeup

LL_DMAMUX2_REQ_GEN_LPTIM5_WKUP

DMAMUX2 Request generator Signal is LPTIM5 Wakeup

LL_DMAMUX2_REQ_GEN_I2C4_WKUP

DMAMUX2 Request generator Signal is I2C4 Wakeup

LL_DMAMUX2_REQ_GEN_SPI6_WKUP

DMAMUX2 Request generator Signal is SPI6 Wakeup

LL_DMAMUX2_REQ_GEN_COMP1_OUT

DMAMUX2 Request generator Signal is Comparator 1 output

LL_DMAMUX2_REQ_GEN_COMP2_OUT

DMAMUX2 Request generator Signal is Comparator 2 output

LL_DMAMUX2_REQ_GEN_RTC_WKUP

DMAMUX2 Request generator Signal is RTC Wakeup

LL_DMAMUX2_REQ_GEN_EXTI0

DMAMUX2 Request generator Signal is EXTI0

LL_DMAMUX2_REQ_GEN_EXTI2

DMAMUX2 Request generator Signal is EXTI2

LL_DMAMUX2_REQ_GEN_I2C4_IT_EVT

DMAMUX2 Request generator Signal is I2C4 IT Event

LL_DMAMUX2_REQ_GEN_SPI6_IT

DMAMUX2 Request generator Signal is SPI6 IT

LL_DMAMUX2_REQ_GEN_LPUART1_TX_IT

DMAMUX2 Request generator Signal is LPUART1 Tx IT

LL_DMAMUX2_REQ_GEN_LPUART1_RX_IT

DMAMUX2 Request generator Signal is LPUART1 Rx IT

LL_DMAMUX2_REQ_GEN_ADC3_IT

DMAMUX2 Request generator Signal is ADC3 IT

LL_DMAMUX2_REQ_GEN_ADC3_AWD1_OUT

DMAMUX2 Request generator Signal is ADC3 Analog Watchdog 1 output

LL_DMAMUX2_REQ_GEN_BDMA_CH0_IT

DMAMUX2 Request generator Signal is BDMA Channel 0 IT

LL_DMAMUX2_REQ_GEN_BDMA_CH1_IT

DMAMUX2 Request generator Signal is BDMA Channel 1 IT

Request Generator Channel

LL_DMAMUX_REQ_GEN_0

LL_DMAMUX_REQ_GEN_1

LL_DMAMUX_REQ_GEN_2

LL_DMAMUX_REQ_GEN_3

LL_DMAMUX_REQ_GEN_4

LL_DMAMUX_REQ_GEN_5

LL_DMAMUX_REQ_GEN_6

LL_DMAMUX_REQ_GEN_7

External Request Signal Generation Polarity

LL_DMAMUX_REQ_GEN_NO_EVENT

No external DMA request generation

LL_DMAMUX_REQ_GEN_POL_RISING

External DMA request generation on event on rising edge

LL_DMAMUX_REQ_GEN_POL_FALLING

External DMA request generation on event on falling edge

LL_DMAMUX_REQ_GEN_POL_RISING_FALLING

External DMA request generation on rising and falling edge

Synchronization Signal Event

LL_DMAMUX1_SYNC_DMAMUX1_CH0_EVT

DMAMUX1 synchronization Signal is DMAMUX1 Channel0 Event

LL_DMAMUX1_SYNC_DMAMUX1_CH1_EVT

DMAMUX1 synchronization Signal is DMAMUX1 Channel1 Event

LL_DMAMUX1_SYNC_DMAMUX1_CH2_EVT

DMAMUX1 synchronization Signal is DMAMUX1 Channel2 Event

LL_DMAMUX1_SYNC_LPTIM1_OUT

DMAMUX1 synchronization Signal is LPTIM1 OUT

LL_DMAMUX1_SYNC_LPTIM2_OUT

DMAMUX1 synchronization Signal is LPTIM2 OUT

LL_DMAMUX1_SYNC_LPTIM3_OUT

DMAMUX1 synchronization Signal is LPTIM3 OUT

LL_DMAMUX1_SYNC_EXTI0

DMAMUX1 synchronization Signal is EXTI0 IT

LL_DMAMUX1_SYNC_TIM12_TRGO

DMAMUX1 synchronization Signal is TIM12 TRGO

LL_DMAMUX2_SYNC_DMAMUX2_CH0_EVT

DMAMUX2 synchronization Signal is DMAMUX2 Channel0 Event

LL_DMAMUX2_SYNC_DMAMUX2_CH1_EVT

DMAMUX2 synchronization Signal is DMAMUX2 Channel1 Event

LL_DMAMUX2_SYNC_DMAMUX2_CH2_EVT

DMAMUX2 synchronization Signal is DMAMUX2 Channel2 Event

LL_DMAMUX2_SYNC_DMAMUX2_CH3_EVT

DMAMUX2 synchronization Signal is DMAMUX2 Channel3 Event

LL_DMAMUX2_SYNC_DMAMUX2_CH4_EVT

DMAMUX2 synchronization Signal is DMAMUX2 Channel4 Event

LL_DMAMUX2_SYNC_DMAMUX2_CH5_EVT

DMAMUX2 synchronization Signal is DMAMUX2 Channel5 Event

LL_DMAMUX2_SYNC_LPUART1_RX_WKUP

DMAMUX2 synchronization Signal is LPUART1 RX Wakeup

LL_DMAMUX2_SYNC_LPUART1_TX_WKUP

DMAMUX2 synchronization Signal is LPUART1 TX Wakeup

LL_DMAMUX2_SYNC_LPTIM2_OUT

DMAMUX2 synchronization Signal is LPTIM2 output

LL_DMAMUX2_SYNC_LPTIM3_OUT

DMAMUX2 synchronization Signal is LPTIM3 output

LL_DMAMUX2_SYNC_I2C4_WKUP

DMAMUX2 synchronization Signal is I2C4 Wakeup

LL_DMAMUX2_SYNC_SPI6_WKUP

DMAMUX2 synchronization Signal is SPI6 Wakeup

LL_DMAMUX2_SYNC_COMP1_OUT

DMAMUX2 synchronization Signal is Comparator 1 output

LL_DMAMUX2_SYNC_RTC_WKUP

DMAMUX2 synchronization Signal is RTC Wakeup

LL_DMAMUX2_SYNC_EXTI0

DMAMUX2 synchronization Signal is EXTI0 IT

LL_DMAMUX2_SYNC_EXTI2

DMAMUX2 synchronization Signal is EXTI2 IT

Synchronization Signal Polarity

LL_DMAMUX_SYNC_NO_EVENT

All requests are blocked

LL_DMAMUX_SYNC_POL_RISING

Synchronization on event on rising edge

LL_DMAMUX_SYNC_POL_FALLING

Synchronization on event on falling edge

LL_DMAMUX_SYNC_POL_RISING_FALLING

Synchronization on event on rising and falling edge

Common Write and read registers macros

LL_DMAMUX_WriteReg

Description:

- Write a value in DMAMUX register.

Parameters:

- __INSTANCE__: DMAMUX Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_DMAMUX_ReadReg

Description:

- Read a value in DMAMUX register.

Parameters:

- __INSTANCE__: DMAMUX Instance
- __REG__: Register to be read

Return value:

- Register: value

107 LL DMA Generic Driver

107.1 DMA Firmware driver registers structures

107.1.1 LL_DMA_InitTypeDef

LL_DMA_InitTypeDef is defined in the stm32h7xx_ll_dma.h

Data Fields

- ***uint32_t PeriphOrM2MSrcAddress***
- ***uint32_t MemoryOrM2MDstAddress***
- ***uint32_t Direction***
- ***uint32_t Mode***
- ***uint32_t PeriphOrM2MSrcIncMode***
- ***uint32_t MemoryOrM2MDstIncMode***
- ***uint32_t PeriphOrM2MSrcDataSize***
- ***uint32_t MemoryOrM2MDstDataSize***
- ***uint32_t NbData***
- ***uint32_t PeriphRequest***
- ***uint32_t Priority***
- ***uint32_t FIFOMode***
- ***uint32_t FIFOThreshold***
- ***uint32_t MemBurst***
- ***uint32_t PeriphBurst***

Field Documentation

- ***uint32_t LL_DMA_InitTypeDef::PeriphOrM2MSrcAddress***

  Specifies the peripheral base address for DMA transfer or as Source base address in case of memory to memory transfer direction.This parameter must be a value between Min_Data = 0 and Max_Data = 0xFFFFFFFF.

- ***uint32_t LL_DMA_InitTypeDef::MemoryOrM2MDstAddress***

  Specifies the memory base address for DMA transfer or as Destination base address in case of memory to memory transfer direction.This parameter must be a value between Min_Data = 0 and Max_Data = 0xFFFFFFFF.

- ***uint32_t LL_DMA_InitTypeDef::Direction***

  Specifies if the data will be transferred from memory to peripheral, from memory to memory or from peripheral to memory. This parameter can be a value of ***DMA_LL_EC_DIRECTION***This feature can be modified afterwards using unitary function **LL_DMA_SetDataTransferDirection()**.

- ***uint32_t LL_DMA_InitTypeDef::Mode***

  Specifies the normal or circular operation mode. This parameter can be a value of ***DMA_LL_EC_MODE***

  Note:

  – The circular buffer mode cannot be used if the memory to memory data transfer direction is configured on the selected Stream

  This feature can be modified afterwards using unitary function **LL_DMA_SetMode()**.

- ***uint32_t LL_DMA_InitTypeDef::PeriphOrM2MSrcIncMode***

  Specifies whether the Peripheral address or Source address in case of memory to memory transfer direction is incremented or not. This parameter can be a value of ***DMA_LL_EC_PERIPH***This feature can be modified afterwards using unitary function **LL_DMA_SetPeriphIncMode()**.

- ***uint32_t LL_DMA_InitTypeDef::MemoryOrM2MDstIncMode***

  Specifies whether the Memory address or Destination address in case of memory to memory transfer direction is incremented or not. This parameter can be a value of ***DMA_LL_EC_MEMORY***This feature can be modified afterwards using unitary function **LL_DMA_SetMemoryIncMode()**.
- ***uint32_t LL_DMA_InitTypeDef::PeriphOrM2MSrcDataSize***

  Specifies the Peripheral data size alignment or Source data size alignment (byte, half word, word) in case of memory to memory transfer direction. This parameter can be a value of ***DMA_LL_EC_PDATAALIGN***This feature can be modified afterwards using unitary function **LL_DMA_SetPeriphSize()**.
- ***uint32_t LL_DMA_InitTypeDef::MemoryOrM2MDstDataSize***

  Specifies the Memory data size alignment or Destination data size alignment (byte, half word, word) in case of memory to memory transfer direction. This parameter can be a value of ***DMA_LL_EC_MDATAALIGN***This feature can be modified afterwards using unitary function **LL_DMA_SetMemorySize()**.
- ***uint32_t LL_DMA_InitTypeDef::NbData***

  Specifies the number of data to transfer, in data unit. The data unit is equal to the source buffer configuration set in PeripheralSize or MemorySize parameters depending in the transfer direction. This parameter must be a value between Min_Data = 0 and Max_Data = 0x0000FFFFThis feature can be modified afterwards using unitary function **LL_DMA_SetDataLength()**.
- ***uint32_t LL_DMA_InitTypeDef::PeriphRequest***

  Specifies the peripheral request. This parameter can be a value of ***DMAMUX1_Request_selection***This feature can be modified afterwards using unitary function **LL_DMA_SetPeriphRequest()**.
- ***uint32_t LL_DMA_InitTypeDef::Priority***

  Specifies the channel priority level. This parameter can be a value of ***DMA_LL_EC_PRIORITY***This feature can be modified afterwards using unitary function **LL_DMA_SetStreamPriorityLevel()**.
- ***uint32_t LL_DMA_InitTypeDef::FIFOMode***

  Specifies if the FIFO mode or Direct mode will be used for the specified stream. This parameter can be a value of ***DMA_LL_FIFOMODE***

  Note:

  – The Direct mode (FIFO mode disabled) cannot be used if the memory-to-memory data transfer is configured on the selected stream

  This feature can be modified afterwards using unitary functions **LL_DMA_EnableFifoMode()** or **LL_DMA_EnableFifoMode()** .
- ***uint32_t LL_DMA_InitTypeDef::FIFOThreshold***

  Specifies the FIFO threshold level. This parameter can be a value of ***DMA_LL_EC_FIFOTHRESHOLD***This feature can be modified afterwards using unitary function **LL_DMA_SetFIFOThreshold()**.
- ***uint32_t LL_DMA_InitTypeDef::MemBurst***

  Specifies the Burst transfer configuration for the memory transfers. It specifies the amount of data to be transferred in a single non interruptible transaction. This parameter can be a value of ***DMA_LL_EC_MBURST***

  Note:

  – The burst mode is possible only if the address Increment mode is enabled.

  This feature can be modified afterwards using unitary function **LL_DMA_SetMemoryBurstxfer()**.
- ***uint32_t LL_DMA_InitTypeDef::PeriphBurst***

  Specifies the Burst transfer configuration for the peripheral transfers. It specifies the amount of data to be transferred in a single non interruptible transaction. This parameter can be a value of ***DMA_LL_EC_PBURST***

  Note:

  – The burst mode is possible only if the address Increment mode is enabled.

  This feature can be modified afterwards using unitary function **LL_DMA_SetPeriphBurstxfer()**.

107.2 DMA Firmware driver API description

The following section lists the various functions of the DMA library.

107.2.1 Detailed description of functions

LL_DMA_EnableStream

Function name

__STATIC_INLINE void LL_DMA_EnableStream (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Enable DMA stream.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR EN LL_DMA_EnableStream

LL_DMA_DisableStream

Function name

__STATIC_INLINE void LL_DMA_DisableStream (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Disable DMA stream.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR EN LL_DMA_DisableStream

LL_DMA_IsEnabledStream

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledStream (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Check if DMA stream is enabled or disabled.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR EN LL_DMA_IsEnabledStream

LL_DMA_ConfigTransfer

Function name

__STATIC_INLINE void LL_DMA_ConfigTransfer (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t Configuration)

Function description

Configure all parameters linked to DMA transfer.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH or LL_DMA_DIRECTION_MEMORY_TO_MEMORY
  - LL_DMA_MODE_NORMAL or LL_DMA_MODE_CIRCULAR or LL_DMA_MODE_PFCTRL
  - LL_DMA_PERIPH_INCREMENT or LL_DMA_PERIPH_NOINCREMENT
  - LL_DMA_MEMORY_INCREMENT or LL_DMA_MEMORY_NOINCREMENT
  - LL_DMA_PDATAALIGN_BYTE or LL_DMA_PDATAALIGN_HALFWORD or LL_DMA_PDATAALIGN_WORD
  - LL_DMA_MDATAALIGN_BYTE or LL_DMA_MDATAALIGN_HALFWORD or LL_DMA_MDATAALIGN_WORD
  - LL_DMA_PRIORITY_LOW or LL_DMA_PRIORITY_MEDIUM or LL_DMA_PRIORITY_HIGH or LL_DMA_PRIORITY_VERYHIGH

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DIR LL_DMA_ConfigTransfer
- CR CIRC LL_DMA_ConfigTransfer
- CR PINC LL_DMA_ConfigTransfer
- CR MINC LL_DMA_ConfigTransfer
- CR PSIZE LL_DMA_ConfigTransfer
- CR MSIZE LL_DMA_ConfigTransfer
- CR PL LL_DMA_ConfigTransfer
- CR PFCTRL LL_DMA_ConfigTransfer

LL_DMA_SetDataTransferDirection

Function name

__STATIC_INLINE void LL_DMA_SetDataTransferDirection (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t Direction)

Function description

Set Data transfer direction (read from peripheral or from memory).

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7
- **Direction:** This parameter can be one of the following values:
  - LL_DMA_DIRECTION_PERIPH_TO_MEMORY
  - LL_DMA_DIRECTION_MEMORY_TO_PERIPH
  - LL_DMA_DIRECTION_MEMORY_TO_MEMORY

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DIR LL_DMA_SetDataTransferDirection

LL_DMA_GetDataTransferDirection

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDataTransferDirection (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Get Data transfer direction (read from peripheral or from memory).

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_DIRECTION_PERIPH_TO_MEMORY
  - LL_DMA_DIRECTION_MEMORY_TO_PERIPH
  - LL_DMA_DIRECTION_MEMORY_TO_MEMORY

Reference Manual to LL API cross reference:

- CR DIR LL_DMA_GetDataTransferDirection

LL_DMA_SetMode

Function name

__STATIC_INLINE void LL_DMA_SetMode (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t Mode)

Function description

Set DMA mode normal, circular or peripheral flow control.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7
- **Mode:** This parameter can be one of the following values:
  - LL_DMA_MODE_NORMAL
  - LL_DMA_MODE_CIRCULAR
  - LL_DMA_MODE_PFCTRL

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CIRC LL_DMA_SetMode
- CR PFCTRL LL_DMA_SetMode

LL_DMA_GetMode

Function name

__STATIC_INLINE uint32_t LL_DMA_GetMode (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Get DMA mode normal, circular or peripheral flow control.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_MODE_NORMAL
  - LL_DMA_MODE_CIRCULAR
  - LL_DMA_MODE_PFCTRL

Reference Manual to LL API cross reference:

- CR CIRC LL_DMA_GetMode
- CR PFCTRL LL_DMA_GetMode

LL_DMA_SetPeriphIncMode

Function name

__STATIC_INLINE void LL_DMA_SetPeriphIncMode (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t IncrementMode)

Function description

Set Peripheral increment mode.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7
- **IncrementMode:** This parameter can be one of the following values:
  - LL_DMA_PERIPH_NOINCREMENT
  - LL_DMA_PERIPH_INCREMENT

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PINC LL_DMA_SetPeriphIncMode

LL_DMA_GetPeriphIncMode

Function name

__STATIC_INLINE uint32_t LL_DMA_GetPeriphIncMode (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Get Peripheral increment mode.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_PERIPH_NOINCREMENT
  - LL_DMA_PERIPH_INCREMENT

Reference Manual to LL API cross reference:

- CR PINC LL_DMA_GetPeriphIncMode

LL_DMA_SetMemoryIncMode

Function name

__STATIC_INLINE void LL_DMA_SetMemoryIncMode (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t IncrementMode)

Function description

Set Memory increment mode.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7
- **IncrementMode:** This parameter can be one of the following values:
  - LL_DMA_MEMORY_NOINCREMENT
  - LL_DMA_MEMORY_INCREMENT

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR MINC LL_DMA_SetMemoryIncMode

LL_DMA_GetMemoryIncMode

Function name

__STATIC_INLINE uint32_t LL_DMA_GetMemoryIncMode (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Get Memory increment mode.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_MEMORY_NOINCREMENT
  - LL_DMA_MEMORY_INCREMENT

Reference Manual to LL API cross reference:

- CR MINC LL_DMA_GetMemoryIncMode

LL_DMA_SetPeriphSize

Function name

__STATIC_INLINE void LL_DMA_SetPeriphSize (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t Size)

Function description

Set Peripheral size.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7
- **Size:** This parameter can be one of the following values:
  - LL_DMA_PDATAALIGN_BYTE
  - LL_DMA_PDATAALIGN_HALFWORD
  - LL_DMA_PDATAALIGN_WORD

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PSIZE LL_DMA_SetPeriphSize

LL_DMA_GetPeriphSize

Function name

__STATIC_INLINE uint32_t LL_DMA_GetPeriphSize (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Get Peripheral size.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_PDATAALIGN_BYTE
  - LL_DMA_PDATAALIGN_HALFWORD
  - LL_DMA_PDATAALIGN_WORD

Reference Manual to LL API cross reference:

- CR PSIZE LL_DMA_GetPeriphSize

LL_DMA_SetMemorySize

Function name

__STATIC_INLINE void LL_DMA_SetMemorySize (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t Size)

Function description

Set Memory size.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7
- **Size:** This parameter can be one of the following values:
  - LL_DMA_MDATAALIGN_BYTE
  - LL_DMA_MDATAALIGN_HALFWORD
  - LL_DMA_MDATAALIGN_WORD

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR MSIZE LL_DMA_SetMemorySize

LL_DMA_GetMemorySize

Function name

__STATIC_INLINE uint32_t LL_DMA_GetMemorySize (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Get Memory size.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_MDATAALIGN_BYTE
  - LL_DMA_MDATAALIGN_HALFWORD
  - LL_DMA_MDATAALIGN_WORD

Reference Manual to LL API cross reference:

- CR MSIZE LL_DMA_GetMemorySize

LL_DMA_SetIncOffsetSize

Function name

__STATIC_INLINE void LL_DMA_SetIncOffsetSize (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t OffsetSize)

Function description

Set Peripheral increment offset size.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7
- **OffsetSize:** This parameter can be one of the following values:
  - LL_DMA_OFFSETSIZE_PSIZE
  - LL_DMA_OFFSETSIZE_FIXEDTO4

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PINCOS LL_DMA_SetIncOffsetSize

LL_DMA_GetIncOffsetSize

Function name

__STATIC_INLINE uint32_t LL_DMA_GetIncOffsetSize (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Get Peripheral increment offset size.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_OFFSETSIZE_PSIZE
  - LL_DMA_OFFSETSIZE_FIXEDTO4

Reference Manual to LL API cross reference:

- CR PINCOS LL_DMA_GetIncOffsetSize

LL_DMA_SetStreamPriorityLevel

Function name

__STATIC_INLINE void LL_DMA_SetStreamPriorityLevel (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t Priority)

Function description

Set Stream priority level.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7
- **Priority:** This parameter can be one of the following values:
  - LL_DMA_PRIORITY_LOW
  - LL_DMA_PRIORITY_MEDIUM
  - LL_DMA_PRIORITY_HIGH
  - LL_DMA_PRIORITY_VERYHIGH

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PL LL_DMA_SetStreamPriorityLevel

LL_DMA_GetStreamPriorityLevel

Function name

__STATIC_INLINE uint32_t LL_DMA_GetStreamPriorityLevel (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Get Stream priority level.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_PRIORITY_LOW
  - LL_DMA_PRIORITY_MEDIUM
  - LL_DMA_PRIORITY_HIGH
  - LL_DMA_PRIORITY_VERYHIGH

Reference Manual to LL API cross reference:

- CR PL LL_DMA_GetStreamPriorityLevel

LL_DMA_EnableBufferableTransfer

Function name

__STATIC_INLINE void LL_DMA_EnableBufferableTransfer (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Enable DMA stream bufferable transfer.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TRBUFF LL_DMA_EnableBufferableTransfer

LL_DMA_DisableBufferableTransfer

Function name

__STATIC_INLINE void LL_DMA_DisableBufferableTransfer (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Disable DMA stream bufferable transfer.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TRBUFF LL_DMA_DisableBufferableTransfer

LL_DMA_SetDataLength

Function name

__STATIC_INLINE void LL_DMA_SetDataLength (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t NbData)

Function description

Set Number of data to transfer.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7
- **NbData:** Between 0 to 0xFFFFFFFF

Return values

- **None:**

Notes

- This action has no effect if stream is enabled.

Reference Manual to LL API cross reference:

- NDTR NDT LL_DMA_SetDataLength

LL_DMA_GetDataLength

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDataLength (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Get Number of data to transfer.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **Between:** 0 to 0xFFFFFFFF

Notes

- Once the stream is enabled, the return value indicate the remaining bytes to be transmitted.

Reference Manual to LL API cross reference:

- NDTR NDT LL_DMA_GetDataLength

LL_DMA_SetPeriphRequest

Function name

__STATIC_INLINE void LL_DMA_SetPeriphRequest (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t Request)

Function description

Set DMA request for DMA Streams on DMAMUX Channel x.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

- **Request:** This parameter can be one of the following values:
  - LL_DMAMUX1_REQ_MEM2MEM
  - LL_DMAMUX1_REQ_GENERATOR0
  - LL_DMAMUX1_REQ_GENERATOR1
  - LL_DMAMUX1_REQ_GENERATOR2
  - LL_DMAMUX1_REQ_GENERATOR3
  - LL_DMAMUX1_REQ_GENERATOR4
  - LL_DMAMUX1_REQ_GENERATOR5
  - LL_DMAMUX1_REQ_GENERATOR6
  - LL_DMAMUX1_REQ_GENERATOR7
  - LL_DMAMUX1_REQ_ADC1
  - LL_DMAMUX1_REQ_ADC2
  - LL_DMAMUX1_REQ_TIM1_CH1
  - LL_DMAMUX1_REQ_TIM1_CH2
  - LL_DMAMUX1_REQ_TIM1_CH3
  - LL_DMAMUX1_REQ_TIM1_CH4
  - LL_DMAMUX1_REQ_TIM1_UP
  - LL_DMAMUX1_REQ_TIM1_TRIG
  - LL_DMAMUX1_REQ_TIM1_COM
  - LL_DMAMUX1_REQ_TIM2_CH1
  - LL_DMAMUX1_REQ_TIM2_CH2
  - LL_DMAMUX1_REQ_TIM2_CH3
  - LL_DMAMUX1_REQ_TIM2_CH4
  - LL_DMAMUX1_REQ_TIM2_UP
  - LL_DMAMUX1_REQ_TIM3_CH1
  - LL_DMAMUX1_REQ_TIM3_CH2
  - LL_DMAMUX1_REQ_TIM3_CH3
  - LL_DMAMUX1_REQ_TIM3_CH4
  - LL_DMAMUX1_REQ_TIM3_UP
  - LL_DMAMUX1_REQ_TIM3_TRIG
  - LL_DMAMUX1_REQ_TIM4_CH1
  - LL_DMAMUX1_REQ_TIM4_CH2
  - LL_DMAMUX1_REQ_TIM4_CH3
  - LL_DMAMUX1_REQ_TIM4_UP
  - LL_DMAMUX1_REQ_I2C1_RX
  - LL_DMAMUX1_REQ_I2C1_TX
  - LL_DMAMUX1_REQ_I2C2_RX
  - LL_DMAMUX1_REQ_I2C2_TX
  - LL_DMAMUX1_REQ_SPI1_RX
  - LL_DMAMUX1_REQ_SPI1_TX
  - LL_DMAMUX1_REQ_SPI2_RX
  - LL_DMAMUX1_REQ_SPI2_TX
  - LL_DMAMUX1_REQ_USART1_RX
  - LL_DMAMUX1_REQ_USART1_TX
  - LL_DMAMUX1_REQ_USART2_RX
  - LL_DMAMUX1_REQ_USART2_TX
  - LL_DMAMUX1_REQ_USART3_RX
  - LL_DMAMUX1_REQ_USART3_TX
  - LL_DMAMUX1_REQ_TIM8_CH1
  - LL_DMAMUX1_REQ_TIM8_CH2
  - LL_DMAMUX1_REQ_TIM8_CH3
  - LL_DMAMUX1_REQ_TIM8_CH4
  - LL_DMAMUX1_REQ_TIM8_UP

Return values

- **None:**

Notes

- DMAMUX channel 0 to 7 are mapped to DMA1 stream 0 to 7. DMAMUX channel 8 to 15 are mapped to DMA2 stream 0 to 7.
- (*) Availability depends on devices.

Reference Manual to LL API cross reference:

- CxCR DMAREQ_ID LL_DMA_SetPeriphRequest

LL_DMA_GetPeriphRequest

Function name

__STATIC_INLINE uint32_t LL_DMA_GetPeriphRequest (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Get DMA request for DMA Channels on DMAMUX Channel x.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMAMUX1_REQ_MEM2MEM
  - LL_DMAMUX1_REQ_GENERATOR0
  - LL_DMAMUX1_REQ_GENERATOR1
  - LL_DMAMUX1_REQ_GENERATOR2
  - LL_DMAMUX1_REQ_GENERATOR3
  - LL_DMAMUX1_REQ_GENERATOR4
  - LL_DMAMUX1_REQ_GENERATOR5
  - LL_DMAMUX1_REQ_GENERATOR6
  - LL_DMAMUX1_REQ_GENERATOR7
  - LL_DMAMUX1_REQ_ADC1
  - LL_DMAMUX1_REQ_ADC2
  - LL_DMAMUX1_REQ_TIM1_CH1
  - LL_DMAMUX1_REQ_TIM1_CH2
  - LL_DMAMUX1_REQ_TIM1_CH3
  - LL_DMAMUX1_REQ_TIM1_CH4
  - LL_DMAMUX1_REQ_TIM1_UP
  - LL_DMAMUX1_REQ_TIM1_TRIG
  - LL_DMAMUX1_REQ_TIM1_COM
  - LL_DMAMUX1_REQ_TIM2_CH1
  - LL_DMAMUX1_REQ_TIM2_CH2
  - LL_DMAMUX1_REQ_TIM2_CH3
  - LL_DMAMUX1_REQ_TIM2_CH4
  - LL_DMAMUX1_REQ_TIM2_UP
  - LL_DMAMUX1_REQ_TIM3_CH1
  - LL_DMAMUX1_REQ_TIM3_CH2
  - LL_DMAMUX1_REQ_TIM3_CH3
  - LL_DMAMUX1_REQ_TIM3_CH4
  - LL_DMAMUX1_REQ_TIM3_UP
  - LL_DMAMUX1_REQ_TIM3_TRIG
  - LL_DMAMUX1_REQ_TIM4_CH1
  - LL_DMAMUX1_REQ_TIM4_CH2
  - LL_DMAMUX1_REQ_TIM4_CH3
  - LL_DMAMUX1_REQ_TIM4_UP
  - LL_DMAMUX1_REQ_I2C1_RX
  - LL_DMAMUX1_REQ_I2C1_TX
  - LL_DMAMUX1_REQ_I2C2_RX
  - LL_DMAMUX1_REQ_I2C2_TX
  - LL_DMAMUX1_REQ_SPI1_RX
  - LL_DMAMUX1_REQ_SPI1_TX
  - LL_DMAMUX1_REQ_SPI2_RX
  - LL_DMAMUX1_REQ_SPI2_TX
  - LL_DMAMUX1_REQ_USART1_RX
  - LL_DMAMUX1_REQ_USART1_TX
  - LL_DMAMUX1_REQ_USART2_RX
  - LL_DMAMUX1_REQ_USART2_TX
  - LL_DMAMUX1_REQ_USART3_RX
  - LL_DMAMUX1_REQ_USART3_TX
  - LL_DMAMUX1_REQ_TIM8_CH1
  - LL_DMAMUX1_REQ_TIM8_CH2
  - LL_DMAMUX1_REQ_TIM8_CH3

Notes

- DMAMUX channel 0 to 7 are mapped to DMA1 stream 0 to 7. DMAMUX channel 8 to 15 are mapped to DMA2 stream 0 to 7.
- (*) Availability depends on devices.

Reference Manual to LL API cross reference:

- CxCR DMAREQ_ID LL_DMA_GetPeriphRequest

LL_DMA_SetMemoryBurstxfer

Function name

__STATIC_INLINE void LL_DMA_SetMemoryBurstxfer (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t Mburst)

Function description

Set Memory burst transfer configuration.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7
- **Mburst:** This parameter can be one of the following values:
  - LL_DMA_MBURST_SINGLE
  - LL_DMA_MBURST_INC4
  - LL_DMA_MBURST_INC8
  - LL_DMA_MBURST_INC16

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR MBURST LL_DMA_SetMemoryBurstxfer

LL_DMA_GetMemoryBurstxfer

Function name

__STATIC_INLINE uint32_t LL_DMA_GetMemoryBurstxfer (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Get Memory burst transfer configuration.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_MBURST_SINGLE
  - LL_DMA_MBURST_INC4
  - LL_DMA_MBURST_INC8
  - LL_DMA_MBURST_INC16

Reference Manual to LL API cross reference:

- CR MBURST LL_DMA_GetMemoryBurstxfer

LL_DMA_SetPeriphBurstxfer

Function name

__STATIC_INLINE void LL_DMA_SetPeriphBurstxfer (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t Pburst)

Function description

Set Peripheral burst transfer configuration.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7
- **Pburst:** This parameter can be one of the following values:
  - LL_DMA_PBURST_SINGLE
  - LL_DMA_PBURST_INC4
  - LL_DMA_PBURST_INC8
  - LL_DMA_PBURST_INC16

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PBURST LL_DMA_SetPeriphBurstxfer

LL_DMA_GetPeriphBurstxfer

Function name

__STATIC_INLINE uint32_t LL_DMA_GetPeriphBurstxfer (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Get Peripheral burst transfer configuration.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_PBURST_SINGLE
  - LL_DMA_PBURST_INC4
  - LL_DMA_PBURST_INC8
  - LL_DMA_PBURST_INC16

Reference Manual to LL API cross reference:

- CR PBURST LL_DMA_GetPeriphBurstxfer

LL_DMA_SetCurrentTargetMem

Function name

__STATIC_INLINE void LL_DMA_SetCurrentTargetMem (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t CurrentMemory)

Function description

Set Current target (only in double buffer mode) to Memory 1 or Memory 0.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7
- **CurrentMemory:** This parameter can be one of the following values:
  - LL_DMA_CURRENTTARGETMEM0
  - LL_DMA_CURRENTTARGETMEM1

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CT LL_DMA_SetCurrentTargetMem

LL_DMA_GetCurrentTargetMem

Function name

__STATIC_INLINE uint32_t LL_DMA_GetCurrentTargetMem (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Set Current target (only in double buffer mode) to Memory 1 or Memory 0.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_CURRENTTARGETMEM0
  - LL_DMA_CURRENTTARGETMEM1

Reference Manual to LL API cross reference:

- CR CT LL_DMA_GetCurrentTargetMem

LL_DMA_EnableDoubleBufferMode

Function name

__STATIC_INLINE void LL_DMA_EnableDoubleBufferMode (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Enable the double buffer mode.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DBM LL_DMA_EnableDoubleBufferMode

LL_DMA_DisableDoubleBufferMode

Function name

__STATIC_INLINE void LL_DMA_DisableDoubleBufferMode (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Disable the double buffer mode.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DBM LL_DMA_DisableDoubleBufferMode

LL_DMA_GetFIFOStatus

Function name

__STATIC_INLINE uint32_t LL_DMA_GetFIFOStatus (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Get FIFO status.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_FIFOSTATUS_0_25
  - LL_DMA_FIFOSTATUS_25_50
  - LL_DMA_FIFOSTATUS_50_75
  - LL_DMA_FIFOSTATUS_75_100
  - LL_DMA_FIFOSTATUS_EMPTY
  - LL_DMA_FIFOSTATUS_FULL

Reference Manual to LL API cross reference:

- FCR FS LL_DMA_GetFIFOStatus

LL_DMA_DisableFifoMode

Function name

__STATIC_INLINE void LL_DMA_DisableFifoMode (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Disable Fifo mode.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- FCR DMDIS LL_DMA_DisableFifoMode

LL_DMA_EnableFifoMode

Function name

__STATIC_INLINE void LL_DMA_EnableFifoMode (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Enable Fifo mode.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- FCR DMDIS LL_DMA_EnableFifoMode

LL_DMA_SetFIFOThreshold

Function name

__STATIC_INLINE void LL_DMA_SetFIFOThreshold (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t Threshold)

Function description

Select FIFO threshold.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7
- **Threshold:** This parameter can be one of the following values:
  - LL_DMA_FIFOTHRESHOLD_1_4
  - LL_DMA_FIFOTHRESHOLD_1_2
  - LL_DMA_FIFOTHRESHOLD_3_4
  - LL_DMA_FIFOTHRESHOLD_FULL

Return values

- **None:**

Reference Manual to LL API cross reference:

- FCR FTH LL_DMA_SetFIFOThreshold

LL_DMA_GetFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_DMA_GetFIFOThreshold (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Get FIFO threshold.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_FIFOTHRESHOLD_1_4
  - LL_DMA_FIFOTHRESHOLD_1_2
  - LL_DMA_FIFOTHRESHOLD_3_4
  - LL_DMA_FIFOTHRESHOLD_FULL

Reference Manual to LL API cross reference:

- FCR FTH LL_DMA_GetFIFOThreshold

LL_DMA_ConfigFifo

Function name

__STATIC_INLINE void LL_DMA_ConfigFifo (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t FifoMode, uint32_t FifoThreshold)

Function description

Configure the FIFO .

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7
- **FifoMode:** This parameter can be one of the following values:
  - LL_DMA_FIFOMODE_ENABLE
  - LL_DMA_FIFOMODE_DISABLE
- **FifoThreshold:** This parameter can be one of the following values:
  - LL_DMA_FIFOTHRESHOLD_1_4
  - LL_DMA_FIFOTHRESHOLD_1_2
  - LL_DMA_FIFOTHRESHOLD_3_4
  - LL_DMA_FIFOTHRESHOLD_FULL

Return values

- **None:**

Reference Manual to LL API cross reference:

- FCR FTH LL_DMA_ConfigFifo
- FCR DMDIS LL_DMA_ConfigFifo

LL_DMA_ConfigAddresses

Function name

__STATIC_INLINE void LL_DMA_ConfigAddresses (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t SrcAddress, uint32_t DstAddress, uint32_t Direction)

Function description

Configure the Source and Destination addresses.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7
- **SrcAddress:** Between 0 to 0xFFFFFFFF
- **DstAddress:** Between 0 to 0xFFFFFFFF
- **Direction:** This parameter can be one of the following values:
  - LL_DMA_DIRECTION_PERIPH_TO_MEMORY
  - LL_DMA_DIRECTION_MEMORY_TO_PERIPH
  - LL_DMA_DIRECTION_MEMORY_TO_MEMORY

Return values

- **None:**

Notes

- This API must not be called when the DMA stream is enabled.

Reference Manual to LL API cross reference:

- M0AR M0A LL_DMA_ConfigAddresses
- PAR PA LL_DMA_ConfigAddresses

LL_DMA_SetMemoryAddress

Function name

__STATIC_INLINE void LL_DMA_SetMemoryAddress (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t MemoryAddress)

Function description

Set the Memory address.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7
- **MemoryAddress:** Between 0 to 0xFFFFFFFF

Return values

- **None:**

Notes

- Interface used for direction LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH only.
- This API must not be called when the DMA stream is enabled.

Reference Manual to LL API cross reference:

- M0AR M0A LL_DMA_SetMemoryAddress

LL_DMA_SetPeriphAddress

Function name

__STATIC_INLINE void LL_DMA_SetPeriphAddress (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t PeriphAddress)

Function description

Set the Peripheral address.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7
- **PeriphAddress:** Between 0 to 0xFFFFFFFF

Return values

- **None:**

Notes

- Interface used for direction LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH only.
- This API must not be called when the DMA stream is enabled.

Reference Manual to LL API cross reference:

- PAR PA LL_DMA_SetPeriphAddress

LL_DMA_GetMemoryAddress

Function name

__STATIC_INLINE uint32_t LL_DMA_GetMemoryAddress (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Get the Memory address.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **Between:** 0 to 0xFFFFFFFF

Notes

- Interface used for direction LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH only.

Reference Manual to LL API cross reference:

- M0AR M0A LL_DMA_GetMemoryAddress

LL_DMA_GetPeriphAddress

Function name

__STATIC_INLINE uint32_t LL_DMA_GetPeriphAddress (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Get the Peripheral address.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **Between:** 0 to 0xFFFFFFFF

Notes

- Interface used for direction LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH only.

Reference Manual to LL API cross reference:

- PAR PA LL_DMA_GetPeriphAddress

LL_DMA_SetM2MSrcAddress

Function name

__STATIC_INLINE void LL_DMA_SetM2MSrcAddress (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t MemoryAddress)

Function description

Set the Memory to Memory Source address.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7
- **MemoryAddress:** Between 0 to 0xFFFFFFFF

Return values

- **None:**

Notes

- Interface used for direction LL_DMA_DIRECTION_MEMORY_TO_MEMORY only.
- This API must not be called when the DMA stream is enabled.

Reference Manual to LL API cross reference:

- PAR PA LL_DMA_SetM2MSrcAddress

LL_DMA_SetM2MDstAddress

Function name

__STATIC_INLINE void LL_DMA_SetM2MDstAddress (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t MemoryAddress)

Function description

Set the Memory to Memory Destination address.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7
- **MemoryAddress:** Between 0 to 0xFFFFFFFF

Return values

- **None:**

Notes

- Interface used for direction LL_DMA_DIRECTION_MEMORY_TO_MEMORY only.
- This API must not be called when the DMA stream is enabled.

Reference Manual to LL API cross reference:

- M0AR M0A LL_DMA_SetM2MDstAddress

LL_DMA_GetM2MSrcAddress

Function name

__STATIC_INLINE uint32_t LL_DMA_GetM2MSrcAddress (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Get the Memory to Memory Source address.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **Between:** 0 to 0xFFFFFFFF

Notes

- Interface used for direction LL_DMA_DIRECTION_MEMORY_TO_MEMORY only.

Reference Manual to LL API cross reference:

- PAR PA LL_DMA_GetM2MSrcAddress

LL_DMA_GetM2MDstAddress

Function name

__STATIC_INLINE uint32_t LL_DMA_GetM2MDstAddress (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Get the Memory to Memory Destination address.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **Between:** 0 to 0xFFFFFFFF

Notes

- Interface used for direction LL_DMA_DIRECTION_MEMORY_TO_MEMORY only.

Reference Manual to LL API cross reference:

- M0AR M0A LL_DMA_GetM2MDstAddress

LL_DMA_SetMemory1Address

Function name

__STATIC_INLINE void LL_DMA_SetMemory1Address (DMA_TypeDef * DMAx, uint32_t Stream, uint32_t Address)

Function description

Set Memory 1 address (used in case of Double buffer mode).

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7
- **Address:** Between 0 to 0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- M1AR M1A LL_DMA_SetMemory1Address

LL_DMA_GetMemory1Address

Function name

__STATIC_INLINE uint32_t LL_DMA_GetMemory1Address (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Get Memory 1 address (used in case of Double buffer mode).

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **Between:** 0 to 0xFFFFFFFF

Reference Manual to LL API cross reference:

- M1AR M1A LL_DMA_GetMemory1Address

LL_DMA_IsActiveFlag_HT0

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT0 (DMA_TypeDef * DMAx)

Function description

Get Stream 0 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LISR HTIF0 LL_DMA_IsActiveFlag_HT0

LL_DMA_IsActiveFlag_HT1

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT1 (DMA_TypeDef * DMAx)

Function description

Get Stream 1 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LISR HTIF1 LL_DMA_IsActiveFlag_HT1

LL_DMA_IsActiveFlag_HT2

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT2 (DMA_TypeDef * DMAx)

Function description

Get Stream 2 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LISR HTIF2 LL_DMA_IsActiveFlag_HT2

LL_DMA_IsActiveFlag_HT3

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT3 (DMA_TypeDef * DMAx)

Function description

Get Stream 3 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LISR HTIF3 LL_DMA_IsActiveFlag_HT3

LL_DMA_IsActiveFlag_HT4

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT4 (DMA_TypeDef * DMAx)

Function description

Get Stream 4 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- HISR HTIF4 LL_DMA_IsActiveFlag_HT4

LL_DMA_IsActiveFlag_HT5

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT5 (DMA_TypeDef * DMAx)

Function description

Get Stream 5 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- HISR HTIF0 LL_DMA_IsActiveFlag_HT5

LL_DMA_IsActiveFlag_HT6

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT6 (DMA_TypeDef * DMAx)

Function description

Get Stream 6 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- HISR HTIF6 LL_DMA_IsActiveFlag_HT6

LL_DMA_IsActiveFlag_HT7

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT7 (DMA_TypeDef * DMAx)

Function description

Get Stream 7 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- HISR HTIF7 LL_DMA_IsActiveFlag_HT7

LL_DMA_IsActiveFlag_TC0

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC0 (DMA_TypeDef * DMAx)

Function description

Get Stream 0 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LISR TCIF0 LL_DMA_IsActiveFlag_TC0

LL_DMA_IsActiveFlag_TC1

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC1 (DMA_TypeDef * DMAx)

Function description

Get Stream 1 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LISR TCIF1 LL_DMA_IsActiveFlag_TC1

LL_DMA_IsActiveFlag_TC2

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC2 (DMA_TypeDef * DMAx)

Function description

Get Stream 2 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LISR TCIF2 LL_DMA_IsActiveFlag_TC2

LL_DMA_IsActiveFlag_TC3

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC3 (DMA_TypeDef * DMAx)

Function description

Get Stream 3 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LISR TCIF3 LL_DMA_IsActiveFlag_TC3

LL_DMA_IsActiveFlag_TC4

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC4 (DMA_TypeDef * DMAx)

Function description

Get Stream 4 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- HISR TCIF4 LL_DMA_IsActiveFlag_TC4

LL_DMA_IsActiveFlag_TC5

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC5 (DMA_TypeDef * DMAx)

Function description

Get Stream 5 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- HISR TCIF0 LL_DMA_IsActiveFlag_TC5

LL_DMA_IsActiveFlag_TC6

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC6 (DMA_TypeDef * DMAx)

Function description

Get Stream 6 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- HISR TCIF6 LL_DMA_IsActiveFlag_TC6

LL_DMA_IsActiveFlag_TC7

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC7 (DMA_TypeDef * DMAx)

Function description

Get Stream 7 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- HISR TCIF7 LL_DMA_IsActiveFlag_TC7

LL_DMA_IsActiveFlag_TE0

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE0 (DMA_TypeDef * DMAx)

Function description

Get Stream 0 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LISR TEIF0 LL_DMA_IsActiveFlag_TE0

LL_DMA_IsActiveFlag_TE1

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE1 (DMA_TypeDef * DMAx)

Function description

Get Stream 1 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LISR TEIF1 LL_DMA_IsActiveFlag_TE1

LL_DMA_IsActiveFlag_TE2

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE2 (DMA_TypeDef * DMAx)

Function description

Get Stream 2 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LISR TEIF2 LL_DMA_IsActiveFlag_TE2

LL_DMA_IsActiveFlag_TE3

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE3 (DMA_TypeDef * DMAx)

Function description

Get Stream 3 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LISR TEIF3 LL_DMA_IsActiveFlag_TE3

LL_DMA_IsActiveFlag_TE4

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE4 (DMA_TypeDef * DMAx)

Function description

Get Stream 4 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- HISR TEIF4 LL_DMA_IsActiveFlag_TE4

LL_DMA_IsActiveFlag_TE5

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE5 (DMA_TypeDef * DMAx)

Function description

Get Stream 5 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- HISR TEIF0 LL_DMA_IsActiveFlag_TE5

LL_DMA_IsActiveFlag_TE6

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE6 (DMA_TypeDef * DMAx)

Function description

Get Stream 6 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- HISR TEIF6 LL_DMA_IsActiveFlag_TE6

LL_DMA_IsActiveFlag_TE7

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE7 (DMA_TypeDef * DMAx)

Function description

Get Stream 7 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- HISR TEIF7 LL_DMA_IsActiveFlag_TE7

LL_DMA_IsActiveFlag_DME0

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_DME0 (DMA_TypeDef * DMAx)

Function description

Get Stream 0 direct mode error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LISR DMEIF0 LL_DMA_IsActiveFlag_DME0

LL_DMA_IsActiveFlag_DME1

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_DME1 (DMA_TypeDef * DMAx)

Function description

Get Stream 1 direct mode error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LISR DMEIF1 LL_DMA_IsActiveFlag_DME1

LL_DMA_IsActiveFlag_DME2

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_DME2 (DMA_TypeDef * DMAx)

Function description

Get Stream 2 direct mode error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LISR DMEIF2 LL_DMA_IsActiveFlag_DME2

LL_DMA_IsActiveFlag_DME3

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_DME3 (DMA_TypeDef * DMAx)

Function description

Get Stream 3 direct mode error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LISR DMEIF3 LL_DMA_IsActiveFlag_DME3

LL_DMA_IsActiveFlag_DME4

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_DME4 (DMA_TypeDef * DMAx)

Function description

Get Stream 4 direct mode error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- HISR DMEIF4 LL_DMA_IsActiveFlag_DME4

LL_DMA_IsActiveFlag_DME5

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_DME5 (DMA_TypeDef * DMAx)

Function description

Get Stream 5 direct mode error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- HISR DMEIF0 LL_DMA_IsActiveFlag_DME5

LL_DMA_IsActiveFlag_DME6

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_DME6 (DMA_TypeDef * DMAx)

Function description

Get Stream 6 direct mode error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- HISR DMEIF6 LL_DMA_IsActiveFlag_DME6

LL_DMA_IsActiveFlag_DME7

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_DME7 (DMA_TypeDef * DMAx)

Function description

Get Stream 7 direct mode error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- HISR DMEIF7 LL_DMA_IsActiveFlag_DME7

LL_DMA_IsActiveFlag_FE0

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_FE0 (DMA_TypeDef * DMAx)

Function description

Get Stream 0 FIFO error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LISR FEIF0 LL_DMA_IsActiveFlag_FE0

LL_DMA_IsActiveFlag_FE1

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_FE1 (DMA_TypeDef * DMAx)

Function description

Get Stream 1 FIFO error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LISR FEIF1 LL_DMA_IsActiveFlag_FE1

LL_DMA_IsActiveFlag_FE2

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_FE2 (DMA_TypeDef * DMAx)

Function description

Get Stream 2 FIFO error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LISR FEIF2 LL_DMA_IsActiveFlag_FE2

LL_DMA_IsActiveFlag_FE3

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_FE3 (DMA_TypeDef * DMAx)

Function description

Get Stream 3 FIFO error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LISR FEIF3 LL_DMA_IsActiveFlag_FE3

LL_DMA_IsActiveFlag_FE4

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_FE4 (DMA_TypeDef * DMAx)

Function description

Get Stream 4 FIFO error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- HISR FEIF4 LL_DMA_IsActiveFlag_FE4

LL_DMA_IsActiveFlag_FE5

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_FE5 (DMA_TypeDef * DMAx)

Function description

Get Stream 5 FIFO error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- HISR FEIF0 LL_DMA_IsActiveFlag_FE5

LL_DMA_IsActiveFlag_FE6

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_FE6 (DMA_TypeDef * DMAx)

Function description

Get Stream 6 FIFO error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- HISR FEIF6 LL_DMA_IsActiveFlag_FE6

LL_DMA_IsActiveFlag_FE7

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_FE7 (DMA_TypeDef * DMAx)

Function description

Get Stream 7 FIFO error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- HISR FEIF7 LL_DMA_IsActiveFlag_FE7

LL_DMA_ClearFlag_HT0

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_HT0 (DMA_TypeDef * DMAx)

Function description

Clear Stream 0 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- LIFCR CHTIF0 LL_DMA_ClearFlag_HT0

LL_DMA_ClearFlag_HT1

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_HT1 (DMA_TypeDef * DMAx)

Function description

Clear Stream 1 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- LIFCR CHTIF1 LL_DMA_ClearFlag_HT1

LL_DMA_ClearFlag_HT2

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_HT2 (DMA_TypeDef * DMAx)

Function description

Clear Stream 2 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- LIFCR CHTIF2 LL_DMA_ClearFlag_HT2

LL_DMA_ClearFlag_HT3

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_HT3 (DMA_TypeDef * DMAx)

Function description

Clear Stream 3 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- LIFCR CHTIF3 LL_DMA_ClearFlag_HT3

LL_DMA_ClearFlag_HT4

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_HT4 (DMA_TypeDef * DMAx)

Function description

Clear Stream 4 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- HIFCR CHTIF4 LL_DMA_ClearFlag_HT4

LL_DMA_ClearFlag_HT5

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_HT5 (DMA_TypeDef * DMAx)

Function description

Clear Stream 5 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- HIFCR CHTIF5 LL_DMA_ClearFlag_HT5

LL_DMA_ClearFlag_HT6

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_HT6 (DMA_TypeDef * DMAx)

Function description

Clear Stream 6 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- HIFCR CHTIF6 LL_DMA_ClearFlag_HT6

LL_DMA_ClearFlag_HT7

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_HT7 (DMA_TypeDef * DMAx)

Function description

Clear Stream 7 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- HIFCR CHTIF7 LL_DMA_ClearFlag_HT7

LL_DMA_ClearFlag_TC0

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TC0 (DMA_TypeDef * DMAx)

Function description

Clear Stream 0 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- LIFCR CTCIF0 LL_DMA_ClearFlag_TC0

LL_DMA_ClearFlag_TC1

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TC1 (DMA_TypeDef * DMAx)

Function description

Clear Stream 1 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- LIFCR CTCIF1 LL_DMA_ClearFlag_TC1

LL_DMA_ClearFlag_TC2

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TC2 (DMA_TypeDef * DMAx)

Function description

Clear Stream 2 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- LIFCR CTCIF2 LL_DMA_ClearFlag_TC2

LL_DMA_ClearFlag_TC3

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TC3 (DMA_TypeDef * DMAx)

Function description

Clear Stream 3 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- LIFCR CTCIF3 LL_DMA_ClearFlag_TC3

LL_DMA_ClearFlag_TC4

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TC4 (DMA_TypeDef * DMAx)

Function description

Clear Stream 4 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- HIFCR CTCIF4 LL_DMA_ClearFlag_TC4

LL_DMA_ClearFlag_TC5

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TC5 (DMA_TypeDef * DMAx)

Function description

Clear Stream 5 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- HIFCR CTCIF5 LL_DMA_ClearFlag_TC5

LL_DMA_ClearFlag_TC6

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TC6 (DMA_TypeDef * DMAx)

Function description

Clear Stream 6 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- HIFCR CTCIF6 LL_DMA_ClearFlag_TC6

LL_DMA_ClearFlag_TC7

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TC7 (DMA_TypeDef * DMAx)

Function description

Clear Stream 7 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- HIFCR CTCIF7 LL_DMA_ClearFlag_TC7

LL_DMA_ClearFlag_TE0

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TE0 (DMA_TypeDef * DMAx)

Function description

Clear Stream 0 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- LIFCR CTEIF0 LL_DMA_ClearFlag_TE0

LL_DMA_ClearFlag_TE1

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TE1 (DMA_TypeDef * DMAx)

Function description

Clear Stream 1 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- LIFCR CTEIF1 LL_DMA_ClearFlag_TE1

LL_DMA_ClearFlag_TE2

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TE2 (DMA_TypeDef * DMAx)

Function description

Clear Stream 2 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- LIFCR CTEIF2 LL_DMA_ClearFlag_TE2

LL_DMA_ClearFlag_TE3

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TE3 (DMA_TypeDef * DMAx)

Function description

Clear Stream 3 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- LIFCR CTEIF3 LL_DMA_ClearFlag_TE3

LL_DMA_ClearFlag_TE4

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TE4 (DMA_TypeDef * DMAx)

Function description

Clear Stream 4 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- HIFCR CTEIF4 LL_DMA_ClearFlag_TE4

LL_DMA_ClearFlag_TE5

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TE5 (DMA_TypeDef * DMAx)

Function description

Clear Stream 5 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- HIFCR CTEIF5 LL_DMA_ClearFlag_TE5

LL_DMA_ClearFlag_TE6

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TE6 (DMA_TypeDef * DMAx)

Function description

Clear Stream 6 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- HIFCR CTEIF6 LL_DMA_ClearFlag_TE6

LL_DMA_ClearFlag_TE7

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TE7 (DMA_TypeDef * DMAx)

Function description

Clear Stream 7 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- HIFCR CTEIF7 LL_DMA_ClearFlag_TE7

LL_DMA_ClearFlag_DME0

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_DME0 (DMA_TypeDef * DMAx)

Function description

Clear Stream 0 direct mode error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- LIFCR CDMEIF0 LL_DMA_ClearFlag_DME0

LL_DMA_ClearFlag_DME1

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_DME1 (DMA_TypeDef * DMAx)

Function description

Clear Stream 1 direct mode error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- LIFCR CDMEIF1 LL_DMA_ClearFlag_DME1

LL_DMA_ClearFlag_DME2

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_DME2 (DMA_TypeDef * DMAx)

Function description

Clear Stream 2 direct mode error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- LIFCR CDMEIF2 LL_DMA_ClearFlag_DME2

LL_DMA_ClearFlag_DME3

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_DME3 (DMA_TypeDef * DMAx)

Function description

Clear Stream 3 direct mode error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- LIFCR CDMEIF3 LL_DMA_ClearFlag_DME3

LL_DMA_ClearFlag_DME4

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_DME4 (DMA_TypeDef * DMAx)

Function description

Clear Stream 4 direct mode error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- HIFCR CDMEIF4 LL_DMA_ClearFlag_DME4

LL_DMA_ClearFlag_DME5

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_DME5 (DMA_TypeDef * DMAx)

Function description

Clear Stream 5 direct mode error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- HIFCR CDMEIF5 LL_DMA_ClearFlag_DME5

LL_DMA_ClearFlag_DME6

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_DME6 (DMA_TypeDef * DMAx)

Function description

Clear Stream 6 direct mode error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- HIFCR CDMEIF6 LL_DMA_ClearFlag_DME6

LL_DMA_ClearFlag_DME7

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_DME7 (DMA_TypeDef * DMAx)

Function description

Clear Stream 7 direct mode error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- HIFCR CDMEIF7 LL_DMA_ClearFlag_DME7

LL_DMA_ClearFlag_FE0

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_FE0 (DMA_TypeDef * DMAx)

Function description

Clear Stream 0 FIFO error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- LIFCR CFEIF0 LL_DMA_ClearFlag_FE0

LL_DMA_ClearFlag_FE1

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_FE1 (DMA_TypeDef * DMAx)

Function description

Clear Stream 1 FIFO error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- LIFCR CFEIF1 LL_DMA_ClearFlag_FE1

LL_DMA_ClearFlag_FE2

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_FE2 (DMA_TypeDef * DMAx)

Function description

Clear Stream 2 FIFO error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- LIFCR CFEIF2 LL_DMA_ClearFlag_FE2

LL_DMA_ClearFlag_FE3

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_FE3 (DMA_TypeDef * DMAx)

Function description

Clear Stream 3 FIFO error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- LIFCR CFEIF3 LL_DMA_ClearFlag_FE3

LL_DMA_ClearFlag_FE4

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_FE4 (DMA_TypeDef * DMAx)

Function description

Clear Stream 4 FIFO error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- HIFCR CFEIF4 LL_DMA_ClearFlag_FE4

LL_DMA_ClearFlag_FE5

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_FE5 (DMA_TypeDef * DMAx)

Function description

Clear Stream 5 FIFO error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- HIFCR CFEIF5 LL_DMA_ClearFlag_FE5

LL_DMA_ClearFlag_FE6

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_FE6 (DMA_TypeDef * DMAx)

Function description

Clear Stream 6 FIFO error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- HIFCR CFEIF6 LL_DMA_ClearFlag_FE6

LL_DMA_ClearFlag_FE7

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_FE7 (DMA_TypeDef * DMAx)

Function description

Clear Stream 7 FIFO error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- HIFCR CFEIF7 LL_DMA_ClearFlag_FE7

LL_DMA_EnableIT_HT

Function name

__STATIC_INLINE void LL_DMA_EnableIT_HT (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Enable Half transfer interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HTIE LL_DMA_EnableIT_HT

LL_DMA_EnableIT_TE

Function name

__STATIC_INLINE void LL_DMA_EnableIT_TE (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Enable Transfer error interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TEIE LL_DMA_EnableIT_TE

LL_DMA_EnableIT_TC

Function name

__STATIC_INLINE void LL_DMA_EnableIT_TC (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Enable Transfer complete interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TCIE LL_DMA_EnableIT_TC

LL_DMA_EnableIT_DME

Function name

__STATIC_INLINE void LL_DMA_EnableIT_DME (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Enable Direct mode error interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMEIE LL_DMA_EnableIT_DME

LL_DMA_EnableIT_FE

Function name

__STATIC_INLINE void LL_DMA_EnableIT_FE (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Enable FIFO error interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- FCR FEIE LL_DMA_EnableIT_FE

LL_DMA_DisableIT_HT

Function name

__STATIC_INLINE void LL_DMA_DisableIT_HT (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Disable Half transfer interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HTIE LL_DMA_DisableIT_HT

LL_DMA_DisableIT_TE

Function name

__STATIC_INLINE void LL_DMA_DisableIT_TE (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Disable Transfer error interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TEIE LL_DMA_DisableIT_TE

LL_DMA_DisableIT_TC

Function name

__STATIC_INLINE void LL_DMA_DisableIT_TC (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Disable Transfer complete interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TCIE LL_DMA_DisableIT_TC

LL_DMA_DisableIT_DME

Function name

__STATIC_INLINE void LL_DMA_DisableIT_DME (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Disable Direct mode error interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMEIE LL_DMA_DisableIT_DME

LL_DMA_DisableIT_FE

Function name

__STATIC_INLINE void LL_DMA_DisableIT_FE (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Disable FIFO error interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- FCR FEIE LL_DMA_DisableIT_FE

LL_DMA_IsEnabledIT_HT

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_HT (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Check if Half transfer interrupt is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR HTIE LL_DMA_IsEnabledIT_HT

LL_DMA_IsEnabledIT_TE

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_TE (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Check if Transfer error nterrup is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR TEIE LL_DMA_IsEnabledIT_TE

LL_DMA_IsEnabledIT_TC

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_TC (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Check if Transfer complete interrupt is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR TCIE LL_DMA_IsEnabledIT_TC

LL_DMA_IsEnabledIT_DME

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_DME (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Check if Direct mode error interrupt is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR DMEIE LL_DMA_IsEnabledIT_DME

LL_DMA_IsEnabledIT_FE

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_FE (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

Check if FIFO error interrupt is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- FCR FEIE LL_DMA_IsEnabledIT_FE

LL_DMA_Init

Function name

uint32_t LL_DMA_Init (DMA_TypeDef * DMAx, uint32_t Stream, LL_DMA_InitTypeDef * DMA_InitStruct)

Function description

Initialize the DMA registers according to the specified parameters in DMA_InitStruct.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7
- **DMA_InitStruct:** pointer to a LL_DMA_InitTypeDef structure.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: DMA registers are initialized
  - ERROR: Not applicable

Notes

- To convert DMAx_Streamy Instance to DMAx Instance and Streamy, use helper macros : __LL_DMA_GET_INSTANCE __LL_DMA_GET_STREAM

LL_DMA_DeInit

Function name

uint32_t LL_DMA_DeInit (DMA_TypeDef * DMAx, uint32_t Stream)

Function description

De-initialize the DMA registers to their default reset values.

Parameters

- **DMAx:** DMAx Instance
- **Stream:** This parameter can be one of the following values:
  - LL_DMA_STREAM_0
  - LL_DMA_STREAM_1
  - LL_DMA_STREAM_2
  - LL_DMA_STREAM_3
  - LL_DMA_STREAM_4
  - LL_DMA_STREAM_5
  - LL_DMA_STREAM_6
  - LL_DMA_STREAM_7
  - LL_DMA_STREAM_ALL

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: DMA registers are de-initialized
  - ERROR: DMA registers are not de-initialized

LL_DMA_StructInit

Function name

void LL_DMA_StructInit (LL_DMA_InitTypeDef * DMA_InitStruct)

Function description

Set each LL_DMA_InitTypeDef field to default value.

Parameters

- **DMA_InitStruct:** Pointer to a LL_DMA_InitTypeDef structure.

Return values

- **None:**

107.3 DMA Firmware driver defines

The following section lists the various define and macros of the module.

107.3.1 DMA

DMA

CURRENTTARGETMEM

LL_BDMA_CURRENTTARGETMEM0

Set CurrentTarget Memory to Memory 0

LL_BDMA_CURRENTTARGETMEM1

Set CurrentTarget Memory to Memory 1

LL_DMA_CURRENTTARGETMEM0

Set CurrentTarget Memory to Memory 0

LL_DMA_CURRENTTARGETMEM1

Set CurrentTarget Memory to Memory 1

DIRECTION

LL_DMA_DIRECTION_PERIPH_TO_MEMORY

Peripheral to memory direction

LL_DMA_DIRECTION_MEMORY_TO_PERIPH

Memory to peripheral direction

LL_DMA_DIRECTION_MEMORY_TO_MEMORY

Memory to memory direction

DOUBLE BUFFER MODE

LL_BDMA_DOUBLEBUFFER_MODE_DISABLE

Disable double buffering mode

LL_BDMA_DOUBLEBUFFER_MODE_ENABLE

Enable double buffering mode

LL_DMA_DOUBLEBUFFER_MODE_DISABLE

Disable double buffering mode

LL_DMA_DOUBLEBUFFER_MODE_ENABLE

Enable double buffering mode

FIFOSTATUS 0

LL_DMA_FIFOSTATUS_0_25

0 < fifo_level < 1/4

LL_DMA_FIFOSTATUS_25_50

1/4 < fifo_level < 1/2

LL_DMA_FIFOSTATUS_50_75

1/2 < fifo_level < 3/4

LL_DMA_FIFOSTATUS_75_100

3/4 < fifo_level < full

LL_DMA_FIFOSTATUS_EMPTY

FIFO is empty

LL_DMA_FIFOSTATUS_FULL

FIFO is full

FIFOTHRESHOLD

LL_DMA_FIFOTHRESHOLD_1_4

FIFO threshold 1 quart full configuration

LL_DMA_FIFOTHRESHOLD_1_2

FIFO threshold half full configuration

LL_DMA_FIFOTHRESHOLD_3_4

FIFO threshold 3 quarts full configuration

LL_DMA_FIFOTHRESHOLD_FULL

FIFO threshold full configuration

MBURST

LL_DMA_MBURST_SINGLE

Memory burst single transfer configuration

LL_DMA_MBURST_INC4

Memory burst of 4 beats transfer configuration

LL_DMA_MBURST_INC8

Memory burst of 8 beats transfer configuration

LL_DMA_MBURST_INC16

Memory burst of 16 beats transfer configuration

MDATAALIGN

LL_DMA_MDATAALIGN_BYTE

Memory data alignment : Byte

LL_DMA_MDATAALIGN_HALFWORD

Memory data alignment : HalfWord

LL_DMA_MDATAALIGN_WORD

Memory data alignment : Word

MEMORY

LL_DMA_MEMORY_NOINCREMENT

Memory increment mode Disable

LL_DMA_MEMORY_INCREMENT

Memory increment mode Enable

MODE

LL_DMA_MODE_NORMAL

Normal Mode

LL_DMA_MODE_CIRCULAR

Circular Mode

LL_DMA_MODE_PFCTRL

Peripheral flow control mode

OFFSETSIZE

LL_DMA_OFFSETSIZE_PSIZE

Peripheral increment offset size is linked to the PSIZE

LL_DMA_OFFSETSIZE_FIXEDTO4

Peripheral increment offset size is fixed to 4 (32-bit alignment)

PBURST

LL_DMA_PBURST_SINGLE

Peripheral burst single transfer configuration

LL_DMA_PBURST_INC4

Peripheral burst of 4 beats transfer configuration

LL_DMA_PBURST_INC8

Peripheral burst of 8 beats transfer configuration

LL_DMA_PBURST_INC16

Peripheral burst of 16 beats transfer configuration

PDATAALIGN

LL_DMA_PDATAALIGN_BYTE

Peripheral data alignment : Byte

LL_DMA_PDATAALIGN_HALFWORD

Peripheral data alignment : HalfWord

LL_DMA_PDATAALIGN_WORD

Peripheral data alignment : Word

PERIPH

LL_DMA_PERIPH_NOINCREMENT

Peripheral increment mode Disable

LL_DMA_PERIPH_INCREMENT

Peripheral increment mode Enable

PRIORITY

LL_DMA_PRIORITY_LOW

Priority level : Low

LL_DMA_PRIORITY_MEDIUM

Priority level : Medium

LL_DMA_PRIORITY_HIGH

Priority level : High

LL_DMA_PRIORITY_VERYHIGH

Priority level : Very_High

STREAM

LL_DMA_STREAM_0

LL_DMA_STREAM_1

LL_DMA_STREAM_2

LL_DMA_STREAM_3

LL_DMA_STREAM_4

LL_DMA_STREAM_5

LL_DMA_STREAM_6

LL_DMA_STREAM_7

LL_DMA_STREAM_ALL

Convert DMAxStreamy

__LL_DMA_GET_INSTANCE

Description:

- Convert DMAx_Streamy into DMAx.

Parameters:

- __STREAM_INSTANCE__: DMAx_Streamy

Return value:

- DMAx

__LL_DMA_GET_STREAM

Description:

- Convert DMAx_Streamy into LL_DMA_STREAM_y.

Parameters:

- __STREAM_INSTANCE__: DMAx_Streamy

Return value:

- LL_DMA_STREAM_y

__LL_DMA_GET_STREAM_INSTANCE

Description:

- Convert DMA Instance DMAx and LL_DMA_STREAM_y into DMAx_Streamy.

Parameters:

- __DMA_INSTANCE__: DMAx
- __STREAM__: LL_DMA_STREAM_y

Return value:

- DMAx_Streamy

Common Write and read registers macros

LL_DMA_WriteReg

Description:

- Write a value in DMA register.

Parameters:

- __INSTANCE__: DMA Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_DMA_ReadReg

Description:

- Read a value in DMA register.

Parameters:

- __INSTANCE__: DMA Instance
- __REG__: Register to be read

Return value:

- Register: value

DMA_LL_FIFOMODE

LL_DMA_FIFOMODE_DISABLE

FIFO mode disable (direct mode is enabled)

LL_DMA_FIFOMODE_ENABLE

FIFO mode enable

108 LL EXTI Generic Driver

108.1 EXTI Firmware driver registers structures

108.1.1 LL_EXTI_InitTypeDef

LL_EXTI_InitTypeDef is defined in the stm32h7xx_ll_exti.h

Data Fields

- ***uint32_t Line_0_31***
- ***uint32_t Line_32_63***
- ***uint32_t Line_64_95***
- ***FunctionalState LineCommand***
- ***uint8_t Mode***
- ***uint8_t Trigger***

Field Documentation

- ***uint32_t LL_EXTI_InitTypeDef::Line_0_31***
  Specifies the EXTI lines to be enabled or disabled for Lines in range 0 to 31 This parameter can be any combination of ***EXTI_LL_EC_LINE***
- ***uint32_t LL_EXTI_InitTypeDef::Line_32_63***
  Specifies the EXTI lines to be enabled or disabled for Lines in range 32 to 63 This parameter can be any combination of ***EXTI_LL_EC_LINE***
- ***uint32_t LL_EXTI_InitTypeDef::Line_64_95***
  Specifies the EXTI lines to be enabled or disabled for Lines in range 64 to 95 This parameter can be any combination of ***EXTI_LL_EC_LINE***
- ***FunctionalState LL_EXTI_InitTypeDef::LineCommand***
  Specifies the new state of the selected EXTI lines. This parameter can be set either to ENABLE or DISABLE
- ***uint8_t LL_EXTI_InitTypeDef::Mode***
  Specifies the mode for the EXTI lines. This parameter can be a value of ***EXTI_LL_EC_MODE***.
- ***uint8_t LL_EXTI_InitTypeDef::Trigger***
  Specifies the trigger signal active edge for the EXTI lines. This parameter can be a value of ***EXTI_LL_EC_TRIGGER***.

108.2 EXTI Firmware driver API description

The following section lists the various functions of the EXTI library.

108.2.1 Detailed description of functions

LL_EXTI_EnableIT_0_31

Function name

__STATIC_INLINE void LL_EXTI_EnableIT_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine Interrupt request for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR1 IMx LL_EXTI_EnableIT_0_31

LL_EXTI_EnableIT_32_63

Function name

__STATIC_INLINE void LL_EXTI_EnableIT_32_63 (uint32_t ExtiLine)

Function description

Enable ExtiLine Interrupt request for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44 (*)
  - LL_EXTI_LINE_46 (*)
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57 (*)
  - LL_EXTI_LINE_58
  - LL_EXTI_LINE_59 (*)
  - LL_EXTI_LINE_60
  - LL_EXTI_LINE_61
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_63
  - LL_EXTI_LINE_ALL_32_63

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR2 IMx LL_EXTI_EnableIT_32_63

LL_EXTI_EnableIT_64_95

Function name

__STATIC_INLINE void LL_EXTI_EnableIT_64_95 (uint32_t ExtiLine)

Function description

Enable ExtiLine Interrupt request for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_64
  - LL_EXTI_LINE_65
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_67
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74
  - LL_EXTI_LINE_75 (*)
  - LL_EXTI_LINE_76 (*)
  - LL_EXTI_LINE_77 (**)
  - LL_EXTI_LINE_78 (**)
  - LL_EXTI_LINE_79 (**)
  - LL_EXTI_LINE_80 (**)
  - LL_EXTI_LINE_82 (**)
  - LL_EXTI_LINE_84 (**)
  - LL_EXTI_LINE_85
  - LL_EXTI_LINE_86 (*)
  - LL_EXTI_LINE_87
  - LL_EXTI_LINE_88 (*)
  - LL_EXTI_LINE_89 (*)
  - LL_EXTI_LINE_90 (*)
  - LL_EXTI_LINE_91 (*)
  - LL_EXTI_LINE_ALL_64_95

  (*) value not defined in all devices. (**) value only defined in dual core devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR3 IMx LL_EXTI_EnableIT_64_95

LL_EXTI_DisableIT_0_31

Function name

__STATIC_INLINE void LL_EXTI_DisableIT_0_31 (uint32_t ExtiLine)

Function description

Disable ExtiLine Interrupt request for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR1 IMx LL_EXTI_DisableIT_0_31

LL_EXTI_DisableIT_32_63

Function name

__STATIC_INLINE void LL_EXTI_DisableIT_32_63 (uint32_t ExtiLine)

Function description

Disable ExtiLine Interrupt request for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44 (*)
  - LL_EXTI_LINE_46 (*)
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57 (*)
  - LL_EXTI_LINE_58
  - LL_EXTI_LINE_59 (*)
  - LL_EXTI_LINE_60
  - LL_EXTI_LINE_61
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_63
  - LL_EXTI_LINE_ALL_32_63

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR2 IMx LL_EXTI_DisableIT_32_63

LL_EXTI_DisableIT_64_95

Function name

__STATIC_INLINE void LL_EXTI_DisableIT_64_95 (uint32_t ExtiLine)

Function description

Disable ExtiLine Interrupt request for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_64
  - LL_EXTI_LINE_65
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_67
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74
  - LL_EXTI_LINE_75 (*)
  - LL_EXTI_LINE_76 (*)
  - LL_EXTI_LINE_77 (**)
  - LL_EXTI_LINE_78 (**)
  - LL_EXTI_LINE_79 (**)
  - LL_EXTI_LINE_80 (**)
  - LL_EXTI_LINE_82 (**)
  - LL_EXTI_LINE_84 (**)
  - LL_EXTI_LINE_85
  - LL_EXTI_LINE_86 (*)
  - LL_EXTI_LINE_87
  - LL_EXTI_LINE_88 (*)
  - LL_EXTI_LINE_89 (*)
  - LL_EXTI_LINE_90 (*)
  - LL_EXTI_LINE_91 (*)
  - LL_EXTI_LINE_ALL_64_95

  (*) value not defined in all devices. (**) value only defined in dual core devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR3 IMx LL_EXTI_DisableIT_64_95

LL_EXTI_IsEnabledIT_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledIT_0_31 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Interrupt request is enabled for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR1 IMx LL_EXTI_IsEnabledIT_0_31

LL_EXTI_IsEnabledIT_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledIT_32_63 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Interrupt request is enabled for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44 (*)
  - LL_EXTI_LINE_46 (*)
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57 (*)
  - LL_EXTI_LINE_58
  - LL_EXTI_LINE_59 (*)
  - LL_EXTI_LINE_60
  - LL_EXTI_LINE_61
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_63
  - LL_EXTI_LINE_ALL_32_63

  (*) value not defined in all devices.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR2 IMx LL_EXTI_IsEnabledIT_32_63

LL_EXTI_IsEnabledIT_64_95

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledIT_64_95 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Interrupt request is enabled for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_64
  - LL_EXTI_LINE_65
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_67
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74
  - LL_EXTI_LINE_75 (*)
  - LL_EXTI_LINE_76 (*)
  - LL_EXTI_LINE_77 (**)
  - LL_EXTI_LINE_78 (**)
  - LL_EXTI_LINE_79 (**)
  - LL_EXTI_LINE_80 (**)
  - LL_EXTI_LINE_82 (**)
  - LL_EXTI_LINE_84 (**)
  - LL_EXTI_LINE_85
  - LL_EXTI_LINE_86 (*)
  - LL_EXTI_LINE_87
  - LL_EXTI_LINE_88 (*)
  - LL_EXTI_LINE_89 (*)
  - LL_EXTI_LINE_90 (*)
  - LL_EXTI_LINE_91 (*)
  - LL_EXTI_LINE_ALL_64_95

  (*) value not defined in all devices. (**) value only defined in dual core devices.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR3 IMx LL_EXTI_IsEnabledIT_64_95

LL_C2_EXTI_EnableIT_0_31

Function name

__STATIC_INLINE void LL_C2_EXTI_EnableIT_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine Interrupt request for Lines in range 0 to 31 for cpu2.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Reference Manual to LL API cross reference:

- C2IMR1 IMx LL_C2_EXTI_EnableIT_0_31

LL_C2_EXTI_EnableIT_32_63

Function name

__STATIC_INLINE void LL_C2_EXTI_EnableIT_32_63 (uint32_t ExtiLine)

Function description

Enable ExtiLine Interrupt request for Lines in range 32 to 63 for cpu2.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57
  - LL_EXTI_LINE_58
  - LL_EXTI_LINE_59
  - LL_EXTI_LINE_60
  - LL_EXTI_LINE_61
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_63
  - LL_EXTI_LINE_ALL_32_63

Return values

- **None:**

Reference Manual to LL API cross reference:

- C2IMR2 IMx LL_C2_EXTI_EnableIT_32_63

LL_C2_EXTI_EnableIT_64_95

Function name

__STATIC_INLINE void LL_C2_EXTI_EnableIT_64_95 (uint32_t ExtiLine)

Function description

Enable ExtiLine Interrupt request for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_64
  - LL_EXTI_LINE_65
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_67
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74
  - LL_EXTI_LINE_75
  - LL_EXTI_LINE_76
  - LL_EXTI_LINE_77
  - LL_EXTI_LINE_78
  - LL_EXTI_LINE_79
  - LL_EXTI_LINE_80
  - LL_EXTI_LINE_82
  - LL_EXTI_LINE_84
  - LL_EXTI_LINE_85
  - LL_EXTI_LINE_86
  - LL_EXTI_LINE_87
  - LL_EXTI_LINE_ALL_64_95

Return values

- **None:**

Reference Manual to LL API cross reference:

- C2IMR3 IMx LL_C2_EXTI_EnableIT_64_95

LL_C2_EXTI_DisableIT_0_31

Function name

__STATIC_INLINE void LL_C2_EXTI_DisableIT_0_31 (uint32_t ExtiLine)

Function description

Disable ExtiLine Interrupt request for Lines in range 0 to 31 for cpu2.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Reference Manual to LL API cross reference:

- C2IMR1 IMx LL_C2_EXTI_DisableIT_0_31

LL_C2_EXTI_DisableIT_32_63

Function name

__STATIC_INLINE void LL_C2_EXTI_DisableIT_32_63 (uint32_t ExtiLine)

Function description

Disable ExtiLine Interrupt request for Lines in range 32 to 63 for cpu2.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57
  - LL_EXTI_LINE_58
  - LL_EXTI_LINE_59
  - LL_EXTI_LINE_60
  - LL_EXTI_LINE_61
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_63
  - LL_EXTI_LINE_ALL_32_63

Return values

- **None:**

Reference Manual to LL API cross reference:

- C2IMR2 IMx LL_C2_EXTI_DisableIT_32_63

LL_C2_EXTI_DisableIT_64_95

Function name

__STATIC_INLINE void LL_C2_EXTI_DisableIT_64_95 (uint32_t ExtiLine)

Function description

Disable ExtiLine Interrupt request for Lines in range 64 to 95 for cpu2.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_64
  - LL_EXTI_LINE_65
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_67
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74
  - LL_EXTI_LINE_75
  - LL_EXTI_LINE_76
  - LL_EXTI_LINE_77
  - LL_EXTI_LINE_78
  - LL_EXTI_LINE_79
  - LL_EXTI_LINE_80
  - LL_EXTI_LINE_82
  - LL_EXTI_LINE_84
  - LL_EXTI_LINE_85
  - LL_EXTI_LINE_86
  - LL_EXTI_LINE_87
  - LL_EXTI_LINE_ALL_64_95

Return values

- **None:**

Reference Manual to LL API cross reference:

- C2IMR3 IMx LL_C2_EXTI_DisableIT_64_95

LL_C2_EXTI_IsEnabledIT_0_31

Function name

__STATIC_INLINE uint32_t LL_C2_EXTI_IsEnabledIT_0_31 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Interrupt request is enabled for Lines in range 0 to 31 for cpu2.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- C2IMR1 IMx LL_C2_EXTI_IsEnabledIT_0_31

LL_C2_EXTI_IsEnabledIT_32_63

Function name

__STATIC_INLINE uint32_t LL_C2_EXTI_IsEnabledIT_32_63 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Interrupt request is enabled for Lines in range 32 to 63 for cpu2.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57
  - LL_EXTI_LINE_58
  - LL_EXTI_LINE_59
  - LL_EXTI_LINE_60
  - LL_EXTI_LINE_61
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_63
  - LL_EXTI_LINE_ALL_32_63

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- C2IMR2 IMx LL_C2_EXTI_IsEnabledIT_32_63

LL_C2_EXTI_IsEnabledIT_64_95

Function name

__STATIC_INLINE uint32_t LL_C2_EXTI_IsEnabledIT_64_95 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Interrupt request is enabled for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_64
  - LL_EXTI_LINE_65
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_67
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74
  - LL_EXTI_LINE_75
  - LL_EXTI_LINE_76
  - LL_EXTI_LINE_77
  - LL_EXTI_LINE_78
  - LL_EXTI_LINE_79
  - LL_EXTI_LINE_80
  - LL_EXTI_LINE_82
  - LL_EXTI_LINE_84
  - LL_EXTI_LINE_85
  - LL_EXTI_LINE_86
  - LL_EXTI_LINE_87
  - LL_EXTI_LINE_ALL_64_95

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- C2IMR3 IMx LL_C2_EXTI_IsEnabledIT_64_95

LL_EXTI_EnableEvent_0_31

Function name

__STATIC_INLINE void LL_EXTI_EnableEvent_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine Event request for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Reference Manual to LL API cross reference:

- EMR1 EMx LL_EXTI_EnableEvent_0_31

LL_EXTI_EnableEvent_32_63

Function name

__STATIC_INLINE void LL_EXTI_EnableEvent_32_63 (uint32_t ExtiLine)

Function description

Enable ExtiLine Event request for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44 (*)
  - LL_EXTI_LINE_46 (*)
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57 (*)
  - LL_EXTI_LINE_58
  - LL_EXTI_LINE_59 (*)
  - LL_EXTI_LINE_60
  - LL_EXTI_LINE_61
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_63
  - LL_EXTI_LINE_ALL_32_63

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- EMR2 EMx LL_EXTI_EnableEvent_32_63

LL_EXTI_EnableEvent_64_95

Function name

__STATIC_INLINE void LL_EXTI_EnableEvent_64_95 (uint32_t ExtiLine)

Function description

Enable ExtiLine Event request for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_64
  - LL_EXTI_LINE_65
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_67
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74
  - LL_EXTI_LINE_75 (*)
  - LL_EXTI_LINE_76 (*)
  - LL_EXTI_LINE_77 (**)
  - LL_EXTI_LINE_78 (**)
  - LL_EXTI_LINE_79 (**)
  - LL_EXTI_LINE_80 (**)
  - LL_EXTI_LINE_82 (**)
  - LL_EXTI_LINE_84 (**)
  - LL_EXTI_LINE_85
  - LL_EXTI_LINE_86 (*)
  - LL_EXTI_LINE_87
  - LL_EXTI_LINE_88 (*)
  - LL_EXTI_LINE_89 (*)
  - LL_EXTI_LINE_90 (*)
  - LL_EXTI_LINE_91 (*)
  - LL_EXTI_LINE_ALL_64_95

  (*) value not defined in all devices. (**) value only defined in dual core devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- EMR3 EMx LL_EXTI_EnableEvent_64_95

LL_EXTI_DisableEvent_0_31

Function name

__STATIC_INLINE void LL_EXTI_DisableEvent_0_31 (uint32_t ExtiLine)

Function description

Disable ExtiLine Event request for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Reference Manual to LL API cross reference:

- EMR1 EMx LL_EXTI_DisableEvent_0_31

LL_EXTI_DisableEvent_32_63

Function name

__STATIC_INLINE void LL_EXTI_DisableEvent_32_63 (uint32_t ExtiLine)

Function description

Disable ExtiLine Event request for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44 (*)
  - LL_EXTI_LINE_46 (*)
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57 (*)
  - LL_EXTI_LINE_58
  - LL_EXTI_LINE_59 (*)
  - LL_EXTI_LINE_60
  - LL_EXTI_LINE_61
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_63
  - LL_EXTI_LINE_ALL_32_63

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- EMR2 EMx LL_EXTI_DisableEvent_32_63

LL_EXTI_DisableEvent_64_95

Function name

__STATIC_INLINE void LL_EXTI_DisableEvent_64_95 (uint32_t ExtiLine)

Function description

Disable ExtiLine Event request for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_64
  - LL_EXTI_LINE_65
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_67
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74
  - LL_EXTI_LINE_75 (*)
  - LL_EXTI_LINE_76 (*)
  - LL_EXTI_LINE_77 (**)
  - LL_EXTI_LINE_78 (**)
  - LL_EXTI_LINE_79 (**)
  - LL_EXTI_LINE_80 (**)
  - LL_EXTI_LINE_82 (**)
  - LL_EXTI_LINE_84 (**)
  - LL_EXTI_LINE_85
  - LL_EXTI_LINE_86 (*)
  - LL_EXTI_LINE_87
  - LL_EXTI_LINE_88 (*)
  - LL_EXTI_LINE_89 (*)
  - LL_EXTI_LINE_90 (*)
  - LL_EXTI_LINE_91 (*)
  - LL_EXTI_LINE_ALL_64_95

  (*) value not defined in all devices. (**) value only defined in dual core devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- EMR3 EMx LL_EXTI_DisableEvent_64_95

LL_EXTI_IsEnabledEvent_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledEvent_0_31 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Event request is enabled for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **State:** of bit (1 or 0).

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- EMR1 EMx LL_EXTI_IsEnabledEvent_0_31

LL_EXTI_IsEnabledEvent_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledEvent_32_63 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Event request is enabled for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44 (*)
  - LL_EXTI_LINE_46 (*)
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57 (*)
  - LL_EXTI_LINE_58
  - LL_EXTI_LINE_59 (*)
  - LL_EXTI_LINE_60
  - LL_EXTI_LINE_61
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_63
  - LL_EXTI_LINE_ALL_32_63

  (*) value not defined in all devices.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EMR2 EMx LL_EXTI_IsEnabledEvent_32_63

LL_EXTI_IsEnabledEvent_64_95

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledEvent_64_95 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Event request is enabled for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_64
  - LL_EXTI_LINE_65
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_67
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74
  - LL_EXTI_LINE_75 (*)
  - LL_EXTI_LINE_76 (*)
  - LL_EXTI_LINE_77 (**)
  - LL_EXTI_LINE_78 (**)
  - LL_EXTI_LINE_79 (**)
  - LL_EXTI_LINE_80 (**)
  - LL_EXTI_LINE_82 (**)
  - LL_EXTI_LINE_84 (**)
  - LL_EXTI_LINE_85
  - LL_EXTI_LINE_86 (*)
  - LL_EXTI_LINE_87
  - LL_EXTI_LINE_88 (*)
  - LL_EXTI_LINE_89 (*)
  - LL_EXTI_LINE_90 (*)
  - LL_EXTI_LINE_91 (*)
  - LL_EXTI_LINE_ALL_64_95

  (*) value not defined in all devices. (**) value only defined in dual core devices.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- EMR3 EMx LL_EXTI_IsEnabledEvent_64_95

LL_C2_EXTI_EnableEvent_0_31

Function name

__STATIC_INLINE void LL_C2_EXTI_EnableEvent_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine Event request for Lines in range 0 to 31 for cpu2.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Reference Manual to LL API cross reference:

- C2EMR1 EMx LL_C2_EXTI_EnableEvent_0_31

LL_C2_EXTI_EnableEvent_32_63

Function name

__STATIC_INLINE void LL_C2_EXTI_EnableEvent_32_63 (uint32_t ExtiLine)

Function description

Enable ExtiLine Event request for Lines in range 32 to 63 for cpu2.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57
  - LL_EXTI_LINE_58
  - LL_EXTI_LINE_59
  - LL_EXTI_LINE_60
  - LL_EXTI_LINE_61
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_63
  - LL_EXTI_LINE_ALL_32_63

Return values

- **None:**

Reference Manual to LL API cross reference:

- C2EMR2 EMx LL_C2_EXTI_EnableEvent_32_63

LL_C2_EXTI_EnableEvent_64_95

Function name

__STATIC_INLINE void LL_C2_EXTI_EnableEvent_64_95 (uint32_t ExtiLine)

Function description

Enable ExtiLine Event request for Lines in range 64 to 95 for cpu2.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_64
  - LL_EXTI_LINE_65
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_67
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74
  - LL_EXTI_LINE_75
  - LL_EXTI_LINE_76
  - LL_EXTI_LINE_77
  - LL_EXTI_LINE_78
  - LL_EXTI_LINE_79
  - LL_EXTI_LINE_80
  - LL_EXTI_LINE_82
  - LL_EXTI_LINE_84
  - LL_EXTI_LINE_85
  - LL_EXTI_LINE_86
  - LL_EXTI_LINE_87
  - LL_EXTI_LINE_ALL_64_95

Return values

- **None:**

Reference Manual to LL API cross reference:

- C2EMR3 EMx LL_C2_EXTI_EnableEvent_64_95

LL_C2_EXTI_DisableEvent_0_31

Function name

__STATIC_INLINE void LL_C2_EXTI_DisableEvent_0_31 (uint32_t ExtiLine)

Function description

Disable ExtiLine Event request for Lines in range 0 to 31 for cpu2.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Reference Manual to LL API cross reference:

- C2EMR1 EMx LL_C2_EXTI_DisableEvent_0_31

LL_C2_EXTI_DisableEvent_32_63

Function name

__STATIC_INLINE void LL_C2_EXTI_DisableEvent_32_63 (uint32_t ExtiLine)

Function description

Disable ExtiLine Event request for Lines in range 32 to 63 for cpu2.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57
  - LL_EXTI_LINE_58
  - LL_EXTI_LINE_59
  - LL_EXTI_LINE_60
  - LL_EXTI_LINE_61
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_63
  - LL_EXTI_LINE_ALL_32_63

Return values

- **None:**

Reference Manual to LL API cross reference:

- C2EMR2 EMx LL_C2_EXTI_DisableEvent_32_63

LL_C2_EXTI_DisableEvent_64_95

Function name

__STATIC_INLINE void LL_C2_EXTI_DisableEvent_64_95 (uint32_t ExtiLine)

Function description

Disable ExtiLine Event request for Lines in range 64 to 95 for cpu2.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_64
  - LL_EXTI_LINE_65
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_67
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74
  - LL_EXTI_LINE_75
  - LL_EXTI_LINE_76
  - LL_EXTI_LINE_77
  - LL_EXTI_LINE_78
  - LL_EXTI_LINE_79
  - LL_EXTI_LINE_80
  - LL_EXTI_LINE_82
  - LL_EXTI_LINE_84
  - LL_EXTI_LINE_85
  - LL_EXTI_LINE_86
  - LL_EXTI_LINE_87
  - LL_EXTI_LINE_ALL_64_95

Return values

- **None:**

Reference Manual to LL API cross reference:

- C2EMR3 EMx LL_C2_EXTI_DisableEvent_64_95

LL_C2_EXTI_IsEnabledEvent_0_31

Function name

__STATIC_INLINE uint32_t LL_C2_EXTI_IsEnabledEvent_0_31 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Event request is enabled for Lines in range 0 to 31 for cpu2.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_26
  - LL_EXTI_LINE_27
  - LL_EXTI_LINE_28
  - LL_EXTI_LINE_29
  - LL_EXTI_LINE_30
  - LL_EXTI_LINE_31
  - LL_EXTI_LINE_ALL_0_31

Return values

- **State:** of bit (1 or 0).

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- C2EMR1 EMx LL_C2_EXTI_IsEnabledEvent_0_31

LL_C2_EXTI_IsEnabledEvent_32_63

Function name

__STATIC_INLINE uint32_t LL_C2_EXTI_IsEnabledEvent_32_63 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Event request is enabled for Lines in range 32 to 63 for cpu2.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_32
  - LL_EXTI_LINE_33
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_36
  - LL_EXTI_LINE_37
  - LL_EXTI_LINE_38
  - LL_EXTI_LINE_39
  - LL_EXTI_LINE_40
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_42
  - LL_EXTI_LINE_43
  - LL_EXTI_LINE_44
  - LL_EXTI_LINE_46
  - LL_EXTI_LINE_47
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
  - LL_EXTI_LINE_54
  - LL_EXTI_LINE_55
  - LL_EXTI_LINE_56
  - LL_EXTI_LINE_57
  - LL_EXTI_LINE_58
  - LL_EXTI_LINE_59
  - LL_EXTI_LINE_60
  - LL_EXTI_LINE_61
  - LL_EXTI_LINE_62
  - LL_EXTI_LINE_63
  - LL_EXTI_LINE_ALL_32_63

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- C2EMR2 EMx LL_C2_EXTI_IsEnabledEvent_32_63

LL_C2_EXTI_IsEnabledEvent_64_95

Function name

__STATIC_INLINE uint32_t LL_C2_EXTI_IsEnabledEvent_64_95 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Event request is enabled for Lines in range 64 to 95 for cpu2.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_64
  - LL_EXTI_LINE_65
  - LL_EXTI_LINE_66
  - LL_EXTI_LINE_67
  - LL_EXTI_LINE_68
  - LL_EXTI_LINE_69
  - LL_EXTI_LINE_70
  - LL_EXTI_LINE_71
  - LL_EXTI_LINE_72
  - LL_EXTI_LINE_73
  - LL_EXTI_LINE_74
  - LL_EXTI_LINE_75
  - LL_EXTI_LINE_76
  - LL_EXTI_LINE_77
  - LL_EXTI_LINE_78
  - LL_EXTI_LINE_79
  - LL_EXTI_LINE_80
  - LL_EXTI_LINE_82
  - LL_EXTI_LINE_84
  - LL_EXTI_LINE_85
  - LL_EXTI_LINE_86
  - LL_EXTI_LINE_87
  - LL_EXTI_LINE_ALL_64_95

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- C2EMR3 EMx LL_C2_EXTI_IsEnabledEvent_64_95

LL_EXTI_EnableRisingTrig_0_31

Function name

__STATIC_INLINE void LL_EXTI_EnableRisingTrig_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine Rising Edge Trigger for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

Reference Manual to LL API cross reference:

- RTSR1 RTx LL_EXTI_EnableRisingTrig_0_31

LL_EXTI_EnableRisingTrig_32_63

Function name

__STATIC_INLINE void LL_EXTI_EnableRisingTrig_32_63 (uint32_t ExtiLine)

Function description

Enable ExtiLine Rising Edge Trigger for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_51

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set.Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

Reference Manual to LL API cross reference:

- RTSR2 RTx LL_EXTI_EnableRisingTrig_32_63

LL_EXTI_EnableRisingTrig_64_95

Function name

__STATIC_INLINE void LL_EXTI_EnableRisingTrig_64_95 (uint32_t ExtiLine)

Function description

Enable ExtiLine Rising Edge Trigger for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_82 (*)
  - LL_EXTI_LINE_84 (*)
  - LL_EXTI_LINE_85
  - LL_EXTI_LINE_86 (**)

  (*) value only defined in dual core devices. (**) value not defined in all devices.

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set.Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

Reference Manual to LL API cross reference:

- RTSR3 RTx LL_EXTI_EnableRisingTrig_64_95

LL_EXTI_DisableRisingTrig_0_31

Function name

__STATIC_INLINE void LL_EXTI_DisableRisingTrig_0_31 (uint32_t ExtiLine)

Function description

Disable ExtiLine Rising Edge Trigger for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

Reference Manual to LL API cross reference:

- RTSR1 RTx LL_EXTI_DisableRisingTrig_0_31

LL_EXTI_DisableRisingTrig_32_63

Function name

__STATIC_INLINE void LL_EXTI_DisableRisingTrig_32_63 (uint32_t ExtiLine)

Function description

Disable ExtiLine Rising Edge Trigger for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_51

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

Reference Manual to LL API cross reference:

- RTSR2 RTx LL_EXTI_DisableRisingTrig_32_63

LL_EXTI_DisableRisingTrig_64_95

Function name

__STATIC_INLINE void LL_EXTI_DisableRisingTrig_64_95 (uint32_t ExtiLine)

Function description

Disable ExtiLine Rising Edge Trigger for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_82 (*)
  - LL_EXTI_LINE_84 (*)
  - LL_EXTI_LINE_85
  - LL_EXTI_LINE_86 (**)

  (*) value only defined in dual core devices. (**) value not defined in all devices.

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

Reference Manual to LL API cross reference:

- RTSR3 RTx LL_EXTI_DisableRisingTrig_64_95

LL_EXTI_IsEnabledRisingTrig_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledRisingTrig_0_31 (uint32_t ExtiLine)

Function description

Check if rising edge trigger is enabled for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTSR1 RTx LL_EXTI_IsEnabledRisingTrig_0_31

LL_EXTI_IsEnabledRisingTrig_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledRisingTrig_32_63 (uint32_t ExtiLine)

Function description

Check if rising edge trigger is enabled for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_51

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTSR2 RTx LL_EXTI_IsEnabledRisingTrig_32_63

LL_EXTI_IsEnabledRisingTrig_64_95

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledRisingTrig_64_95 (uint32_t ExtiLine)

Function description

Check if rising edge trigger is enabled for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_82 (*)
  - LL_EXTI_LINE_84 (*)
  - LL_EXTI_LINE_85
  - LL_EXTI_LINE_86 (**)

  (*) value only defined in dual core devices. (**) value not defined in all devices.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTSR3 RTx LL_EXTI_IsEnabledRisingTrig_64_95

LL_EXTI_EnableFallingTrig_0_31

Function name

__STATIC_INLINE void LL_EXTI_EnableFallingTrig_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine Falling Edge Trigger for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- FTSR1 FTx LL_EXTI_EnableFallingTrig_0_31

LL_EXTI_EnableFallingTrig_32_63

Function name

__STATIC_INLINE void LL_EXTI_EnableFallingTrig_32_63 (uint32_t ExtiLine)

Function description

Enable ExtiLine Falling Edge Trigger for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_51

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a Falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

Reference Manual to LL API cross reference:

- FTSR2 FTx LL_EXTI_EnableFallingTrig_32_63

LL_EXTI_EnableFallingTrig_64_95

Function name

__STATIC_INLINE void LL_EXTI_EnableFallingTrig_64_95 (uint32_t ExtiLine)

Function description

Enable ExtiLine Falling Edge Trigger for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_82 (*)
  - LL_EXTI_LINE_84 (*)
  - LL_EXTI_LINE_85
  - LL_EXTI_LINE_86 (**)

  (*) value only defined in dual core devices. (**) value not defined in all devices.

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a Falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

Reference Manual to LL API cross reference:

- FTSR3 FTx LL_EXTI_EnableFallingTrig_64_95

LL_EXTI_DisableFallingTrig_0_31

Function name

__STATIC_INLINE void LL_EXTI_DisableFallingTrig_0_31 (uint32_t ExtiLine)

Function description

Disable ExtiLine Falling Edge Trigger for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a Falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- FTSR1 FTx LL_EXTI_DisableFallingTrig_0_31

LL_EXTI_DisableFallingTrig_32_63

Function name

__STATIC_INLINE void LL_EXTI_DisableFallingTrig_32_63 (uint32_t ExtiLine)

Function description

Disable ExtiLine Falling Edge Trigger for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_51

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a Falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

Reference Manual to LL API cross reference:

- FTSR2 FTx LL_EXTI_DisableFallingTrig_32_63

LL_EXTI_DisableFallingTrig_64_95

Function name

__STATIC_INLINE void LL_EXTI_DisableFallingTrig_64_95 (uint32_t ExtiLine)

Function description

Disable ExtiLine Falling Edge Trigger for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_82 (*)
  - LL_EXTI_LINE_84 (*)
  - LL_EXTI_LINE_85
  - LL_EXTI_LINE_86 (**)

  (*) value only defined in dual core devices. (**) value not defined in all devices.

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a Falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

Reference Manual to LL API cross reference:

- FTSR3 FTx LL_EXTI_DisableFallingTrig_64_95

LL_EXTI_IsEnabledFallingTrig_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledFallingTrig_0_31 (uint32_t ExtiLine)

Function description

Check if falling edge trigger is enabled for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21

Return values

- **State:** of bit (1 or 0).

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- FTSR1 FTx LL_EXTI_IsEnabledFallingTrig_0_31

LL_EXTI_IsEnabledFallingTrig_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledFallingTrig_32_63 (uint32_t ExtiLine)

Function description

Check if falling edge trigger is enabled for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_51

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- FTSR2 FTx LL_EXTI_IsEnabledFallingTrig_32_63

LL_EXTI_IsEnabledFallingTrig_64_95

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledFallingTrig_64_95 (uint32_t ExtiLine)

Function description

Check if falling edge trigger is enabled for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_82 (*)
  - LL_EXTI_LINE_84 (*)
  - LL_EXTI_LINE_85
  - LL_EXTI_LINE_86 (**)

  (*) value only defined in dual core devices. (**) value not defined in all devices.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- FTSR3 FTx LL_EXTI_IsEnabledFallingTrig_64_95

LL_EXTI_GenerateSWI_0_31

Function name

__STATIC_INLINE void LL_EXTI_GenerateSWI_0_31 (uint32_t ExtiLine)

Function description

Generate a software Interrupt Event for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21

Return values

- **None:**

Notes

- If the interrupt is enabled on this line in the EXTI_C1IMR1, writing a 1 to this bit when it is at '0' sets the corresponding pending bit in EXTI_PR1 resulting in an interrupt request generation. This bit is cleared by clearing the corresponding bit in the EXTI_PR1 register (by writing a 1 into the bit)
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- SWIER1 SWIx LL_EXTI_GenerateSWI_0_31

LL_EXTI_GenerateSWI_32_63

Function name

__STATIC_INLINE void LL_EXTI_GenerateSWI_32_63 (uint32_t ExtiLine)

Function description

Generate a software Interrupt Event for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_51

Return values

- **None:**

Notes

- If the interrupt is enabled on this line in the EXTI_IMR2, writing a 1 to this bit when it is at '0' sets the corresponding pending bit in EXTI_PR2 resulting in an interrupt request generation. This bit is cleared by clearing the corresponding bit in the EXTI_PR2 register (by writing a 1 into the bit)

Reference Manual to LL API cross reference:

- SWIER2 SWIx LL_EXTI_GenerateSWI_32_63

LL_EXTI_GenerateSWI_64_95

Function name

__STATIC_INLINE void LL_EXTI_GenerateSWI_64_95 (uint32_t ExtiLine)

Function description

Generate a software Interrupt Event for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_82 (*)
  - LL_EXTI_LINE_84 (*)
  - LL_EXTI_LINE_85
  - LL_EXTI_LINE_86 (**)

  (*) value only defined in dual core devices. (**) value not defined in all devices.

Return values

- **None:**

Notes

- If the interrupt is enabled on this line in the EXTI_IMR2, writing a 1 to this bit when it is at '0' sets the corresponding pending bit in EXTI_PR2 resulting in an interrupt request generation. This bit is cleared by clearing the corresponding bit in the EXTI_PR3 register (by writing a 1 into the bit)

Reference Manual to LL API cross reference:

- SWIER3 SWIx LL_EXTI_GenerateSWI_64_95

LL_EXTI_IsActiveFlag_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsActiveFlag_0_31 (uint32_t ExtiLine)

Function description

Check if the ExtLine Flag is set or not for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_ALL_0_31

Return values

- **State:** of bit (1 or 0).

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.

Reference Manual to LL API cross reference:

- PR1 PIFx LL_EXTI_IsActiveFlag_0_31

LL_EXTI_IsActiveFlag_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsActiveFlag_32_63 (uint32_t ExtiLine)

Function description

Check if the ExtLine Flag is set or not for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_ALL_32_63

Return values

- **State:** of bit (1 or 0).

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.

Reference Manual to LL API cross reference:

- PR2 PIFx LL_EXTI_IsActiveFlag_32_63

LL_EXTI_IsActiveFlag_64_95

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsActiveFlag_64_95 (uint32_t ExtiLine)

Function description

Check if the ExtLine Flag is set or not for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_82 (*)
  - LL_EXTI_LINE_84 (*)
  - LL_EXTI_LINE_85
  - LL_EXTI_LINE_86 (**)

  (*) value only defined in dual core devices. (**) value not defined in all devices.

Return values

- **State:** of bit (1 or 0).

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.

Reference Manual to LL API cross reference:

- PR3 PIFx LL_EXTI_IsActiveFlag_64_95

LL_EXTI_ReadFlag_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_ReadFlag_0_31 (uint32_t ExtiLine)

Function description

Read ExtLine Combination Flag for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21

Return values

- **@note:** This bit is set when the selected edge event arrives on the interrupt

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.

Reference Manual to LL API cross reference:

- PR1 PIFx LL_EXTI_ReadFlag_0_31

LL_EXTI_ReadFlag_32_63

Function name

__STATIC_INLINE uint32_t LL_EXTI_ReadFlag_32_63 (uint32_t ExtiLine)

Function description

Read ExtLine Combination Flag for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_51

Return values

- **@note:** This bit is set when the selected edge event arrives on the interrupt

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.

Reference Manual to LL API cross reference:

- PR2 PIFx LL_EXTI_ReadFlag_32_63

LL_EXTI_ReadFlag_64_95

Function name

__STATIC_INLINE uint32_t LL_EXTI_ReadFlag_64_95 (uint32_t ExtiLine)

Function description

Read ExtLine Combination Flag for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_82 (*)
  - LL_EXTI_LINE_84 (*)
  - LL_EXTI_LINE_85
  - LL_EXTI_LINE_86 (**)

  (*) value only defined in dual core devices. (**) value not defined in all devices.

Return values

- **@note:** This bit is set when the selected edge event arrives on the interrupt

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.

Reference Manual to LL API cross reference:

- PR3 PIFx LL_EXTI_ReadFlag_64_95

LL_EXTI_ClearFlag_0_31

Function name

__STATIC_INLINE void LL_EXTI_ClearFlag_0_31 (uint32_t ExtiLine)

Function description

Clear ExtLine Flags for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21

Return values

- **None:**

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.

Reference Manual to LL API cross reference:

- PR1 PIFx LL_EXTI_ClearFlag_0_31

LL_EXTI_ClearFlag_32_63

Function name

__STATIC_INLINE void LL_EXTI_ClearFlag_32_63 (uint32_t ExtiLine)

Function description

Clear ExtLine Flags for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_51

Return values

- **None:**

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.

Reference Manual to LL API cross reference:

- PR2 PIFx LL_EXTI_ClearFlag_32_63

LL_EXTI_ClearFlag_64_95

Function name

__STATIC_INLINE void LL_EXTI_ClearFlag_64_95 (uint32_t ExtiLine)

Function description

Clear ExtLine Flags for Lines in range 64 to 95.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_82 (*)
  - LL_EXTI_LINE_84 (*)
  - LL_EXTI_LINE_85
  - LL_EXTI_LINE_86 (**)

  (*) value only defined in dual core devices. (**) value not defined in all devices.

Return values

- **None:**

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.

Reference Manual to LL API cross reference:

- PR3 PIFx LL_EXTI_ClearFlag_64_95

LL_C2_EXTI_IsActiveFlag_0_31

Function name

__STATIC_INLINE uint32_t LL_C2_EXTI_IsActiveFlag_0_31 (uint32_t ExtiLine)

Function description

Check if the ExtLine Flag is set or not for Lines in range 0 to 31 for cpu2.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_ALL_0_31

Return values

- **State:** of bit (1 or 0).

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.

Reference Manual to LL API cross reference:

- C2PR1 PIFx LL_C2_EXTI_IsActiveFlag_0_31

LL_C2_EXTI_IsActiveFlag_32_63

Function name

__STATIC_INLINE uint32_t LL_C2_EXTI_IsActiveFlag_32_63 (uint32_t ExtiLine)

Function description

Check if the ExtLine Flag is set or not for Lines in range 32 to 63 for cpu2.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_ALL_32_63

Return values

- **State:** of bit (1 or 0).

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.

Reference Manual to LL API cross reference:

- C2PR2 PIFx LL_C2_EXTI_IsActiveFlag_32_63

LL_C2_EXTI_IsActiveFlag_64_95

Function name

__STATIC_INLINE uint32_t LL_C2_EXTI_IsActiveFlag_64_95 (uint32_t ExtiLine)

Function description

Check if the ExtLine Flag is set or not for Lines in range 64 to 95 for cpu2.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_82
  - LL_EXTI_LINE_84
  - LL_EXTI_LINE_85
  - LL_EXTI_LINE_86
  - LL_EXTI_LINE_ALL_64_95

Return values

- **State:** of bit (1 or 0).

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.

Reference Manual to LL API cross reference:

- C2PR3 PIFx LL_C2_EXTI_IsActiveFlag_64_95

LL_C2_EXTI_ReadFlag_0_31

Function name

__STATIC_INLINE uint32_t LL_C2_EXTI_ReadFlag_0_31 (uint32_t ExtiLine)

Function description

Read ExtLine Combination Flag for Lines in range 0 to 31 for cpu2.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21

Return values

- **@note:** This bit is set when the selected edge event arrives on the interrupt

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.

Reference Manual to LL API cross reference:

- C2PR1 PIFx LL_C2_EXTI_ReadFlag_0_31

LL_C2_EXTI_ReadFlag_32_63

Function name

__STATIC_INLINE uint32_t LL_C2_EXTI_ReadFlag_32_63 (uint32_t ExtiLine)

Function description

Read ExtLine Combination Flag for Lines in range 32 to 63 for cpu2.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_51

Return values

- **@note:** This bit is set when the selected edge event arrives on the interrupt

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.

Reference Manual to LL API cross reference:

- C2PR2 PIFx LL_C2_EXTI_ReadFlag_32_63

LL_C2_EXTI_ReadFlag_64_95

Function name

__STATIC_INLINE uint32_t LL_C2_EXTI_ReadFlag_64_95 (uint32_t ExtiLine)

Function description

Read ExtLine Combination Flag for Lines in range 64 to 95 for cpu2.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_82
  - LL_EXTI_LINE_84
  - LL_EXTI_LINE_85
  - LL_EXTI_LINE_86

Return values

- **@note:** This bit is set when the selected edge event arrives on the interrupt

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.

Reference Manual to LL API cross reference:

- C2PR3 PIFx LL_C2_EXTI_ReadFlag_64_95

LL_C2_EXTI_ClearFlag_0_31

Function name

__STATIC_INLINE void LL_C2_EXTI_ClearFlag_0_31 (uint32_t ExtiLine)

Function description

Clear ExtLine Flags for Lines in range 0 to 31 for cpu2.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21

Return values

- **None:**

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.

Reference Manual to LL API cross reference:

- C2PR1 PIFx LL_C2_EXTI_ClearFlag_0_31

LL_C2_EXTI_ClearFlag_32_63

Function name

__STATIC_INLINE void LL_C2_EXTI_ClearFlag_32_63 (uint32_t ExtiLine)

Function description

Clear ExtLine Flags for Lines in range 32 to 63 for cpu2.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_51

Return values

- **None:**

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.

Reference Manual to LL API cross reference:

- C2PR2 PIFx LL_C2_EXTI_ClearFlag_32_63

LL_C2_EXTI_ClearFlag_64_95

Function name

__STATIC_INLINE void LL_C2_EXTI_ClearFlag_64_95 (uint32_t ExtiLine)

Function description

Clear ExtLine Flags for Lines in range 64 to 95 for cpu2.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_82
  - LL_EXTI_LINE_84
  - LL_EXTI_LINE_85
  - LL_EXTI_LINE_86

Return values

- **None:**

Notes

- This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.

Reference Manual to LL API cross reference:

- C2PR3 PIFx LL_C2_EXTI_ClearFlag_64_95

LL_D3_EXTI_EnablePendMask_0_31

Function name

__STATIC_INLINE void LL_D3_EXTI_EnablePendMask_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine D3 Pending Mask for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_25

Return values

- **None:**

Reference Manual to LL API cross reference:

- D3PMR1 MRx LL_D3_EXTI_EnablePendMask_0_31

LL_D3_EXTI_EnablePendMask_32_63

Function name

__STATIC_INLINE void LL_D3_EXTI_EnablePendMask_32_63 (uint32_t ExtiLine)

Function description

Enable ExtiLine D3 Pending Mask for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53

Return values

- **None:**

Reference Manual to LL API cross reference:

- D3PMR2 MRx LL_D3_EXTI_EnablePendMask_32_63

LL_D3_EXTI_DisablePendMask_0_31

Function name

__STATIC_INLINE void LL_D3_EXTI_DisablePendMask_0_31 (uint32_t ExtiLine)

Function description

Disable ExtiLine D3 Pending Mask for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_25

Return values

- **None:**

Reference Manual to LL API cross reference:

- D3PMR1 MRx LL_D3_EXTI_DisablePendMask_0_31

LL_D3_EXTI_DisablePendMask_32_63

Function name

__STATIC_INLINE void LL_D3_EXTI_DisablePendMask_32_63 (uint32_t ExtiLine)

Function description

Disable ExtiLine D3 Pending Mask for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53

Return values

- **None:**

Reference Manual to LL API cross reference:

- D3PMR2 MRx LL_D3_EXTI_DisablePendMask_32_63

LL_D3_EXTI_IsEnabledPendMask_0_31

Function name

__STATIC_INLINE uint32_t LL_D3_EXTI_IsEnabledPendMask_0_31 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine D3 Pending Mask is enabled for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_25

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- D3PMR1 MRx LL_D3_EXTI_IsEnabledPendMask_0_31

LL_D3_EXTI_IsEnabledPendMask_32_63

Function name

__STATIC_INLINE uint32_t LL_D3_EXTI_IsEnabledPendMask_32_63 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine D3 Pending Mask is enabled for Lines in range 32 to 63.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_41
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- D3PMR2 MRx LL_D3_EXTI_IsEnabledPendMask_32_63

LL_D3_EXTI_SetPendClearSel_0_15

Function name

__STATIC_INLINE void LL_D3_EXTI_SetPendClearSel_0_15 (uint32_t ExtiLine, uint32_t ClrSrc)

Function description

Set ExtLine D3 Domain Pend Clear Source selection for Lines in range 0 to 15.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
- **ClrSrc:** This parameter can be one of the following values:
  - LL_EXTI_D3_PEND_CLR_DMACH6
  - LL_EXTI_D3_PEND_CLR_DMACH7
  - LL_EXTI_D3_PEND_CLR_LPTIM4 (*)
  - LL_EXTI_D3_PEND_CLR_LPTIM5 (*)
  - LL_EXTI_D3_PEND_CLR_LPTIM2 (*)
  - LL_EXTI_D3_PEND_CLR_LPTIM3 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- D3PCR1L PCSx LL_D3_EXTI_SetPendClearSel_0_15

LL_D3_EXTI_SetPendClearSel_16_31

Function name

__STATIC_INLINE void LL_D3_EXTI_SetPendClearSel_16_31 (uint32_t ExtiLine, uint32_t ClrSrc)

Function description

Set ExtLine D3 Domain Pend Clear Source selection for Lines in range 16 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_25
- **ClrSrc:** This parameter can be one of the following values:
  - LL_EXTI_D3_PEND_CLR_DMACH6
  - LL_EXTI_D3_PEND_CLR_DMACH7
  - LL_EXTI_D3_PEND_CLR_LPTIM4 (*)
  - LL_EXTI_D3_PEND_CLR_LPTIM5 (*)
  - LL_EXTI_D3_PEND_CLR_LPTIM2 (*)
  - LL_EXTI_D3_PEND_CLR_LPTIM3 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- D3PCR1H PCSx LL_D3_EXTI_SetPendClearSel_16_31

LL_D3_EXTI_SetPendClearSel_32_47

Function name

__STATIC_INLINE void LL_D3_EXTI_SetPendClearSel_32_47 (uint32_t ExtiLine, uint32_t ClrSrc)

Function description

Set ExtLine D3 Domain Pend Clear Source selection for Lines in range 32 to 47.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_41
- **ClrSrc:** This parameter can be one of the following values:
  - LL_EXTI_D3_PEND_CLR_DMACH6
  - LL_EXTI_D3_PEND_CLR_DMACH7
  - LL_EXTI_D3_PEND_CLR_LPTIM4 (*)
  - LL_EXTI_D3_PEND_CLR_LPTIM5 (*)
  - LL_EXTI_D3_PEND_CLR_LPTIM2 (*)
  - LL_EXTI_D3_PEND_CLR_LPTIM3 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- D3PCR2L PCSx LL_D3_EXTI_SetPendClearSel_32_47

LL_D3_EXTI_SetPendClearSel_48_63

Function name

__STATIC_INLINE void LL_D3_EXTI_SetPendClearSel_48_63 (uint32_t ExtiLine, uint32_t ClrSrc)

Function description

Set ExtLine D3 Domain Pend Clear Source selection for Lines in range 48 to 63.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53
- **ClrSrc:** This parameter can be one of the following values:
  - LL_EXTI_D3_PEND_CLR_DMACH6
  - LL_EXTI_D3_PEND_CLR_DMACH7
  - LL_EXTI_D3_PEND_CLR_LPTIM4 (*)
  - LL_EXTI_D3_PEND_CLR_LPTIM5 (*)
  - LL_EXTI_D3_PEND_CLR_LPTIM2 (*)
  - LL_EXTI_D3_PEND_CLR_LPTIM3 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- D3PCR2H PCSx LL_D3_EXTI_SetPendClearSel_48_63

LL_D3_EXTI_GetPendClearSel_0_15

Function name

__STATIC_INLINE uint32_t LL_D3_EXTI_GetPendClearSel_0_15 (uint32_t ExtiLine)

Function description

Get ExtLine D3 Domain Pend Clear Source selection for Lines in range 0 to 15.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15

Return values

- **Returned:** value can be one of the following values:
  - LL_EXTI_D3_PEND_CLR_DMACH6
  - LL_EXTI_D3_PEND_CLR_DMACH7
  - LL_EXTI_D3_PEND_CLR_LPTIM4 (*)
  - LL_EXTI_D3_PEND_CLR_LPTIM5 (*)
  - LL_EXTI_D3_PEND_CLR_LPTIM2 (*)
  - LL_EXTI_D3_PEND_CLR_LPTIM3 (*)

  (*) value not defined in all devices.

Reference Manual to LL API cross reference:

- D3PCR1L PCSx LL_D3_EXTI_GetPendClearSel_0_15

LL_D3_EXTI_GetPendClearSel_16_31

Function name

__STATIC_INLINE uint32_t LL_D3_EXTI_GetPendClearSel_16_31 (uint32_t ExtiLine)

Function description

Get ExtLine D3 Domain Pend Clear Source selection for Lines in range 16 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_25

Return values

- **Returned:** value can be one of the following values:
  - LL_EXTI_D3_PEND_CLR_DMACH6
  - LL_EXTI_D3_PEND_CLR_DMACH7
  - LL_EXTI_D3_PEND_CLR_LPTIM4 (*)
  - LL_EXTI_D3_PEND_CLR_LPTIM5 (*)
  - LL_EXTI_D3_PEND_CLR_LPTIM2 (*)
  - LL_EXTI_D3_PEND_CLR_LPTIM3 (*)

  (*) value not defined in all devices.

Reference Manual to LL API cross reference:

- D3PCR1H PCSx LL_D3_EXTI_GetPendClearSel_16_31

LL_D3_EXTI_GetPendClearSel_32_47

Function name

__STATIC_INLINE uint32_t LL_D3_EXTI_GetPendClearSel_32_47 (uint32_t ExtiLine)

Function description

Get ExtLine D3 Domain Pend Clear Source selection for Lines in range 32 to 47.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_34
  - LL_EXTI_LINE_35
  - LL_EXTI_LINE_41

Return values

- **Returned:** value can be one of the following values:
  - LL_EXTI_D3_PEND_CLR_DMACH6
  - LL_EXTI_D3_PEND_CLR_DMACH7
  - LL_EXTI_D3_PEND_CLR_LPTIM4 (*)
  - LL_EXTI_D3_PEND_CLR_LPTIM5 (*)
  - LL_EXTI_D3_PEND_CLR_LPTIM2 (*)
  - LL_EXTI_D3_PEND_CLR_LPTIM3 (*)

  (*) value not defined in all devices.

Reference Manual to LL API cross reference:

- D3PCR2L PCSx LL_D3_EXTI_GetPendClearSel_32_47

LL_D3_EXTI_GetPendClearSel_48_63

Function name

__STATIC_INLINE uint32_t LL_D3_EXTI_GetPendClearSel_48_63 (uint32_t ExtiLine)

Function description

Get ExtLine D3 Domain Pend Clear Source selection for Lines in range 48 to 63.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_48
  - LL_EXTI_LINE_49
  - LL_EXTI_LINE_50
  - LL_EXTI_LINE_51
  - LL_EXTI_LINE_52
  - LL_EXTI_LINE_53

Return values

- **Returned:** value can be one of the following values:
  - LL_EXTI_D3_PEND_CLR_DMACH6
  - LL_EXTI_D3_PEND_CLR_DMACH7
  - LL_EXTI_D3_PEND_CLR_LPTIM4 (*)
  - LL_EXTI_D3_PEND_CLR_LPTIM5 (*)
  - LL_EXTI_D3_PEND_CLR_LPTIM2 (*)
  - LL_EXTI_D3_PEND_CLR_LPTIM3 (*)

  (*) value not defined in all devices.

Reference Manual to LL API cross reference:

- D3PCR2H PCSx LL_D3_EXTI_GetPendClearSel_48_63

LL_EXTI_Init

Function name

ErrorStatus LL_EXTI_Init (LL_EXTI_InitTypeDef * EXTI_InitStruct)

Function description

Initialize the EXTI registers according to the specified parameters in EXTI_InitStruct.

Parameters

- **EXTI_InitStruct:** pointer to a LL_EXTI_InitTypeDef structure.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: EXTI registers are initialized
  - ERROR: not applicable

LL_EXTI_DeInit

Function name

ErrorStatus LL_EXTI_DeInit (void)

Function description

De-initialize the EXTI registers to their default reset values.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: EXTI registers are de-initialized
  - ERROR: not applicable

LL_EXTI_StructInit

Function name

void LL_EXTI_StructInit (LL_EXTI_InitTypeDef * EXTI_InitStruct)

Function description

Set each LL_EXTI_InitTypeDef field to default value.

Parameters

- **EXTI_InitStruct:** Pointer to a LL_EXTI_InitTypeDef structure.

Return values

- **None:**

108.3 EXTI Firmware driver defines

The following section lists the various define and macros of the module.

108.3.1 EXTI

EXTI

D3 Pend Clear Source

LL_EXTI_D3_PEND_CLR_DMACH6

DMA ch6 event selected as D3 domain pendclear source

LL_EXTI_D3_PEND_CLR_DMACH7

DMA ch7 event selected as D3 domain pendclear source

LL_EXTI_D3_PEND_CLR_LPTIM4

LPTIM4 out selected as D3 domain pendclear source

LL_EXTI_D3_PEND_CLR_LPTIM5

LPTIM5 out selected as D3 domain pendclear source

LINE

LL_EXTI_LINE_0

Extended line 0

LL_EXTI_LINE_1

Extended line 1

LL_EXTI_LINE_2

Extended line 2

LL_EXTI_LINE_3

Extended line 3

LL_EXTI_LINE_4

Extended line 4

LL_EXTI_LINE_5

Extended line 5

LL_EXTI_LINE_6

Extended line 6

LL_EXTI_LINE_7

Extended line 7

LL_EXTI_LINE_8

Extended line 8

LL_EXTI_LINE_9

Extended line 9

LL_EXTI_LINE_10

Extended line 10

LL_EXTI_LINE_11

Extended line 11

LL_EXTI_LINE_12

Extended line 12

LL_EXTI_LINE_13

Extended line 13

LL_EXTI_LINE_14

Extended line 14

LL_EXTI_LINE_15

Extended line 15

LL_EXTI_LINE_16

Extended line 16

LL_EXTI_LINE_17

Extended line 17

LL_EXTI_LINE_18

Extended line 18

LL_EXTI_LINE_19

Extended line 19

LL_EXTI_LINE_20

Extended line 20

LL_EXTI_LINE_21

Extended line 21

LL_EXTI_LINE_22

Extended line 22

LL_EXTI_LINE_23

Extended line 23

LL_EXTI_LINE_24

Extended line 24

LL_EXTI_LINE_25

Extended line 25

LL_EXTI_LINE_26

Extended line 26

LL_EXTI_LINE_27

Extended line 27

LL_EXTI_LINE_28

Extended line 28

LL_EXTI_LINE_29

Extended line 29

LL_EXTI_LINE_30

Extended line 30

LL_EXTI_LINE_31

Extended line 31

LL_EXTI_LINE_ALL_0_31

All Extended line not reserved

LL_EXTI_LINE_32

Extended line 32

LL_EXTI_LINE_33

Extended line 33

LL_EXTI_LINE_34

Extended line 34

LL_EXTI_LINE_35

Extended line 35

LL_EXTI_LINE_36

Extended line 36

LL_EXTI_LINE_37

Extended line 37

LL_EXTI_LINE_38

Extended line 38

LL_EXTI_LINE_39

Extended line 39

LL_EXTI_LINE_40

Extended line 40

LL_EXTI_LINE_41

Extended line 41

LL_EXTI_LINE_42

Extended line 42

LL_EXTI_LINE_43

Extended line 43

LL_EXTI_LINE_44

Extended line 44

LL_EXTI_LINE_47

Extended line 47

LL_EXTI_LINE_48

Extended line 48

LL_EXTI_LINE_49

Extended line 49

LL_EXTI_LINE_50

Extended line 50

LL_EXTI_LINE_51

Extended line 51

LL_EXTI_LINE_52

Extended line 52

LL_EXTI_LINE_53

Extended line 53

LL_EXTI_LINE_54

Extended line 54

LL_EXTI_LINE_55

Extended line 55

LL_EXTI_LINE_56

Extended line 56

LL_EXTI_LINE_57

Extended line 57

LL_EXTI_LINE_58

Extended line 58

LL_EXTI_LINE_59

Extended line 59

LL_EXTI_LINE_60

Extended line 60

LL_EXTI_LINE_61

Extended line 61

LL_EXTI_LINE_62

Extended line 62

LL_EXTI_LINE_63

Extended line 63

LL_EXTI_LINE_ALL_32_63

All Extended line not reserved

LL_EXTI_LINE_64

Extended line 64

LL_EXTI_LINE_65

Extended line 65

LL_EXTI_LINE_66

Extended line 66

LL_EXTI_LINE_67

Extended line 67

LL_EXTI_LINE_68

Extended line 68

LL_EXTI_LINE_69

Extended line 69

LL_EXTI_LINE_70

Extended line 70

LL_EXTI_LINE_71

Extended line 71

LL_EXTI_LINE_72

Extended line 72

LL_EXTI_LINE_73

Extended line 73

LL_EXTI_LINE_74

Extended line 74

LL_EXTI_LINE_75

Extended line 75

LL_EXTI_LINE_76

Extended line 76

LL_EXTI_LINE_77

Extended line 77

LL_EXTI_LINE_78

Extended line 78

LL_EXTI_LINE_79

Extended line 79

LL_EXTI_LINE_80

Extended line 80

LL_EXTI_LINE_82

Extended line 82

LL_EXTI_LINE_84

Extended line 84

LL_EXTI_LINE_85

Extended line 85

LL_EXTI_LINE_86

Extended line 86

LL_EXTI_LINE_87

Extended line 87

LL_EXTI_LINE_ALL_64_95

All Extended line not reserved

LL_EXTI_LINE_ALL

All Extended line

LL_EXTI_LINE_NONE

None Extended line

Mode

LL_EXTI_MODE_IT

Cortex-M7 Interrupt Mode

LL_EXTI_MODE_EVENT

Cortex-M7 Event Mode

LL_EXTI_MODE_IT_EVENT

Cortex-M7 Interrupt & Event Mode

LL_EXTI_MODE_C1_IT

Cortex-M7 Interrupt Mode

LL_EXTI_MODE_C1_EVENT

Cortex-M7 Event Mode

LL_EXTI_MODE_C1_IT_EVENT

Cortex-M7 Interrupt & Event Mode

LL_EXTI_MODE_C2_IT

Cortex-M4 Interrupt Mode

LL_EXTI_MODE_C2_EVENT

Cortex-M4 Event Mode

LL_EXTI_MODE_C2_IT_EVENT

Cortex-M4 Interrupt & Event Mode

Edge Trigger

LL_EXTI_TRIGGER_NONE

No Trigger Mode

LL_EXTI_TRIGGER_RISING

Trigger Rising Mode

LL_EXTI_TRIGGER_FALLING

Trigger Falling Mode

LL_EXTI_TRIGGER_RISING_FALLING

Trigger Rising & Falling Mode

Common Write and read registers Macros

LL_EXTI_WriteReg

Description:

- Write a value in EXTI register.

Parameters:

- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_EXTI_ReadReg

Description:

- Read a value in EXTI register.

Parameters:

- __REG__: Register to be read

Return value:

- Register: value

109 LL FMAC Generic Driver

109.1 FMAC Firmware driver API description

The following section lists the various functions of the FMAC library.

109.1.1 Detailed description of functions

LL_FMAC_SetX1FullWatermark

Function name

__STATIC_INLINE void LL_FMAC_SetX1FullWatermark (FMAC_TypeDef * FMACx, uint32_t Watermark)

Function description

Configure X1 full watermark.

Parameters

- **FMACx:** FMAC instance
- **Watermark:** This parameter can be one of the following values:
  - LL_FMAC_WM_0_THRESHOLD_1
  - LL_FMAC_WM_1_THRESHOLD_2
  - LL_FMAC_WM_2_THRESHOLD_4
  - LL_FMAC_WM_3_THRESHOLD_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- X1BUFCFG FULL_WM LL_FMAC_SetX1FullWatermark

LL_FMAC_GetX1FullWatermark

Function name

__STATIC_INLINE uint32_t LL_FMAC_GetX1FullWatermark (FMAC_TypeDef * FMACx)

Function description

Return X1 full watermark.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** Returned value can be one of the following values:
  - LL_FMAC_WM_0_THRESHOLD_1
  - LL_FMAC_WM_1_THRESHOLD_2
  - LL_FMAC_WM_2_THRESHOLD_4
  - LL_FMAC_WM_3_THRESHOLD_8

Reference Manual to LL API cross reference:

- X1BUFCFG FULL_WM LL_FMAC_GetX1FullWatermark

LL_FMAC_SetX1BufferSize

Function name

__STATIC_INLINE void LL_FMAC_SetX1BufferSize (FMAC_TypeDef * FMACx, uint8_t BufferSize)

Function description

Configure X1 buffer size.

Parameters

- **FMACx:** FMAC instance
- **BufferSize:** Number of 16-bit words allocated to the input buffer (including the optional "headroom"). This parameter must be a number between Min_Data=0x01 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- X1BUFCFG X1_BUF_SIZE LL_FMAC_SetX1BufferSize

LL_FMAC_GetX1BufferSize

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetX1BufferSize (FMAC_TypeDef * FMACx)

Function description

Return X1 buffer size.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Number of 16-bit words allocated to the input buffer (including the optional "headroom") (value between Min_Data=0x01 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- X1BUFCFG X1_BUF_SIZE LL_FMAC_GetX1BufferSize

LL_FMAC_SetX1Base

Function name

__STATIC_INLINE void LL_FMAC_SetX1Base (FMAC_TypeDef * FMACx, uint8_t Base)

Function description

Configure X1 base.

Parameters

- **FMACx:** FMAC instance
- **Base:** Base address of the input buffer (X1) within the internal memory. This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- X1BUFCFG X1_BASE LL_FMAC_SetX1Base

LL_FMAC_GetX1Base

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetX1Base (FMAC_TypeDef * FMACx)

Function description

Return X1 base.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Base address of the input buffer (X1) within the internal memory (value between Min_Data=0x00 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- X1BUFCFG X1_BASE LL_FMAC_GetX1Base

LL_FMAC_SetX2BufferSize

Function name

__STATIC_INLINE void LL_FMAC_SetX2BufferSize (FMAC_TypeDef * FMACx, uint8_t BufferSize)

Function description

Configure X2 buffer size.

Parameters

- **FMACx:** FMAC instance
- **BufferSize:** Number of 16-bit words allocated to the coefficient buffer. This parameter must be a number between Min_Data=0x01 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- X2BUFCFG X2_BUF_SIZE LL_FMAC_SetX2BufferSize

LL_FMAC_GetX2BufferSize

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetX2BufferSize (FMAC_TypeDef * FMACx)

Function description

Return X2 buffer size.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Number of 16-bit words allocated to the coefficient buffer (value between Min_Data=0x01 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- X2BUFCFG X2_BUF_SIZE LL_FMAC_GetX2BufferSize

LL_FMAC_SetX2Base

Function name

__STATIC_INLINE void LL_FMAC_SetX2Base (FMAC_TypeDef * FMACx, uint8_t Base)

Function description

Configure X2 base.

Parameters

- **FMACx:** FMAC instance
- **Base:** Base address of the coefficient buffer (X2) within the internal memory. This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- X2BUFCFG X2_BASE LL_FMAC_SetX2Base

LL_FMAC_GetX2Base

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetX2Base (FMAC_TypeDef * FMACx)

Function description

Return X2 base.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Base address of the coefficient buffer (X2) within the internal memory (value between Min_Data=0x00 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- X2BUFCFG X2_BASE LL_FMAC_GetX2Base

LL_FMAC_SetYEmptyWatermark

Function name

__STATIC_INLINE void LL_FMAC_SetYEmptyWatermark (FMAC_TypeDef * FMACx, uint32_t Watermark)

Function description

Configure Y empty watermark.

Parameters

- **FMACx:** FMAC instance
- **Watermark:** This parameter can be one of the following values:
  - LL_FMAC_WM_0_THRESHOLD_1
  - LL_FMAC_WM_1_THRESHOLD_2
  - LL_FMAC_WM_2_THRESHOLD_4
  - LL_FMAC_WM_3_THRESHOLD_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- YBUFCFG EMPTY_WM LL_FMAC_SetYEmptyWatermark

LL_FMAC_GetYEmptyWatermark

Function name

__STATIC_INLINE uint32_t LL_FMAC_GetYEmptyWatermark (FMAC_TypeDef * FMACx)

Function description

Return Y empty watermark.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** Returned value can be one of the following values:
  - LL_FMAC_WM_0_THRESHOLD_1
  - LL_FMAC_WM_1_THRESHOLD_2
  - LL_FMAC_WM_2_THRESHOLD_4
  - LL_FMAC_WM_3_THRESHOLD_8

Reference Manual to LL API cross reference:

- YBUFCFG EMPTY_WM LL_FMAC_GetYEmptyWatermark

LL_FMAC_SetYBufferSize

Function name

__STATIC_INLINE void LL_FMAC_SetYBufferSize (FMAC_TypeDef * FMACx, uint8_t BufferSize)

Function description

Configure Y buffer size.

Parameters

- **FMACx:** FMAC instance
- **BufferSize:** Number of 16-bit words allocated to the output buffer (including the optional "headroom"). This parameter must be a number between Min_Data=0x01 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- YBUFCFG Y_BUF_SIZE LL_FMAC_SetYBufferSize

LL_FMAC_GetYBufferSize

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetYBufferSize (FMAC_TypeDef * FMACx)

Function description

Return Y buffer size.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Number of 16-bit words allocated to the output buffer (including the optional "headroom" - value between Min_Data=0x01 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- YBUFCFG Y_BUF_SIZE LL_FMAC_GetYBufferSize

LL_FMAC_SetYBase

Function name

__STATIC_INLINE void LL_FMAC_SetYBase (FMAC_TypeDef * FMACx, uint8_t Base)

Function description

Configure Y base.

Parameters

- **FMACx:** FMAC instance
- **Base:** Base address of the output buffer (Y) within the internal memory. This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- YBUFCFG Y_BASE LL_FMAC_SetYBase

LL_FMAC_GetYBase

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetYBase (FMAC_TypeDef * FMACx)

Function description

Return Y base.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Base address of the output buffer (Y) within the internal memory (value between Min_Data=0x00 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- YBUFCFG Y_BASE LL_FMAC_GetYBase

LL_FMAC_EnableStart

Function name

__STATIC_INLINE void LL_FMAC_EnableStart (FMAC_TypeDef * FMACx)

Function description

Start FMAC processing.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- PARAM START LL_FMAC_EnableStart

LL_FMAC_DisableStart

Function name

__STATIC_INLINE void LL_FMAC_DisableStart (FMAC_TypeDef * FMACx)

Function description

Stop FMAC processing.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- PARAM START LL_FMAC_DisableStart

LL_FMAC_IsEnabledStart

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledStart (FMAC_TypeDef * FMACx)

Function description

Check the state of FMAC processing.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- PARAM START LL_FMAC_IsEnabledStart

LL_FMAC_SetFunction

Function name

__STATIC_INLINE void LL_FMAC_SetFunction (FMAC_TypeDef * FMACx, uint32_t Function)

Function description

Configure function.

Parameters

- **FMACx:** FMAC instance
- **Function:** This parameter can be one of the following values:
  - LL_FMAC_FUNC_LOAD_X1
  - LL_FMAC_FUNC_LOAD_X2
  - LL_FMAC_FUNC_LOAD_Y
  - LL_FMAC_FUNC_CONVO_FIR
  - LL_FMAC_FUNC_IIR_DIRECT_FORM_1

Return values

- **None:**

Reference Manual to LL API cross reference:

- PARAM FUNC LL_FMAC_SetFunction

LL_FMAC_GetFunction

Function name

__STATIC_INLINE uint32_t LL_FMAC_GetFunction (FMAC_TypeDef * FMACx)

Function description

Return function.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** Returned value can be one of the following values:
  - LL_FMAC_FUNC_LOAD_X1
  - LL_FMAC_FUNC_LOAD_X2
  - LL_FMAC_FUNC_LOAD_Y
  - LL_FMAC_FUNC_CONVO_FIR
  - LL_FMAC_FUNC_IIR_DIRECT_FORM_1

Reference Manual to LL API cross reference:

- PARAM FUNC LL_FMAC_GetFunction

LL_FMAC_SetParamR

Function name

__STATIC_INLINE void LL_FMAC_SetParamR (FMAC_TypeDef * FMACx, uint8_t Param)

Function description

Configure input parameter R.

Parameters

- **FMACx:** FMAC instance
- **Param:** Parameter R (gain, etc.). This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PARAM R LL_FMAC_SetParamR

LL_FMAC_GetParamR

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetParamR (FMAC_TypeDef * FMACx)

Function description

Return input parameter R.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Parameter R (gain, etc.) (value between Min_Data=0x00 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- PARAM R LL_FMAC_GetParamR

LL_FMAC_SetParamQ

Function name

__STATIC_INLINE void LL_FMAC_SetParamQ (FMAC_TypeDef * FMACx, uint8_t Param)

Function description

Configure input parameter Q.

Parameters

- **FMACx:** FMAC instance
- **Param:** Parameter Q (vector length, etc.). This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PARAM Q LL_FMAC_SetParamQ

LL_FMAC_GetParamQ

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetParamQ (FMAC_TypeDef * FMACx)

Function description

Return input parameter Q.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Parameter Q (vector length, etc.) (value between Min_Data=0x00 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- PARAM Q LL_FMAC_GetParamQ

LL_FMAC_SetParamP

Function name

__STATIC_INLINE void LL_FMAC_SetParamP (FMAC_TypeDef * FMACx, uint8_t Param)

Function description

Configure input parameter P.

Parameters

- **FMACx:** FMAC instance
- **Param:** Parameter P (vector length, number of filter taps, etc.). This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PARAM P LL_FMAC_SetParamP

LL_FMAC_GetParamP

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetParamP (FMAC_TypeDef * FMACx)

Function description

Return input parameter P.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Parameter P (vector length, number of filter taps, etc.) (value between Min_Data=0x00 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- PARAM P LL_FMAC_GetParamP

LL_FMAC_EnableReset

Function name

__STATIC_INLINE void LL_FMAC_EnableReset (FMAC_TypeDef * FMACx)

Function description

Start the FMAC reset.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RESET LL_FMAC_EnableReset

LL_FMAC_IsEnabledReset

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledReset (FMAC_TypeDef * FMACx)

Function description

Check the state of the FMAC reset.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR RESET LL_FMAC_IsEnabledReset

LL_FMAC_EnableClipping

Function name

__STATIC_INLINE void LL_FMAC_EnableClipping (FMAC_TypeDef * FMACx)

Function description

Enable Clipping.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CLIPEN LL_FMAC_EnableClipping

LL_FMAC_DisableClipping

Function name

__STATIC_INLINE void LL_FMAC_DisableClipping (FMAC_TypeDef * FMACx)

Function description

Disable Clipping.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CLIPEN LL_FMAC_DisableClipping

LL_FMAC_IsEnabledClipping

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledClipping (FMAC_TypeDef * FMACx)

Function description

Check Clipping State.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CLIPEN LL_FMAC_IsEnabledClipping

LL_FMAC_EnableDMAReq_WRITE

Function name

__STATIC_INLINE void LL_FMAC_EnableDMAReq_WRITE (FMAC_TypeDef * FMACx)

Function description

Enable FMAC DMA write channel request.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAWEN LL_FMAC_EnableDMAReq_WRITE

LL_FMAC_DisableDMAReq_WRITE

Function name

__STATIC_INLINE void LL_FMAC_DisableDMAReq_WRITE (FMAC_TypeDef * FMACx)

Function description

Disable FMAC DMA write channel request.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAWEN LL_FMAC_DisableDMAReq_WRITE

LL_FMAC_IsEnabledDMAReq_WRITE

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledDMAReq_WRITE (FMAC_TypeDef * FMACx)

Function description

Check FMAC DMA write channel request state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR DMAWEN LL_FMAC_IsEnabledDMAReq_WRITE

LL_FMAC_EnableDMAReq_READ

Function name

__STATIC_INLINE void LL_FMAC_EnableDMAReq_READ (FMAC_TypeDef * FMACx)

Function description

Enable FMAC DMA read channel request.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAREN LL_FMAC_EnableDMAReq_READ

LL_FMAC_DisableDMAReq_READ

Function name

__STATIC_INLINE void LL_FMAC_DisableDMAReq_READ (FMAC_TypeDef * FMACx)

Function description

Disable FMAC DMA read channel request.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAREN LL_FMAC_DisableDMAReq_READ

LL_FMAC_IsEnabledDMAReq_READ

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledDMAReq_READ (FMAC_TypeDef * FMACx)

Function description

Check FMAC DMA read channel request state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR DMAREN LL_FMAC_IsEnabledDMAReq_READ

LL_FMAC_EnableIT_SAT

Function name

__STATIC_INLINE void LL_FMAC_EnableIT_SAT (FMAC_TypeDef * FMACx)

Function description

Enable FMAC saturation error interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SATIEN LL_FMAC_EnableIT_SAT

LL_FMAC_DisableIT_SAT

Function name

__STATIC_INLINE void LL_FMAC_DisableIT_SAT (FMAC_TypeDef * FMACx)

Function description

Disable FMAC saturation error interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SATIEN LL_FMAC_DisableIT_SAT

LL_FMAC_IsEnabledIT_SAT

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledIT_SAT (FMAC_TypeDef * FMACx)

Function description

Check FMAC saturation error interrupt state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR SATIEN LL_FMAC_IsEnabledIT_SAT

LL_FMAC_EnableIT_UNFL

Function name

__STATIC_INLINE void LL_FMAC_EnableIT_UNFL (FMAC_TypeDef * FMACx)

Function description

Enable FMAC underflow error interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR UNFLIEN LL_FMAC_EnableIT_UNFL

LL_FMAC_DisableIT_UNFL

Function name

__STATIC_INLINE void LL_FMAC_DisableIT_UNFL (FMAC_TypeDef * FMACx)

Function description

Disable FMAC underflow error interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR UNFLIEN LL_FMAC_DisableIT_UNFL

LL_FMAC_IsEnabledIT_UNFL

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledIT_UNFL (FMAC_TypeDef * FMACx)

Function description

Check FMAC underflow error interrupt state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR UNFLIEN LL_FMAC_IsEnabledIT_UNFL

LL_FMAC_EnableIT_OVFL

Function name

__STATIC_INLINE void LL_FMAC_EnableIT_OVFL (FMAC_TypeDef * FMACx)

Function description

Enable FMAC overflow error interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR OVFLIEN LL_FMAC_EnableIT_OVFL

LL_FMAC_DisableIT_OVFL

Function name

__STATIC_INLINE void LL_FMAC_DisableIT_OVFL (FMAC_TypeDef * FMACx)

Function description

Disable FMAC overflow error interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR OVFLIEN LL_FMAC_DisableIT_OVFL

LL_FMAC_IsEnabledIT_OVFL

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledIT_OVFL (FMAC_TypeDef * FMACx)

Function description

Check FMAC overflow error interrupt state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR OVFLIEN LL_FMAC_IsEnabledIT_OVFL

LL_FMAC_EnableIT_WR

Function name

__STATIC_INLINE void LL_FMAC_EnableIT_WR (FMAC_TypeDef * FMACx)

Function description

Enable FMAC write interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR WIEN LL_FMAC_EnableIT_WR

LL_FMAC_DisableIT_WR

Function name

__STATIC_INLINE void LL_FMAC_DisableIT_WR (FMAC_TypeDef * FMACx)

Function description

Disable FMAC write interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR WIEN LL_FMAC_DisableIT_WR

LL_FMAC_IsEnabledIT_WR

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledIT_WR (FMAC_TypeDef * FMACx)

Function description

Check FMAC write interrupt state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR WIEN LL_FMAC_IsEnabledIT_WR

LL_FMAC_EnableIT_RD

Function name

__STATIC_INLINE void LL_FMAC_EnableIT_RD (FMAC_TypeDef * FMACx)

Function description

Enable FMAC read interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RIEN LL_FMAC_EnableIT_RD

LL_FMAC_DisableIT_RD

Function name

__STATIC_INLINE void LL_FMAC_DisableIT_RD (FMAC_TypeDef * FMACx)

Function description

Disable FMAC read interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RIEN LL_FMAC_DisableIT_RD

LL_FMAC_IsEnabledIT_RD

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledIT_RD (FMAC_TypeDef * FMACx)

Function description

Check FMAC read interrupt state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR RIEN LL_FMAC_IsEnabledIT_RD

LL_FMAC_IsActiveFlag_SAT

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsActiveFlag_SAT (FMAC_TypeDef * FMACx)

Function description

Check FMAC saturation error flag state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR SAT LL_FMAC_IsActiveFlag_SAT

LL_FMAC_IsActiveFlag_UNFL

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsActiveFlag_UNFL (FMAC_TypeDef * FMACx)

Function description

Check FMAC underflow error flag state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR UNFL LL_FMAC_IsActiveFlag_UNFL

LL_FMAC_IsActiveFlag_OVFL

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsActiveFlag_OVFL (FMAC_TypeDef * FMACx)

Function description

Check FMAC overflow error flag state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR OVFL LL_FMAC_IsActiveFlag_OVFL

LL_FMAC_IsActiveFlag_X1FULL

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsActiveFlag_X1FULL (FMAC_TypeDef * FMACx)

Function description

Check FMAC X1 buffer full flag state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR X1FULL LL_FMAC_IsActiveFlag_X1FULL

LL_FMAC_IsActiveFlag_YEMPTY

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsActiveFlag_YEMPTY (FMAC_TypeDef * FMACx)

Function description

Check FMAC Y buffer empty flag state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR YEMPTY LL_FMAC_IsActiveFlag_YEMPTY

LL_FMAC_WriteData

Function name

__STATIC_INLINE void LL_FMAC_WriteData (FMAC_TypeDef * FMACx, uint16_t InData)

Function description

Write 16-bit input data for the FMAC processing.

Parameters

- **FMACx:** FMAC instance
- **InData:** 16-bit value to be provided as input data for FMAC processing. This parameter must be a number between Min_Data=0x0000 and Max_Data=0xFFFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WDATA WDATA LL_FMAC_WriteData

LL_FMAC_ReadData

Function name

__STATIC_INLINE uint16_t LL_FMAC_ReadData (FMAC_TypeDef * FMACx)

Function description

Return 16-bit output data of FMAC processing.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint16_t:** 16-bit output data of FMAC processing (value between Min_Data=0x0000 and Max_Data=0xFFFF).

Reference Manual to LL API cross reference:

- RDATA RDATA LL_FMAC_ReadData

LL_FMAC_ConfigX1

Function name

__STATIC_INLINE void LL_FMAC_ConfigX1 (FMAC_TypeDef * FMACx, uint32_t Watermark, uint8_t Base, uint8_t BufferSize)

Function description

Configure memory for X1 buffer.

Parameters

- **FMACx:** FMAC instance
- **Watermark:** This parameter can be one of the following values:
  - LL_FMAC_WM_0_THRESHOLD_1
  - LL_FMAC_WM_1_THRESHOLD_2
  - LL_FMAC_WM_2_THRESHOLD_4
  - LL_FMAC_WM_3_THRESHOLD_8
- **Base:** Base address of the input buffer (X1) within the internal memory. This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.
- **BufferSize:** Number of 16-bit words allocated to the input buffer (including the optional "headroom"). This parameter must be a number between Min_Data=0x01 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- X1BUFCFG FULL_WM LL_FMAC_ConfigX1
- X1BUFCFG X1_BASE LL_FMAC_ConfigX1
- X1BUFCFG X1_BUF_SIZE LL_FMAC_ConfigX1

LL_FMAC_ConfigX2

Function name

__STATIC_INLINE void LL_FMAC_ConfigX2 (FMAC_TypeDef * FMACx, uint8_t Base, uint8_t BufferSize)

Function description

Configure memory for X2 buffer.

Parameters

- **FMACx:** FMAC instance
- **Base:** Base address of the coefficient buffer (X2) within the internal memory. This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.
- **BufferSize:** Number of 16-bit words allocated to the coefficient buffer. This parameter must be a number between Min_Data=0x01 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- X2BUFCFG X2_BASE LL_FMAC_ConfigX2
- X2BUFCFG X2_BUF_SIZE LL_FMAC_ConfigX2

LL_FMAC_ConfigY

Function name

__STATIC_INLINE void LL_FMAC_ConfigY (FMAC_TypeDef * FMACx, uint32_t Watermark, uint8_t Base, uint8_t BufferSize)

Function description

Configure memory for Y buffer.

Parameters

- **FMACx:** FMAC instance
- **Watermark:** This parameter can be one of the following values:
  - LL_FMAC_WM_0_THRESHOLD_1
  - LL_FMAC_WM_1_THRESHOLD_2
  - LL_FMAC_WM_2_THRESHOLD_4
  - LL_FMAC_WM_3_THRESHOLD_8
- **Base:** Base address of the output buffer (Y) within the internal memory. This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.
- **BufferSize:** Number of 16-bit words allocated to the output buffer (including the optional "headroom"). This parameter must be a number between Min_Data=0x01 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- YBUFCFG EMPTY_WM LL_FMAC_ConfigY
- YBUFCFG Y_BASE LL_FMAC_ConfigY
- YBUFCFG Y_BUF_SIZE LL_FMAC_ConfigY

LL_FMAC_ConfigFunc

Function name

__STATIC_INLINE void LL_FMAC_ConfigFunc (FMAC_TypeDef * FMACx, uint8_t Start, uint32_t Function, uint8_t ParamP, uint8_t ParamQ, uint8_t ParamR)

Function description

Configure the FMAC processing.

Parameters

- **FMACx:** FMAC instance
- **Start:** This parameter can be one of the following values:
  - LL_FMAC_PROCESSING_STOP
  - LL_FMAC_PROCESSING_START
- **Function:** This parameter can be one of the following values:
  - LL_FMAC_FUNC_LOAD_X1
  - LL_FMAC_FUNC_LOAD_X2
  - LL_FMAC_FUNC_LOAD_Y
  - LL_FMAC_FUNC_CONVO_FIR
  - LL_FMAC_FUNC_IIR_DIRECT_FORM_1
- **ParamP:** Parameter P (vector length, number of filter taps, etc.). This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.
- **ParamQ:** Parameter Q (vector length, etc.). This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.
- **ParamR:** Parameter R (gain, etc.). This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PARAM START LL_FMAC_ConfigFunc
- PARAM FUNC LL_FMAC_ConfigFunc
- PARAM P LL_FMAC_ConfigFunc
- PARAM Q LL_FMAC_ConfigFunc
- PARAM R LL_FMAC_ConfigFunc

LL_FMAC_Init

Function name

ErrorStatus LL_FMAC_Init (FMAC_TypeDef * FMACx)

Function description

Initialize FMAC peripheral registers to their default reset values.

Parameters

- **FMACx:** FMAC Instance

Return values

- **ErrorStatus:** enumeration value:
  - SUCCESS: FMAC registers are initialized
  - ERROR: FMAC registers are not initialized

LL_FMAC_DeInit

Function name

ErrorStatus LL_FMAC_DeInit (FMAC_TypeDef * FMACx)

Function description

De-Initialize FMAC peripheral registers to their default reset values.

Parameters

- **FMACx:** FMAC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: FMAC registers are de-initialized
  - ERROR: FMAC registers are not de-initialized

109.2 FMAC Firmware driver defines

The following section lists the various define and macros of the module.

109.2.1 FMAC

FMAC

FMAC functions

LL_FMAC_FUNC_LOAD_X1

Load X1 buffer

LL_FMAC_FUNC_LOAD_X2

Load X2 buffer

LL_FMAC_FUNC_LOAD_Y

Load Y buffer

LL_FMAC_FUNC_CONVO_FIR

Convolution (FIR filter)

LL_FMAC_FUNC_IIR_DIRECT_FORM_1

IIR filter (direct form 1)

Get Flag Defines

LL_FMAC_SR_SAT

Saturation Error Flag (this helps in debugging a filter)

LL_FMAC_SR_UNFL

Underflow Error Flag

LL_FMAC_SR_OVFL

Overflow Error Flag

LL_FMAC_SR_X1FULL

X1 Buffer Full Flag

LL_FMAC_SR_YEMPTY

Y Buffer Empty Flag

IT Defines

LL_FMAC_CR_SATIEN

Saturation Error Interrupt Enable (this helps in debugging a filter)

LL_FMAC_CR_UNFLIEN

Underflow Error Interrupt Enable

LL_FMAC_CR_OVFLIEN

Overflow Error Interrupt Enable

LL_FMAC_CR_WIEN

Write Interrupt Enable

LL_FMAC_CR_RIEN

Read Interrupt Enable

FMAC processing

LL_FMAC_PROCESSING_STOP

Stop FMAC Processing

LL_FMAC_PROCESSING_START

Start FMAC Processing

FMAC watermarks

LL_FMAC_WM_0_THRESHOLD_1

Buffer full/empty flag set if there is less than 1 free/unread space.

LL_FMAC_WM_1_THRESHOLD_2

Buffer full/empty flag set if there are less than 2 free/unread spaces.

LL_FMAC_WM_2_THRESHOLD_4

Buffer full/empty flag set if there are less than 4 free/unread spaces.

LL_FMAC_WM_3_THRESHOLD_8

Buffer full/empty flag set if there are less than 8 free/empty spaces.

Common Write and read registers Macros

LL_FMAC_WriteReg

Description:

- Write a value in FMAC register.

Parameters:

- __INSTANCE__: FMAC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_FMAC_ReadReg

Description:

- Read a value in FMAC register.

Parameters:

- __INSTANCE__: FMAC Instance
- __REG__: Register to be read

Return value:

- Register: value

110 LL GPIO Generic Driver

110.1 GPIO Firmware driver registers structures

110.1.1 LL_GPIO_InitTypeDef

LL_GPIO_InitTypeDef is defined in the stm32h7xx_ll_gpio.h

Data Fields

- ***uint32_t Pin***
- ***uint32_t Mode***
- ***uint32_t Speed***
- ***uint32_t OutputType***
- ***uint32_t Pull***
- ***uint32_t Alternate***

Field Documentation

- ***uint32_t LL_GPIO_InitTypeDef::Pin***

  Specifies the GPIO pins to be configured. This parameter can be any value of ***GPIO_LL_EC_PIN***
- ***uint32_t LL_GPIO_InitTypeDef::Mode***

  Specifies the operating mode for the selected pins. This parameter can be a value of ***GPIO_LL_EC_MODE***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetPinMode()**.
- ***uint32_t LL_GPIO_InitTypeDef::Speed***

  Specifies the speed for the selected pins. This parameter can be a value of ***GPIO_LL_EC_SPEED***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetPinSpeed()**.
- ***uint32_t LL_GPIO_InitTypeDef::OutputType***

  Specifies the operating output type for the selected pins. This parameter can be a value of ***GPIO_LL_EC_OUTPUT***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetPinOutputType()**.
- ***uint32_t LL_GPIO_InitTypeDef::Pull***

  Specifies the operating Pull-up/Pull down for the selected pins. This parameter can be a value of ***GPIO_LL_EC_PULL***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetPinPull()**.
- ***uint32_t LL_GPIO_InitTypeDef::Alternate***

  Specifies the Peripheral to be connected to the selected pins. This parameter can be a value of ***GPIO_LL_EC_AF***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetAFPin_0_7()** and **LL_GPIO_SetAFPin_8_15()**.

110.2 GPIO Firmware driver API description

The following section lists the various functions of the GPIO library.

110.2.1 Detailed description of functions

LL_GPIO_SetPinMode

Function name

__STATIC_INLINE void LL_GPIO_SetPinMode (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Mode)

Function description

Configure gpio mode for a dedicated pin on dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
- **Mode:** This parameter can be one of the following values:
  - LL_GPIO_MODE_INPUT
  - LL_GPIO_MODE_OUTPUT
  - LL_GPIO_MODE_ALTERNATE
  - LL_GPIO_MODE_ANALOG

Return values

- **None:**

Notes

- I/O mode can be Input mode, General purpose output, Alternate function mode or Analog.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- MODER MODEy LL_GPIO_SetPinMode

LL_GPIO_GetPinMode

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetPinMode (GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio mode for a dedicated pin on dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_MODE_INPUT
  - LL_GPIO_MODE_OUTPUT
  - LL_GPIO_MODE_ALTERNATE
  - LL_GPIO_MODE_ANALOG

Notes

- I/O mode can be Input mode, General purpose output, Alternate function mode or Analog.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- MODER MODEy LL_GPIO_GetPinMode

LL_GPIO_SetPinOutputType

Function name

__STATIC_INLINE void LL_GPIO_SetPinOutputType (GPIO_TypeDef * GPIOx, uint32_t PinMask, uint32_t OutputType)

Function description

Configure gpio output type for several pins on dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL
- **OutputType:** This parameter can be one of the following values:
  - LL_GPIO_OUTPUT_PUSHPULL
  - LL_GPIO_OUTPUT_OPENDRAIN

Return values

- **None:**

Notes

- Output type as to be set when gpio pin is in output or alternate modes. Possible type are Push-pull or Open-drain.

Reference Manual to LL API cross reference:

- OTYPER OTy LL_GPIO_SetPinOutputType

LL_GPIO_GetPinOutputType

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetPinOutputType (GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio output type for several pins on dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_OUTPUT_PUSHPULL
  - LL_GPIO_OUTPUT_OPENDRAIN

Notes

- Output type as to be set when gpio pin is in output or alternate modes. Possible type are Push-pull or Open-drain.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- OTYPER OTy LL_GPIO_GetPinOutputType

LL_GPIO_SetPinSpeed

Function name

__STATIC_INLINE void LL_GPIO_SetPinSpeed (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Speed)

Function description

Configure gpio speed for a dedicated pin on dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
- **Speed:** This parameter can be one of the following values:
  - LL_GPIO_SPEED_FREQ_LOW
  - LL_GPIO_SPEED_FREQ_MEDIUM
  - LL_GPIO_SPEED_FREQ_HIGH
  - LL_GPIO_SPEED_FREQ_VERY_HIGH

Return values

- **None:**

Notes

- I/O speed can be Low, Medium, Fast or High speed.
- Warning: only one pin can be passed as parameter.
- Refer to datasheet for frequency specifications and the power supply and load conditions for each speed.

Reference Manual to LL API cross reference:

- OSPEEDR OSPEEDy LL_GPIO_SetPinSpeed

LL_GPIO_GetPinSpeed

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetPinSpeed (GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio speed for a dedicated pin on dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_SPEED_FREQ_LOW
  - LL_GPIO_SPEED_FREQ_MEDIUM
  - LL_GPIO_SPEED_FREQ_HIGH
  - LL_GPIO_SPEED_FREQ_VERY_HIGH

Notes

- I/O speed can be Low, Medium, Fast or High speed.
- Warning: only one pin can be passed as parameter.
- Refer to datasheet for frequency specifications and the power supply and load conditions for each speed.

Reference Manual to LL API cross reference:

- OSPEEDR OSPEEDy LL_GPIO_GetPinSpeed

LL_GPIO_SetPinPull

Function name

__STATIC_INLINE void LL_GPIO_SetPinPull (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Pull)

Function description

Configure gpio pull-up or pull-down for a dedicated pin on a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
- **Pull:** This parameter can be one of the following values:
  - LL_GPIO_PULL_NO
  - LL_GPIO_PULL_UP
  - LL_GPIO_PULL_DOWN

Return values

- **None:**

Notes

- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- PUPDR PUPDy LL_GPIO_SetPinPull

LL_GPIO_GetPinPull

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetPinPull (GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio pull-up or pull-down for a dedicated pin on a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_PULL_NO
  - LL_GPIO_PULL_UP
  - LL_GPIO_PULL_DOWN

Notes

- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- PUPDR PUPDy LL_GPIO_GetPinPull

LL_GPIO_SetAFPin_0_7

Function name

__STATIC_INLINE void LL_GPIO_SetAFPin_0_7 (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Alternate)

Function description

Configure gpio alternate function of a dedicated pin from 0 to 7 for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
- **Alternate:** This parameter can be one of the following values:
  - LL_GPIO_AF_0
  - LL_GPIO_AF_1
  - LL_GPIO_AF_2
  - LL_GPIO_AF_3
  - LL_GPIO_AF_4
  - LL_GPIO_AF_5
  - LL_GPIO_AF_6
  - LL_GPIO_AF_7
  - LL_GPIO_AF_8
  - LL_GPIO_AF_9
  - LL_GPIO_AF_10
  - LL_GPIO_AF_11
  - LL_GPIO_AF_12
  - LL_GPIO_AF_13
  - LL_GPIO_AF_14
  - LL_GPIO_AF_15

Return values

- **None:**

Notes

- Possible values are from AF0 to AF15 depending on target.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- AFRL AFSELy LL_GPIO_SetAFPin_0_7

LL_GPIO_GetAFPin_0_7

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetAFPin_0_7 (GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio alternate function of a dedicated pin from 0 to 7 for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_AF_0
  - LL_GPIO_AF_1
  - LL_GPIO_AF_2
  - LL_GPIO_AF_3
  - LL_GPIO_AF_4
  - LL_GPIO_AF_5
  - LL_GPIO_AF_6
  - LL_GPIO_AF_7
  - LL_GPIO_AF_8
  - LL_GPIO_AF_9
  - LL_GPIO_AF_10
  - LL_GPIO_AF_11
  - LL_GPIO_AF_12
  - LL_GPIO_AF_13
  - LL_GPIO_AF_14
  - LL_GPIO_AF_15

Reference Manual to LL API cross reference:

- AFRL AFSELy LL_GPIO_GetAFPin_0_7

LL_GPIO_SetAFPin_8_15

Function name

__STATIC_INLINE void LL_GPIO_SetAFPin_8_15 (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Alternate)

Function description

Configure gpio alternate function of a dedicated pin from 8 to 15 for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
- **Alternate:** This parameter can be one of the following values:
  - LL_GPIO_AF_0
  - LL_GPIO_AF_1
  - LL_GPIO_AF_2
  - LL_GPIO_AF_3
  - LL_GPIO_AF_4
  - LL_GPIO_AF_5
  - LL_GPIO_AF_6
  - LL_GPIO_AF_7
  - LL_GPIO_AF_8
  - LL_GPIO_AF_9
  - LL_GPIO_AF_10
  - LL_GPIO_AF_11
  - LL_GPIO_AF_12
  - LL_GPIO_AF_13
  - LL_GPIO_AF_14
  - LL_GPIO_AF_15

Return values

- **None:**

Notes

- Possible values are from AF0 to AF15 depending on target.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- AFRH AFSELy LL_GPIO_SetAFPin_8_15

LL_GPIO_GetAFPin_8_15

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetAFPin_8_15 (GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio alternate function of a dedicated pin from 8 to 15 for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_AF_0
  - LL_GPIO_AF_1
  - LL_GPIO_AF_2
  - LL_GPIO_AF_3
  - LL_GPIO_AF_4
  - LL_GPIO_AF_5
  - LL_GPIO_AF_6
  - LL_GPIO_AF_7
  - LL_GPIO_AF_8
  - LL_GPIO_AF_9
  - LL_GPIO_AF_10
  - LL_GPIO_AF_11
  - LL_GPIO_AF_12
  - LL_GPIO_AF_13
  - LL_GPIO_AF_14
  - LL_GPIO_AF_15

Notes

- Possible values are from AF0 to AF15 depending on target.

Reference Manual to LL API cross reference:

- AFRH AFSELy LL_GPIO_GetAFPin_8_15

LL_GPIO_LockPin

Function name

__STATIC_INLINE void LL_GPIO_LockPin (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Lock configuration of several pins for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **None:**

Notes

- When the lock sequence has been applied on a port bit, the value of this port bit can no longer be modified until the next reset.
- Each lock bit freezes a specific configuration register (control and alternate function registers).

Reference Manual to LL API cross reference:

- LCKR LCKK LL_GPIO_LockPin

LL_GPIO_IsPinLocked

Function name

__STATIC_INLINE uint32_t LL_GPIO_IsPinLocked (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Return 1 if all pins passed as parameter, of a dedicated port, are locked.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LCKR LCKy LL_GPIO_IsPinLocked

LL_GPIO_IsAnyPinLocked

Function name

__STATIC_INLINE uint32_t LL_GPIO_IsAnyPinLocked (GPIO_TypeDef * GPIOx)

Function description

Return 1 if one of the pin of a dedicated port is locked.

Parameters

- **GPIOx:** GPIO Port

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LCKR LCKK LL_GPIO_IsAnyPinLocked

LL_GPIO_ReadInputPort

Function name

__STATIC_INLINE uint32_t LL_GPIO_ReadInputPort (GPIO_TypeDef * GPIOx)

Function description

Return full input data register value for a dedicated port.

Parameters

- **GPIOx:** GPIO Port

Return values

- **Input:** data register value of port

Reference Manual to LL API cross reference:

- IDR IDy LL_GPIO_ReadInputPort

LL_GPIO_IsInputPinSet

Function name

__STATIC_INLINE uint32_t LL_GPIO_IsInputPinSet (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Return if input data level for several pins of dedicated port is high or low.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IDR IDy LL_GPIO_IsInputPinSet

LL_GPIO_WriteOutputPort

Function name

__STATIC_INLINE void LL_GPIO_WriteOutputPort (GPIO_TypeDef * GPIOx, uint32_t PortValue)

Function description

Write output data register for the port.

Parameters

- **GPIOx:** GPIO Port
- **PortValue:** Level value for each pin of the port

Return values

- **None:**

Reference Manual to LL API cross reference:

- ODR ODy LL_GPIO_WriteOutputPort

LL_GPIO_ReadOutputPort

Function name

__STATIC_INLINE uint32_t LL_GPIO_ReadOutputPort (GPIO_TypeDef * GPIOx)

Function description

Return full output data register value for a dedicated port.

Parameters

- **GPIOx:** GPIO Port

Return values

- **Output:** data register value of port

Reference Manual to LL API cross reference:

- ODR ODy LL_GPIO_ReadOutputPort

LL_GPIO_IsOutputPinSet

Function name

__STATIC_INLINE uint32_t LL_GPIO_IsOutputPinSet (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Return if input data level for several pins of dedicated port is high or low.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ODR ODy LL_GPIO_IsOutputPinSet

LL_GPIO_SetOutputPin

Function name

__STATIC_INLINE void LL_GPIO_SetOutputPin (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Set several pins to high level on dedicated gpio port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- BSRR BSy LL_GPIO_SetOutputPin

LL_GPIO_ResetOutputPin

Function name

__STATIC_INLINE void LL_GPIO_ResetOutputPin (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Set several pins to low level on dedicated gpio port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- BSRR BRy LL_GPIO_ResetOutputPin

LL_GPIO_TogglePin

Function name

__STATIC_INLINE void LL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Toggle data value for several pin of dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- ODR ODy LL_GPIO_TogglePin

LL_GPIO_DeInit

Function name

ErrorStatus LL_GPIO_DeInit (GPIO_TypeDef * GPIOx)

Function description

De-initialize GPIO registers (Registers restored to their default values).

Parameters

- **GPIOx:** GPIO Port

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: GPIO registers are de-initialized
  - ERROR: Wrong GPIO Port

LL_GPIO_Init

Function name

ErrorStatus LL_GPIO_Init (GPIO_TypeDef * GPIOx, LL_GPIO_InitTypeDef * GPIO_InitStruct)

Function description

Initialize GPIO registers according to the specified parameters in GPIO_InitStruct.

Parameters

- **GPIOx:** GPIO Port
- **GPIO_InitStruct:** pointer to a LL_GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: GPIO registers are initialized according to GPIO_InitStruct content
  - ERROR: Not applicable

LL_GPIO_StructInit

Function name

void LL_GPIO_StructInit (LL_GPIO_InitTypeDef * GPIO_InitStruct)

Function description

Set each LL_GPIO_InitTypeDef field to default value.

Parameters

- **GPIO_InitStruct:** pointer to a LL_GPIO_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

110.3 GPIO Firmware driver defines

The following section lists the various define and macros of the module.

110.3.1 GPIO

GPIO

Alternate Function

LL_GPIO_AF_0

Select alternate function 0

LL_GPIO_AF_1

Select alternate function 1

LL_GPIO_AF_2

Select alternate function 2

LL_GPIO_AF_3

Select alternate function 3

LL_GPIO_AF_4

Select alternate function 4

LL_GPIO_AF_5

Select alternate function 5

LL_GPIO_AF_6

Select alternate function 6

LL_GPIO_AF_7

Select alternate function 7

LL_GPIO_AF_8

Select alternate function 8

LL_GPIO_AF_9

Select alternate function 9

LL_GPIO_AF_10

Select alternate function 10

LL_GPIO_AF_11

Select alternate function 11

LL_GPIO_AF_12

Select alternate function 12

LL_GPIO_AF_13

Select alternate function 13

LL_GPIO_AF_14

Select alternate function 14

LL_GPIO_AF_15

Select alternate function 15

Mode

LL_GPIO_MODE_INPUT

Select input mode

LL_GPIO_MODE_OUTPUT

Select output mode

LL_GPIO_MODE_ALTERNATE

Select alternate function mode

LL_GPIO_MODE_ANALOG

Select analog mode

Output Type

LL_GPIO_OUTPUT_PUSHPULL

Select push-pull as output type

LL_GPIO_OUTPUT_OPENDRAIN

Select open-drain as output type

PIN

LL_GPIO_PIN_0

Select pin 0

LL_GPIO_PIN_1

Select pin 1

LL_GPIO_PIN_2

Select pin 2

LL_GPIO_PIN_3

Select pin 3

LL_GPIO_PIN_4

Select pin 4

LL_GPIO_PIN_5

Select pin 5

LL_GPIO_PIN_6

Select pin 6

LL_GPIO_PIN_7

Select pin 7

LL_GPIO_PIN_8

Select pin 8

LL_GPIO_PIN_9

Select pin 9

LL_GPIO_PIN_10

Select pin 10

LL_GPIO_PIN_11

Select pin 11

LL_GPIO_PIN_12

Select pin 12

LL_GPIO_PIN_13

Select pin 13

LL_GPIO_PIN_14

Select pin 14

LL_GPIO_PIN_15

Select pin 15

LL_GPIO_PIN_ALL

Select all pins

Pull Up Pull Down

LL_GPIO_PULL_NO

Select I/O no pull

LL_GPIO_PULL_UP

Select I/O pull up

LL_GPIO_PULL_DOWN

Select I/O pull down

Output Speed

LL_GPIO_SPEED_FREQ_LOW

Select I/O low output speed

LL_GPIO_SPEED_FREQ_MEDIUM

Select I/O medium output speed

LL_GPIO_SPEED_FREQ_HIGH

Select I/O fast output speed

LL_GPIO_SPEED_FREQ_VERY_HIGH

Select I/O high output speed

Common Write and read registers Macros

LL_GPIO_WriteReg

Description:

- Write a value in GPIO register.

Parameters:

- __INSTANCE__: GPIO Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_GPIO_ReadReg

Description:

- Read a value in GPIO register.

Parameters:

- __INSTANCE__: GPIO Instance
- __REG__: Register to be read

Return value:

- Register: value

GPIO Exported Constants

LL_GPIO_SPEED_LOW

LL_GPIO_SPEED_MEDIUM

LL_GPIO_SPEED_FAST

LL_GPIO_SPEED_HIGH

111 LL HRTIM Generic Driver

111.1 HRTIM Firmware driver API description

The following section lists the various functions of the HRTIM library.

111.1.1 Detailed description of functions

LL_HRTIM_SetSyncInSrc

Function name

__STATIC_INLINE void LL_HRTIM_SetSyncInSrc (HRTIM_TypeDef * HRTIMx, uint32_t SyncInSrc)

Function description

Select the HRTIM synchronization input source.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **SyncInSrc:** This parameter can be one of the following values:
  - LL_HRTIM_SYNCIN_SRC_NONE
  - LL_HRTIM_SYNCIN_SRC_TIM_EVENT
  - LL_HRTIM_SYNCIN_SRC_EXTERNAL_EVENT

Return values

- **None:**

Notes

- This function must not be called when the concerned timer(s) is (are) enabled .

Reference Manual to LL API cross reference:

- MCR SYNCIN LL_HRTIM_SetSyncInSrc

LL_HRTIM_GetSyncInSrc

Function name

__STATIC_INLINE uint32_t LL_HRTIM_GetSyncInSrc (const HRTIM_TypeDef * HRTIMx)

Function description

Get actual HRTIM synchronization input source.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **SyncInSrc:** Returned value can be one of the following values:
  - LL_HRTIM_SYNCIN_SRC_NONE
  - LL_HRTIM_SYNCIN_SRC_TIM_EVENT
  - LL_HRTIM_SYNCIN_SRC_EXTERNAL_EVENT

Reference Manual to LL API cross reference:

- MCR SYNCIN LL_HRTIM_SetSyncInSrc

LL_HRTIM_ConfigSyncOut

Function name

__STATIC_INLINE void LL_HRTIM_ConfigSyncOut (HRTIM_TypeDef * HRTIMx, uint32_t Config, uint32_t Src)

Function description

Configure the HRTIM synchronization output.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Config:** This parameter can be one of the following values:
  - LL_HRTIM_SYNCOUT_DISABLED
  - LL_HRTIM_SYNCOUT_POSITIVE_PULSE
  - LL_HRTIM_SYNCOUT_NEGATIVE_PULSE
- **Src:** This parameter can be one of the following values:
  - LL_HRTIM_SYNCOUT_SRC_MASTER_START
  - LL_HRTIM_SYNCOUT_SRC_MASTER_CMP1
  - LL_HRTIM_SYNCOUT_SRC_TIMA_START
  - LL_HRTIM_SYNCOUT_SRC_TIMA_CMP1

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR SYNCSRC LL_HRTIM_ConfigSyncOut
- MCR SYNCOUT LL_HRTIM_ConfigSyncOut

LL_HRTIM_SetSyncOutConfig

Function name

__STATIC_INLINE void LL_HRTIM_SetSyncOutConfig (HRTIM_TypeDef * HRTIMx, uint32_t SyncOutConfig)

Function description

Set the routing and conditioning of the synchronization output event.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **SyncOutConfig:** This parameter can be one of the following values:
  - LL_HRTIM_SYNCOUT_DISABLED
  - LL_HRTIM_SYNCOUT_POSITIVE_PULSE
  - LL_HRTIM_SYNCOUT_NEGATIVE_PULSE

Return values

- **None:**

Notes

- This function can be called only when the master timer is enabled.

Reference Manual to LL API cross reference:

- MCR SYNCOUT LL_HRTIM_SetSyncOutConfig

LL_HRTIM_GetSyncOutConfig

Function name

__STATIC_INLINE uint32_t LL_HRTIM_GetSyncOutConfig (const HRTIM_TypeDef * HRTIMx)

Function description

Get actual routing and conditioning of the synchronization output event.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **SyncOutConfig:** Returned value can be one of the following values:
  - LL_HRTIM_SYNCOUT_DISABLED
  - LL_HRTIM_SYNCOUT_POSITIVE_PULSE
  - LL_HRTIM_SYNCOUT_NEGATIVE_PULSE

Reference Manual to LL API cross reference:

- MCR SYNCOUT LL_HRTIM_GetSyncOutConfig

LL_HRTIM_SetSyncOutSrc

Function name

__STATIC_INLINE void LL_HRTIM_SetSyncOutSrc (HRTIM_TypeDef * HRTIMx, uint32_t SyncOutSrc)

Function description

Set the source and event to be sent on the HRTIM synchronization output.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **SyncOutSrc:** This parameter can be one of the following values:
  - LL_HRTIM_SYNCOUT_SRC_MASTER_START
  - LL_HRTIM_SYNCOUT_SRC_MASTER_CMP1
  - LL_HRTIM_SYNCOUT_SRC_TIMA_START
  - LL_HRTIM_SYNCOUT_SRC_TIMA_CMP1

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR SYNCSRC LL_HRTIM_SetSyncOutSrc

LL_HRTIM_GetSyncOutSrc

Function name

__STATIC_INLINE uint32_t LL_HRTIM_GetSyncOutSrc (const HRTIM_TypeDef * HRTIMx)

Function description

Get actual source and event sent on the HRTIM synchronization output.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **SyncOutSrc:** Returned value can be one of the following values:
  - LL_HRTIM_SYNCOUT_SRC_MASTER_START
  - LL_HRTIM_SYNCOUT_SRC_MASTER_CMP1
  - LL_HRTIM_SYNCOUT_SRC_TIMA_START
  - LL_HRTIM_SYNCOUT_SRC_TIMA_CMP1

Reference Manual to LL API cross reference:

- MCR SYNCSRC LL_HRTIM_GetSyncOutSrc

LL_HRTIM_SuspendUpdate

Function name

__STATIC_INLINE void LL_HRTIM_SuspendUpdate (HRTIM_TypeDef * HRTIMx, uint32_t Timers)

Function description

Disable (temporarily) update event generation.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timers:** This parameter can be a combination of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Notes

- Allow to temporarily disable the transfer from preload to active registers, whatever the selected update event. This allows to modify several registers in multiple timers.

Reference Manual to LL API cross reference:

- CR1 MUDIS LL_HRTIM_SuspendUpdate
- CR1 TAUDIS LL_HRTIM_SuspendUpdate
- CR1 TBUDIS LL_HRTIM_SuspendUpdate
- CR1 TCUDIS LL_HRTIM_SuspendUpdate
- CR1 TDUDIS LL_HRTIM_SuspendUpdate
- CR1 TEUDIS LL_HRTIM_SuspendUpdate

LL_HRTIM_ResumeUpdate

Function name

__STATIC_INLINE void LL_HRTIM_ResumeUpdate (HRTIM_TypeDef * HRTIMx, uint32_t Timers)

Function description

Enable update event generation.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timers:** This parameter can be a combination of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Notes

- The regular update event takes place.

Reference Manual to LL API cross reference:

- CR1 MUDIS LL_HRTIM_ResumeUpdate
- CR1 TAUDIS LL_HRTIM_ResumeUpdate
- CR1 TBUDIS LL_HRTIM_ResumeUpdate
- CR1 TCUDIS LL_HRTIM_ResumeUpdate
- CR1 TDUDIS LL_HRTIM_ResumeUpdate
- CR1 TEUDIS LL_HRTIM_ResumeUpdate

LL_HRTIM_ForceUpdate

Function name

__STATIC_INLINE void LL_HRTIM_ForceUpdate (HRTIM_TypeDef * HRTIMx, uint32_t Timers)

Function description

Force an immediate transfer from the preload to the active register .

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timers:** This parameter can be a combination of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Notes

- Any pending update request is cancelled.

Reference Manual to LL API cross reference:

- CR2 MSWU LL_HRTIM_ForceUpdate
- CR2 TASWU LL_HRTIM_ForceUpdate
- CR2 TBSWU LL_HRTIM_ForceUpdate
- CR2 TCSWU LL_HRTIM_ForceUpdate
- CR2 TDSWU LL_HRTIM_ForceUpdate
- CR2 TESWU LL_HRTIM_ForceUpdate

LL_HRTIM_CounterReset

Function name

__STATIC_INLINE void LL_HRTIM_CounterReset (HRTIM_TypeDef * HRTIMx, uint32_t Timers)

Function description

Reset the HRTIM timer(s) counter.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timers:** This parameter can be a combination of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 MRST LL_HRTIM_CounterReset
- CR2 TARST LL_HRTIM_CounterReset
- CR2 TBRST LL_HRTIM_CounterReset
- CR2 TCRST LL_HRTIM_CounterReset
- CR2 TDRST LL_HRTIM_CounterReset
- CR2 TERST LL_HRTIM_CounterReset

LL_HRTIM_EnableOutput

Function name

__STATIC_INLINE void LL_HRTIM_EnableOutput (HRTIM_TypeDef * HRTIMx, uint32_t Outputs)

Function description

Enable the HRTIM timer(s) output(s) .

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Outputs:** This parameter can be a combination of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2

Return values

- **None:**

Reference Manual to LL API cross reference:

- OENR TA1OEN LL_HRTIM_EnableOutput
- OENR TA2OEN LL_HRTIM_EnableOutput
- OENR TB1OEN LL_HRTIM_EnableOutput
- OENR TB2OEN LL_HRTIM_EnableOutput
- OENR TC1OEN LL_HRTIM_EnableOutput
- OENR TC2OEN LL_HRTIM_EnableOutput
- OENR TD1OEN LL_HRTIM_EnableOutput
- OENR TD2OEN LL_HRTIM_EnableOutput
- OENR TE1OEN LL_HRTIM_EnableOutput
- OENR TE2OEN LL_HRTIM_EnableOutput

LL_HRTIM_DisableOutput

Function name

__STATIC_INLINE void LL_HRTIM_DisableOutput (HRTIM_TypeDef * HRTIMx, uint32_t Outputs)

Function description

Disable the HRTIM timer(s) output(s) .

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Outputs:** This parameter can be a combination of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2

Return values

- **None:**

Reference Manual to LL API cross reference:

- OENR TA1OEN LL_HRTIM_DisableOutput
- OENR TA2OEN LL_HRTIM_DisableOutput
- OENR TB1OEN LL_HRTIM_DisableOutput
- OENR TB2OEN LL_HRTIM_DisableOutput
- OENR TC1OEN LL_HRTIM_DisableOutput
- OENR TC2OEN LL_HRTIM_DisableOutput
- OENR TD1OEN LL_HRTIM_DisableOutput
- OENR TD2OEN LL_HRTIM_DisableOutput
- OENR TE1OEN LL_HRTIM_DisableOutput
- OENR TE2OEN LL_HRTIM_DisableOutput

LL_HRTIM_IsEnabledOutput

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledOutput (const HRTIM_TypeDef * HRTIMx, uint32_t Output)

Function description

Indicates whether the HRTIM timer output is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2

Return values

- **State:** of TxyOEN bit in HRTIM_OENR register (1 or 0).

Reference Manual to LL API cross reference:

- OENR TA1OEN LL_HRTIM_IsEnabledOutput
- OENR TA2OEN LL_HRTIM_IsEnabledOutput
- OENR TB1OEN LL_HRTIM_IsEnabledOutput
- OENR TB2OEN LL_HRTIM_IsEnabledOutput
- OENR TC1OEN LL_HRTIM_IsEnabledOutput
- OENR TC2OEN LL_HRTIM_IsEnabledOutput
- OENR TD1OEN LL_HRTIM_IsEnabledOutput
- OENR TD2OEN LL_HRTIM_IsEnabledOutput
- OENR TE1OEN LL_HRTIM_IsEnabledOutput
- OENR TE2OEN LL_HRTIM_IsEnabledOutput

LL_HRTIM_IsDisabledOutput

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsDisabledOutput (const HRTIM_TypeDef * HRTIMx, uint32_t Output)

Function description

Indicates whether the HRTIM timer output is disabled.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2

Return values

- **State:** of TxyODS bit in HRTIM_OENR register (1 or 0).

Reference Manual to LL API cross reference:

- ODISR TA1ODIS LL_HRTIM_IsDisabledOutput
- ODISR TA2ODIS LL_HRTIM_IsDisabledOutput
- ODISR TB1ODIS LL_HRTIM_IsDisabledOutput
- ODISR TB2ODIS LL_HRTIM_IsDisabledOutput
- ODISR TC1ODIS LL_HRTIM_IsDisabledOutput
- ODISR TC2ODIS LL_HRTIM_IsDisabledOutput
- ODISR TD1ODIS LL_HRTIM_IsDisabledOutput
- ODISR TD2ODIS LL_HRTIM_IsDisabledOutput
- ODISR TE1ODIS LL_HRTIM_IsDisabledOutput
- ODISR TE2ODIS LL_HRTIM_IsDisabledOutput

LL_HRTIM_ConfigADCTrig

Function name

__STATIC_INLINE void LL_HRTIM_ConfigADCTrig (HRTIM_TypeDef * HRTIMx, uint32_t ADCTrig, uint32_t Update, uint32_t Src)

Function description

Configure an ADC trigger.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **ADCTrig:** This parameter can be one of the following values:
  - LL_HRTIM_ADCTRIG_1
  - LL_HRTIM_ADCTRIG_2
  - LL_HRTIM_ADCTRIG_3
  - LL_HRTIM_ADCTRIG_4
- **Update:** This parameter can be one of the following values:
  - LL_HRTIM_ADCTRIG_UPDATE_MASTER
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_A
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_B
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_C
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_D
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_E
- **Src:** This parameter can be a combination of the following values:

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 ADC1USRC LL_HRTIM_ConfigADCTrig
- CR1 ADC2USRC LL_HRTIM_ConfigADCTrig
- CR1 ADC3USRC LL_HRTIM_ConfigADCTrig
- CR1 ADC4USRC LL_HRTIM_ConfigADCTrig
- ADC1R ADC1MC1 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1MC2 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1MC3 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1MC4 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1MPER LL_HRTIM_ConfigADCTrig
- ADC1R ADC1EEV1 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1EEV2 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1EEV3 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1EEV4 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1EEV5 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TAC2 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TAC3 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TAC4 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TAPER LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TARST LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TBC2 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TBC3 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TBC4 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TBPER LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TBRST LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TCC2 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TCC3 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TCC4 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TCPER LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TDC2 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TDC3 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TDC4 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TDPER LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TEC2 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TEC3 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TEC4 LL_HRTIM_ConfigADCTrig
- ADC1R ADC1TEPER LL_HRTIM_ConfigADCTrig
- ADC2R ADC2MC1 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2MC2 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2MC3 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2MC4 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2MPER LL_HRTIM_ConfigADCTrig
- ADC2R ADC2EEV6 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2EEV7 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2EEV8 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2EEV9 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2EEV10 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TAC2 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TAC3 LL_HRTIM_ConfigADCTrig

- ADC2R ADC2TAC4 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TAPER LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TBC2 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TBC3 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TBC4 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TBPER LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TCC2 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TCC3 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TCC4 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TCPER LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TCRST LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TDC2 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TDC3 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TDC4 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TDPER LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TDRST LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TEC2 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TEC3 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TEC4 LL_HRTIM_ConfigADCTrig
- ADC2R ADC2TERST LL_HRTIM_ConfigADCTrig
- ADC3R ADC3MC1 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3MC2 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3MC3 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3MC4 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3MPER LL_HRTIM_ConfigADCTrig
- ADC3R ADC3EEV1 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3EEV2 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3EEV3 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3EEV4 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3EEV5 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TAC2 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TAC3 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TAC4 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TAPER LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TARST LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TBC2 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TBC3 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TBC4 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TBPER LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TBRST LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TCC2 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TCC3 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TCC4 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TCPER LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TDC2 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TDC3 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TDC4 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TDPER LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TEC2 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TEC3 LL_HRTIM_ConfigADCTrig

- ADC3R ADC3TEC4 LL_HRTIM_ConfigADCTrig
- ADC3R ADC3TEPER LL_HRTIM_ConfigADCTrig
- ADC4R ADC4MC1 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4MC2 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4MC3 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4MC4 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4MPER LL_HRTIM_ConfigADCTrig
- ADC4R ADC4EEV6 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4EEV7 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4EEV8 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4EEV9 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4EEV10 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TAC2 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TAC3 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TAC4 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TAPER LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TBC2 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TBC3 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TBC4 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TBPER LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TCC2 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TCC3 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TCC4 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TCPER LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TCRST LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TDC2 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TDC3 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TDC4 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TDPER LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TDRST LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TEC2 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TEC3 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TEC4 LL_HRTIM_ConfigADCTrig
- ADC4R ADC4TERST LL_HRTIM_ConfigADCTrig

LL_HRTIM_SetADCTrigUpdate

Function name

__STATIC_INLINE void LL_HRTIM_SetADCTrigUpdate (HRTIM_TypeDef * HRTIMx, uint32_t ADCTrig, uint32_t Update)

Function description

Associate the ADCx trigger to a timer triggering the update of the HRTIM_ADCxR register.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **ADCTrig:** This parameter can be one of the following values:
  - LL_HRTIM_ADCTRIG_1
  - LL_HRTIM_ADCTRIG_2
  - LL_HRTIM_ADCTRIG_3
  - LL_HRTIM_ADCTRIG_4
- **Update:** This parameter can be one of the following values:
  - LL_HRTIM_ADCTRIG_UPDATE_MASTER
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_A
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_B
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_C
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_D
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_E

Return values

- **None:**

Notes

- When the preload is disabled in the source timer, the HRTIM_ADCxR registers are not preloaded either: a write access will result in an immediate update of the trigger source.

Reference Manual to LL API cross reference:

- CR1 ADC1USRC LL_HRTIM_SetADCTrigUpdate
- CR1 ADC2USRC LL_HRTIM_SetADCTrigUpdate
- CR1 ADC3USRC LL_HRTIM_SetADCTrigUpdate
- CR1 ADC4USRC LL_HRTIM_SetADCTrigUpdate
-

LL_HRTIM_GetADCTrigUpdate

Function name

__STATIC_INLINE uint32_t LL_HRTIM_GetADCTrigUpdate (const HRTIM_TypeDef * HRTIMx, uint32_t ADCTrig)

Function description

Get the source timer triggering the update of the HRTIM_ADCxR register.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **ADCTrig:** This parameter can be one of the following values:
  - LL_HRTIM_ADCTRIG_1
  - LL_HRTIM_ADCTRIG_2
  - LL_HRTIM_ADCTRIG_3
  - LL_HRTIM_ADCTRIG_4

Return values

- **Update:** Returned value can be one of the following values:
  - LL_HRTIM_ADCTRIG_UPDATE_MASTER
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_A
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_B
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_C
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_D
  - LL_HRTIM_ADCTRIG_UPDATE_TIMER_E

Reference Manual to LL API cross reference:

- CR1 ADC1USRC LL_HRTIM_GetADCTrigUpdate
- CR1 ADC2USRC LL_HRTIM_GetADCTrigUpdate
- CR1 ADC3USRC LL_HRTIM_GetADCTrigUpdate
- CR1 ADC4USRC LL_HRTIM_GetADCTrigUpdate
-

LL_HRTIM_SetADCTrigSrc

Function name

__STATIC_INLINE void LL_HRTIM_SetADCTrigSrc (HRTIM_TypeDef * HRTIMx, uint32_t ADCTrig, uint32_t Src)

Function description

Specify which events (timer events and/or external events) are used as triggers for ADC conversion.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **ADCTrig:** This parameter can be one of the following values:
  - LL_HRTIM_ADCTRIG_1
  - LL_HRTIM_ADCTRIG_2
  - LL_HRTIM_ADCTRIG_3
  - LL_HRTIM_ADCTRIG_4

- **Src:** For ADC trigger 1 and ADC trigger 3 this parameter can be a combination of the following values:
  - LL_HRTIM_ADCTRIG_SRC13_NONE
  - LL_HRTIM_ADCTRIG_SRC13_MCMP1
  - LL_HRTIM_ADCTRIG_SRC13_MCMP2
  - LL_HRTIM_ADCTRIG_SRC13_MCMP3
  - LL_HRTIM_ADCTRIG_SRC13_MCMP4
  - LL_HRTIM_ADCTRIG_SRC13_MPER
  - LL_HRTIM_ADCTRIG_SRC13_EEV1
  - LL_HRTIM_ADCTRIG_SRC13_EEV2
  - LL_HRTIM_ADCTRIG_SRC13_EEV3
  - LL_HRTIM_ADCTRIG_SRC13_EEV4
  - LL_HRTIM_ADCTRIG_SRC13_EEV5
  - LL_HRTIM_ADCTRIG_SRC13_TIMACMP2
  - LL_HRTIM_ADCTRIG_SRC13_TIMACMP3
  - LL_HRTIM_ADCTRIG_SRC13_TIMACMP4
  - LL_HRTIM_ADCTRIG_SRC13_TIMAPER
  - LL_HRTIM_ADCTRIG_SRC13_TIMARST
  - LL_HRTIM_ADCTRIG_SRC13_TIMBCMP2
  - LL_HRTIM_ADCTRIG_SRC13_TIMBCMP3
  - LL_HRTIM_ADCTRIG_SRC13_TIMBCMP4
  - LL_HRTIM_ADCTRIG_SRC13_TIMBPER
  - LL_HRTIM_ADCTRIG_SRC13_TIMBRST
  - LL_HRTIM_ADCTRIG_SRC13_TIMCCMP2
  - LL_HRTIM_ADCTRIG_SRC13_TIMCCMP3
  - LL_HRTIM_ADCTRIG_SRC13_TIMCCMP4
  - LL_HRTIM_ADCTRIG_SRC13_TIMCPER
  - LL_HRTIM_ADCTRIG_SRC13_TIMDCMP2
  - LL_HRTIM_ADCTRIG_SRC13_TIMDCMP3
  - LL_HRTIM_ADCTRIG_SRC13_TIMDCMP4
  - LL_HRTIM_ADCTRIG_SRC13_TIMDPER
  - LL_HRTIM_ADCTRIG_SRC13_TIMECMP2
  - LL_HRTIM_ADCTRIG_SRC13_TIMECMP3
  - LL_HRTIM_ADCTRIG_SRC13_TIMECMP4
  - LL_HRTIM_ADCTRIG_SRC13_TIMEPER

  For ADC trigger 2 and ADC trigger 4 this parameter can be a combination of the following values:
  - LL_HRTIM_ADCTRIG_SRC24_NONE
  - LL_HRTIM_ADCTRIG_SRC24_MCMP1
  - LL_HRTIM_ADCTRIG_SRC24_MCMP2
  - LL_HRTIM_ADCTRIG_SRC24_MCMP3
  - LL_HRTIM_ADCTRIG_SRC24_MCMP4
  - LL_HRTIM_ADCTRIG_SRC24_MPER
  - LL_HRTIM_ADCTRIG_SRC24_EEV6
  - LL_HRTIM_ADCTRIG_SRC24_EEV7
  - LL_HRTIM_ADCTRIG_SRC24_EEV8
  - LL_HRTIM_ADCTRIG_SRC24_EEV9
  - LL_HRTIM_ADCTRIG_SRC24_EEV10
  - LL_HRTIM_ADCTRIG_SRC24_TIMACMP2
  - LL_HRTIM_ADCTRIG_SRC24_TIMACMP3
  - LL_HRTIM_ADCTRIG_SRC24_TIMACMP4
  - LL_HRTIM_ADCTRIG_SRC24_TIMAPER
  - LL_HRTIM_ADCTRIG_SRC24_TIMBCMP2
  - LL_HRTIM_ADCTRIG_SRC24_TIMBCMP3

Return values

- **None:**

Reference Manual to LL API cross reference:

- ADC1R ADC1MC1 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1MC2 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1MC3 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1MC4 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1MPER LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1EEV1 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1EEV2 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1EEV3 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1EEV4 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1EEV5 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TAC2 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TAC3 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TAC4 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TAPER LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TARST LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TBC2 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TBC3 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TBC4 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TBPER LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TBRST LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TCC2 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TCC3 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TCC4 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TCPER LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TDC2 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TDC3 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TDC4 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TDPER LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TEC2 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TEC3 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TEC4 LL_HRTIM_SetADCTrigSrc
- ADC1R ADC1TEPER LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2MC1 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2MC2 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2MC3 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2MC4 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2MPER LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2EEV6 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2EEV7 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2EEV8 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2EEV9 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2EEV10 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TAC2 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TAC3 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TAC4 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TAPER LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TBC2 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TBC3 LL_HRTIM_SetADCTrigSrc

- ADC2R ADC2TBC4 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TBPER LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TCC2 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TCC3 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TCC4 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TCPER LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TCRST LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TDC2 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TDC3 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TDC4 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TDPER LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TDRST LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TEC2 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TEC3 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TEC4 LL_HRTIM_SetADCTrigSrc
- ADC2R ADC2TERST LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3MC1 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3MC2 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3MC3 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3MC4 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3MPER LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3EEV1 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3EEV2 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3EEV3 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3EEV4 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3EEV5 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TAC2 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TAC3 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TAC4 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TAPER LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TARST LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TBC2 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TBC3 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TBC4 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TBPER LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TBRST LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TCC2 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TCC3 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TCC4 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TCPER LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TDC2 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TDC3 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TDC4 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TDPER LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TEC2 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TEC3 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TEC4 LL_HRTIM_SetADCTrigSrc
- ADC3R ADC3TEPER LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4MC1 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4MC2 LL_HRTIM_SetADCTrigSrc

- ADC4R ADC4MC3 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4MC4 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4MPER LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4EEV6 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4EEV7 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4EEV8 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4EEV9 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4EEV10 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TAC2 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TAC3 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TAC4 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TAPER LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TBC2 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TBC3 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TBC4 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TBPER LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TCC2 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TCC3 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TCC4 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TCPER LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TCRST LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TDC2 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TDC3 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TDC4 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TDPER LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TDRST LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TEC2 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TEC3 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TEC4 LL_HRTIM_SetADCTrigSrc
- ADC4R ADC4TERST LL_HRTIM_SetADCTrigSrc
-

LL_HRTIM_GetADCTrigSrc

Function name

__STATIC_INLINE uint32_t LL_HRTIM_GetADCTrigSrc (const HRTIM_TypeDef * HRTIMx, uint32_t ADCTrig)

Function description

Indicate which events (timer events and/or external events) are currently used as triggers for ADC conversion.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **ADCTrig:** This parameter can be one of the following values:
  - LL_HRTIM_ADCTRIG_1
  - LL_HRTIM_ADCTRIG_2
  - LL_HRTIM_ADCTRIG_3
  - LL_HRTIM_ADCTRIG_4

Return values

- **Src:** This parameter can be a combination of the following values:

Reference Manual to LL API cross reference:

- ADC1R ADC1MC1 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1MC2 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1MC3 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1MC4 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1MPER LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1EEV1 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1EEV2 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1EEV3 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1EEV4 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1EEV5 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TAC2 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TAC3 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TAC4 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TAPER LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TARST LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TBC2 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TBC3 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TBC4 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TBPER LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TBRST LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TCC2 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TCC3 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TCC4 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TCPER LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TDC2 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TDC3 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TDC4 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TDPER LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TEC2 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TEC3 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TEC4 LL_HRTIM_GetADCTrigSrc
- ADC1R ADC1TEPER LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2MC1 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2MC2 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2MC3 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2MC4 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2MPER LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2EEV6 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2EEV7 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2EEV8 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2EEV9 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2EEV10 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TAC2 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TAC3 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TAC4 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TAPER LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TBC2 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TBC3 LL_HRTIM_GetADCTrigSrc

- ADC2R ADC2TBC4 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TBPER LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TCC2 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TCC3 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TCC4 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TCPER LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TCRST LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TDC2 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TDC3 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TDC4 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TDPER LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TDRST LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TEC2 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TEC3 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TEC4 LL_HRTIM_GetADCTrigSrc
- ADC2R ADC2TERST LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3MC1 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3MC2 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3MC3 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3MC4 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3MPER LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3EEV1 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3EEV2 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3EEV3 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3EEV4 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3EEV5 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TAC2 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TAC3 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TAC4 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TAPER LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TARST LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TBC2 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TBC3 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TBC4 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TBPER LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TBRST LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TCC2 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TCC3 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TCC4 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TCPER LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TDC2 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TDC3 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TDC4 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TDPER LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TEC2 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TEC3 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TEC4 LL_HRTIM_GetADCTrigSrc
- ADC3R ADC3TEPER LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4MC1 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4MC2 LL_HRTIM_GetADCTrigSrc

- ADC4R ADC4MC3 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4MC4 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4MPER LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4EEV6 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4EEV7 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4EEV8 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4EEV9 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4EEV10 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TAC2 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TAC3 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TAC4 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TAPER LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TBC2 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TBC3 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TBC4 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TBPER LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TCC2 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TCC3 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TCC4 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TCPER LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TCRST LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TDC2 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TDC3 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TDC4 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TDPER LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TDRST LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TEC2 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TEC3 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TEC4 LL_HRTIM_GetADCTrigSrc
- ADC4R ADC4TERST LL_HRTIM_GetADCTrigSrc

LL_HRTIM_TIM_CounterEnable

Function name

__STATIC_INLINE void LL_HRTIM_TIM_CounterEnable (HRTIM_TypeDef * HRTIMx, uint32_t Timers)

Function description

Enable timer(s) counter.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timers:** This parameter can be a combination of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER TECEN LL_HRTIM_TIM_CounterEnable
- MDIER TDCEN LL_HRTIM_TIM_CounterEnable
- MDIER TCCEN LL_HRTIM_TIM_CounterEnable
- MDIER TBCEN LL_HRTIM_TIM_CounterEnable
- MDIER TACEN LL_HRTIM_TIM_CounterEnable
- MDIER MCEN LL_HRTIM_TIM_CounterEnable

LL_HRTIM_TIM_CounterDisable

Function name

__STATIC_INLINE void LL_HRTIM_TIM_CounterDisable (HRTIM_TypeDef * HRTIMx, uint32_t Timers)

Function description

Disable timer(s) counter.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timers:** This parameter can be a combination of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER TECEN LL_HRTIM_TIM_CounterDisable
- MDIER TDCEN LL_HRTIM_TIM_CounterDisable
- MDIER TCCEN LL_HRTIM_TIM_CounterDisable
- MDIER TBCEN LL_HRTIM_TIM_CounterDisable
- MDIER TACEN LL_HRTIM_TIM_CounterDisable
- MDIER MCEN LL_HRTIM_TIM_CounterDisable

LL_HRTIM_TIM_IsCounterEnabled

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_IsCounterEnabled (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the timer counter is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of MCEN or TxCEN bit HRTIM_MCR register (1 or 0).

Reference Manual to LL API cross reference:

- MDIER TECEN LL_HRTIM_TIM_IsCounterEnabled
- MDIER TDCEN LL_HRTIM_TIM_IsCounterEnabled
- MDIER TCCEN LL_HRTIM_TIM_IsCounterEnabled
- MDIER TBCEN LL_HRTIM_TIM_IsCounterEnabled
- MDIER TACEN LL_HRTIM_TIM_IsCounterEnabled
- MDIER MCEN LL_HRTIM_TIM_IsCounterEnabled

LL_HRTIM_TIM_SetPrescaler

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetPrescaler (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Prescaler)

Function description

Set the timer clock prescaler ratio.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **Prescaler:** This parameter can be one of the following values:
  - LL_HRTIM_PRESCALERRATIO_DIV1
  - LL_HRTIM_PRESCALERRATIO_DIV2
  - LL_HRTIM_PRESCALERRATIO_DIV4

Return values

- **None:**

Notes

- The counter clock equivalent frequency (CK_CNT) is equal to fHRCK / 2^CKPSC[2:0].
- The prescaling ratio cannot be modified once the timer counter is enabled.

Reference Manual to LL API cross reference:

- MCR CKPSC LL_HRTIM_TIM_SetPrescaler
- TIMxCR CKPSC LL_HRTIM_TIM_SetPrescaler

LL_HRTIM_TIM_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetPrescaler (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get the timer clock prescaler ratio.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **Prescaler:** Returned value can be one of the following values:
  - LL_HRTIM_PRESCALERRATIO_DIV1
  - LL_HRTIM_PRESCALERRATIO_DIV2
  - LL_HRTIM_PRESCALERRATIO_DIV4

Reference Manual to LL API cross reference:

- MCR CKPSC LL_HRTIM_TIM_GetPrescaler
- TIMxCR CKPSC LL_HRTIM_TIM_GetPrescaler

LL_HRTIM_TIM_SetCounterMode

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetCounterMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Mode)

Function description

Set the counter operating mode mode (single-shot, continuous or re-triggerable).

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **Mode:** This parameter can be one of the following values:
  - LL_HRTIM_MODE_CONTINUOUS
  - LL_HRTIM_MODE_SINGLESHOT
  - LL_HRTIM_MODE_RETRIGGERABLE

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR CONT LL_HRTIM_TIM_SetCounterMode
- MCR RETRIG LL_HRTIM_TIM_SetCounterMode
- TIMxCR CONT LL_HRTIM_TIM_SetCounterMode
- TIMxCR RETRIG LL_HRTIM_TIM_SetCounterMode

LL_HRTIM_TIM_GetCounterMode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetCounterMode (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get the counter operating mode mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **Mode:** Returned value can be one of the following values:
  - LL_HRTIM_MODE_CONTINUOUS
  - LL_HRTIM_MODE_SINGLESHOT
  - LL_HRTIM_MODE_RETRIGGERABLE

Reference Manual to LL API cross reference:

- MCR CONT LL_HRTIM_TIM_GetCounterMode
- MCR RETRIG LL_HRTIM_TIM_GetCounterMode
- TIMxCR CONT LL_HRTIM_TIM_GetCounterMode
- TIMxCR RETRIG LL_HRTIM_TIM_GetCounterMode

LL_HRTIM_TIM_EnableHalfMode

Function name

__STATIC_INLINE void LL_HRTIM_TIM_EnableHalfMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the half duty-cycle mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Notes

- When the half mode is enabled, HRTIM_MCMP1R (or HRTIM_CMP1xR) active register is automatically updated with HRTIM_MPER/2 (or HRTIM_PERxR/2) value when HRTIM_MPER (or HRTIM_PERxR) register is written.

Reference Manual to LL API cross reference:

- MCR HALF LL_HRTIM_TIM_EnableHalfMode
- TIMxCR HALF LL_HRTIM_TIM_EnableHalfMode

LL_HRTIM_TIM_DisableHalfMode

Function name

__STATIC_INLINE void LL_HRTIM_TIM_DisableHalfMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the half duty-cycle mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR HALF LL_HRTIM_TIM_DisableHalfMode
- TIMxCR HALF LL_HRTIM_TIM_DisableHalfMode

LL_HRTIM_TIM_IsEnabledHalfMode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_IsEnabledHalfMode (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether half duty-cycle mode is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of HALF bit to 1 in HRTIM_MCR or HRTIM_TIMxCR register (1 or 0).

Reference Manual to LL API cross reference:

- MCR HALF LL_HRTIM_TIM_IsEnabledHalfMode
- TIMxCR HALF LL_HRTIM_TIM_IsEnabledHalfMode

LL_HRTIM_TIM_EnableStartOnSync

Function name

__STATIC_INLINE void LL_HRTIM_TIM_EnableStartOnSync (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the timer start when receiving a synchronization input event.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR SYNCSTRTM LL_HRTIM_TIM_EnableStartOnSync
- TIMxCR SYNSTRTA LL_HRTIM_TIM_EnableStartOnSync

LL_HRTIM_TIM_DisableStartOnSync

Function name

__STATIC_INLINE void LL_HRTIM_TIM_DisableStartOnSync (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the timer start when receiving a synchronization input event.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR SYNCSTRTM LL_HRTIM_TIM_DisableStartOnSync
- TIMxCR SYNSTRTA LL_HRTIM_TIM_DisableStartOnSync

LL_HRTIM_TIM_IsEnabledStartOnSync

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_IsEnabledStartOnSync (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the timer start when receiving a synchronization input event.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of SYNCSTRTx bit in HRTIM_MCR or HRTIM_TIMxCR register (1 or 0).

Reference Manual to LL API cross reference:

- MCR SYNCSTRTM LL_HRTIM_TIM_IsEnabledStartOnSync
- TIMxCR SYNSTRTA LL_HRTIM_TIM_IsEnabledStartOnSync

LL_HRTIM_TIM_EnableResetOnSync

Function name

__STATIC_INLINE void LL_HRTIM_TIM_EnableResetOnSync (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the timer reset when receiving a synchronization input event.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR SYNCRSTM LL_HRTIM_TIM_EnableResetOnSync
- TIMxCR SYNCRSTA LL_HRTIM_TIM_EnableResetOnSync

LL_HRTIM_TIM_DisableResetOnSync

Function name

__STATIC_INLINE void LL_HRTIM_TIM_DisableResetOnSync (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the timer reset when receiving a synchronization input event.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR SYNCRSTM LL_HRTIM_TIM_DisableResetOnSync
- TIMxCR SYNCRSTA LL_HRTIM_TIM_DisableResetOnSync

LL_HRTIM_TIM_IsEnabledResetOnSync

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_IsEnabledResetOnSync (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the timer reset when receiving a synchronization input event.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR SYNCRSTM LL_HRTIM_TIM_IsEnabledResetOnSync
- TIMxCR SYNCRSTA LL_HRTIM_TIM_IsEnabledResetOnSync

LL_HRTIM_TIM_SetDACTrig

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetDACTrig (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t DACTrig)

Function description

Set the HRTIM output the DAC synchronization event is generated on (DACtrigOutx).

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **DACTrig:** This parameter can be one of the following values:
  - LL_HRTIM_DACTRIG_NONE
  - LL_HRTIM_DACTRIG_DACTRIGOUT_1
  - LL_HRTIM_DACTRIG_DACTRIGOUT_2
  - LL_HRTIM_DACTRIG_DACTRIGOUT_3

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR DACSYNC LL_HRTIM_TIM_SetDACTrig
- TIMxCR DACSYNC LL_HRTIM_TIM_SetDACTrig

LL_HRTIM_TIM_GetDACTrig

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetDACTrig (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get the HRTIM output the DAC synchronization event is generated on (DACtrigOutx).

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **DACTrig:** Returned value can be one of the following values:
  - LL_HRTIM_DACTRIG_NONE
  - LL_HRTIM_DACTRIG_DACTRIGOUT_1
  - LL_HRTIM_DACTRIG_DACTRIGOUT_2
  - LL_HRTIM_DACTRIG_DACTRIGOUT_3

Reference Manual to LL API cross reference:

- MCR DACSYNC LL_HRTIM_TIM_GetDACTrig
- TIMxCR DACSYNC LL_HRTIM_TIM_GetDACTrig

LL_HRTIM_TIM_EnablePreload

Function name

__STATIC_INLINE void LL_HRTIM_TIM_EnablePreload (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the timer registers preload mechanism.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Notes

- When the preload mode is enabled, accessed registers are shadow registers. Their content is transferred into the active register after an update request, either software or synchronized with an event.

Reference Manual to LL API cross reference:

- MCR PREEN LL_HRTIM_TIM_EnablePreload
- TIMxCR PREEN LL_HRTIM_TIM_EnablePreload

LL_HRTIM_TIM_DisablePreload

Function name

__STATIC_INLINE void LL_HRTIM_TIM_DisablePreload (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the timer registers preload mechanism.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR PREEN LL_HRTIM_TIM_DisablePreload
- TIMxCR PREEN LL_HRTIM_TIM_DisablePreload

LL_HRTIM_TIM_IsEnabledPreload

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_IsEnabledPreload (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the timer registers preload mechanism is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of PREEN bit in HRTIM_MCR or HRTIM_TIMxCR register (1 or 0).

Reference Manual to LL API cross reference:

- MCR PREEN LL_HRTIM_TIM_IsEnabledPreload
- TIMxCR PREEN LL_HRTIM_TIM_IsEnabledPreload

LL_HRTIM_TIM_SetUpdateTrig

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetUpdateTrig (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t UpdateTrig)

Function description

Set the timer register update trigger.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **UpdateTrig:** This parameter can be one of the following values:

Reference Manual to LL API cross reference:

- MCR MREPU LL_HRTIM_TIM_SetUpdateTrig
- TIMxCR TAU LL_HRTIM_TIM_SetUpdateTrig
- TIMxCR TBU LL_HRTIM_TIM_SetUpdateTrig
- TIMxCR TCU LL_HRTIM_TIM_SetUpdateTrig
- TIMxCR TDU LL_HRTIM_TIM_SetUpdateTrig
- TIMxCR TEU LL_HRTIM_TIM_SetUpdateTrig
- TIMxCR MSTU LL_HRTIM_TIM_SetUpdateTrig

LL_HRTIM_TIM_GetUpdateTrig

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetUpdateTrig (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get the timer register update trigger.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **UpdateTrig:** Returned value can be one of the following values:

Reference Manual to LL API cross reference:

- MCR MREPU LL_HRTIM_TIM_GetUpdateTrig
- TIMxCR TBU LL_HRTIM_TIM_GetUpdateTrig
- TIMxCR TCU LL_HRTIM_TIM_GetUpdateTrig
- TIMxCR TDU LL_HRTIM_TIM_GetUpdateTrig
- TIMxCR TEU LL_HRTIM_TIM_GetUpdateTrig
- TIMxCR MSTU LL_HRTIM_TIM_GetUpdateTrig

LL_HRTIM_TIM_SetUpdateGating

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetUpdateGating (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t UpdateGating)

Function description

Set the timer registers update condition (how the registers update occurs relatively to the burst DMA transaction or an external update request received on one of the update enable inputs (UPD_EN[3:1])).

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **UpdateGating:** This parameter can be one of the following values:

Reference Manual to LL API cross reference:

- MCR BRSTDMA LL_HRTIM_TIM_SetUpdateGating
- TIMxCR UPDGAT LL_HRTIM_TIM_SetUpdateGating

LL_HRTIM_TIM_GetUpdateGating

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetUpdateGating (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get the timer registers update condition.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **UpdateGating:** Returned value can be one of the following values:

Reference Manual to LL API cross reference:

- MCR BRSTDMA LL_HRTIM_TIM_GetUpdateGating
- TIMxCR UPDGAT LL_HRTIM_TIM_GetUpdateGating

LL_HRTIM_TIM_EnablePushPullMode

Function name

__STATIC_INLINE void LL_HRTIM_TIM_EnablePushPullMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the push-pull mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxCR PSHPLL LL_HRTIM_TIM_EnablePushPullMode

LL_HRTIM_TIM_DisablePushPullMode

Function name

__STATIC_INLINE void LL_HRTIM_TIM_DisablePushPullMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the push-pull mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxCR PSHPLL LL_HRTIM_TIM_DisablePushPullMode

LL_HRTIM_TIM_IsEnabledPushPullMode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_IsEnabledPushPullMode (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the push-pull mode is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of PSHPLL bit in HRTIM_TIMxCR register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxCR PSHPLL LL_HRTIM_TIM_IsEnabledPushPullMode
-

LL_HRTIM_TIM_SetCompareMode

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetCompareMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t CompareUnit, uint32_t Mode)

Function description

Set the functioning mode of the compare unit (CMP2 or CMP4 can operate in standard mode or in auto delayed mode).

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **CompareUnit:** This parameter can be one of the following values:
  - LL_HRTIM_COMPAREUNIT_2
  - LL_HRTIM_COMPAREUNIT_4
- **Mode:** This parameter can be one of the following values:
  - LL_HRTIM_COMPAREMODE_REGULAR
  - LL_HRTIM_COMPAREMODE_DELAY_NOTIMEOUT
  - LL_HRTIM_COMPAREMODE_DELAY_CMP1
  - LL_HRTIM_COMPAREMODE_DELAY_CMP3

Return values

- **None:**

Notes

- In auto-delayed mode the compare match occurs independently from the timer counter value.

Reference Manual to LL API cross reference:

- TIMxCR DELCMP2 LL_HRTIM_TIM_SetCompareMode
- TIMxCR DELCMP4 LL_HRTIM_TIM_SetCompareMode

LL_HRTIM_TIM_GetCompareMode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetCompareMode (const HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t CompareUnit)

Function description

Get the functioning mode of the compare unit.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **CompareUnit:** This parameter can be one of the following values:
  - LL_HRTIM_COMPAREUNIT_2
  - LL_HRTIM_COMPAREUNIT_4

Return values

- **Mode:** Returned value can be one of the following values:
  - LL_HRTIM_COMPAREMODE_REGULAR
  - LL_HRTIM_COMPAREMODE_DELAY_NOTIMEOUT
  - LL_HRTIM_COMPAREMODE_DELAY_CMP1
  - LL_HRTIM_COMPAREMODE_DELAY_CMP3

Reference Manual to LL API cross reference:

- TIMxCR DELCMP2 LL_HRTIM_TIM_GetCompareMode
- TIMxCR DELCMP4 LL_HRTIM_TIM_GetCompareMode

LL_HRTIM_TIM_SetCounter

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetCounter (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Counter)

Function description

Set the timer counter value.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **Counter:** Value between 0 and 0xFFFF

Return values

- **None:**

Notes

- This function can only be called when the timer is stopped.
- For HR clock prescaling ratio below 32 (CKPSC[2:0] < 5), the least significant bits of the counter are not significant. They cannot be written and return 0 when read.
- The timer behavior is not guaranteed if the counter value is set above the period.

Reference Manual to LL API cross reference:

- MCNTR MCNT LL_HRTIM_TIM_SetCounter
- CNTxR CNTx LL_HRTIM_TIM_SetCounter

LL_HRTIM_TIM_GetCounter

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetCounter (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual timer counter value.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **Counter:** Value between 0 and 0xFFFF

Reference Manual to LL API cross reference:

- MCNTR MCNT LL_HRTIM_TIM_GetCounter
- CNTxR CNTx LL_HRTIM_TIM_GetCounter

LL_HRTIM_TIM_SetPeriod

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetPeriod (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Period)

Function description

Set the timer period value.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **Period:** Value between 0 and 0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- MPER MPER LL_HRTIM_TIM_SetPeriod
- PERxR PERx LL_HRTIM_TIM_SetPeriod

LL_HRTIM_TIM_GetPeriod

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetPeriod (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual timer period value.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **Period:** Value between 0 and 0xFFFF

Reference Manual to LL API cross reference:

- MPER MPER LL_HRTIM_TIM_GetPeriod
- PERxR PERx LL_HRTIM_TIM_GetPeriod

LL_HRTIM_TIM_SetRepetition

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetRepetition (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Repetition)

Function description

Set the timer repetition period value.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **Repetition:** Value between 0 and 0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- MREP MREP LL_HRTIM_TIM_SetRepetition
- REPxR REPx LL_HRTIM_TIM_SetRepetition

LL_HRTIM_TIM_GetRepetition

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetRepetition (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual timer repetition period value.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **Repetition:** Value between 0 and 0xFF

Reference Manual to LL API cross reference:

- MREP MREP LL_HRTIM_TIM_GetRepetition
- REPxR REPx LL_HRTIM_TIM_GetRepetition

LL_HRTIM_TIM_SetCompare1

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetCompare1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t CompareValue)

Function description

Set the compare value of the compare unit 1.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **CompareValue:** Compare value must be above or equal to 3 periods of the fHRTIM clock, that is 0x60 if CKPSC[2:0] = 0, 0x30 if CKPSC[2:0] = 1, 0x18 if CKPSC[2:0] = 2,...

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCMP1R MCMP1 LL_HRTIM_TIM_SetCompare1
- CMP1xR CMP1x LL_HRTIM_TIM_SetCompare1

LL_HRTIM_TIM_GetCompare1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetCompare1 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual compare value of the compare unit 1.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **CompareValue:** Compare value must be above or equal to 3 periods of the fHRTIM clock, that is 0x60 if CKPSC[2:0] = 0, 0x30 if CKPSC[2:0] = 1, 0x18 if CKPSC[2:0] = 2,...

Reference Manual to LL API cross reference:

- MCMP1R MCMP1 LL_HRTIM_TIM_GetCompare1
- CMP1xR CMP1x LL_HRTIM_TIM_GetCompare1

LL_HRTIM_TIM_SetCompare2

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetCompare2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t CompareValue)

Function description

Set the compare value of the compare unit 2.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **CompareValue:** Compare value must be above or equal to 3 periods of the fHRTIM clock, that is 0x60 if CKPSC[2:0] = 0, 0x30 if CKPSC[2:0] = 1, 0x18 if CKPSC[2:0] = 2,...

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCMP2R MCMP2 LL_HRTIM_TIM_SetCompare2
- CMP2xR CMP2x LL_HRTIM_TIM_SetCompare2

LL_HRTIM_TIM_GetCompare2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetCompare2 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual compare value of the compare unit 2.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **CompareValue:** Compare value must be above or equal to 3 periods of the fHRTIM clock, that is 0x60 if CKPSC[2:0] = 0, 0x30 if CKPSC[2:0] = 1, 0x18 if CKPSC[2:0] = 2,...

Reference Manual to LL API cross reference:

- MCMP2R MCMP2 LL_HRTIM_TIM_GetCompare2
- CMP2xR CMP2x LL_HRTIM_TIM_GetCompare2
-

LL_HRTIM_TIM_SetCompare3

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetCompare3 (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t CompareValue)

Function description

Set the compare value of the compare unit 3.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **CompareValue:** Compare value must be above or equal to 3 periods of the fHRTIM clock, that is 0x60 if CKPSC[2:0] = 0, 0x30 if CKPSC[2:0] = 1, 0x18 if CKPSC[2:0] = 2,...

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCMP3R MCMP3 LL_HRTIM_TIM_SetCompare3
- CMP3xR CMP3x LL_HRTIM_TIM_SetCompare3

LL_HRTIM_TIM_GetCompare3

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetCompare3 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual compare value of the compare unit 3.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **CompareValue:** Compare value must be above or equal to 3 periods of the fHRTIM clock, that is 0x60 if CKPSC[2:0] = 0, 0x30 if CKPSC[2:0] = 1, 0x18 if CKPSC[2:0] = 2,...

Reference Manual to LL API cross reference:

- MCMP3R MCMP3 LL_HRTIM_TIM_GetCompare3
- CMP3xR CMP3x LL_HRTIM_TIM_GetCompare3

LL_HRTIM_TIM_SetCompare4

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetCompare4 (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t CompareValue)

Function description

Set the compare value of the compare unit 4.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **CompareValue:** Compare value must be above or equal to 3 periods of the fHRTIM clock, that is 0x60 if CKPSC[2:0] = 0, 0x30 if CKPSC[2:0] = 1, 0x18 if CKPSC[2:0] = 2,...

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCMP4R MCMP4 LL_HRTIM_TIM_SetCompare4
- CMP4xR CMP4x LL_HRTIM_TIM_SetCompare4

LL_HRTIM_TIM_GetCompare4

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetCompare4 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual compare value of the compare unit 4.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **CompareValue:** Compare value must be above or equal to 3 periods of the fHRTIM clock, that is 0x60 if CKPSC[2:0] = 0, 0x30 if CKPSC[2:0] = 1, 0x18 if CKPSC[2:0] = 2,...

Reference Manual to LL API cross reference:

- MCMP4R MCMP4 LL_HRTIM_TIM_GetCompare4
- CMP4xR CMP4x LL_HRTIM_TIM_GetCompare4

LL_HRTIM_TIM_SetResetTrig

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetResetTrig (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t ResetTrig)

Function description

Set the reset trigger of a timer counter.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **ResetTrig:** This parameter can be a combination of the following values:
  - LL_HRTIM_RESETTRIG_NONE
  - LL_HRTIM_RESETTRIG_UPDATE
  - LL_HRTIM_RESETTRIG_CMP2
  - LL_HRTIM_RESETTRIG_CMP4
  - LL_HRTIM_RESETTRIG_MASTER_PER
  - LL_HRTIM_RESETTRIG_MASTER_CMP1
  - LL_HRTIM_RESETTRIG_MASTER_CMP2
  - LL_HRTIM_RESETTRIG_MASTER_CMP3
  - LL_HRTIM_RESETTRIG_MASTER_CMP4
  - LL_HRTIM_RESETTRIG_EEV_1
  - LL_HRTIM_RESETTRIG_EEV_2
  - LL_HRTIM_RESETTRIG_EEV_3
  - LL_HRTIM_RESETTRIG_EEV_4
  - LL_HRTIM_RESETTRIG_EEV_5
  - LL_HRTIM_RESETTRIG_EEV_6
  - LL_HRTIM_RESETTRIG_EEV_7
  - LL_HRTIM_RESETTRIG_EEV_8
  - LL_HRTIM_RESETTRIG_EEV_9
  - LL_HRTIM_RESETTRIG_EEV_10
  - LL_HRTIM_RESETTRIG_OTHER1_CMP1
  - LL_HRTIM_RESETTRIG_OTHER1_CMP2
  - LL_HRTIM_RESETTRIG_OTHER1_CMP4
  - LL_HRTIM_RESETTRIG_OTHER2_CMP1
  - LL_HRTIM_RESETTRIG_OTHER2_CMP2
  - LL_HRTIM_RESETTRIG_OTHER2_CMP4
  - LL_HRTIM_RESETTRIG_OTHER3_CMP1
  - LL_HRTIM_RESETTRIG_OTHER3_CMP2
  - LL_HRTIM_RESETTRIG_OTHER3_CMP4
  - LL_HRTIM_RESETTRIG_OTHER4_CMP1
  - LL_HRTIM_RESETTRIG_OTHER4_CMP2
  - LL_HRTIM_RESETTRIG_OTHER4_CMP4

Return values

- **None:**

Notes

- The reset of the timer counter can be triggered by up to 30 events that can be selected among the following sources: The timing unit: Compare 2, Compare 4 and Update (3 events). The master timer: Reset and Compare 1..4 (5 events). The external events EXTEVNT1..10 (10 events). All other timing units (e.g. Timer B..E for timer A): Compare 1, 2 and 4 (12 events).

Reference Manual to LL API cross reference:

- RSTxR UPDT LL_HRTIM_TIM_SetResetTrig
- RSTxR CMP2 LL_HRTIM_TIM_SetResetTrig
- RSTxR CMP4 LL_HRTIM_TIM_SetResetTrig
- RSTxR MSTPER LL_HRTIM_TIM_SetResetTrig
- RSTxR MSTCMP1 LL_HRTIM_TIM_SetResetTrig
- RSTxR MSTCMP2 LL_HRTIM_TIM_SetResetTrig
- RSTxR MSTCMP3 LL_HRTIM_TIM_SetResetTrig
- RSTxR MSTCMP4 LL_HRTIM_TIM_SetResetTrig
- RSTxR EXTEVNT1 LL_HRTIM_TIM_SetResetTrig
- RSTxR EXTEVNT2 LL_HRTIM_TIM_SetResetTrig
- RSTxR EXTEVNT3 LL_HRTIM_TIM_SetResetTrig
- RSTxR EXTEVNT4 LL_HRTIM_TIM_SetResetTrig
- RSTxR EXTEVNT5 LL_HRTIM_TIM_SetResetTrig
- RSTxR EXTEVNT6 LL_HRTIM_TIM_SetResetTrig
- RSTxR EXTEVNT7 LL_HRTIM_TIM_SetResetTrig
- RSTxR EXTEVNT8 LL_HRTIM_TIM_SetResetTrig
- RSTxR EXTEVNT9 LL_HRTIM_TIM_SetResetTrig
- RSTxR EXTEVNT10 LL_HRTIM_TIM_SetResetTrig
- RSTxR TIMBCMP1 LL_HRTIM_TIM_SetResetTrig
- RSTxR TIMBCMP2 LL_HRTIM_TIM_SetResetTrig
- RSTxR TIMBCMP4 LL_HRTIM_TIM_SetResetTrig
- RSTxR TIMCCMP1 LL_HRTIM_TIM_SetResetTrig
- RSTxR TIMCCMP2 LL_HRTIM_TIM_SetResetTrig
- RSTxR TIMCCMP4 LL_HRTIM_TIM_SetResetTrig
- RSTxR TIMDCMP1 LL_HRTIM_TIM_SetResetTrig
- RSTxR TIMDCMP2 LL_HRTIM_TIM_SetResetTrig
- RSTxR TIMDCMP4 LL_HRTIM_TIM_SetResetTrig
- RSTxR TIMECMP1 LL_HRTIM_TIM_SetResetTrig
- RSTxR TIMECMP2 LL_HRTIM_TIM_SetResetTrig
- RSTxR TIMECMP4 LL_HRTIM_TIM_SetResetTrig

LL_HRTIM_TIM_GetResetTrig

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetResetTrig (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual reset trigger of a timer counter.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **ResetTrig:** Returned value can be one of the following values:
  - LL_HRTIM_RESETTRIG_NONE
  - LL_HRTIM_RESETTRIG_UPDATE
  - LL_HRTIM_RESETTRIG_CMP2
  - LL_HRTIM_RESETTRIG_CMP4
  - LL_HRTIM_RESETTRIG_MASTER_PER
  - LL_HRTIM_RESETTRIG_MASTER_CMP1
  - LL_HRTIM_RESETTRIG_MASTER_CMP2
  - LL_HRTIM_RESETTRIG_MASTER_CMP3
  - LL_HRTIM_RESETTRIG_MASTER_CMP4
  - LL_HRTIM_RESETTRIG_EEV_1
  - LL_HRTIM_RESETTRIG_EEV_2
  - LL_HRTIM_RESETTRIG_EEV_3
  - LL_HRTIM_RESETTRIG_EEV_4
  - LL_HRTIM_RESETTRIG_EEV_5
  - LL_HRTIM_RESETTRIG_EEV_6
  - LL_HRTIM_RESETTRIG_EEV_7
  - LL_HRTIM_RESETTRIG_EEV_8
  - LL_HRTIM_RESETTRIG_EEV_9
  - LL_HRTIM_RESETTRIG_EEV_10
  - LL_HRTIM_RESETTRIG_OTHER1_CMP1
  - LL_HRTIM_RESETTRIG_OTHER1_CMP2
  - LL_HRTIM_RESETTRIG_OTHER1_CMP4
  - LL_HRTIM_RESETTRIG_OTHER2_CMP1
  - LL_HRTIM_RESETTRIG_OTHER2_CMP2
  - LL_HRTIM_RESETTRIG_OTHER2_CMP4
  - LL_HRTIM_RESETTRIG_OTHER3_CMP1
  - LL_HRTIM_RESETTRIG_OTHER3_CMP2
  - LL_HRTIM_RESETTRIG_OTHER3_CMP4
  - LL_HRTIM_RESETTRIG_OTHER4_CMP1
  - LL_HRTIM_RESETTRIG_OTHER4_CMP2
  - LL_HRTIM_RESETTRIG_OTHER4_CMP4

Reference Manual to LL API cross reference:

- RSTxR UPDT LL_HRTIM_TIM_GetResetTrig
- RSTxR CMP2 LL_HRTIM_TIM_GetResetTrig
- RSTxR CMP4 LL_HRTIM_TIM_GetResetTrig
- RSTxR MSTPER LL_HRTIM_TIM_GetResetTrig
- RSTxR MSTCMP1 LL_HRTIM_TIM_GetResetTrig
- RSTxR MSTCMP2 LL_HRTIM_TIM_GetResetTrig
- RSTxR MSTCMP3 LL_HRTIM_TIM_GetResetTrig
- RSTxR MSTCMP4 LL_HRTIM_TIM_GetResetTrig
- RSTxR EXTEVNT1 LL_HRTIM_TIM_GetResetTrig
- RSTxR EXTEVNT2 LL_HRTIM_TIM_GetResetTrig
- RSTxR EXTEVNT3 LL_HRTIM_TIM_GetResetTrig
- RSTxR EXTEVNT4 LL_HRTIM_TIM_GetResetTrig
- RSTxR EXTEVNT5 LL_HRTIM_TIM_GetResetTrig
- RSTxR EXTEVNT6 LL_HRTIM_TIM_GetResetTrig
- RSTxR EXTEVNT7 LL_HRTIM_TIM_GetResetTrig
- RSTxR EXTEVNT8 LL_HRTIM_TIM_GetResetTrig
- RSTxR EXTEVNT9 LL_HRTIM_TIM_GetResetTrig
- RSTxR EXTEVNT10 LL_HRTIM_TIM_GetResetTrig
- RSTxR TIMBCMP1 LL_HRTIM_TIM_GetResetTrig
- RSTxR TIMBCMP2 LL_HRTIM_TIM_GetResetTrig
- RSTxR TIMBCMP4 LL_HRTIM_TIM_GetResetTrig
- RSTxR TIMCCMP1 LL_HRTIM_TIM_GetResetTrig
- RSTxR TIMCCMP2 LL_HRTIM_TIM_GetResetTrig
- RSTxR TIMCCMP4 LL_HRTIM_TIM_GetResetTrig
- RSTxR TIMDCMP1 LL_HRTIM_TIM_GetResetTrig
- RSTxR TIMDCMP2 LL_HRTIM_TIM_GetResetTrig
- RSTxR TIMDCMP4 LL_HRTIM_TIM_GetResetTrig
- RSTxR TIMECMP1 LL_HRTIM_TIM_GetResetTrig
- RSTxR TIMECMP2 LL_HRTIM_TIM_GetResetTrig
- RSTxR TIMECMP4 LL_HRTIM_TIM_GetResetTrig

LL_HRTIM_TIM_GetCapture1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetCapture1 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get captured value for capture unit 1.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **Captured:** value

Reference Manual to LL API cross reference:

- CPT1xR CPT1x LL_HRTIM_TIM_GetCapture1

LL_HRTIM_TIM_GetCapture2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetCapture2 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get captured value for capture unit 2.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **Captured:** value

Reference Manual to LL API cross reference:

- CPT2xR CPT2x LL_HRTIM_TIM_GetCapture2

LL_HRTIM_TIM_SetCaptureTrig

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetCaptureTrig (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t CaptureUnit, uint32_t CaptureTrig)

Function description

Set the trigger of a capture unit for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **CaptureUnit:** This parameter can be one of the following values:
  - LL_HRTIM_CAPTUREUNIT_1
  - LL_HRTIM_CAPTUREUNIT_2
- **CaptureTrig:** This parameter can be a combination of the following values:
  - LL_HRTIM_CAPTURETRIG_NONE
  - LL_HRTIM_CAPTURETRIG_UPDATE
  - LL_HRTIM_CAPTURETRIG_EEV_1
  - LL_HRTIM_CAPTURETRIG_EEV_2
  - LL_HRTIM_CAPTURETRIG_EEV_3
  - LL_HRTIM_CAPTURETRIG_EEV_4
  - LL_HRTIM_CAPTURETRIG_EEV_5
  - LL_HRTIM_CAPTURETRIG_EEV_6
  - LL_HRTIM_CAPTURETRIG_EEV_7
  - LL_HRTIM_CAPTURETRIG_EEV_8
  - LL_HRTIM_CAPTURETRIG_EEV_9
  - LL_HRTIM_CAPTURETRIG_EEV_10
  - LL_HRTIM_CAPTURETRIG_TA1_SET
  - LL_HRTIM_CAPTURETRIG_TA1_RESET
  - LL_HRTIM_CAPTURETRIG_TIMA_CMP1
  - LL_HRTIM_CAPTURETRIG_TIMA_CMP2
  - LL_HRTIM_CAPTURETRIG_TB1_SET
  - LL_HRTIM_CAPTURETRIG_TB1_RESET
  - LL_HRTIM_CAPTURETRIG_TIMB_CMP1
  - LL_HRTIM_CAPTURETRIG_TIMB_CMP2
  - LL_HRTIM_CAPTURETRIG_TC1_SET
  - LL_HRTIM_CAPTURETRIG_TC1_RESET
  - LL_HRTIM_CAPTURETRIG_TIMC_CMP1
  - LL_HRTIM_CAPTURETRIG_TIMC_CMP2
  - LL_HRTIM_CAPTURETRIG_TD1_SET
  - LL_HRTIM_CAPTURETRIG_TD1_RESET
  - LL_HRTIM_CAPTURETRIG_TIMD_CMP1
  - LL_HRTIM_CAPTURETRIG_TIMD_CMP2
  - LL_HRTIM_CAPTURETRIG_TE1_SET
  - LL_HRTIM_CAPTURETRIG_TE1_RESET
  - LL_HRTIM_CAPTURETRIG_TIME_CMP1
  - LL_HRTIM_CAPTURETRIG_TIME_CMP2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CPT1xCR SWCPT LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR UPDCPT LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR EXEV1CPT LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR EXEV2CPT LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR EXEV3CPT LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR EXEV4CPT LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR EXEV5CPT LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR EXEV6CPT LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR EXEV7CPT LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR EXEV8CPT LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR EXEV9CPT LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR EXEV10CPT LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TA1SET LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TA1RST LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TACMP1 LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TACMP2 LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TB1SET LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TB1RST LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TBCMP1 LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TBCMP2 LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TC1SET LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TC1RST LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TCCMP1 LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TCCMP2 LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TD1SET LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TD1RST LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TDCMP1 LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TDCMP2 LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TE1SET LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TE1RST LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TECMP1 LL_HRTIM_TIM_SetCaptureTrig
- CPT1xCR TECMP2 LL_HRTIM_TIM_SetCaptureTrig

LL_HRTIM_TIM_GetCaptureTrig

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetCaptureTrig (const HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t CaptureUnit)

Function description

Get actual trigger of a capture unit for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **CaptureUnit:** This parameter can be one of the following values:
  - LL_HRTIM_CAPTUREUNIT_1
  - LL_HRTIM_CAPTUREUNIT_2

Return values

- **CaptureTrig:** This parameter can be a combination of the following values:
  - LL_HRTIM_CAPTURETRIG_NONE
  - LL_HRTIM_CAPTURETRIG_UPDATE
  - LL_HRTIM_CAPTURETRIG_EEV_1
  - LL_HRTIM_CAPTURETRIG_EEV_2
  - LL_HRTIM_CAPTURETRIG_EEV_3
  - LL_HRTIM_CAPTURETRIG_EEV_4
  - LL_HRTIM_CAPTURETRIG_EEV_5
  - LL_HRTIM_CAPTURETRIG_EEV_6
  - LL_HRTIM_CAPTURETRIG_EEV_7
  - LL_HRTIM_CAPTURETRIG_EEV_8
  - LL_HRTIM_CAPTURETRIG_EEV_9
  - LL_HRTIM_CAPTURETRIG_EEV_10
  - LL_HRTIM_CAPTURETRIG_TA1_SET
  - LL_HRTIM_CAPTURETRIG_TA1_RESET
  - LL_HRTIM_CAPTURETRIG_TIMA_CMP1
  - LL_HRTIM_CAPTURETRIG_TIMA_CMP2
  - LL_HRTIM_CAPTURETRIG_TB1_SET
  - LL_HRTIM_CAPTURETRIG_TB1_RESET
  - LL_HRTIM_CAPTURETRIG_TIMB_CMP1
  - LL_HRTIM_CAPTURETRIG_TIMB_CMP2
  - LL_HRTIM_CAPTURETRIG_TC1_SET
  - LL_HRTIM_CAPTURETRIG_TC1_RESET
  - LL_HRTIM_CAPTURETRIG_TIMC_CMP1
  - LL_HRTIM_CAPTURETRIG_TIMC_CMP2
  - LL_HRTIM_CAPTURETRIG_TD1_SET
  - LL_HRTIM_CAPTURETRIG_TD1_RESET
  - LL_HRTIM_CAPTURETRIG_TIMD_CMP1
  - LL_HRTIM_CAPTURETRIG_TIMD_CMP2
  - LL_HRTIM_CAPTURETRIG_TE1_SET
  - LL_HRTIM_CAPTURETRIG_TE1_RESET
  - LL_HRTIM_CAPTURETRIG_TIME_CMP1
  - LL_HRTIM_CAPTURETRIG_TIME_CMP2

Reference Manual to LL API cross reference:

- CPT1xCR SWCPT LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR UPDCPT LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR EXEV1CPT LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR EXEV2CPT LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR EXEV3CPT LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR EXEV4CPT LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR EXEV5CPT LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR EXEV6CPT LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR EXEV7CPT LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR EXEV8CPT LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR EXEV9CPT LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR EXEV10CPT LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TA1SET LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TA1RST LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TACMP1 LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TACMP2 LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TB1SET LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TB1RST LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TBCMP1 LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TBCMP2 LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TC1SET LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TC1RST LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TCCMP1 LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TCCMP2 LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TD1SET LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TD1RST LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TDCMP1 LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TDCMP2 LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TE1SET LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TE1RST LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TECMP1 LL_HRTIM_TIM_GetCaptureTrig
- CPT1xCR TECMP2 LL_HRTIM_TIM_GetCaptureTrig

LL_HRTIM_TIM_EnableDeadTime

Function name

__STATIC_INLINE void LL_HRTIM_TIM_EnableDeadTime (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable deadtime insertion for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- OUTxR DTEN LL_HRTIM_TIM_EnableDeadTime

LL_HRTIM_TIM_DisableDeadTime

Function name

__STATIC_INLINE void LL_HRTIM_TIM_DisableDeadTime (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable deadtime insertion for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- OUTxR DTEN LL_HRTIM_TIM_DisableDeadTime

LL_HRTIM_TIM_IsEnabledDeadTime

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_IsEnabledDeadTime (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether deadtime insertion is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of DTEN bit in HRTIM_OUTxR register (1 or 0).

Reference Manual to LL API cross reference:

- OUTxR DTEN LL_HRTIM_TIM_IsEnabledDeadTime

LL_HRTIM_TIM_SetDLYPRTMode

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetDLYPRTMode (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t DLYPRTMode)

Function description

Set the delayed protection (DLYPRT) mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **DLYPRTMode:** Delayed protection (DLYPRT) mode

Notes

- This function must be called prior enabling the delayed protection
- Balanced Idle mode is only available in push-pull mode

Reference Manual to LL API cross reference:

- OUTxR DLYPRTEN LL_HRTIM_TIM_SetDLYPRTMode
- OUTxR DLYPRT LL_HRTIM_TIM_SetDLYPRTMode

LL_HRTIM_TIM_GetDLYPRTMode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetDLYPRTMode (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get the delayed protection (DLYPRT) mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **DLYPRTMode:** Delayed protection (DLYPRT) mode

Reference Manual to LL API cross reference:

- OUTxR DLYPRTEN LL_HRTIM_TIM_GetDLYPRTMode
- OUTxR DLYPRT LL_HRTIM_TIM_GetDLYPRTMode

LL_HRTIM_TIM_EnableDLYPRT

Function name

__STATIC_INLINE void LL_HRTIM_TIM_EnableDLYPRT (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable delayed protection (DLYPRT) for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Notes

- This function must not be called once the concerned timer is enabled

Reference Manual to LL API cross reference:

- OUTxR DLYPRTEN LL_HRTIM_TIM_EnableDLYPRT

LL_HRTIM_TIM_DisableDLYPRT

Function name

__STATIC_INLINE void LL_HRTIM_TIM_DisableDLYPRT (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable delayed protection (DLYPRT) for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Notes

- This function must not be called once the concerned timer is enabled

Reference Manual to LL API cross reference:

- OUTxR DLYPRTEN LL_HRTIM_TIM_DisableDLYPRT

LL_HRTIM_TIM_IsEnabledDLYPRT

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_IsEnabledDLYPRT (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether delayed protection (DLYPRT) is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of DLYPRTEN bit in HRTIM_OUTxR register (1 or 0).

Reference Manual to LL API cross reference:

- OUTxR DLYPRTEN LL_HRTIM_TIM_IsEnabledDLYPRT

LL_HRTIM_TIM_EnableFault

Function name

__STATIC_INLINE void LL_HRTIM_TIM_EnableFault (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Faults)

Function description

Enable the fault channel(s) for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **Faults:** This parameter can be a combination of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLTxR FLT1EN LL_HRTIM_TIM_EnableFault
- FLTxR FLT2EN LL_HRTIM_TIM_EnableFault
- FLTxR FLT3EN LL_HRTIM_TIM_EnableFault
- FLTxR FLT4EN LL_HRTIM_TIM_EnableFault
- FLTxR FLT5EN LL_HRTIM_TIM_EnableFault

LL_HRTIM_TIM_DisableFault

Function name

__STATIC_INLINE void LL_HRTIM_TIM_DisableFault (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Faults)

Function description

Disable the fault channel(s) for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **Faults:** This parameter can be a combination of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLTxR FLT1EN LL_HRTIM_TIM_DisableFault
- FLTxR FLT2EN LL_HRTIM_TIM_DisableFault
- FLTxR FLT3EN LL_HRTIM_TIM_DisableFault
- FLTxR FLT4EN LL_HRTIM_TIM_DisableFault
- FLTxR FLT5EN LL_HRTIM_TIM_DisableFault

LL_HRTIM_TIM_IsEnabledFault

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_IsEnabledFault (const HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Fault)

Function description

Indicate whether the fault channel is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5

Return values

- **State:** of FLTxEN bit in HRTIM_FLTxR register (1 or 0).

Reference Manual to LL API cross reference:

- FLTxR FLT1EN LL_HRTIM_TIM_IsEnabledFault
- FLTxR FLT2EN LL_HRTIM_TIM_IsEnabledFault
- FLTxR FLT3EN LL_HRTIM_TIM_IsEnabledFault
- FLTxR FLT4EN LL_HRTIM_TIM_IsEnabledFault
- FLTxR FLT5EN LL_HRTIM_TIM_IsEnabledFault

LL_HRTIM_TIM_LockFault

Function name

__STATIC_INLINE void LL_HRTIM_TIM_LockFault (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Lock the fault conditioning set-up for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Notes

- Timer fault-related set-up is frozen until the next HRTIM or system reset

Reference Manual to LL API cross reference:

- FLTxR FLTLCK LL_HRTIM_TIM_LockFault

LL_HRTIM_TIM_SetBurstModeOption

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetBurstModeOption (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t BurtsModeOption)

Function description

Define how the timer behaves during a burst mode operation.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **BurtsModeOption:** This parameter can be one of the following values:
  - LL_HRTIM_BURSTMODE_MAINTAINCLOCK
  - LL_HRTIM_BURSTMODE_RESETCOUNTER

Return values

- **None:**

Notes

- This function must not be called when the burst mode is enabled

Reference Manual to LL API cross reference:

- BMCR MTBM LL_HRTIM_TIM_SetBurstModeOption
- BMCR TABM LL_HRTIM_TIM_SetBurstModeOption
- BMCR TBBM LL_HRTIM_TIM_SetBurstModeOption
- BMCR TCBM LL_HRTIM_TIM_SetBurstModeOption
- BMCR TDBM LL_HRTIM_TIM_SetBurstModeOption
- BMCR TEBM LL_HRTIM_TIM_SetBurstModeOption

LL_HRTIM_TIM_GetBurstModeOption

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetBurstModeOption (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Retrieve how the timer behaves during a burst mode operation.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **BurtsMode:** This parameter can be one of the following values:
  - LL_HRTIM_BURSTMODE_MAINTAINCLOCK
  - LL_HRTIM_BURSTMODE_RESETCOUNTER

Reference Manual to LL API cross reference:

- BMCR MCR LL_HRTIM_TIM_GetBurstModeOption
- BMCR TABM LL_HRTIM_TIM_GetBurstModeOption
- BMCR TBBM LL_HRTIM_TIM_GetBurstModeOption
- BMCR TCBM LL_HRTIM_TIM_GetBurstModeOption
- BMCR TDBM LL_HRTIM_TIM_GetBurstModeOption
- BMCR TEBM LL_HRTIM_TIM_GetBurstModeOption

LL_HRTIM_TIM_ConfigBurstDMA

Function name

__STATIC_INLINE void LL_HRTIM_TIM_ConfigBurstDMA (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Registers)

Function description

Program which registers are to be written by Burst DMA transfers.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **Registers:** Registers to be updated by the DMA request

Reference Manual to LL API cross reference:

- BDMUPDR MTBM LL_HRTIM_TIM_ConfigBurstDMA
- BDMUPDR MICR LL_HRTIM_TIM_ConfigBurstDMA
- BDMUPDR MDIER LL_HRTIM_TIM_ConfigBurstDMA
- BDMUPDR MCNT LL_HRTIM_TIM_ConfigBurstDMA
- BDMUPDR MPER LL_HRTIM_TIM_ConfigBurstDMA
- BDMUPDR MREP LL_HRTIM_TIM_ConfigBurstDMA
- BDMUPDR MCMP1 LL_HRTIM_TIM_ConfigBurstDMA
- BDMUPDR MCMP2 LL_HRTIM_TIM_ConfigBurstDMA
- BDMUPDR MCMP3 LL_HRTIM_TIM_ConfigBurstDMA
- BDMUPDR MCMP4 LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxCR LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxICR LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxDIER LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxCNT LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxPER LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxREP LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxCMP1 LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxCMP2 LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxCMP3 LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxCMP4 LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxDTR LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxSET1R LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxRST1R LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxSET2R LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxRST2R LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxEEFR1 LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxEEFR2 LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxRSTR LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxOUTR LL_HRTIM_TIM_ConfigBurstDMA
- BDTxUPDR TIMxLTCH LL_HRTIM_TIM_ConfigBurstDMA

LL_HRTIM_TIM_GetCurrentPushPullStatus

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetCurrentPushPullStatus (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate on which output the signal is currently applied.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **CPPSTAT:** This parameter can be one of the following values:
  - LL_HRTIM_CPPSTAT_OUTPUT1
  - LL_HRTIM_CPPSTAT_OUTPUT2

Notes

- Only significant when the timer operates in push-pull mode.

Reference Manual to LL API cross reference:

- TIMxISR CPPSTAT LL_HRTIM_TIM_GetCurrentPushPullStatus

LL_HRTIM_TIM_GetIdlePushPullStatus

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetIdlePushPullStatus (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate on which output the signal was applied, in push-pull mode, balanced fault mode or delayed idle mode, when the protection was triggered.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **IPPSTAT:** This parameter can be one of the following values:
  - LL_HRTIM_IPPSTAT_OUTPUT1
  - LL_HRTIM_IPPSTAT_OUTPUT2

Reference Manual to LL API cross reference:

- TIMxISR IPPSTAT LL_HRTIM_TIM_GetIdlePushPullStatus

LL_HRTIM_TIM_SetEventFilter

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetEventFilter (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Event, uint32_t Filter)

Function description

Set the event filter for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_1
  - LL_HRTIM_EVENT_2
  - LL_HRTIM_EVENT_3
  - LL_HRTIM_EVENT_4
  - LL_HRTIM_EVENT_5
  - LL_HRTIM_EVENT_6
  - LL_HRTIM_EVENT_7
  - LL_HRTIM_EVENT_8
  - LL_HRTIM_EVENT_9
  - LL_HRTIM_EVENT_10
- **Filter:** This parameter can be one of the following values:
  - LL_HRTIM_EEFLTR_NONE
  - LL_HRTIM_EEFLTR_BLANKINGCMP1
  - LL_HRTIM_EEFLTR_BLANKINGCMP2
  - LL_HRTIM_EEFLTR_BLANKINGCMP3
  - LL_HRTIM_EEFLTR_BLANKINGCMP4
  - LL_HRTIM_EEFLTR_BLANKINGFLTR1
  - LL_HRTIM_EEFLTR_BLANKINGFLTR2
  - LL_HRTIM_EEFLTR_BLANKINGFLTR3
  - LL_HRTIM_EEFLTR_BLANKINGFLTR4
  - LL_HRTIM_EEFLTR_BLANKINGFLTR5
  - LL_HRTIM_EEFLTR_BLANKINGFLTR6
  - LL_HRTIM_EEFLTR_BLANKINGFLTR7
  - LL_HRTIM_EEFLTR_BLANKINGFLTR8
  - LL_HRTIM_EEFLTR_WINDOWINGCMP2
  - LL_HRTIM_EEFLTR_WINDOWINGCMP3
  - LL_HRTIM_EEFLTR_WINDOWINGTIM

Return values

- **None:**

Notes

- This function must not be called when the timer counter is enabled.

Reference Manual to LL API cross reference:

- EEFxR1 EE1LTCH LL_HRTIM_TIM_SetEventFilter
- EEFxR1 EE2LTCH LL_HRTIM_TIM_SetEventFilter
- EEFxR1 EE3LTCH LL_HRTIM_TIM_SetEventFilter
- EEFxR1 EE4LTCH LL_HRTIM_TIM_SetEventFilter
- EEFxR1 EE5LTCH LL_HRTIM_TIM_SetEventFilter
- EEFxR2 EE6LTCH LL_HRTIM_TIM_SetEventFilter
- EEFxR2 EE7LTCH LL_HRTIM_TIM_SetEventFilter
- EEFxR2 EE8LTCH LL_HRTIM_TIM_SetEventFilter
- EEFxR2 EE9LTCH LL_HRTIM_TIM_SetEventFilter
- EEFxR2 EE10LTCH LL_HRTIM_TIM_SetEventFilter

LL_HRTIM_TIM_GetEventFilter

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetEventFilter (const HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Event)

Function description

Get actual event filter settings for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_1
  - LL_HRTIM_EVENT_2
  - LL_HRTIM_EVENT_3
  - LL_HRTIM_EVENT_4
  - LL_HRTIM_EVENT_5
  - LL_HRTIM_EVENT_6
  - LL_HRTIM_EVENT_7
  - LL_HRTIM_EVENT_8
  - LL_HRTIM_EVENT_9
  - LL_HRTIM_EVENT_10

Return values

- **Filter:** This parameter can be one of the following values:
  - LL_HRTIM_EEFLTR_NONE
  - LL_HRTIM_EEFLTR_BLANKINGCMP1
  - LL_HRTIM_EEFLTR_BLANKINGCMP2
  - LL_HRTIM_EEFLTR_BLANKINGCMP3
  - LL_HRTIM_EEFLTR_BLANKINGCMP4
  - LL_HRTIM_EEFLTR_BLANKINGFLTR1
  - LL_HRTIM_EEFLTR_BLANKINGFLTR2
  - LL_HRTIM_EEFLTR_BLANKINGFLTR3
  - LL_HRTIM_EEFLTR_BLANKINGFLTR4
  - LL_HRTIM_EEFLTR_BLANKINGFLTR5
  - LL_HRTIM_EEFLTR_BLANKINGFLTR6
  - LL_HRTIM_EEFLTR_BLANKINGFLTR7
  - LL_HRTIM_EEFLTR_BLANKINGFLTR8
  - LL_HRTIM_EEFLTR_WINDOWINGCMP2
  - LL_HRTIM_EEFLTR_WINDOWINGCMP3
  - LL_HRTIM_EEFLTR_WINDOWINGTIM

Reference Manual to LL API cross reference:

- EEFxR1 EE1FLTR LL_HRTIM_TIM_GetEventFilter
- EEFxR1 EE2FLTR LL_HRTIM_TIM_GetEventFilter
- EEFxR1 EE3FLTR LL_HRTIM_TIM_GetEventFilter
- EEFxR1 EE4FLTR LL_HRTIM_TIM_GetEventFilter
- EEFxR1 EE5FLTR LL_HRTIM_TIM_GetEventFilter
- EEFxR2 EE6FLTR LL_HRTIM_TIM_GetEventFilter
- EEFxR2 EE7FLTR LL_HRTIM_TIM_GetEventFilter
- EEFxR2 EE8FLTR LL_HRTIM_TIM_GetEventFilter
- EEFxR2 EE9FLTR LL_HRTIM_TIM_GetEventFilter
- EEFxR2 EE10FLTR LL_HRTIM_TIM_GetEventFilter

LL_HRTIM_TIM_SetEventLatchStatus

Function name

__STATIC_INLINE void LL_HRTIM_TIM_SetEventLatchStatus (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Event, uint32_t LatchStatus)

Function description

Enable or disable event latch mechanism for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_1
  - LL_HRTIM_EVENT_2
  - LL_HRTIM_EVENT_3
  - LL_HRTIM_EVENT_4
  - LL_HRTIM_EVENT_5
  - LL_HRTIM_EVENT_6
  - LL_HRTIM_EVENT_7
  - LL_HRTIM_EVENT_8
  - LL_HRTIM_EVENT_9
  - LL_HRTIM_EVENT_10
- **LatchStatus:** This parameter can be one of the following values:
  - LL_HRTIM_EELATCH_DISABLED
  - LL_HRTIM_EELATCH_ENABLED

Return values

- **None:**

Notes

- This function must not be called when the timer counter is enabled.

Reference Manual to LL API cross reference:

- EEFxR1 EE1LTCH LL_HRTIM_TIM_SetEventLatchStatus
- EEFxR1 EE2LTCH LL_HRTIM_TIM_SetEventLatchStatus
- EEFxR1 EE3LTCH LL_HRTIM_TIM_SetEventLatchStatus
- EEFxR1 EE4LTCH LL_HRTIM_TIM_SetEventLatchStatus
- EEFxR1 EE5LTCH LL_HRTIM_TIM_SetEventLatchStatus
- EEFxR2 EE6LTCH LL_HRTIM_TIM_SetEventLatchStatus
- EEFxR2 EE7LTCH LL_HRTIM_TIM_SetEventLatchStatus
- EEFxR2 EE8LTCH LL_HRTIM_TIM_SetEventLatchStatus
- EEFxR2 EE9LTCH LL_HRTIM_TIM_SetEventLatchStatus
- EEFxR2 EE10LTCH LL_HRTIM_TIM_SetEventLatchStatus

LL_HRTIM_TIM_GetEventLatchStatus

Function name

__STATIC_INLINE uint32_t LL_HRTIM_TIM_GetEventLatchStatus (const HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Event)

Function description

Get actual event latch status for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_1
  - LL_HRTIM_EVENT_2
  - LL_HRTIM_EVENT_3
  - LL_HRTIM_EVENT_4
  - LL_HRTIM_EVENT_5
  - LL_HRTIM_EVENT_6
  - LL_HRTIM_EVENT_7
  - LL_HRTIM_EVENT_8
  - LL_HRTIM_EVENT_9
  - LL_HRTIM_EVENT_10

Return values

- **LatchStatus:** This parameter can be one of the following values:
  - LL_HRTIM_EELATCH_DISABLED
  - LL_HRTIM_EELATCH_ENABLED

Reference Manual to LL API cross reference:

- EEFxR1 EE1LTCH LL_HRTIM_TIM_GetEventLatchStatus
- EEFxR1 EE2LTCH LL_HRTIM_TIM_GetEventLatchStatus
- EEFxR1 EE3LTCH LL_HRTIM_TIM_GetEventLatchStatus
- EEFxR1 EE4LTCH LL_HRTIM_TIM_GetEventLatchStatus
- EEFxR1 EE5LTCH LL_HRTIM_TIM_GetEventLatchStatus
- EEFxR2 EE6LTCH LL_HRTIM_TIM_GetEventLatchStatus
- EEFxR2 EE7LTCH LL_HRTIM_TIM_GetEventLatchStatus
- EEFxR2 EE8LTCH LL_HRTIM_TIM_GetEventLatchStatus
- EEFxR2 EE9LTCH LL_HRTIM_TIM_GetEventLatchStatus
- EEFxR2 EE10LTCH LL_HRTIM_TIM_GetEventLatchStatus

LL_HRTIM_DT_Config

Function name

__STATIC_INLINE void LL_HRTIM_DT_Config (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Configuration)

Function description

Configure the dead time insertion feature for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_HRTIM_DT_PRESCALER_MUL8 or ... or LL_HRTIM_DT_PRESCALER_DIV16
  - LL_HRTIM_DT_RISING_POSITIVE or LL_HRTIM_DT_RISING_NEGATIVE
  - LL_HRTIM_DT_FALLING_POSITIVE or LL_HRTIM_DT_FALLING_NEGATIVE

Return values

- **None:**

Reference Manual to LL API cross reference:

- DTxR DTPRSC LL_HRTIM_DT_Config
- DTxR SDTF LL_HRTIM_DT_Config
- DTxR SDRT LL_HRTIM_DT_Config

LL_HRTIM_DT_SetPrescaler

Function name

__STATIC_INLINE void LL_HRTIM_DT_SetPrescaler (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Prescaler)

Function description

Set the deadtime prescaler value.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **Prescaler:** This parameter can be one of the following values:
  - LL_HRTIM_DT_PRESCALER_MUL8
  - LL_HRTIM_DT_PRESCALER_MUL4
  - LL_HRTIM_DT_PRESCALER_MUL2
  - LL_HRTIM_DT_PRESCALER_DIV1
  - LL_HRTIM_DT_PRESCALER_DIV2
  - LL_HRTIM_DT_PRESCALER_DIV4
  - LL_HRTIM_DT_PRESCALER_DIV8
  - LL_HRTIM_DT_PRESCALER_DIV16

Return values

- **None:**

Reference Manual to LL API cross reference:

- DTxR DTPRSC LL_HRTIM_DT_SetPrescaler

LL_HRTIM_DT_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_HRTIM_DT_GetPrescaler (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual deadtime prescaler value.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **Prescaler:** This parameter can be one of the following values:
  - LL_HRTIM_DT_PRESCALER_MUL8
  - LL_HRTIM_DT_PRESCALER_MUL4
  - LL_HRTIM_DT_PRESCALER_MUL2
  - LL_HRTIM_DT_PRESCALER_DIV1
  - LL_HRTIM_DT_PRESCALER_DIV2
  - LL_HRTIM_DT_PRESCALER_DIV4
  - LL_HRTIM_DT_PRESCALER_DIV8
  - LL_HRTIM_DT_PRESCALER_DIV16

Reference Manual to LL API cross reference:

- DTxR DTPRSC LL_HRTIM_DT_GetPrescaler

LL_HRTIM_DT_SetRisingValue

Function name

__STATIC_INLINE void LL_HRTIM_DT_SetRisingValue (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t RisingValue)

Function description

Set the deadtime rising value.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **RisingValue:** Value between 0 and 0x1FF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DTxR DTR LL_HRTIM_DT_SetRisingValue

LL_HRTIM_DT_GetRisingValue

Function name

__STATIC_INLINE uint32_t LL_HRTIM_DT_GetRisingValue (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual deadtime rising value.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **RisingValue:** Value between 0 and 0x1FF

Reference Manual to LL API cross reference:

- DTxR DTR LL_HRTIM_DT_GetRisingValue

LL_HRTIM_DT_SetRisingSign

Function name

__STATIC_INLINE void LL_HRTIM_DT_SetRisingSign (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t RisingSign)

Function description

Set the deadtime sign on rising edge.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **RisingSign:** This parameter can be one of the following values:
  - LL_HRTIM_DT_RISING_POSITIVE
  - LL_HRTIM_DT_RISING_NEGATIVE

Return values

- **None:**

Reference Manual to LL API cross reference:

- DTxR SDTR LL_HRTIM_DT_SetRisingSign

LL_HRTIM_DT_GetRisingSign

Function name

__STATIC_INLINE uint32_t LL_HRTIM_DT_GetRisingSign (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual deadtime sign on rising edge.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **RisingSign:** This parameter can be one of the following values:
  - LL_HRTIM_DT_RISING_POSITIVE
  - LL_HRTIM_DT_RISING_NEGATIVE

Reference Manual to LL API cross reference:

- DTxR SDTR LL_HRTIM_DT_GetRisingSign

LL_HRTIM_DT_SetFallingValue

Function name

__STATIC_INLINE void LL_HRTIM_DT_SetFallingValue (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t FallingValue)

Function description

Set the deadime falling value.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **FallingValue:** Value between 0 and 0x1FF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DTxR DTF LL_HRTIM_DT_SetFallingValue

LL_HRTIM_DT_GetFallingValue

Function name

__STATIC_INLINE uint32_t LL_HRTIM_DT_GetFallingValue (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual deadtime falling value.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **FallingValue:** Value between 0 and 0x1FF

Reference Manual to LL API cross reference:

- DTxR DTF LL_HRTIM_DT_GetFallingValue

LL_HRTIM_DT_SetFallingSign

Function name

__STATIC_INLINE void LL_HRTIM_DT_SetFallingSign (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t FallingSign)

Function description

Set the deadtime sign on falling edge.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **FallingSign:** This parameter can be one of the following values:
  - LL_HRTIM_DT_FALLING_POSITIVE
  - LL_HRTIM_DT_FALLING_NEGATIVE

Return values

- **None:**

Reference Manual to LL API cross reference:

- DTxR SDTF LL_HRTIM_DT_SetFallingSign

LL_HRTIM_DT_GetFallingSign

Function name

__STATIC_INLINE uint32_t LL_HRTIM_DT_GetFallingSign (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual deadtime sign on falling edge.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **FallingSign:** This parameter can be one of the following values:
  - LL_HRTIM_DT_FALLING_POSITIVE
  - LL_HRTIM_DT_FALLING_NEGATIVE

Reference Manual to LL API cross reference:

- DTxR SDTF LL_HRTIM_DT_GetFallingSign

LL_HRTIM_DT_LockRising

Function name

__STATIC_INLINE void LL_HRTIM_DT_LockRising (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Lock the deadtime value and sign on rising edge.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- DTxR DTRLK LL_HRTIM_DT_LockRising

LL_HRTIM_DT_LockRisingSign

Function name

__STATIC_INLINE void LL_HRTIM_DT_LockRisingSign (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Lock the deadtime sign on rising edge.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- DTxR DTRSLK LL_HRTIM_DT_LockRisingSign

LL_HRTIM_DT_LockFalling

Function name

__STATIC_INLINE void LL_HRTIM_DT_LockFalling (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Lock the deadtime value and sign on falling edge.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- DTxR DTFLK LL_HRTIM_DT_LockFalling

LL_HRTIM_DT_LockFallingSign

Function name

__STATIC_INLINE void LL_HRTIM_DT_LockFallingSign (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Lock the deadtime sign on falling edge.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- DTxR DTFSLK LL_HRTIM_DT_LockFallingSign

LL_HRTIM_CHP_Config

Function name

__STATIC_INLINE void LL_HRTIM_CHP_Config (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Configuration)

Function description

Configure the chopper stage for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_HRTIM_CHP_PRESCALER_DIV16 or ... or LL_HRTIM_CHP_PRESCALER_DIV256
  - LL_HRTIM_CHP_DUTYCYCLE_0 or ... or LL_HRTIM_CHP_DUTYCYCLE_875
  - LL_HRTIM_CHP_PULSEWIDTH_16 or ... or LL_HRTIM_CHP_PULSEWIDTH_256

Return values

- **None:**

Notes

- This function must not be called if the chopper mode is already enabled for one of the timer outputs.

Reference Manual to LL API cross reference:

- CHPxR CARFRQ LL_HRTIM_CHP_Config
- CHPxR CARDTY LL_HRTIM_CHP_Config
- CHPxR STRTPW LL_HRTIM_CHP_Config

LL_HRTIM_CHP_SetPrescaler

Function name

__STATIC_INLINE void LL_HRTIM_CHP_SetPrescaler (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t Prescaler)

Function description

Set prescaler determining the carrier frequency to be added on top of the timer output signals when chopper mode is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **Prescaler:** This parameter can be one of the following values:
  - LL_HRTIM_CHP_PRESCALER_DIV16
  - LL_HRTIM_CHP_PRESCALER_DIV32
  - LL_HRTIM_CHP_PRESCALER_DIV48
  - LL_HRTIM_CHP_PRESCALER_DIV64
  - LL_HRTIM_CHP_PRESCALER_DIV80
  - LL_HRTIM_CHP_PRESCALER_DIV96
  - LL_HRTIM_CHP_PRESCALER_DIV112
  - LL_HRTIM_CHP_PRESCALER_DIV128
  - LL_HRTIM_CHP_PRESCALER_DIV144
  - LL_HRTIM_CHP_PRESCALER_DIV160
  - LL_HRTIM_CHP_PRESCALER_DIV176
  - LL_HRTIM_CHP_PRESCALER_DIV192
  - LL_HRTIM_CHP_PRESCALER_DIV208
  - LL_HRTIM_CHP_PRESCALER_DIV224
  - LL_HRTIM_CHP_PRESCALER_DIV240
  - LL_HRTIM_CHP_PRESCALER_DIV256

Return values

- **None:**

Notes

- This function must not be called if the chopper mode is already enabled for one of the timer outputs.

Reference Manual to LL API cross reference:

- CHPxR CARFRQ LL_HRTIM_CHP_SetPrescaler

LL_HRTIM_CHP_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_HRTIM_CHP_GetPrescaler (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual chopper stage prescaler value.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **Prescaler:** This parameter can be one of the following values:
  - LL_HRTIM_CHP_PRESCALER_DIV16
  - LL_HRTIM_CHP_PRESCALER_DIV32
  - LL_HRTIM_CHP_PRESCALER_DIV48
  - LL_HRTIM_CHP_PRESCALER_DIV64
  - LL_HRTIM_CHP_PRESCALER_DIV80
  - LL_HRTIM_CHP_PRESCALER_DIV96
  - LL_HRTIM_CHP_PRESCALER_DIV112
  - LL_HRTIM_CHP_PRESCALER_DIV128
  - LL_HRTIM_CHP_PRESCALER_DIV144
  - LL_HRTIM_CHP_PRESCALER_DIV160
  - LL_HRTIM_CHP_PRESCALER_DIV176
  - LL_HRTIM_CHP_PRESCALER_DIV192
  - LL_HRTIM_CHP_PRESCALER_DIV208
  - LL_HRTIM_CHP_PRESCALER_DIV224
  - LL_HRTIM_CHP_PRESCALER_DIV240
  - LL_HRTIM_CHP_PRESCALER_DIV256

Reference Manual to LL API cross reference:

- CHPxR CARFRQ LL_HRTIM_CHP_GetPrescaler

LL_HRTIM_CHP_SetDutyCycle

Function name

__STATIC_INLINE void LL_HRTIM_CHP_SetDutyCycle (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t DutyCycle)

Function description

Set the chopper duty cycle.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **DutyCycle:** This parameter can be one of the following values:
  - LL_HRTIM_CHP_DUTYCYCLE_0
  - LL_HRTIM_CHP_DUTYCYCLE_125
  - LL_HRTIM_CHP_DUTYCYCLE_250
  - LL_HRTIM_CHP_DUTYCYCLE_375
  - LL_HRTIM_CHP_DUTYCYCLE_500
  - LL_HRTIM_CHP_DUTYCYCLE_625
  - LL_HRTIM_CHP_DUTYCYCLE_750
  - LL_HRTIM_CHP_DUTYCYCLE_875

Return values

- **None:**

Notes

- Duty cycle can be adjusted by 1/8 step (from 0/8 up to 7/8)
- This function must not be called if the chopper mode is already enabled for one of the timer outputs.

Reference Manual to LL API cross reference:

- CHPxR CARDTY LL_HRTIM_CHP_SetDutyCycle

LL_HRTIM_CHP_GetDutyCycle

Function name

__STATIC_INLINE uint32_t LL_HRTIM_CHP_GetDutyCycle (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual chopper duty cycle.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **DutyCycle:** This parameter can be one of the following values:
  - LL_HRTIM_CHP_DUTYCYCLE_0
  - LL_HRTIM_CHP_DUTYCYCLE_125
  - LL_HRTIM_CHP_DUTYCYCLE_250
  - LL_HRTIM_CHP_DUTYCYCLE_375
  - LL_HRTIM_CHP_DUTYCYCLE_500
  - LL_HRTIM_CHP_DUTYCYCLE_625
  - LL_HRTIM_CHP_DUTYCYCLE_750
  - LL_HRTIM_CHP_DUTYCYCLE_875

Reference Manual to LL API cross reference:

- CHPxR CARDTY LL_HRTIM_CHP_GetDutyCycle

LL_HRTIM_CHP_SetPulseWidth

Function name

__STATIC_INLINE void LL_HRTIM_CHP_SetPulseWidth (HRTIM_TypeDef * HRTIMx, uint32_t Timer, uint32_t PulseWidth)

Function description

Set the start pulse width.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E
- **PulseWidth:** This parameter can be one of the following values:
  - LL_HRTIM_CHP_PULSEWIDTH_16
  - LL_HRTIM_CHP_PULSEWIDTH_32
  - LL_HRTIM_CHP_PULSEWIDTH_48
  - LL_HRTIM_CHP_PULSEWIDTH_64
  - LL_HRTIM_CHP_PULSEWIDTH_80
  - LL_HRTIM_CHP_PULSEWIDTH_96
  - LL_HRTIM_CHP_PULSEWIDTH_112
  - LL_HRTIM_CHP_PULSEWIDTH_128
  - LL_HRTIM_CHP_PULSEWIDTH_144
  - LL_HRTIM_CHP_PULSEWIDTH_160
  - LL_HRTIM_CHP_PULSEWIDTH_176
  - LL_HRTIM_CHP_PULSEWIDTH_192
  - LL_HRTIM_CHP_PULSEWIDTH_208
  - LL_HRTIM_CHP_PULSEWIDTH_224
  - LL_HRTIM_CHP_PULSEWIDTH_240
  - LL_HRTIM_CHP_PULSEWIDTH_256

Return values

- **None:**

Notes

- This function must not be called if the chopper mode is already enabled for one of the timer outputs.

Reference Manual to LL API cross reference:

- CHPxR STRPW LL_HRTIM_CHP_SetPulseWidth

LL_HRTIM_CHP_GetPulseWidth

Function name

__STATIC_INLINE uint32_t LL_HRTIM_CHP_GetPulseWidth (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Get actual start pulse width.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **PulseWidth:** This parameter can be one of the following values:
  - LL_HRTIM_CHP_PULSEWIDTH_16
  - LL_HRTIM_CHP_PULSEWIDTH_32
  - LL_HRTIM_CHP_PULSEWIDTH_48
  - LL_HRTIM_CHP_PULSEWIDTH_64
  - LL_HRTIM_CHP_PULSEWIDTH_80
  - LL_HRTIM_CHP_PULSEWIDTH_96
  - LL_HRTIM_CHP_PULSEWIDTH_112
  - LL_HRTIM_CHP_PULSEWIDTH_128
  - LL_HRTIM_CHP_PULSEWIDTH_144
  - LL_HRTIM_CHP_PULSEWIDTH_160
  - LL_HRTIM_CHP_PULSEWIDTH_176
  - LL_HRTIM_CHP_PULSEWIDTH_192
  - LL_HRTIM_CHP_PULSEWIDTH_208
  - LL_HRTIM_CHP_PULSEWIDTH_224
  - LL_HRTIM_CHP_PULSEWIDTH_240
  - LL_HRTIM_CHP_PULSEWIDTH_256

Reference Manual to LL API cross reference:

- CHPxR STRPW LL_HRTIM_CHP_GetPulseWidth

LL_HRTIM_OUT_SetOutputSetSrc

Function name

__STATIC_INLINE void LL_HRTIM_OUT_SetOutputSetSrc (HRTIM_TypeDef * HRTIMx, uint32_t Output, uint32_t SetSrc)

Function description

Set the timer output set source.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
- **SetSrc:** This parameter can be a combination of the following values:
  - LL_HRTIM_CROSSBAR_NONE
  - LL_HRTIM_CROSSBAR_RESYNC
  - LL_HRTIM_CROSSBAR_TIMPER
  - LL_HRTIM_CROSSBAR_TIMCMP1
  - LL_HRTIM_CROSSBAR_TIMCMP2
  - LL_HRTIM_CROSSBAR_TIMCMP3
  - LL_HRTIM_CROSSBAR_TIMCMP4
  - LL_HRTIM_CROSSBAR_MASTERPER
  - LL_HRTIM_CROSSBAR_MASTERCMP1
  - LL_HRTIM_CROSSBAR_MASTERCMP2
  - LL_HRTIM_CROSSBAR_MASTERCMP3
  - LL_HRTIM_CROSSBAR_MASTERCMP4
  - LL_HRTIM_CROSSBAR_TIMEV_1
  - LL_HRTIM_CROSSBAR_TIMEV_2
  - LL_HRTIM_CROSSBAR_TIMEV_3
  - LL_HRTIM_CROSSBAR_TIMEV_4
  - LL_HRTIM_CROSSBAR_TIMEV_5
  - LL_HRTIM_CROSSBAR_TIMEV_6
  - LL_HRTIM_CROSSBAR_TIMEV_7
  - LL_HRTIM_CROSSBAR_TIMEV_8
  - LL_HRTIM_CROSSBAR_TIMEV_9
  - LL_HRTIM_CROSSBAR_EEV_1
  - LL_HRTIM_CROSSBAR_EEV_2
  - LL_HRTIM_CROSSBAR_EEV_3
  - LL_HRTIM_CROSSBAR_EEV_4
  - LL_HRTIM_CROSSBAR_EEV_5
  - LL_HRTIM_CROSSBAR_EEV_6
  - LL_HRTIM_CROSSBAR_EEV_7
  - LL_HRTIM_CROSSBAR_EEV_8
  - LL_HRTIM_CROSSBAR_EEV_9
  - LL_HRTIM_CROSSBAR_EEV_10
  - LL_HRTIM_CROSSBAR_UPDATE

Return values

- **None:**

Reference Manual to LL API cross reference:

- SETx1R SST LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R RESYNC LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R PER LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R CMP1 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R CMP2 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R CMP3 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R CMP4 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R MSTPER LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R MSTCMP1 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R MSTCMP2 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R MSTCMP3 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R MSTCMP4 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT1 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT2 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT3 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT4 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT5 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT6 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT7 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT8 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT9 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT1 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT2 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT3 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT4 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT5 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT6 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT7 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT8 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT9 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT10 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R UPDATE LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R SST LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R RESYNC LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R PER LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R CMP1 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R CMP2 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R CMP3 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R CMP4 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R MSTPER LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R MSTCMP1 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R MSTCMP2 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R MSTCMP3 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R MSTCMP4 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT1 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT2 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT3 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT4 LL_HRTIM_OUT_SetOutputSetSrc

- SETx1R TIMEVNT5 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT6 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT7 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT8 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R TIMEVNT9 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT1 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT2 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT3 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT4 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT5 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT6 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT7 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT8 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT9 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R EXEVNT10 LL_HRTIM_OUT_SetOutputSetSrc
- SETx1R UPDATE LL_HRTIM_OUT_SetOutputSetSrc

LL_HRTIM_OUT_GetOutputSetSrc

Function name

__STATIC_INLINE uint32_t LL_HRTIM_OUT_GetOutputSetSrc (const HRTIM_TypeDef * HRTIMx, uint32_t Output)

Function description

Get the timer output set source.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2

Return values

- **SetSrc:** This parameter can be a combination of the following values:
  - LL_HRTIM_CROSSBAR_NONE
  - LL_HRTIM_CROSSBAR_RESYNC
  - LL_HRTIM_CROSSBAR_TIMPER
  - LL_HRTIM_CROSSBAR_TIMCMP1
  - LL_HRTIM_CROSSBAR_TIMCMP2
  - LL_HRTIM_CROSSBAR_TIMCMP3
  - LL_HRTIM_CROSSBAR_TIMCMP4
  - LL_HRTIM_CROSSBAR_MASTERPER
  - LL_HRTIM_CROSSBAR_MASTERCMP1
  - LL_HRTIM_CROSSBAR_MASTERCMP2
  - LL_HRTIM_CROSSBAR_MASTERCMP3
  - LL_HRTIM_CROSSBAR_MASTERCMP4
  - LL_HRTIM_CROSSBAR_TIMEV_1
  - LL_HRTIM_CROSSBAR_TIMEV_2
  - LL_HRTIM_CROSSBAR_TIMEV_3
  - LL_HRTIM_CROSSBAR_TIMEV_4
  - LL_HRTIM_CROSSBAR_TIMEV_5
  - LL_HRTIM_CROSSBAR_TIMEV_6
  - LL_HRTIM_CROSSBAR_TIMEV_7
  - LL_HRTIM_CROSSBAR_TIMEV_8
  - LL_HRTIM_CROSSBAR_TIMEV_9
  - LL_HRTIM_CROSSBAR_EEV_1
  - LL_HRTIM_CROSSBAR_EEV_2
  - LL_HRTIM_CROSSBAR_EEV_3
  - LL_HRTIM_CROSSBAR_EEV_4
  - LL_HRTIM_CROSSBAR_EEV_5
  - LL_HRTIM_CROSSBAR_EEV_6
  - LL_HRTIM_CROSSBAR_EEV_7
  - LL_HRTIM_CROSSBAR_EEV_8
  - LL_HRTIM_CROSSBAR_EEV_9
  - LL_HRTIM_CROSSBAR_EEV_10
  - LL_HRTIM_CROSSBAR_UPDATE

Reference Manual to LL API cross reference:

- SETx1R SST LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R RESYNC LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R PER LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R CMP1 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R CMP2 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R CMP3 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R CMP4 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R MSTPER LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R MSTCMP1 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R MSTCMP2 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R MSTCMP3 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R MSTCMP4 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT1 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT2 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT3 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT4 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT5 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT6 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT7 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT8 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT9 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT1 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT2 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT3 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT4 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT5 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT6 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT7 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT8 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT9 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT10 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R UPDATE LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R SST LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R RESYNC LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R PER LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R CMP1 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R CMP2 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R CMP3 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R CMP4 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R MSTPER LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R MSTCMP1 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R MSTCMP2 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R MSTCMP3 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R MSTCMP4 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT1 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT2 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT3 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT4 LL_HRTIM_OUT_GetOutputSetSrc

- SETx1R TIMEVNT5 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT6 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT7 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT8 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R TIMEVNT9 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT1 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT2 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT3 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT4 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT5 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT6 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT7 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT8 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT9 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R EXEVNT10 LL_HRTIM_OUT_GetOutputSetSrc
- SETx1R UPDATE LL_HRTIM_OUT_GetOutputSetSrc

LL_HRTIM_OUT_SetOutputResetSrc

Function name

__STATIC_INLINE void LL_HRTIM_OUT_SetOutputResetSrc (HRTIM_TypeDef * HRTIMx, uint32_t Output, uint32_t ResetSrc)

Function description

Set the timer output reset source.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
- **ResetSrc:** This parameter can be a combination of the following values:
  - LL_HRTIM_CROSSBAR_NONE
  - LL_HRTIM_CROSSBAR_RESYNC
  - LL_HRTIM_CROSSBAR_TIMPER
  - LL_HRTIM_CROSSBAR_TIMCMP1
  - LL_HRTIM_CROSSBAR_TIMCMP2
  - LL_HRTIM_CROSSBAR_TIMCMP3
  - LL_HRTIM_CROSSBAR_TIMCMP4
  - LL_HRTIM_CROSSBAR_MASTERPER
  - LL_HRTIM_CROSSBAR_MASTERCMP1
  - LL_HRTIM_CROSSBAR_MASTERCMP2
  - LL_HRTIM_CROSSBAR_MASTERCMP3
  - LL_HRTIM_CROSSBAR_MASTERCMP4
  - LL_HRTIM_CROSSBAR_TIMEV_1
  - LL_HRTIM_CROSSBAR_TIMEV_2
  - LL_HRTIM_CROSSBAR_TIMEV_3
  - LL_HRTIM_CROSSBAR_TIMEV_4
  - LL_HRTIM_CROSSBAR_TIMEV_5
  - LL_HRTIM_CROSSBAR_TIMEV_6
  - LL_HRTIM_CROSSBAR_TIMEV_7
  - LL_HRTIM_CROSSBAR_TIMEV_8
  - LL_HRTIM_CROSSBAR_TIMEV_9
  - LL_HRTIM_CROSSBAR_EEV_1
  - LL_HRTIM_CROSSBAR_EEV_2
  - LL_HRTIM_CROSSBAR_EEV_3
  - LL_HRTIM_CROSSBAR_EEV_4
  - LL_HRTIM_CROSSBAR_EEV_5
  - LL_HRTIM_CROSSBAR_EEV_6
  - LL_HRTIM_CROSSBAR_EEV_7
  - LL_HRTIM_CROSSBAR_EEV_8
  - LL_HRTIM_CROSSBAR_EEV_9
  - LL_HRTIM_CROSSBAR_EEV_10
  - LL_HRTIM_CROSSBAR_UPDATE

Return values

- **None:**

Reference Manual to LL API cross reference:

- RSTx1R RST LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R RESYNC LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R PER LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R CMP1 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R CMP2 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R CMP3 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R CMP4 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R MSTPER LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R MSTCMP1 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R MSTCMP2 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R MSTCMP3 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R MSTCMP4 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT1 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT2 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT3 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT4 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT5 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT6 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT7 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT8 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT9 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT1 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT2 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT3 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT4 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT5 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT6 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT7 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT8 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT9 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT10 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R UPDATE LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R RST LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R RESYNC LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R PER LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R CMP1 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R CMP2 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R CMP3 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R CMP4 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R MSTPER LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R MSTCMP1 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R MSTCMP2 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R MSTCMP3 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R MSTCMP4 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT1 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT2 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT3 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT4 LL_HRTIM_OUT_SetOutputResetSrc

- RSTx1R TIMEVNT5 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT6 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT7 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT8 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R TIMEVNT9 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT1 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT2 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT3 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT4 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT5 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT6 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT7 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT8 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT9 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R EXEVNT10 LL_HRTIM_OUT_SetOutputResetSrc
- RSTx1R UPDATE LL_HRTIM_OUT_SetOutputResetSrc

LL_HRTIM_OUT_GetOutputResetSrc

Function name

__STATIC_INLINE uint32_t LL_HRTIM_OUT_GetOutputResetSrc (const HRTIM_TypeDef * HRTIMx, uint32_t Output)

Function description

Get the timer output set source.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2

Return values

- **ResetSrc:** This parameter can be a combination of the following values:
  - LL_HRTIM_CROSSBAR_NONE
  - LL_HRTIM_CROSSBAR_RESYNC
  - LL_HRTIM_CROSSBAR_TIMPER
  - LL_HRTIM_CROSSBAR_TIMCMP1
  - LL_HRTIM_CROSSBAR_TIMCMP2
  - LL_HRTIM_CROSSBAR_TIMCMP3
  - LL_HRTIM_CROSSBAR_TIMCMP4
  - LL_HRTIM_CROSSBAR_MASTERPER
  - LL_HRTIM_CROSSBAR_MASTERCMP1
  - LL_HRTIM_CROSSBAR_MASTERCMP2
  - LL_HRTIM_CROSSBAR_MASTERCMP3
  - LL_HRTIM_CROSSBAR_MASTERCMP4
  - LL_HRTIM_CROSSBAR_TIMEV_1
  - LL_HRTIM_CROSSBAR_TIMEV_2
  - LL_HRTIM_CROSSBAR_TIMEV_3
  - LL_HRTIM_CROSSBAR_TIMEV_4
  - LL_HRTIM_CROSSBAR_TIMEV_5
  - LL_HRTIM_CROSSBAR_TIMEV_6
  - LL_HRTIM_CROSSBAR_TIMEV_7
  - LL_HRTIM_CROSSBAR_TIMEV_8
  - LL_HRTIM_CROSSBAR_TIMEV_9
  - LL_HRTIM_CROSSBAR_EEV_1
  - LL_HRTIM_CROSSBAR_EEV_2
  - LL_HRTIM_CROSSBAR_EEV_3
  - LL_HRTIM_CROSSBAR_EEV_4
  - LL_HRTIM_CROSSBAR_EEV_5
  - LL_HRTIM_CROSSBAR_EEV_6
  - LL_HRTIM_CROSSBAR_EEV_7
  - LL_HRTIM_CROSSBAR_EEV_8
  - LL_HRTIM_CROSSBAR_EEV_9
  - LL_HRTIM_CROSSBAR_EEV_10
  - LL_HRTIM_CROSSBAR_UPDATE

Reference Manual to LL API cross reference:

- RSTx1R RST LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R RESYNC LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R PER LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R CMP1 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R CMP2 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R CMP3 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R CMP4 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R MSTPER LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R MSTCMP1 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R MSTCMP2 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R MSTCMP3 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R MSTCMP4 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT1 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT2 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT3 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT4 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT5 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT6 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT7 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT8 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT9 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT1 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT2 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT3 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT4 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT5 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT6 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT7 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT8 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT9 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT10 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R UPDATE LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R RST LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R RESYNC LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R PER LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R CMP1 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R CMP2 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R CMP3 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R CMP4 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R MSTPER LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R MSTCMP1 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R MSTCMP2 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R MSTCMP3 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R MSTCMP4 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT1 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT2 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT3 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT4 LL_HRTIM_OUT_GetOutputResetSrc

- RSTx1R TIMEVNT5 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT6 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT7 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT8 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R TIMEVNT9 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT1 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT2 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT3 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT4 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT5 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT6 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT7 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT8 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT9 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R EXEVNT10 LL_HRTIM_OUT_GetOutputResetSrc
- RSTx1R UPDATE LL_HRTIM_OUT_GetOutputResetSrc

LL_HRTIM_OUT_Config

Function name

__STATIC_INLINE void LL_HRTIM_OUT_Config (HRTIM_TypeDef * HRTIMx, uint32_t Output, uint32_t Configuration)

Function description

Configure a timer output.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_HRTIM_OUT_POSITIVE_POLARITY or LL_HRTIM_OUT_NEGATIVE_POLARITY
  - LL_HRTIM_OUT_NO_IDLE or LL_HRTIM_OUT_IDLE_WHEN_BURST
  - LL_HRTIM_OUT_IDLELEVEL_INACTIVE or LL_HRTIM_OUT_IDLELEVEL_ACTIVE
  - LL_HRTIM_OUT_FAULTSTATE_NO_ACTION or LL_HRTIM_OUT_FAULTSTATE_ACTIVE or LL_HRTIM_OUT_FAULTSTATE_INACTIVE or LL_HRTIM_OUT_FAULTSTATE_HIGHZ
  - LL_HRTIM_OUT_CHOPPERMODE_DISABLED or LL_HRTIM_OUT_CHOPPERMODE_ENABLED
  - LL_HRTIM_OUT_BM_ENTRYMODE_REGULAR or LL_HRTIM_OUT_BM_ENTRYMODE_DELAYED

Return values

- **None:**

Reference Manual to LL API cross reference:

- OUTxR POL1 LL_HRTIM_OUT_Config
- OUTxR IDLEM1 LL_HRTIM_OUT_Config
- OUTxR IDLES1 LL_HRTIM_OUT_Config
- OUTxR FAULT1 LL_HRTIM_OUT_Config
- OUTxR CHP1 LL_HRTIM_OUT_Config
- OUTxR DIDL1 LL_HRTIM_OUT_Config
- OUTxR POL2 LL_HRTIM_OUT_Config
- OUTxR IDLEM2 LL_HRTIM_OUT_Config
- OUTxR IDLES2 LL_HRTIM_OUT_Config
- OUTxR FAULT2 LL_HRTIM_OUT_Config
- OUTxR CHP2 LL_HRTIM_OUT_Config
- OUTxR DIDL2 LL_HRTIM_OUT_Config

LL_HRTIM_OUT_SetPolarity

Function name

__STATIC_INLINE void LL_HRTIM_OUT_SetPolarity (HRTIM_TypeDef * HRTIMx, uint32_t Output, uint32_t Polarity)

Function description

Set the polarity of a timer output.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
- **Polarity:** This parameter can be one of the following values:
  - LL_HRTIM_OUT_POSITIVE_POLARITY
  - LL_HRTIM_OUT_NEGATIVE_POLARITY

Return values

- **None:**

Reference Manual to LL API cross reference:

- OUTxR POL1 LL_HRTIM_OUT_SetPolarity
- OUTxR POL2 LL_HRTIM_OUT_SetPolarity

LL_HRTIM_OUT_GetPolarity

Function name

__STATIC_INLINE uint32_t LL_HRTIM_OUT_GetPolarity (const HRTIM_TypeDef * HRTIMx, uint32_t Output)

Function description

Get actual polarity of the timer output.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2

Return values

- **Polarity:** This parameter can be one of the following values:
  - LL_HRTIM_OUT_POSITIVE_POLARITY
  - LL_HRTIM_OUT_NEGATIVE_POLARITY

Reference Manual to LL API cross reference:

- OUTxR POL1 LL_HRTIM_OUT_GetPolarity
- OUTxR POL2 LL_HRTIM_OUT_GetPolarity

LL_HRTIM_OUT_SetIdleMode

Function name

__STATIC_INLINE void LL_HRTIM_OUT_SetIdleMode (HRTIM_TypeDef * HRTIMx, uint32_t Output, uint32_t IdleMode)

Function description

Set the output IDLE mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
- **IdleMode:** This parameter can be one of the following values:
  - LL_HRTIM_OUT_NO_IDLE
  - LL_HRTIM_OUT_IDLE_WHEN_BURST

Return values

- **None:**

Notes

- This function must not be called when the burst mode is active

Reference Manual to LL API cross reference:

- OUTxR IDLEM1 LL_HRTIM_OUT_SetIdleMode
- OUTxR IDLEM2 LL_HRTIM_OUT_SetIdleMode

LL_HRTIM_OUT_GetIdleMode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_OUT_GetIdleMode (const HRTIM_TypeDef * HRTIMx, uint32_t Output)

Function description

Get actual output IDLE mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2

Return values

- **IdleMode:** This parameter can be one of the following values:
  - LL_HRTIM_OUT_NO_IDLE
  - LL_HRTIM_OUT_IDLE_WHEN_BURST

Reference Manual to LL API cross reference:

- OUTxR IDLEM1 LL_HRTIM_OUT_GetIdleMode
- OUTxR IDLEM2 LL_HRTIM_OUT_GetIdleMode

LL_HRTIM_OUT_SetIdleLevel

Function name

__STATIC_INLINE void LL_HRTIM_OUT_SetIdleLevel (HRTIM_TypeDef * HRTIMx, uint32_t Output, uint32_t IdleLevel)

Function description

Set the output IDLE level.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
- **IdleLevel:** This parameter can be one of the following values:
  - LL_HRTIM_OUT_IDLELEVEL_INACTIVE
  - LL_HRTIM_OUT_IDLELEVEL_ACTIVE

Return values

- **None:**

Notes

- This function must be called prior enabling the timer.
- Idle level isn't relevant when the output idle mode is set to LL_HRTIM_OUT_NO_IDLE.

Reference Manual to LL API cross reference:

- OUTxR IDLES1 LL_HRTIM_OUT_SetIdleLevel
- OUTxR IDLES2 LL_HRTIM_OUT_SetIdleLevel

LL_HRTIM_OUT_GetIdleLevel

Function name

__STATIC_INLINE uint32_t LL_HRTIM_OUT_GetIdleLevel (const HRTIM_TypeDef * HRTIMx, uint32_t Output)

Function description

Get actual output IDLE level.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2

Return values

- **IdleLevel:** This parameter can be one of the following values:
  - LL_HRTIM_OUT_IDLELEVEL_INACTIVE
  - LL_HRTIM_OUT_IDLELEVEL_ACTIVE

Reference Manual to LL API cross reference:

- OUTxR IDLES1 LL_HRTIM_OUT_GetIdleLevel
- OUTxR IDLES2 LL_HRTIM_OUT_GetIdleLevel

LL_HRTIM_OUT_SetFaultState

Function name

__STATIC_INLINE void LL_HRTIM_OUT_SetFaultState (HRTIM_TypeDef * HRTIMx, uint32_t Output, uint32_t FaultState)

Function description

Set the output FAULT state.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
- **FaultState:** This parameter can be one of the following values:
  - LL_HRTIM_OUT_FAULTSTATE_NO_ACTION
  - LL_HRTIM_OUT_FAULTSTATE_ACTIVE
  - LL_HRTIM_OUT_FAULTSTATE_INACTIVE
  - LL_HRTIM_OUT_FAULTSTATE_HIGHZ

Return values

- **None:**

Notes

- This function must not called when the timer is enabled and a fault channel is enabled at timer level.

Reference Manual to LL API cross reference:

- OUTxR FAULT1 LL_HRTIM_OUT_SetFaultState
- OUTxR FAULT2 LL_HRTIM_OUT_SetFaultState

LL_HRTIM_OUT_GetFaultState

Function name

__STATIC_INLINE uint32_t LL_HRTIM_OUT_GetFaultState (const HRTIM_TypeDef * HRTIMx, uint32_t Output)

Function description

Get actual FAULT state.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2

Return values

- **FaultState:** This parameter can be one of the following values:
  - LL_HRTIM_OUT_FAULTSTATE_NO_ACTION
  - LL_HRTIM_OUT_FAULTSTATE_ACTIVE
  - LL_HRTIM_OUT_FAULTSTATE_INACTIVE
  - LL_HRTIM_OUT_FAULTSTATE_HIGHZ

Reference Manual to LL API cross reference:

- OUTxR FAULT1 LL_HRTIM_OUT_GetFaultState
- OUTxR FAULT2 LL_HRTIM_OUT_GetFaultState

LL_HRTIM_OUT_SetChopperMode

Function name

__STATIC_INLINE void LL_HRTIM_OUT_SetChopperMode (HRTIM_TypeDef * HRTIMx, uint32_t Output, uint32_t ChopperMode)

Function description

Set the output chopper mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
- **ChopperMode:** This parameter can be one of the following values:
  - LL_HRTIM_OUT_CHOPPERMODE_DISABLED
  - LL_HRTIM_OUT_CHOPPERMODE_ENABLED

Return values

- **None:**

Notes

- This function must not called when the timer is enabled.

Reference Manual to LL API cross reference:

- OUTxR CHP1 LL_HRTIM_OUT_SetChopperMode
- OUTxR CHP2 LL_HRTIM_OUT_SetChopperMode

LL_HRTIM_OUT_GetChopperMode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_OUT_GetChopperMode (const HRTIM_TypeDef * HRTIMx, uint32_t Output)

Function description

Get actual output chopper mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2

Return values

- **ChopperMode:** This parameter can be one of the following values:
  - LL_HRTIM_OUT_CHOPPERMODE_DISABLED
  - LL_HRTIM_OUT_CHOPPERMODE_ENABLED

Reference Manual to LL API cross reference:

- OUTxR CHP1 LL_HRTIM_OUT_GetChopperMode
- OUTxR CHP2 LL_HRTIM_OUT_GetChopperMode

LL_HRTIM_OUT_SetBMEntryMode

Function name

__STATIC_INLINE void LL_HRTIM_OUT_SetBMEntryMode (HRTIM_TypeDef * HRTIMx, uint32_t Output, uint32_t BMEntryMode)

Function description

Set the output burst mode entry mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
- **BMEntryMode:** This parameter can be one of the following values:
  - LL_HRTIM_OUT_BM_ENTRYMODE_REGULAR
  - LL_HRTIM_OUT_BM_ENTRYMODE_DELAYED

Return values

- **None:**

Notes

- This function must not called when the timer is enabled.

Reference Manual to LL API cross reference:

- OUTxR DIDL1 LL_HRTIM_OUT_SetBMEntryMode
- OUTxR DIDL2 LL_HRTIM_OUT_SetBMEntryMode

LL_HRTIM_OUT_GetBMEntryMode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_OUT_GetBMEntryMode (const HRTIM_TypeDef * HRTIMx, uint32_t Output)

Function description

Get actual output burst mode entry mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2

Return values

- **BMEntryMode:** This parameter can be one of the following values:
  - LL_HRTIM_OUT_BM_ENTRYMODE_REGULAR
  - LL_HRTIM_OUT_BM_ENTRYMODE_DELAYED

Reference Manual to LL API cross reference:

- OUTxR DIDL1 LL_HRTIM_OUT_GetBMEntryMode
- OUTxR DIDL2 LL_HRTIM_OUT_GetBMEntryMode

LL_HRTIM_OUT_GetDLYPRTOutStatus

Function name

__STATIC_INLINE uint32_t LL_HRTIM_OUT_GetDLYPRTOutStatus (const HRTIM_TypeDef * HRTIMx, uint32_t Output)

Function description

Get the level (active or inactive) of the designated output when the delayed protection was triggered.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2

Return values

- **OutputLevel:** This parameter can be one of the following values:
  - LL_HRTIM_OUT_LEVEL_INACTIVE
  - LL_HRTIM_OUT_LEVEL_ACTIVE

Reference Manual to LL API cross reference:

- TIMxISR O1SRSR LL_HRTIM_OUT_GetDLYPRTOutStatus
- TIMxISR O2SRSR LL_HRTIM_OUT_GetDLYPRTOutStatus

LL_HRTIM_OUT_ForceLevel

Function name

__STATIC_INLINE void LL_HRTIM_OUT_ForceLevel (HRTIM_TypeDef * HRTIMx, uint32_t Output, uint32_t OutputLevel)

Function description

Force the timer output to its active or inactive level.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2
- **OutputLevel:** This parameter can be one of the following values:
  - LL_HRTIM_OUT_LEVEL_INACTIVE
  - LL_HRTIM_OUT_LEVEL_ACTIVE

Return values

- **None:**

Reference Manual to LL API cross reference:

- SETx1R SST LL_HRTIM_OUT_ForceLevel
- RSTx1R SRT LL_HRTIM_OUT_ForceLevel
- SETx2R SST LL_HRTIM_OUT_ForceLevel
- RSTx2R SRT LL_HRTIM_OUT_ForceLevel

LL_HRTIM_OUT_GetLevel

Function name

__STATIC_INLINE uint32_t LL_HRTIM_OUT_GetLevel (const HRTIM_TypeDef * HRTIMx, uint32_t Output)

Function description

Get actual output level, before the output stage (chopper, polarity).

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Output:** This parameter can be one of the following values:
  - LL_HRTIM_OUTPUT_TA1
  - LL_HRTIM_OUTPUT_TA2
  - LL_HRTIM_OUTPUT_TB1
  - LL_HRTIM_OUTPUT_TB2
  - LL_HRTIM_OUTPUT_TC1
  - LL_HRTIM_OUTPUT_TC2
  - LL_HRTIM_OUTPUT_TD1
  - LL_HRTIM_OUTPUT_TD2
  - LL_HRTIM_OUTPUT_TE1
  - LL_HRTIM_OUTPUT_TE2

Return values

- **OutputLevel:** This parameter can be one of the following values:
  - LL_HRTIM_OUT_LEVEL_INACTIVE
  - LL_HRTIM_OUT_LEVEL_ACTIVE

Reference Manual to LL API cross reference:

- TIMxISR O1CPY LL_HRTIM_OUT_GetLevel
- TIMxISR O2CPY LL_HRTIM_OUT_GetLevel

LL_HRTIM_EE_Config

Function name

__STATIC_INLINE void LL_HRTIM_EE_Config (HRTIM_TypeDef * HRTIMx, uint32_t Event, uint32_t Configuration)

Function description

Configure external event conditioning.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_1
  - LL_HRTIM_EVENT_2
  - LL_HRTIM_EVENT_3
  - LL_HRTIM_EVENT_4
  - LL_HRTIM_EVENT_5
  - LL_HRTIM_EVENT_6
  - LL_HRTIM_EVENT_7
  - LL_HRTIM_EVENT_8
  - LL_HRTIM_EVENT_9
  - LL_HRTIM_EVENT_10
- **Configuration:** This parameter must be a combination of all the following values:
  - External event source 1 or External event source 2 or External event source 3 or External event source 4
  - LL_HRTIM_EE_POLARITY_HIGH or LL_HRTIM_EE_POLARITY_LOW
  - LL_HRTIM_EE_SENSITIVITY_LEVEL or LL_HRTIM_EE_SENSITIVITY_RISINGEDGE or LL_HRTIM_EE_SENSITIVITY_FALLINGEDGE or LL_HRTIM_EE_SENSITIVITY_BOTHEDGES
  - LL_HRTIM_EE_FASTMODE_DISABLE or LL_HRTIM_EE_FASTMODE_ENABLE

Return values

- **None:**

Notes

- This function must not be called when the timer counter is enabled.
- Event source (EExSrc1..EExSRC4) mapping depends on configured event channel.
- Fast mode is available only for LL_HRTIM_EVENT_1..5.

Reference Manual to LL API cross reference:

- EECR1 EE1SRC LL_HRTIM_EE_Config
- EECR1 EE1POL LL_HRTIM_EE_Config
- EECR1 EE1SNS LL_HRTIM_EE_Config
- EECR1 EE1FAST LL_HRTIM_EE_Config
- EECR1 EE2SRC LL_HRTIM_EE_Config
- EECR1 EE2POL LL_HRTIM_EE_Config
- EECR1 EE2SNS LL_HRTIM_EE_Config
- EECR1 EE2FAST LL_HRTIM_EE_Config
- EECR1 EE3SRC LL_HRTIM_EE_Config
- EECR1 EE3POL LL_HRTIM_EE_Config
- EECR1 EE3SNS LL_HRTIM_EE_Config
- EECR1 EE3FAST LL_HRTIM_EE_Config
- EECR1 EE4SRC LL_HRTIM_EE_Config
- EECR1 EE4POL LL_HRTIM_EE_Config
- EECR1 EE4SNS LL_HRTIM_EE_Config
- EECR1 EE4FAST LL_HRTIM_EE_Config
- EECR1 EE5SRC LL_HRTIM_EE_Config
- EECR1 EE5POL LL_HRTIM_EE_Config
- EECR1 EE5SNS LL_HRTIM_EE_Config
- EECR1 EE5FAST LL_HRTIM_EE_Config
- EECR2 EE6SRC LL_HRTIM_EE_Config
- EECR2 EE6POL LL_HRTIM_EE_Config
- EECR2 EE6SNS LL_HRTIM_EE_Config
- EECR2 EE6FAST LL_HRTIM_EE_Config
- EECR2 EE7SRC LL_HRTIM_EE_Config
- EECR2 EE7POL LL_HRTIM_EE_Config
- EECR2 EE7SNS LL_HRTIM_EE_Config
- EECR2 EE7FAST LL_HRTIM_EE_Config
- EECR2 EE8SRC LL_HRTIM_EE_Config
- EECR2 EE8POL LL_HRTIM_EE_Config
- EECR2 EE8SNS LL_HRTIM_EE_Config
- EECR2 EE8FAST LL_HRTIM_EE_Config
- EECR2 EE9SRC LL_HRTIM_EE_Config
- EECR2 EE9POL LL_HRTIM_EE_Config
- EECR2 EE9SNS LL_HRTIM_EE_Config
- EECR2 EE9FAST LL_HRTIM_EE_Config
- EECR2 EE10SRC LL_HRTIM_EE_Config
- EECR2 EE10POL LL_HRTIM_EE_Config
- EECR2 EE10SNS LL_HRTIM_EE_Config
- EECR2 EE10FAST LL_HRTIM_EE_Config

LL_HRTIM_EE_SetSrc

Function name

__STATIC_INLINE void LL_HRTIM_EE_SetSrc (HRTIM_TypeDef * HRTIMx, uint32_t Event, uint32_t Src)

Function description

Set the external event source.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_1
  - LL_HRTIM_EVENT_2
  - LL_HRTIM_EVENT_3
  - LL_HRTIM_EVENT_4
  - LL_HRTIM_EVENT_5
  - LL_HRTIM_EVENT_6
  - LL_HRTIM_EVENT_7
  - LL_HRTIM_EVENT_8
  - LL_HRTIM_EVENT_9
  - LL_HRTIM_EVENT_10
- **Src:** This parameter can be one of the following values:
  - External event source 1
  - External event source 2
  - External event source 3
  - External event source 4

Return values

- **None:**

Reference Manual to LL API cross reference:

- EECR1 EE1SRC LL_HRTIM_EE_SetSrc
- EECR1 EE2SRC LL_HRTIM_EE_SetSrc
- EECR1 EE3SRC LL_HRTIM_EE_SetSrc
- EECR1 EE4SRC LL_HRTIM_EE_SetSrc
- EECR1 EE5SRC LL_HRTIM_EE_SetSrc
- EECR2 EE6SRC LL_HRTIM_EE_SetSrc
- EECR2 EE7SRC LL_HRTIM_EE_SetSrc
- EECR2 EE8SRC LL_HRTIM_EE_SetSrc
- EECR2 EE9SRC LL_HRTIM_EE_SetSrc
- EECR2 EE10SRC LL_HRTIM_EE_SetSrc

LL_HRTIM_EE_GetSrc

Function name

__STATIC_INLINE uint32_t LL_HRTIM_EE_GetSrc (const HRTIM_TypeDef * HRTIMx, uint32_t Event)

Function description

Get actual external event source.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_1
  - LL_HRTIM_EVENT_2
  - LL_HRTIM_EVENT_3
  - LL_HRTIM_EVENT_4
  - LL_HRTIM_EVENT_5
  - LL_HRTIM_EVENT_6
  - LL_HRTIM_EVENT_7
  - LL_HRTIM_EVENT_8
  - LL_HRTIM_EVENT_9
  - LL_HRTIM_EVENT_10

Return values

- **EventSrc:** This parameter can be one of the following values:
  - External event source 1
  - External event source 2
  - External event source 3
  - External event source 4

Reference Manual to LL API cross reference:

- EECR1 EE1SRC LL_HRTIM_EE_GetSrc
- EECR1 EE2SRC LL_HRTIM_EE_GetSrc
- EECR1 EE3SRC LL_HRTIM_EE_GetSrc
- EECR1 EE4SRC LL_HRTIM_EE_GetSrc
- EECR1 EE5SRC LL_HRTIM_EE_GetSrc
- EECR2 EE6SRC LL_HRTIM_EE_GetSrc
- EECR2 EE7SRC LL_HRTIM_EE_GetSrc
- EECR2 EE8SRC LL_HRTIM_EE_GetSrc
- EECR2 EE9SRC LL_HRTIM_EE_GetSrc
- EECR2 EE10SRC LL_HRTIM_EE_GetSrc

LL_HRTIM_EE_SetPolarity

Function name

__STATIC_INLINE void LL_HRTIM_EE_SetPolarity (HRTIM_TypeDef * HRTIMx, uint32_t Event, uint32_t Polarity)

Function description

Set the polarity of an external event.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_1
  - LL_HRTIM_EVENT_2
  - LL_HRTIM_EVENT_3
  - LL_HRTIM_EVENT_4
  - LL_HRTIM_EVENT_5
  - LL_HRTIM_EVENT_6
  - LL_HRTIM_EVENT_7
  - LL_HRTIM_EVENT_8
  - LL_HRTIM_EVENT_9
  - LL_HRTIM_EVENT_10
- **Polarity:** This parameter can be one of the following values:
  - LL_HRTIM_EE_POLARITY_HIGH
  - LL_HRTIM_EE_POLARITY_LOW

Return values

- **None:**

Notes

- This function must not be called when the timer counter is enabled.
- Event polarity is only significant when event detection is level-sensitive.

Reference Manual to LL API cross reference:

- EECR1 EE1POL LL_HRTIM_EE_SetPolarity
- EECR1 EE2POL LL_HRTIM_EE_SetPolarity
- EECR1 EE3POL LL_HRTIM_EE_SetPolarity
- EECR1 EE4POL LL_HRTIM_EE_SetPolarity
- EECR1 EE5POL LL_HRTIM_EE_SetPolarity
- EECR2 EE6POL LL_HRTIM_EE_SetPolarity
- EECR2 EE7POL LL_HRTIM_EE_SetPolarity
- EECR2 EE8POL LL_HRTIM_EE_SetPolarity
- EECR2 EE9POL LL_HRTIM_EE_SetPolarity
- EECR2 EE10POL LL_HRTIM_EE_SetPolarity

LL_HRTIM_EE_GetPolarity

Function name

__STATIC_INLINE uint32_t LL_HRTIM_EE_GetPolarity (const HRTIM_TypeDef * HRTIMx, uint32_t Event)

Function description

Get actual polarity setting of an external event.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_1
  - LL_HRTIM_EVENT_2
  - LL_HRTIM_EVENT_3
  - LL_HRTIM_EVENT_4
  - LL_HRTIM_EVENT_5
  - LL_HRTIM_EVENT_6
  - LL_HRTIM_EVENT_7
  - LL_HRTIM_EVENT_8
  - LL_HRTIM_EVENT_9
  - LL_HRTIM_EVENT_10

Return values

- **Polarity:** This parameter can be one of the following values:
  - LL_HRTIM_EE_POLARITY_HIGH
  - LL_HRTIM_EE_POLARITY_LOW

Reference Manual to LL API cross reference:

- EECR1 EE1POL LL_HRTIM_EE_GetPolarity
- EECR1 EE2POL LL_HRTIM_EE_GetPolarity
- EECR1 EE3POL LL_HRTIM_EE_GetPolarity
- EECR1 EE4POL LL_HRTIM_EE_GetPolarity
- EECR1 EE5POL LL_HRTIM_EE_GetPolarity
- EECR2 EE6POL LL_HRTIM_EE_GetPolarity
- EECR2 EE7POL LL_HRTIM_EE_GetPolarity
- EECR2 EE8POL LL_HRTIM_EE_GetPolarity
- EECR2 EE9POL LL_HRTIM_EE_GetPolarity
- EECR2 EE10POL LL_HRTIM_EE_GetPolarity

LL_HRTIM_EE_SetSensitivity

Function name

__STATIC_INLINE void LL_HRTIM_EE_SetSensitivity (HRTIM_TypeDef * HRTIMx, uint32_t Event, uint32_t Sensitivity)

Function description

Set the sensitivity of an external event.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_1
  - LL_HRTIM_EVENT_2
  - LL_HRTIM_EVENT_3
  - LL_HRTIM_EVENT_4
  - LL_HRTIM_EVENT_5
  - LL_HRTIM_EVENT_6
  - LL_HRTIM_EVENT_7
  - LL_HRTIM_EVENT_8
  - LL_HRTIM_EVENT_9
  - LL_HRTIM_EVENT_10
- **Sensitivity:** This parameter can be one of the following values:
  - LL_HRTIM_EE_SENSITIVITY_LEVEL
  - LL_HRTIM_EE_SENSITIVITY_RISINGEDGE
  - LL_HRTIM_EE_SENSITIVITY_FALLINGEDGE
  - LL_HRTIM_EE_SENSITIVITY_BOTHEDGES

Return values

- **None:**

Reference Manual to LL API cross reference:

- EECR1 EE1SNS LL_HRTIM_EE_SetSensitivity
- EECR1 EE2SNS LL_HRTIM_EE_SetSensitivity
- EECR1 EE3SNS LL_HRTIM_EE_SetSensitivity
- EECR1 EE4SNS LL_HRTIM_EE_SetSensitivity
- EECR1 EE5SNS LL_HRTIM_EE_SetSensitivity
- EECR2 EE6SNS LL_HRTIM_EE_SetSensitivity
- EECR2 EE7SNS LL_HRTIM_EE_SetSensitivity
- EECR2 EE8SNS LL_HRTIM_EE_SetSensitivity
- EECR2 EE9SNS LL_HRTIM_EE_SetSensitivity
- EECR2 EE10SNS LL_HRTIM_EE_SetSensitivity

LL_HRTIM_EE_GetSensitivity

Function name

__STATIC_INLINE uint32_t LL_HRTIM_EE_GetSensitivity (const HRTIM_TypeDef * HRTIMx, uint32_t Event)

Function description

Get actual sensitivity setting of an external event.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_1
  - LL_HRTIM_EVENT_2
  - LL_HRTIM_EVENT_3
  - LL_HRTIM_EVENT_4
  - LL_HRTIM_EVENT_5
  - LL_HRTIM_EVENT_6
  - LL_HRTIM_EVENT_7
  - LL_HRTIM_EVENT_8
  - LL_HRTIM_EVENT_9
  - LL_HRTIM_EVENT_10

Return values

- **Polarity:** This parameter can be one of the following values:
  - LL_HRTIM_EE_SENSITIVITY_LEVEL
  - LL_HRTIM_EE_SENSITIVITY_RISINGEDGE
  - LL_HRTIM_EE_SENSITIVITY_FALLINGEDGE
  - LL_HRTIM_EE_SENSITIVITY_BOTHEDGES

Reference Manual to LL API cross reference:

- EECR1 EE1SNS LL_HRTIM_EE_GetSensitivity
- EECR1 EE2SNS LL_HRTIM_EE_GetSensitivity
- EECR1 EE3SNS LL_HRTIM_EE_GetSensitivity
- EECR1 EE4SNS LL_HRTIM_EE_GetSensitivity
- EECR1 EE5SNS LL_HRTIM_EE_GetSensitivity
- EECR2 EE6SNS LL_HRTIM_EE_GetSensitivity
- EECR2 EE7SNS LL_HRTIM_EE_GetSensitivity
- EECR2 EE8SNS LL_HRTIM_EE_GetSensitivity
- EECR2 EE9SNS LL_HRTIM_EE_GetSensitivity
- EECR2 EE10SNS LL_HRTIM_EE_GetSensitivity

LL_HRTIM_EE_SetFastMode

Function name

__STATIC_INLINE void LL_HRTIM_EE_SetFastMode (HRTIM_TypeDef * HRTIMx, uint32_t Event, uint32_t FastMode)

Function description

Set the fast mode of an external event.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_1
  - LL_HRTIM_EVENT_2
  - LL_HRTIM_EVENT_3
  - LL_HRTIM_EVENT_4
  - LL_HRTIM_EVENT_5
- **FastMode:** This parameter can be one of the following values:
  - LL_HRTIM_EE_FASTMODE_DISABLE
  - LL_HRTIM_EE_FASTMODE_ENABLE

Return values

- **None:**

Notes

- This function must not be called when the timer counter is enabled.

Reference Manual to LL API cross reference:

- EECR1 EE1FAST LL_HRTIM_EE_SetFastMode
- EECR1 EE2FAST LL_HRTIM_EE_SetFastMode
- EECR1 EE3FAST LL_HRTIM_EE_SetFastMode
- EECR1 EE4FAST LL_HRTIM_EE_SetFastMode
- EECR1 EE5FAST LL_HRTIM_EE_SetFastMode
- EECR2 EE6FAST LL_HRTIM_EE_SetFastMode
- EECR2 EE7FAST LL_HRTIM_EE_SetFastMode
- EECR2 EE8FAST LL_HRTIM_EE_SetFastMode
- EECR2 EE9FAST LL_HRTIM_EE_SetFastMode
- EECR2 EE10FAST LL_HRTIM_EE_SetFastMode

LL_HRTIM_EE_GetFastMode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_EE_GetFastMode (const HRTIM_TypeDef * HRTIMx, uint32_t Event)

Function description

Get actual fast mode setting of an external event.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_1
  - LL_HRTIM_EVENT_2
  - LL_HRTIM_EVENT_3
  - LL_HRTIM_EVENT_4
  - LL_HRTIM_EVENT_5

Return values

- **FastMode:** This parameter can be one of the following values:
  - LL_HRTIM_EE_FASTMODE_DISABLE
  - LL_HRTIM_EE_FASTMODE_ENABLE

Reference Manual to LL API cross reference:

- EECR1 EE1FAST LL_HRTIM_EE_GetFastMode
- EECR1 EE2FAST LL_HRTIM_EE_GetFastMode
- EECR1 EE3FAST LL_HRTIM_EE_GetFastMode
- EECR1 EE4FAST LL_HRTIM_EE_GetFastMode
- EECR1 EE5FAST LL_HRTIM_EE_GetFastMode
- EECR2 EE6FAST LL_HRTIM_EE_GetFastMode
- EECR2 EE7FAST LL_HRTIM_EE_GetFastMode
- EECR2 EE8FAST LL_HRTIM_EE_GetFastMode
- EECR2 EE9FAST LL_HRTIM_EE_GetFastMode
- EECR2 EE10FAST LL_HRTIM_EE_GetFastMode

LL_HRTIM_EE_SetFilter

Function name

__STATIC_INLINE void LL_HRTIM_EE_SetFilter (HRTIM_TypeDef * HRTIMx, uint32_t Event, uint32_t Filter)

Function description

Set the digital noise filter of a external event.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_6
  - LL_HRTIM_EVENT_7
  - LL_HRTIM_EVENT_8
  - LL_HRTIM_EVENT_9
  - LL_HRTIM_EVENT_10
- **Filter:** This parameter can be one of the following values:
  - LL_HRTIM_EE_FILTER_NONE
  - LL_HRTIM_EE_FILTER_1
  - LL_HRTIM_EE_FILTER_2
  - LL_HRTIM_EE_FILTER_3
  - LL_HRTIM_EE_FILTER_4
  - LL_HRTIM_EE_FILTER_5
  - LL_HRTIM_EE_FILTER_6
  - LL_HRTIM_EE_FILTER_7
  - LL_HRTIM_EE_FILTER_8
  - LL_HRTIM_EE_FILTER_9
  - LL_HRTIM_EE_FILTER_10
  - LL_HRTIM_EE_FILTER_11
  - LL_HRTIM_EE_FILTER_12
  - LL_HRTIM_EE_FILTER_13
  - LL_HRTIM_EE_FILTER_14
  - LL_HRTIM_EE_FILTER_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- EECR3 EE6F LL_HRTIM_EE_SetFilter
- EECR3 EE7F LL_HRTIM_EE_SetFilter
- EECR3 EE8F LL_HRTIM_EE_SetFilter
- EECR3 EE9F LL_HRTIM_EE_SetFilter
- EECR3 EE10F LL_HRTIM_EE_SetFilter

LL_HRTIM_EE_GetFilter

Function name

__STATIC_INLINE uint32_t LL_HRTIM_EE_GetFilter (const HRTIM_TypeDef * HRTIMx, uint32_t Event)

Function description

Get actual digital noise filter setting of a external event.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Event:** This parameter can be one of the following values:
  - LL_HRTIM_EVENT_6
  - LL_HRTIM_EVENT_7
  - LL_HRTIM_EVENT_8
  - LL_HRTIM_EVENT_9
  - LL_HRTIM_EVENT_10

Return values

- **Filter:** This parameter can be one of the following values:
  - LL_HRTIM_EE_FILTER_NONE
  - LL_HRTIM_EE_FILTER_1
  - LL_HRTIM_EE_FILTER_2
  - LL_HRTIM_EE_FILTER_3
  - LL_HRTIM_EE_FILTER_4
  - LL_HRTIM_EE_FILTER_5
  - LL_HRTIM_EE_FILTER_6
  - LL_HRTIM_EE_FILTER_7
  - LL_HRTIM_EE_FILTER_8
  - LL_HRTIM_EE_FILTER_9
  - LL_HRTIM_EE_FILTER_10
  - LL_HRTIM_EE_FILTER_11
  - LL_HRTIM_EE_FILTER_12
  - LL_HRTIM_EE_FILTER_13
  - LL_HRTIM_EE_FILTER_14
  - LL_HRTIM_EE_FILTER_15

Reference Manual to LL API cross reference:

- EECR3 EE6F LL_HRTIM_EE_GetFilter
- EECR3 EE7F LL_HRTIM_EE_GetFilter
- EECR3 EE8F LL_HRTIM_EE_GetFilter
- EECR3 EE9F LL_HRTIM_EE_GetFilter
- EECR3 EE10F LL_HRTIM_EE_GetFilter

LL_HRTIM_EE_SetPrescaler

Function name

__STATIC_INLINE void LL_HRTIM_EE_SetPrescaler (HRTIM_TypeDef * HRTIMx, uint32_t Prescaler)

Function description

Set the external event prescaler.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Prescaler:** This parameter can be one of the following values:
  - LL_HRTIM_EE_PRESCALER_DIV1
  - LL_HRTIM_EE_PRESCALER_DIV2
  - LL_HRTIM_EE_PRESCALER_DIV4
  - LL_HRTIM_EE_PRESCALER_DIV8

Return values

- **None:**

Reference Manual to LL API cross reference:

- EECR3 EEVSD LL_HRTIM_EE_SetPrescaler

LL_HRTIM_EE_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_HRTIM_EE_GetPrescaler (const HRTIM_TypeDef * HRTIMx)

Function description

Get actual external event prescaler setting.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **Prescaler:** This parameter can be one of the following values:
  - LL_HRTIM_EE_PRESCALER_DIV1
  - LL_HRTIM_EE_PRESCALER_DIV2
  - LL_HRTIM_EE_PRESCALER_DIV4
  - LL_HRTIM_EE_PRESCALER_DIV8

Reference Manual to LL API cross reference:

- EECR3 EEVSD LL_HRTIM_EE_GetPrescaler

LL_HRTIM_FLT_Config

Function name

__STATIC_INLINE void LL_HRTIM_FLT_Config (HRTIM_TypeDef * HRTIMx, uint32_t Fault, uint32_t Configuration)

Function description

Configure fault signal conditioning Polarity and Source.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_HRTIM_FLT_SRC_DIGITALINPUT..LL_HRTIM_FLT_SRC_INTERNAL
  - LL_HRTIM_FLT_POLARITY_LOW..LL_HRTIM_FLT_POLARITY_HIGH

Return values

- **None:**

Notes

- This function must not be called when the fault channel is enabled.

Reference Manual to LL API cross reference:

- FLTINR1 FLT1P LL_HRTIM_FLT_Config
- FLTINR1 FLT1SRC LL_HRTIM_FLT_Config
- FLTINR1 FLT2P LL_HRTIM_FLT_Config
- FLTINR1 FLT2SRC LL_HRTIM_FLT_Config
- FLTINR1 FLT3P LL_HRTIM_FLT_Config
- FLTINR1 FLT3SRC LL_HRTIM_FLT_Config
- FLTINR1 FLT4P LL_HRTIM_FLT_Config
- FLTINR1 FLT4SRC LL_HRTIM_FLT_Config
- FLTINR2 FLT5P LL_HRTIM_FLT_Config
- FLTINR2 FLT5SRC LL_HRTIM_FLT_Config

LL_HRTIM_FLT_SetSrc

Function name

__STATIC_INLINE void LL_HRTIM_FLT_SetSrc (HRTIM_TypeDef * HRTIMx, uint32_t Fault, uint32_t Src)

Function description

Set the source of a fault signal.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5
- **Src:** This parameter can be one of the following values:
  - LL_HRTIM_FLT_SRC_DIGITALINPUT
  - LL_HRTIM_FLT_SRC_INTERNAL

Return values

- **None:**

Notes

- This function must not be called when the fault channel is enabled.

Reference Manual to LL API cross reference:

- FLTINR1 FLT1SRC LL_HRTIM_FLT_SetSrc
- FLTINR1 FLT2SRC LL_HRTIM_FLT_SetSrc
- FLTINR1 FLT3SRC LL_HRTIM_FLT_SetSrc
- FLTINR1 FLT4SRC LL_HRTIM_FLT_SetSrc
- FLTINR2 FLT5SRC LL_HRTIM_FLT_SetSrc

LL_HRTIM_FLT_GetSrc

Function name

__STATIC_INLINE uint32_t LL_HRTIM_FLT_GetSrc (const HRTIM_TypeDef * HRTIMx, uint32_t Fault)

Function description

Get actual source of a fault signal.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5

Return values

- **Source:** This parameter can be one of the following values:
  - LL_HRTIM_FLT_SRC_DIGITALINPUT
  - LL_HRTIM_FLT_SRC_INTERNAL

Reference Manual to LL API cross reference:

- FLTINR1 FLT1SRC LL_HRTIM_FLT_GetSrc
- FLTINR1 FLT2SRC LL_HRTIM_FLT_GetSrc
- FLTINR1 FLT3SRC LL_HRTIM_FLT_GetSrc
- FLTINR1 FLT4SRC LL_HRTIM_FLT_GetSrc
- FLTINR2 FLT5SRC LL_HRTIM_FLT_GetSrc

LL_HRTIM_FLT_SetPolarity

Function name

__STATIC_INLINE void LL_HRTIM_FLT_SetPolarity (HRTIM_TypeDef * HRTIMx, uint32_t Fault, uint32_t Polarity)

Function description

Set the polarity of a fault signal.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5
- **Polarity:** This parameter can be one of the following values:
  - LL_HRTIM_FLT_POLARITY_LOW
  - LL_HRTIM_FLT_POLARITY_HIGH

Return values

- **None:**

Notes

- This function must not be called when the fault channel is enabled.

Reference Manual to LL API cross reference:

- FLTINR1 FLT1P LL_HRTIM_FLT_SetPolarity
- FLTINR1 FLT2P LL_HRTIM_FLT_SetPolarity
- FLTINR1 FLT3P LL_HRTIM_FLT_SetPolarity
- FLTINR1 FLT4P LL_HRTIM_FLT_SetPolarity
- FLTINR2 FLT5P LL_HRTIM_FLT_SetPolarity

LL_HRTIM_FLT_GetPolarity

Function name

__STATIC_INLINE uint32_t LL_HRTIM_FLT_GetPolarity (const HRTIM_TypeDef * HRTIMx, uint32_t Fault)

Function description

Get actual polarity of a fault signal.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5

Return values

- **Polarity:** This parameter can be one of the following values:
  - LL_HRTIM_FLT_POLARITY_LOW
  - LL_HRTIM_FLT_POLARITY_HIGH

Reference Manual to LL API cross reference:

- FLTINR1 FLT1P LL_HRTIM_FLT_GetPolarity
- FLTINR1 FLT2P LL_HRTIM_FLT_GetPolarity
- FLTINR1 FLT3P LL_HRTIM_FLT_GetPolarity
- FLTINR1 FLT4P LL_HRTIM_FLT_GetPolarity
- FLTINR2 FLT5P LL_HRTIM_FLT_GetPolarity

LL_HRTIM_FLT_SetFilter

Function name

__STATIC_INLINE void LL_HRTIM_FLT_SetFilter (HRTIM_TypeDef * HRTIMx, uint32_t Fault, uint32_t Filter)

Function description

Set the digital noise filter of a fault signal.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5
- **Filter:** This parameter can be one of the following values:
  - LL_HRTIM_FLT_FILTER_NONE
  - LL_HRTIM_FLT_FILTER_1
  - LL_HRTIM_FLT_FILTER_2
  - LL_HRTIM_FLT_FILTER_3
  - LL_HRTIM_FLT_FILTER_4
  - LL_HRTIM_FLT_FILTER_5
  - LL_HRTIM_FLT_FILTER_6
  - LL_HRTIM_FLT_FILTER_7
  - LL_HRTIM_FLT_FILTER_8
  - LL_HRTIM_FLT_FILTER_9
  - LL_HRTIM_FLT_FILTER_10
  - LL_HRTIM_FLT_FILTER_11
  - LL_HRTIM_FLT_FILTER_12
  - LL_HRTIM_FLT_FILTER_13
  - LL_HRTIM_FLT_FILTER_14
  - LL_HRTIM_FLT_FILTER_15

Return values

- **None:**

Notes

- This function must not be called when the fault channel is enabled.

Reference Manual to LL API cross reference:

- FLTINR1 FLT1F LL_HRTIM_FLT_SetFilter
- FLTINR1 FLT2F LL_HRTIM_FLT_SetFilter
- FLTINR1 FLT3F LL_HRTIM_FLT_SetFilter
- FLTINR1 FLT4F LL_HRTIM_FLT_SetFilter
- FLTINR2 FLT5F LL_HRTIM_FLT_SetFilter

LL_HRTIM_FLT_GetFilter

Function name

__STATIC_INLINE uint32_t LL_HRTIM_FLT_GetFilter (const HRTIM_TypeDef * HRTIMx, uint32_t Fault)

Function description

Get actual digital noise filter setting of a fault signal.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5

Return values

- **Filter:** This parameter can be one of the following values:
  - LL_HRTIM_FLT_FILTER_NONE
  - LL_HRTIM_FLT_FILTER_1
  - LL_HRTIM_FLT_FILTER_2
  - LL_HRTIM_FLT_FILTER_3
  - LL_HRTIM_FLT_FILTER_4
  - LL_HRTIM_FLT_FILTER_5
  - LL_HRTIM_FLT_FILTER_6
  - LL_HRTIM_FLT_FILTER_7
  - LL_HRTIM_FLT_FILTER_8
  - LL_HRTIM_FLT_FILTER_9
  - LL_HRTIM_FLT_FILTER_10
  - LL_HRTIM_FLT_FILTER_11
  - LL_HRTIM_FLT_FILTER_12
  - LL_HRTIM_FLT_FILTER_13
  - LL_HRTIM_FLT_FILTER_14
  - LL_HRTIM_FLT_FILTER_15

Reference Manual to LL API cross reference:

- FLTINR1 FLT1F LL_HRTIM_FLT_GetFilter
- FLTINR1 FLT2F LL_HRTIM_FLT_GetFilter
- FLTINR1 FLT3F LL_HRTIM_FLT_GetFilter
- FLTINR1 FLT4F LL_HRTIM_FLT_GetFilter
- FLTINR2 FLT5F LL_HRTIM_FLT_GetFilter

LL_HRTIM_FLT_SetPrescaler

Function name

__STATIC_INLINE void LL_HRTIM_FLT_SetPrescaler (HRTIM_TypeDef * HRTIMx, uint32_t Prescaler)

Function description

Set the fault circuitry prescaler.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Prescaler:** This parameter can be one of the following values:
  - LL_HRTIM_FLT_PRESCALER_DIV1
  - LL_HRTIM_FLT_PRESCALER_DIV2
  - LL_HRTIM_FLT_PRESCALER_DIV4
  - LL_HRTIM_FLT_PRESCALER_DIV8

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLTINR2 FLTSD LL_HRTIM_FLT_SetPrescaler

LL_HRTIM_FLT_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_HRTIM_FLT_GetPrescaler (const HRTIM_TypeDef * HRTIMx)

Function description

Get actual fault circuitry prescaler setting.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **Prescaler:** This parameter can be one of the following values:
  - LL_HRTIM_FLT_PRESCALER_DIV1
  - LL_HRTIM_FLT_PRESCALER_DIV2
  - LL_HRTIM_FLT_PRESCALER_DIV4
  - LL_HRTIM_FLT_PRESCALER_DIV8

Reference Manual to LL API cross reference:

- FLTINR2 FLTSD LL_HRTIM_FLT_GetPrescaler

LL_HRTIM_FLT_Lock

Function name

__STATIC_INLINE void LL_HRTIM_FLT_Lock (HRTIM_TypeDef * HRTIMx, uint32_t Fault)

Function description

Lock the fault signal conditioning settings.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLTINR1 FLT1LCK LL_HRTIM_FLT_Lock
- FLTINR1 FLT2LCK LL_HRTIM_FLT_Lock
- FLTINR1 FLT3LCK LL_HRTIM_FLT_Lock
- FLTINR1 FLT4LCK LL_HRTIM_FLT_Lock
- FLTINR2 FLT5LCK LL_HRTIM_FLT_Lock

LL_HRTIM_FLT_Enable

Function name

__STATIC_INLINE void LL_HRTIM_FLT_Enable (HRTIM_TypeDef * HRTIMx, uint32_t Fault)

Function description

Enable the fault circuitry for the designated fault input.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLTINR1 FLT1E LL_HRTIM_FLT_Enable
- FLTINR1 FLT2E LL_HRTIM_FLT_Enable
- FLTINR1 FLT3E LL_HRTIM_FLT_Enable
- FLTINR1 FLT4E LL_HRTIM_FLT_Enable
- FLTINR2 FLT5E LL_HRTIM_FLT_Enable

LL_HRTIM_FLT_Disable

Function name

__STATIC_INLINE void LL_HRTIM_FLT_Disable (HRTIM_TypeDef * HRTIMx, uint32_t Fault)

Function description

Disable the fault circuitry for for the designated fault input.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLTINR1 FLT1E LL_HRTIM_FLT_Disable
- FLTINR1 FLT2E LL_HRTIM_FLT_Disable
- FLTINR1 FLT3E LL_HRTIM_FLT_Disable
- FLTINR1 FLT4E LL_HRTIM_FLT_Disable
- FLTINR2 FLT5E LL_HRTIM_FLT_Disable

LL_HRTIM_FLT_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_HRTIM_FLT_IsEnabled (const HRTIM_TypeDef * HRTIMx, uint32_t Fault)

Function description

Indicate whether the fault circuitry is enabled for a given fault input.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Fault:** This parameter can be one of the following values:
  - LL_HRTIM_FAULT_1
  - LL_HRTIM_FAULT_2
  - LL_HRTIM_FAULT_3
  - LL_HRTIM_FAULT_4
  - LL_HRTIM_FAULT_5

Return values

- **State:** of FLTxEN bit in HRTIM_FLTINRx register (1 or 0).

Reference Manual to LL API cross reference:

- FLTINR1 FLT1E LL_HRTIM_FLT_IsEnabled
- FLTINR1 FLT2E LL_HRTIM_FLT_IsEnabled
- FLTINR1 FLT3E LL_HRTIM_FLT_IsEnabled
- FLTINR1 FLT4E LL_HRTIM_FLT_IsEnabled
- FLTINR2 FLT5E LL_HRTIM_FLT_IsEnabled

LL_HRTIM_BM_Config

Function name

__STATIC_INLINE void LL_HRTIM_BM_Config (HRTIM_TypeDef * HRTIMx, uint32_t Configuration)

Function description

Configure the burst mode controller.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_HRTIM_BM_MODE_SINGLESHOT or LL_HRTIM_BM_MODE_CONTINOUS
  - LL_HRTIM_BM_CLKSRC_MASTER or ... or LL_HRTIM_BM_CLKSRC_FHRTIM
  - LL_HRTIM_BM_PRESCALER_DIV1 or ... LL_HRTIM_BM_PRESCALER_DIV32768

Return values

- **None:**

Reference Manual to LL API cross reference:

- BMCR BMOM LL_HRTIM_BM_Config
- BMCR BMCLK LL_HRTIM_BM_Config
- BMCR BMPRSC LL_HRTIM_BM_Config

LL_HRTIM_BM_SetMode

Function name

__STATIC_INLINE void LL_HRTIM_BM_SetMode (HRTIM_TypeDef * HRTIMx, uint32_t Mode)

Function description

Set the burst mode controller operating mode.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Mode:** This parameter can be one of the following values:
  - LL_HRTIM_BM_MODE_SINGLESHOT
  - LL_HRTIM_BM_MODE_CONTINOUS

Return values

- **None:**

Reference Manual to LL API cross reference:

- BMCR BMOM LL_HRTIM_BM_SetMode

LL_HRTIM_BM_GetMode

Function name

__STATIC_INLINE uint32_t LL_HRTIM_BM_GetMode (const HRTIM_TypeDef * HRTIMx)

Function description

Get actual burst mode controller operating mode.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **Mode:** This parameter can be one of the following values:
  - LL_HRTIM_BM_MODE_SINGLESHOT
  - LL_HRTIM_BM_MODE_CONTINOUS

Reference Manual to LL API cross reference:

- BMCR BMOM LL_HRTIM_BM_GetMode

LL_HRTIM_BM_SetClockSrc

Function name

__STATIC_INLINE void LL_HRTIM_BM_SetClockSrc (HRTIM_TypeDef * HRTIMx, uint32_t ClockSrc)

Function description

Set the burst mode controller clock source.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **ClockSrc:** This parameter can be one of the following values:
  - LL_HRTIM_BM_CLKSRC_MASTER
  - LL_HRTIM_BM_CLKSRC_TIMER_A
  - LL_HRTIM_BM_CLKSRC_TIMER_B
  - LL_HRTIM_BM_CLKSRC_TIMER_C
  - LL_HRTIM_BM_CLKSRC_TIMER_D
  - LL_HRTIM_BM_CLKSRC_TIMER_E
  - LL_HRTIM_BM_CLKSRC_TIM16_OC
  - LL_HRTIM_BM_CLKSRC_TIM17_OC
  - LL_HRTIM_BM_CLKSRC_TIM7_TRGO
  - LL_HRTIM_BM_CLKSRC_FHRTIM

Return values

- **None:**

Reference Manual to LL API cross reference:

- BMCR BMCLK LL_HRTIM_BM_SetClockSrc

LL_HRTIM_BM_GetClockSrc

Function name

__STATIC_INLINE uint32_t LL_HRTIM_BM_GetClockSrc (const HRTIM_TypeDef * HRTIMx)

Function description

Get actual burst mode controller clock source.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **ClockSrc:** This parameter can be one of the following values:
  - LL_HRTIM_BM_CLKSRC_MASTER
  - LL_HRTIM_BM_CLKSRC_TIMER_A
  - LL_HRTIM_BM_CLKSRC_TIMER_B
  - LL_HRTIM_BM_CLKSRC_TIMER_C
  - LL_HRTIM_BM_CLKSRC_TIMER_D
  - LL_HRTIM_BM_CLKSRC_TIMER_E
  - LL_HRTIM_BM_CLKSRC_TIM16_OC
  - LL_HRTIM_BM_CLKSRC_TIM17_OC
  - LL_HRTIM_BM_CLKSRC_TIM7_TRGO
  - LL_HRTIM_BM_CLKSRC_FHRTIM
- **ClockSrc:** This parameter can be one of the following values:
  - LL_HRTIM_BM_CLKSRC_MASTER
  - LL_HRTIM_BM_CLKSRC_TIMER_A
  - LL_HRTIM_BM_CLKSRC_TIMER_B
  - LL_HRTIM_BM_CLKSRC_TIMER_C
  - LL_HRTIM_BM_CLKSRC_TIMER_D
  - LL_HRTIM_BM_CLKSRC_TIMER_E
  - LL_HRTIM_BM_CLKSRC_TIM16_OC
  - LL_HRTIM_BM_CLKSRC_TIM17_OC
  - LL_HRTIM_BM_CLKSRC_TIM7_TRGO
  - LL_HRTIM_BM_CLKSRC_FHRTIM

Reference Manual to LL API cross reference:

- BMCR BMCLK LL_HRTIM_BM_GetClockSrc

LL_HRTIM_BM_SetPrescaler

Function name

__STATIC_INLINE void LL_HRTIM_BM_SetPrescaler (HRTIM_TypeDef * HRTIMx, uint32_t Prescaler)

Function description

Set the burst mode controller prescaler.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Prescaler:** This parameter can be one of the following values:
  - LL_HRTIM_BM_PRESCALER_DIV1
  - LL_HRTIM_BM_PRESCALER_DIV2
  - LL_HRTIM_BM_PRESCALER_DIV4
  - LL_HRTIM_BM_PRESCALER_DIV8
  - LL_HRTIM_BM_PRESCALER_DIV16
  - LL_HRTIM_BM_PRESCALER_DIV32
  - LL_HRTIM_BM_PRESCALER_DIV64
  - LL_HRTIM_BM_PRESCALER_DIV128
  - LL_HRTIM_BM_PRESCALER_DIV256
  - LL_HRTIM_BM_PRESCALER_DIV512
  - LL_HRTIM_BM_PRESCALER_DIV1024
  - LL_HRTIM_BM_PRESCALER_DIV2048
  - LL_HRTIM_BM_PRESCALER_DIV4096
  - LL_HRTIM_BM_PRESCALER_DIV8192
  - LL_HRTIM_BM_PRESCALER_DIV16384
  - LL_HRTIM_BM_PRESCALER_DIV32768

Return values

- **None:**

Reference Manual to LL API cross reference:

- BMCR BMPRSC LL_HRTIM_BM_SetPrescaler

LL_HRTIM_BM_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_HRTIM_BM_GetPrescaler (const HRTIM_TypeDef * HRTIMx)

Function description

Get actual burst mode controller prescaler setting.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **Prescaler:** This parameter can be one of the following values:
  - LL_HRTIM_BM_PRESCALER_DIV1
  - LL_HRTIM_BM_PRESCALER_DIV2
  - LL_HRTIM_BM_PRESCALER_DIV4
  - LL_HRTIM_BM_PRESCALER_DIV8
  - LL_HRTIM_BM_PRESCALER_DIV16
  - LL_HRTIM_BM_PRESCALER_DIV32
  - LL_HRTIM_BM_PRESCALER_DIV64
  - LL_HRTIM_BM_PRESCALER_DIV128
  - LL_HRTIM_BM_PRESCALER_DIV256
  - LL_HRTIM_BM_PRESCALER_DIV512
  - LL_HRTIM_BM_PRESCALER_DIV1024
  - LL_HRTIM_BM_PRESCALER_DIV2048
  - LL_HRTIM_BM_PRESCALER_DIV4096
  - LL_HRTIM_BM_PRESCALER_DIV8192
  - LL_HRTIM_BM_PRESCALER_DIV16384
  - LL_HRTIM_BM_PRESCALER_DIV32768

Reference Manual to LL API cross reference:

- BMCR BMPRSC LL_HRTIM_BM_GetPrescaler

LL_HRTIM_BM_EnablePreload

Function name

__STATIC_INLINE void LL_HRTIM_BM_EnablePreload (HRTIM_TypeDef * HRTIMx)

Function description

Enable burst mode compare and period registers preload.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- BMCR BMPREN LL_HRTIM_BM_EnablePreload

LL_HRTIM_BM_DisablePreload

Function name

__STATIC_INLINE void LL_HRTIM_BM_DisablePreload (HRTIM_TypeDef * HRTIMx)

Function description

Disable burst mode compare and period registers preload.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- BMCR BMPREN LL_HRTIM_BM_DisablePreload

LL_HRTIM_BM_IsEnabledPreload

Function name

__STATIC_INLINE uint32_t LL_HRTIM_BM_IsEnabledPreload (const HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether burst mode compare and period registers are preloaded.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of BMPREN bit in HRTIM_BMCR register (1 or 0).

Reference Manual to LL API cross reference:

- BMCR BMPREN LL_HRTIM_BM_IsEnabledPreload

LL_HRTIM_BM_SetTrig

Function name

__STATIC_INLINE void LL_HRTIM_BM_SetTrig (HRTIM_TypeDef * HRTIMx, uint32_t Trig)

Function description

Set the burst mode controller trigger.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Trig:** This parameter can be a combination of the following values:
  - LL_HRTIM_BM_TRIG_NONE
  - LL_HRTIM_BM_TRIG_MASTER_RESET
  - LL_HRTIM_BM_TRIG_MASTER_REPETITION
  - LL_HRTIM_BM_TRIG_MASTER_CMP1
  - LL_HRTIM_BM_TRIG_MASTER_CMP2
  - LL_HRTIM_BM_TRIG_MASTER_CMP3
  - LL_HRTIM_BM_TRIG_MASTER_CMP4
  - LL_HRTIM_BM_TRIG_TIMA_RESET
  - LL_HRTIM_BM_TRIG_TIMA_REPETITION
  - LL_HRTIM_BM_TRIG_TIMA_CMP1
  - LL_HRTIM_BM_TRIG_TIMA_CMP2
  - LL_HRTIM_BM_TRIG_TIMB_RESET
  - LL_HRTIM_BM_TRIG_TIMB_REPETITION
  - LL_HRTIM_BM_TRIG_TIMB_CMP1
  - LL_HRTIM_BM_TRIG_TIMB_CMP2
  - LL_HRTIM_BM_TRIG_TIMC_RESET
  - LL_HRTIM_BM_TRIG_TIMC_REPETITION
  - LL_HRTIM_BM_TRIG_TIMC_CMP1
  - LL_HRTIM_BM_TRIG_TIMC_CMP2
  - LL_HRTIM_BM_TRIG_TIMD_RESET
  - LL_HRTIM_BM_TRIG_TIMD_REPETITION
  - LL_HRTIM_BM_TRIG_TIMD_CMP1
  - LL_HRTIM_BM_TRIG_TIMD_CMP2
  - LL_HRTIM_BM_TRIG_TIME_RESET
  - LL_HRTIM_BM_TRIG_TIME_REPETITION
  - LL_HRTIM_BM_TRIG_TIME_CMP1
  - LL_HRTIM_BM_TRIG_TIME_CMP2
  - LL_HRTIM_BM_TRIG_TIMA_EVENT7
  - LL_HRTIM_BM_TRIG_TIMD_EVENT8
  - LL_HRTIM_BM_TRIG_EVENT_7
  - LL_HRTIM_BM_TRIG_EVENT_8
  - LL_HRTIM_BM_TRIG_EVENT_ONCHIP

Return values

- **None:**

Reference Manual to LL API cross reference:

- BMTRGR SW LL_HRTIM_BM_SetTrig
- BMTRGR MSTRST LL_HRTIM_BM_SetTrig
- BMTRGR MSTREP LL_HRTIM_BM_SetTrig
- BMTRGR MSTCMP1 LL_HRTIM_BM_SetTrig
- BMTRGR MSTCMP2 LL_HRTIM_BM_SetTrig
- BMTRGR MSTCMP3 LL_HRTIM_BM_SetTrig
- BMTRGR MSTCMP4 LL_HRTIM_BM_SetTrig
- BMTRGR TARST LL_HRTIM_BM_SetTrig
- BMTRGR TAREP LL_HRTIM_BM_SetTrig
- BMTRGR TACMP1 LL_HRTIM_BM_SetTrig
- BMTRGR TACMP2 LL_HRTIM_BM_SetTrig
- BMTRGR TBRST LL_HRTIM_BM_SetTrig
- BMTRGR TBREP LL_HRTIM_BM_SetTrig
- BMTRGR TBCMP1 LL_HRTIM_BM_SetTrig
- BMTRGR TBCMP2 LL_HRTIM_BM_SetTrig
- BMTRGR TCRST LL_HRTIM_BM_SetTrig
- BMTRGR TCREP LL_HRTIM_BM_SetTrig
- BMTRGR TCCMP1 LL_HRTIM_BM_SetTrig
- BMTRGR TCCMP2 LL_HRTIM_BM_SetTrig
- BMTRGR TDRST LL_HRTIM_BM_SetTrig
- BMTRGR TDREP LL_HRTIM_BM_SetTrig
- BMTRGR TDCMP1 LL_HRTIM_BM_SetTrig
- BMTRGR TDCMP2 LL_HRTIM_BM_SetTrig
- BMTRGR TERST LL_HRTIM_BM_SetTrig
- BMTRGR TEREP LL_HRTIM_BM_SetTrig
- BMTRGR TECMP1 LL_HRTIM_BM_SetTrig
- BMTRGR TECMP2 LL_HRTIM_BM_SetTrig
- BMTRGR TAEEV7 LL_HRTIM_BM_SetTrig
- BMTRGR TAEEV8 LL_HRTIM_BM_SetTrig
- BMTRGR EEV7 LL_HRTIM_BM_SetTrig
- BMTRGR EEV8 LL_HRTIM_BM_SetTrig
- BMTRGR OCHIPEV LL_HRTIM_BM_SetTrig

LL_HRTIM_BM_GetTrig

Function name

__STATIC_INLINE uint32_t LL_HRTIM_BM_GetTrig (const HRTIM_TypeDef * HRTIMx)

Function description

Get actual burst mode controller trigger.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **Trig:** This parameter can be a combination of the following values:
  - LL_HRTIM_BM_TRIG_NONE
  - LL_HRTIM_BM_TRIG_MASTER_RESET
  - LL_HRTIM_BM_TRIG_MASTER_REPETITION
  - LL_HRTIM_BM_TRIG_MASTER_CMP1
  - LL_HRTIM_BM_TRIG_MASTER_CMP2
  - LL_HRTIM_BM_TRIG_MASTER_CMP3
  - LL_HRTIM_BM_TRIG_MASTER_CMP4
  - LL_HRTIM_BM_TRIG_TIMA_RESET
  - LL_HRTIM_BM_TRIG_TIMA_REPETITION
  - LL_HRTIM_BM_TRIG_TIMA_CMP1
  - LL_HRTIM_BM_TRIG_TIMA_CMP2
  - LL_HRTIM_BM_TRIG_TIMB_RESET
  - LL_HRTIM_BM_TRIG_TIMB_REPETITION
  - LL_HRTIM_BM_TRIG_TIMB_CMP1
  - LL_HRTIM_BM_TRIG_TIMB_CMP2
  - LL_HRTIM_BM_TRIG_TIMC_RESET
  - LL_HRTIM_BM_TRIG_TIMC_REPETITION
  - LL_HRTIM_BM_TRIG_TIMC_CMP1
  - LL_HRTIM_BM_TRIG_TIMC_CMP2
  - LL_HRTIM_BM_TRIG_TIMD_RESET
  - LL_HRTIM_BM_TRIG_TIMD_REPETITION
  - LL_HRTIM_BM_TRIG_TIMD_CMP1
  - LL_HRTIM_BM_TRIG_TIMD_CMP2
  - LL_HRTIM_BM_TRIG_TIME_RESET
  - LL_HRTIM_BM_TRIG_TIME_REPETITION
  - LL_HRTIM_BM_TRIG_TIME_CMP1
  - LL_HRTIM_BM_TRIG_TIME_CMP2
  - LL_HRTIM_BM_TRIG_TIMA_EVENT7
  - LL_HRTIM_BM_TRIG_TIMD_EVENT8
  - LL_HRTIM_BM_TRIG_EVENT_7
  - LL_HRTIM_BM_TRIG_EVENT_8
  - LL_HRTIM_BM_TRIG_EVENT_ONCHIP

Reference Manual to LL API cross reference:

- BMTRGR SW LL_HRTIM_BM_GetTrig
- BMTRGR MSTRST LL_HRTIM_BM_GetTrig
- BMTRGR MSTREP LL_HRTIM_BM_GetTrig
- BMTRGR MSTCMP1 LL_HRTIM_BM_GetTrig
- BMTRGR MSTCMP2 LL_HRTIM_BM_GetTrig
- BMTRGR MSTCMP3 LL_HRTIM_BM_GetTrig
- BMTRGR MSTCMP4 LL_HRTIM_BM_GetTrig
- BMTRGR TARST LL_HRTIM_BM_GetTrig
- BMTRGR TAREP LL_HRTIM_BM_GetTrig
- BMTRGR TACMP1 LL_HRTIM_BM_GetTrig
- BMTRGR TACMP2 LL_HRTIM_BM_GetTrig
- BMTRGR TBRST LL_HRTIM_BM_GetTrig
- BMTRGR TBREP LL_HRTIM_BM_GetTrig
- BMTRGR TBCMP1 LL_HRTIM_BM_GetTrig
- BMTRGR TBCMP2 LL_HRTIM_BM_GetTrig
- BMTRGR TCRST LL_HRTIM_BM_GetTrig
- BMTRGR TCREP LL_HRTIM_BM_GetTrig
- BMTRGR TCCMP1 LL_HRTIM_BM_GetTrig
- BMTRGR TCCMP2 LL_HRTIM_BM_GetTrig
- BMTRGR TDRST LL_HRTIM_BM_GetTrig
- BMTRGR TDREP LL_HRTIM_BM_GetTrig
- BMTRGR TDCMP1 LL_HRTIM_BM_GetTrig
- BMTRGR TDCMP2 LL_HRTIM_BM_GetTrig
- BMTRGR TERST LL_HRTIM_BM_GetTrig
- BMTRGR TEREP LL_HRTIM_BM_GetTrig
- BMTRGR TECMP1 LL_HRTIM_BM_GetTrig
- BMTRGR TECMP2 LL_HRTIM_BM_GetTrig
- BMTRGR TAEEV7 LL_HRTIM_BM_GetTrig
- BMTRGR TAEEV8 LL_HRTIM_BM_GetTrig
- BMTRGR EEV7 LL_HRTIM_BM_GetTrig
- BMTRGR EEV8 LL_HRTIM_BM_GetTrig
- BMTRGR OCHIPEV LL_HRTIM_BM_GetTrig

LL_HRTIM_BM_SetCompare

Function name

__STATIC_INLINE void LL_HRTIM_BM_SetCompare (HRTIM_TypeDef * HRTIMx, uint32_t CompareValue)

Function description

Set the burst mode controller compare value.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **CompareValue:** Compare value must be above or equal to 3 periods of the fHRTIM clock, that is 0x60 if CKPSC[2:0] = 0, 0x30 if CKPSC[2:0] = 1, 0x18 if CKPSC[2:0] = 2,...

Return values

- **None:**

Reference Manual to LL API cross reference:

- BMCMPR BMCMP LL_HRTIM_BM_SetCompare

LL_HRTIM_BM_GetCompare

Function name

__STATIC_INLINE uint32_t LL_HRTIM_BM_GetCompare (const HRTIM_TypeDef * HRTIMx)

Function description

Get actual burst mode controller compare value.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **CompareValue:** Compare value must be above or equal to 3 periods of the fHRTIM clock, that is 0x60 if CKPSC[2:0] = 0, 0x30 if CKPSC[2:0] = 1, 0x18 if CKPSC[2:0] = 2,...

Reference Manual to LL API cross reference:

- BMCMPR BMCMP LL_HRTIM_BM_GetCompare

LL_HRTIM_BM_SetPeriod

Function name

__STATIC_INLINE void LL_HRTIM_BM_SetPeriod (HRTIM_TypeDef * HRTIMx, uint32_t Period)

Function description

Set the burst mode controller period.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Period:** The period value must be above or equal to 3 periods of the fHRTIM clock, that is 0x60 if CKPSC[2:0] = 0, 0x30 if CKPSC[2:0] = 1, 0x18 if CKPSC[2:0] = 2,... The maximum value is 0x0000 FFDF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BMPER BMPER LL_HRTIM_BM_SetPeriod

LL_HRTIM_BM_GetPeriod

Function name

__STATIC_INLINE uint32_t LL_HRTIM_BM_GetPeriod (const HRTIM_TypeDef * HRTIMx)

Function description

Get actual burst mode controller period.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **The:** period value must be above or equal to 3 periods of the fHRTIM clock, that is 0x60 if CKPSC[2:0] = 0, 0x30 if CKPSC[2:0] = 1, 0x18 if CKPSC[2:0] = 2,... The maximum value is 0x0000 FFDF.

Reference Manual to LL API cross reference:

- BMPER BMPER LL_HRTIM_BM_GetPeriod

LL_HRTIM_BM_Enable

Function name

__STATIC_INLINE void LL_HRTIM_BM_Enable (HRTIM_TypeDef * HRTIMx)

Function description

Enable the burst mode controller.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- BMCR BME LL_HRTIM_BM_Enable

LL_HRTIM_BM_Disable

Function name

__STATIC_INLINE void LL_HRTIM_BM_Disable (HRTIM_TypeDef * HRTIMx)

Function description

Disable the burst mode controller.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- BMCR BME LL_HRTIM_BM_Disable

LL_HRTIM_BM_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_HRTIM_BM_IsEnabled (const HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether the burst mode controller is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of BME bit in HRTIM_BMCR register (1 or 0).

Reference Manual to LL API cross reference:

- BMCR BME LL_HRTIM_BM_IsEnabled

LL_HRTIM_BM_Start

Function name

__STATIC_INLINE void LL_HRTIM_BM_Start (HRTIM_TypeDef * HRTIMx)

Function description

Trigger the burst operation (software trigger)

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- BMTRGR SW LL_HRTIM_BM_Start

LL_HRTIM_BM_Stop

Function name

__STATIC_INLINE void LL_HRTIM_BM_Stop (HRTIM_TypeDef * HRTIMx)

Function description

Stop the burst mode operation.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Notes

- Causes a burst mode early termination.

Reference Manual to LL API cross reference:

- BMCR BMSTAT LL_HRTIM_BM_Stop

LL_HRTIM_BM_GetStatus

Function name

__STATIC_INLINE uint32_t LL_HRTIM_BM_GetStatus (const HRTIM_TypeDef * HRTIMx)

Function description

Get actual burst mode status.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **Status:** This parameter can be one of the following values:
  - LL_HRTIM_BM_STATUS_NORMAL
  - LL_HRTIM_BM_STATUS_BURST_ONGOING

Reference Manual to LL API cross reference:

- BMCR BMSTAT LL_HRTIM_BM_GetStatus

LL_HRTIM_ClearFlag_FLT1

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_FLT1 (HRTIM_TypeDef * HRTIMx)

Function description

Clear the Fault 1 interrupt flag.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR FLT1C LL_HRTIM_ClearFlag_FLT1

LL_HRTIM_IsActiveFlag_FLT1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_FLT1 (const HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether Fault 1 interrupt occurred.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of FLT1 bit in HRTIM_ISR register (1 or 0).

Reference Manual to LL API cross reference:

- ICR FLT1 LL_HRTIM_IsActiveFlag_FLT1

LL_HRTIM_ClearFlag_FLT2

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_FLT2 (HRTIM_TypeDef * HRTIMx)

Function description

Clear the Fault 2 interrupt flag.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR FLT2C LL_HRTIM_ClearFlag_FLT2

LL_HRTIM_IsActiveFlag_FLT2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_FLT2 (const HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether Fault 2 interrupt occurred.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of FLT2 bit in HRTIM_ISR register (1 or 0).

Reference Manual to LL API cross reference:

- ICR FLT2 LL_HRTIM_IsActiveFlag_FLT2

LL_HRTIM_ClearFlag_FLT3

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_FLT3 (HRTIM_TypeDef * HRTIMx)

Function description

Clear the Fault 3 interrupt flag.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR FLT3C LL_HRTIM_ClearFlag_FLT3

LL_HRTIM_IsActiveFlag_FLT3

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_FLT3 (const HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether Fault 3 interrupt occurred.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of FLT3 bit in HRTIM_ISR register (1 or 0).

Reference Manual to LL API cross reference:

- ICR FLT3 LL_HRTIM_IsActiveFlag_FLT3

LL_HRTIM_ClearFlag_FLT4

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_FLT4 (HRTIM_TypeDef * HRTIMx)

Function description

Clear the Fault 4 interrupt flag.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR FLT4C LL_HRTIM_ClearFlag_FLT4

LL_HRTIM_IsActiveFlag_FLT4

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_FLT4 (const HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether Fault 4 interrupt occurred.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of FLT4 bit in HRTIM_ISR register (1 or 0).

Reference Manual to LL API cross reference:

- ICR FLT4 LL_HRTIM_IsActiveFlag_FLT4

LL_HRTIM_ClearFlag_FLT5

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_FLT5 (HRTIM_TypeDef * HRTIMx)

Function description

Clear the Fault 5 interrupt flag.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR FLT5C LL_HRTIM_ClearFlag_FLT5

LL_HRTIM_IsActiveFlag_FLT5

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_FLT5 (const HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether Fault 5 interrupt occurred.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of FLT5 bit in HRTIM_ISR register (1 or 0).

Reference Manual to LL API cross reference:

- ICR FLT5 LL_HRTIM_IsActiveFlag_FLT5

LL_HRTIM_ClearFlag_SYSFLT

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_SYSFLT (HRTIM_TypeDef * HRTIMx)

Function description

Clear the System Fault interrupt flag.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR SYSFLTC LL_HRTIM_ClearFlag_SYSFLT

LL_HRTIM_IsActiveFlag_SYSFLT

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_SYSFLT (const HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether System Fault interrupt occurred.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of SYSFLT bit in HRTIM_ISR register (1 or 0).

Reference Manual to LL API cross reference:

- ISR SYSFLT LL_HRTIM_IsActiveFlag_SYSFLT

LL_HRTIM_ClearFlag_BMPER

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_BMPER (HRTIM_TypeDef * HRTIMx)

Function description

Clear the Burst Mode period interrupt flag.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR BMPERC LL_HRTIM_ClearFlag_BMPER

LL_HRTIM_IsActiveFlag_BMPER

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_BMPER (const HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether Burst Mode period interrupt occurred.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of BMPER bit in HRTIM_ISR register (1 or 0).

Reference Manual to LL API cross reference:

- ISR BMPER LL_HRTIM_IsActiveFlag_BMPER

LL_HRTIM_ClearFlag_SYNC

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_SYNC (HRTIM_TypeDef * HRTIMx)

Function description

Clear the Synchronization Input interrupt flag.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- MICR SYNCC LL_HRTIM_ClearFlag_SYNC

LL_HRTIM_IsActiveFlag_SYNC

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_SYNC (const HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether the Synchronization Input interrupt occurred.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of SYNC bit in HRTIM_MISR register (1 or 0).

Reference Manual to LL API cross reference:

- MISR SYNC LL_HRTIM_IsActiveFlag_SYNC

LL_HRTIM_ClearFlag_UPDATE

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_UPDATE (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Clear the update interrupt flag for a given timer (including the master timer) .

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MICR MUPDC LL_HRTIM_ClearFlag_UPDATE
- TIMxICR UPDC LL_HRTIM_ClearFlag_UPDATE

LL_HRTIM_IsActiveFlag_UPDATE

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_UPDATE (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the update interrupt has occurred for a given timer (including the master timer) .

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of MUPD/UPD bit in HRTIM_MISR/HRTIM_TIMxISR register (1 or 0).

Reference Manual to LL API cross reference:

- MISR MUPD LL_HRTIM_IsActiveFlag_UPDATE
- TIMxISR UPD LL_HRTIM_IsActiveFlag_UPDATE

LL_HRTIM_ClearFlag_REP

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_REP (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Clear the repetition interrupt flag for a given timer (including the master timer) .

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MICR MREPC LL_HRTIM_ClearFlag_REP
- TIMxICR REPC LL_HRTIM_ClearFlag_REP

LL_HRTIM_IsActiveFlag_REP

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_REP (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the repetition interrupt has occurred for a given timer (including the master timer) .

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of MREP/REP bit in HRTIM_MISR/HRTIM_TIMxISR register (1 or 0).

Reference Manual to LL API cross reference:

- MISR MREP LL_HRTIM_IsActiveFlag_REP
- TIMxISR REP LL_HRTIM_IsActiveFlag_REP

LL_HRTIM_ClearFlag_CMP1

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_CMP1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Clear the compare 1 match interrupt for a given timer (including the master timer).

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MICR MCMP1C LL_HRTIM_ClearFlag_CMP1
- TIMxICR CMP1C LL_HRTIM_ClearFlag_CMP1

LL_HRTIM_IsActiveFlag_CMP1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_CMP1 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the compare match 1 interrupt has occurred for a given timer (including the master timer) .

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of MCMP1/CMP1 bit in HRTIM_MISR/HRTIM_TIMxISR register (1 or 0).

Reference Manual to LL API cross reference:

- MISR MCMP1 LL_HRTIM_IsActiveFlag_CMP1
- TIMxISR CMP1 LL_HRTIM_IsActiveFlag_CMP1

LL_HRTIM_ClearFlag_CMP2

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_CMP2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Clear the compare 2 match interrupt for a given timer (including the master timer).

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MICR MCMP2C LL_HRTIM_ClearFlag_CMP2
- TIMxICR CMP2C LL_HRTIM_ClearFlag_CMP2

LL_HRTIM_IsActiveFlag_CMP2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_CMP2 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the compare match 2 interrupt has occurred for a given timer (including the master timer) .

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of MCMP2/CMP2 bit in HRTIM_MISR/HRTIM_TIMxISR register (1 or 0).

Reference Manual to LL API cross reference:

- MISR MCMP2 LL_HRTIM_IsActiveFlag_CMP2
- TIMxISR CMP2 LL_HRTIM_IsActiveFlag_CMP2

LL_HRTIM_ClearFlag_CMP3

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_CMP3 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Clear the compare 3 match interrupt for a given timer (including the master timer).

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MICR MCMP3C LL_HRTIM_ClearFlag_CMP3
- TIMxICR CMP3C LL_HRTIM_ClearFlag_CMP3

LL_HRTIM_IsActiveFlag_CMP3

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_CMP3 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the compare match 3 interrupt has occurred for a given timer (including the master timer) .

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of MCMP3/CMP3 bit in HRTIM_MISR/HRTIM_TIMxISR register (1 or 0).

Reference Manual to LL API cross reference:

- MISR MCMP3 LL_HRTIM_IsActiveFlag_CMP3
- TIMxISR CMP3 LL_HRTIM_IsActiveFlag_CMP3

LL_HRTIM_ClearFlag_CMP4

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_CMP4 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Clear the compare 4 match interrupt for a given timer (including the master timer).

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MICR MCMP4C LL_HRTIM_ClearFlag_CMP4
- TIMxICR CMP4C LL_HRTIM_ClearFlag_CMP4

LL_HRTIM_IsActiveFlag_CMP4

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_CMP4 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the compare match 4 interrupt has occurred for a given timer (including the master timer) .

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of MCMP4/CMP4 bit in HRTIM_MISR/HRTIM_TIMxISR register (1 or 0).

Reference Manual to LL API cross reference:

- MISR MCMP4 LL_HRTIM_IsActiveFlag_CMP4
- TIMxISR CMP4 LL_HRTIM_IsActiveFlag_CMP4

LL_HRTIM_ClearFlag_CPT1

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_CPT1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Clear the capture 1 interrupt flag for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxICR CPT1C LL_HRTIM_ClearFlag_CPT1

LL_HRTIM_IsActiveFlag_CPT1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_CPT1 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the capture 1 interrupt occurred for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of CPT1 bit in HRTIM_TIMxISR register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxISR CPT1 LL_HRTIM_IsActiveFlag_CPT1

LL_HRTIM_ClearFlag_CPT2

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_CPT2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Clear the capture 2 interrupt flag for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxICR CPT2C LL_HRTIM_ClearFlag_CPT2

LL_HRTIM_IsActiveFlag_CPT2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_CPT2 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the capture 2 interrupt occurred for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of CPT2 bit in HRTIM_TIMxISR register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxISR CPT2 LL_HRTIM_IsActiveFlag_CPT2

LL_HRTIM_ClearFlag_SET1

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_SET1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Clear the output 1 set interrupt flag for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxICR SET1C LL_HRTIM_ClearFlag_SET1

LL_HRTIM_IsActiveFlag_SET1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_SET1 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the output 1 set interrupt occurred for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of SETx1 bit in HRTIM_TIMxISR register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxISR SET1 LL_HRTIM_IsActiveFlag_SET1

LL_HRTIM_ClearFlag_RST1

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_RST1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Clear the output 1 reset interrupt flag for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxICR RST1C LL_HRTIM_ClearFlag_RST1

LL_HRTIM_IsActiveFlag_RST1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_RST1 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the output 1 reset interrupt occurred for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of RSTx1 bit in HRTIM_TIMxISR register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxISR RST1 LL_HRTIM_IsActiveFlag_RST1

LL_HRTIM_ClearFlag_SET2

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_SET2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Clear the output 2 set interrupt flag for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxICR SET2C LL_HRTIM_ClearFlag_SET2

LL_HRTIM_IsActiveFlag_SET2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_SET2 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the output 2 set interrupt occurred for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of SETx2 bit in HRTIM_TIMxISR register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxISR SET2 LL_HRTIM_IsActiveFlag_SET2

LL_HRTIM_ClearFlag_RST2

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_RST2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Clear the output 2reset interrupt flag for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxICR RST2C LL_HRTIM_ClearFlag_RST2

LL_HRTIM_IsActiveFlag_RST2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_RST2 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the output 2 reset interrupt occurred for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of RSTx2 bit in HRTIM_TIMxISR register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxISR RST2 LL_HRTIM_IsActiveFlag_RST2

LL_HRTIM_ClearFlag_RST

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_RST (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Clear the reset and/or roll-over interrupt flag for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxICR RSTC LL_HRTIM_ClearFlag_RST

LL_HRTIM_IsActiveFlag_RST

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_RST (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the reset and/or roll-over interrupt occurred for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of RST bit in HRTIM_TIMxISR register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxISR RST LL_HRTIM_IsActiveFlag_RST

LL_HRTIM_ClearFlag_DLYPRT

Function name

__STATIC_INLINE void LL_HRTIM_ClearFlag_DLYPRT (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Clear the delayed protection interrupt flag for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxICR DLYPRTC LL_HRTIM_ClearFlag_DLYPRT

LL_HRTIM_IsActiveFlag_DLYPRT

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsActiveFlag_DLYPRT (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the delayed protection interrupt occurred for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of DLYPRT bit in HRTIM_TIMxISR register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxISR DLYPRT LL_HRTIM_IsActiveFlag_DLYPRT

LL_HRTIM_EnableIT_FLT1

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_FLT1 (HRTIM_TypeDef * HRTIMx)

Function description

Enable the fault 1 interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER FLT1IE LL_HRTIM_EnableIT_FLT1

LL_HRTIM_DisableIT_FLT1

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_FLT1 (HRTIM_TypeDef * HRTIMx)

Function description

Disable the fault 1 interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER FLT1IE LL_HRTIM_DisableIT_FLT1

LL_HRTIM_IsEnabledIT_FLT1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_FLT1 (const HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether the fault 1 interrupt is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of FLT1IE bit in HRTIM_IER register (1 or 0).

Reference Manual to LL API cross reference:

- IER FLT1IE LL_HRTIM_IsEnabledIT_FLT1

LL_HRTIM_EnableIT_FLT2

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_FLT2 (HRTIM_TypeDef * HRTIMx)

Function description

Enable the fault 2 interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER FLT2IE LL_HRTIM_EnableIT_FLT2

LL_HRTIM_DisableIT_FLT2

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_FLT2 (HRTIM_TypeDef * HRTIMx)

Function description

Disable the fault 2 interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER FLT2IE LL_HRTIM_DisableIT_FLT2

LL_HRTIM_IsEnabledIT_FLT2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_FLT2 (const HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether the fault 2 interrupt is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of FLT2IE bit in HRTIM_IER register (1 or 0).

Reference Manual to LL API cross reference:

- IER FLT2IE LL_HRTIM_IsEnabledIT_FLT2

LL_HRTIM_EnableIT_FLT3

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_FLT3 (HRTIM_TypeDef * HRTIMx)

Function description

Enable the fault 3 interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER FLT3IE LL_HRTIM_EnableIT_FLT3

LL_HRTIM_DisableIT_FLT3

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_FLT3 (HRTIM_TypeDef * HRTIMx)

Function description

Disable the fault 3 interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER FLT3IE LL_HRTIM_DisableIT_FLT3

LL_HRTIM_IsEnabledIT_FLT3

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_FLT3 (const HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether the fault 3 interrupt is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of FLT3IE bit in HRTIM_IER register (1 or 0).

Reference Manual to LL API cross reference:

- IER FLT3IE LL_HRTIM_IsEnabledIT_FLT3

LL_HRTIM_EnableIT_FLT4

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_FLT4 (HRTIM_TypeDef * HRTIMx)

Function description

Enable the fault 4 interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER FLT4IE LL_HRTIM_EnableIT_FLT4

LL_HRTIM_DisableIT_FLT4

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_FLT4 (HRTIM_TypeDef * HRTIMx)

Function description

Disable the fault 4 interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER FLT4IE LL_HRTIM_DisableIT_FLT4

LL_HRTIM_IsEnabledIT_FLT4

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_FLT4 (const HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether the fault 4 interrupt is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of FLT4IE bit in HRTIM_IER register (1 or 0).

Reference Manual to LL API cross reference:

- IER FLT4IE LL_HRTIM_IsEnabledIT_FLT4

LL_HRTIM_EnableIT_FLT5

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_FLT5 (HRTIM_TypeDef * HRTIMx)

Function description

Enable the fault 5 interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER FLT5IE LL_HRTIM_EnableIT_FLT5

LL_HRTIM_DisableIT_FLT5

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_FLT5 (HRTIM_TypeDef * HRTIMx)

Function description

Disable the fault 5 interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER FLT5IE LL_HRTIM_DisableIT_FLT5

LL_HRTIM_IsEnabledIT_FLT5

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_FLT5 (const HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether the fault 5 interrupt is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of FLT5IE bit in HRTIM_IER register (1 or 0).

Reference Manual to LL API cross reference:

- IER FLT5IE LL_HRTIM_IsEnabledIT_FLT5

LL_HRTIM_EnableIT_SYSFLT

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_SYSFLT (HRTIM_TypeDef * HRTIMx)

Function description

Enable the system fault interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER SYSFLTIE LL_HRTIM_EnableIT_SYSFLT

LL_HRTIM_DisableIT_SYSFLT

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_SYSFLT (HRTIM_TypeDef * HRTIMx)

Function description

Disable the system fault interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER SYSFLTIE LL_HRTIM_DisableIT_SYSFLT

LL_HRTIM_IsEnabledIT_SYSFLT

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_SYSFLT (const HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether the system fault interrupt is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of SYSFLTIE bit in HRTIM_IER register (1 or 0).

Reference Manual to LL API cross reference:

- IER SYSFLTIE LL_HRTIM_IsEnabledIT_SYSFLT

LL_HRTIM_EnableIT_BMPER

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_BMPER (HRTIM_TypeDef * HRTIMx)

Function description

Enable the burst mode period interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER BMPERIE LL_HRTIM_EnableIT_BMPER

LL_HRTIM_DisableIT_BMPER

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_BMPER (HRTIM_TypeDef * HRTIMx)

Function description

Disable the burst mode period interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER BMPERIE LL_HRTIM_DisableIT_BMPER

LL_HRTIM_IsEnabledIT_BMPER

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_BMPER (const HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether the burst mode period interrupt is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of BMPERIE bit in HRTIM_IER register (1 or 0).

Reference Manual to LL API cross reference:

- IER BMPERIE LL_HRTIM_IsEnabledIT_BMPER

LL_HRTIM_EnableIT_SYNC

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_SYNC (HRTIM_TypeDef * HRTIMx)

Function description

Enable the synchronization input interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER SYNCIE LL_HRTIM_EnableIT_SYNC

LL_HRTIM_DisableIT_SYNC

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_SYNC (HRTIM_TypeDef * HRTIMx)

Function description

Disable the synchronization input interrupt.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER SYNCIE LL_HRTIM_DisableIT_SYNC

LL_HRTIM_IsEnabledIT_SYNC

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_SYNC (const HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether the synchronization input interrupt is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of SYNCIE bit in HRTIM_MDIER register (1 or 0).

Reference Manual to LL API cross reference:

- MDIER SYNCIE LL_HRTIM_IsEnabledIT_SYNC

LL_HRTIM_EnableIT_UPDATE

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_UPDATE (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the update interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MUPDIE LL_HRTIM_EnableIT_UPDATE
- TIMxDIER UPDIE LL_HRTIM_EnableIT_UPDATE

LL_HRTIM_DisableIT_UPDATE

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_UPDATE (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the update interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MUPDIE LL_HRTIM_DisableIT_UPDATE
- TIMxDIER UPDIE LL_HRTIM_DisableIT_UPDATE

LL_HRTIM_IsEnabledIT_UPDATE

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_UPDATE (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the update interrupt is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of MUPDIE/UPDIE bit in HRTIM_MDIER/HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- MDIER MUPDIE LL_HRTIM_IsEnabledIT_UPDATE
- TIMxDIER UPDIE LL_HRTIM_IsEnabledIT_UPDATE

LL_HRTIM_EnableIT_REP

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_REP (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the repetition interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MREPIE LL_HRTIM_EnableIT_REP
- TIMxDIER REPIE LL_HRTIM_EnableIT_REP

LL_HRTIM_DisableIT_REP

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_REP (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the repetition interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MREPIE LL_HRTIM_DisableIT_REP
- TIMxDIER REPIE LL_HRTIM_DisableIT_REP

LL_HRTIM_IsEnabledIT_REP

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_REP (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the repetition interrupt is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of MREPIE/REPIE bit in HRTIM_MDIER/HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- MDIER MREPIE LL_HRTIM_IsEnabledIT_REP
- TIMxDIER REPIE LL_HRTIM_IsEnabledIT_REP

LL_HRTIM_EnableIT_CMP1

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_CMP1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the compare 1 interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP1IE LL_HRTIM_EnableIT_CMP1
- TIMxDIER CMP1IE LL_HRTIM_EnableIT_CMP1

LL_HRTIM_DisableIT_CMP1

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_CMP1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the compare 1 interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP1IE LL_HRTIM_DisableIT_CMP1
- TIMxDIER CMP1IE LL_HRTIM_DisableIT_CMP1

LL_HRTIM_IsEnabledIT_CMP1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_CMP1 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the compare 1 interrupt is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of MCMP1IE/CMP1IE bit in HRTIM_MDIER/HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- MDIER MCMP1IE LL_HRTIM_IsEnabledIT_CMP1
- TIMxDIER CMP1IE LL_HRTIM_IsEnabledIT_CMP1

LL_HRTIM_EnableIT_CMP2

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_CMP2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the compare 2 interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP2IE LL_HRTIM_EnableIT_CMP2
- TIMxDIER CMP2IE LL_HRTIM_EnableIT_CMP2

LL_HRTIM_DisableIT_CMP2

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_CMP2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the compare 2 interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP2IE LL_HRTIM_DisableIT_CMP2
- TIMxDIER CMP2IE LL_HRTIM_DisableIT_CMP2

LL_HRTIM_IsEnabledIT_CMP2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_CMP2 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the compare 2 interrupt is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of MCMP2IE/CMP2IE bit in HRTIM_MDIER/HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- MDIER MCMP2IE LL_HRTIM_IsEnabledIT_CMP2
- TIMxDIER CMP2IE LL_HRTIM_IsEnabledIT_CMP2

LL_HRTIM_EnableIT_CMP3

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_CMP3 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the compare 3 interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP3IE LL_HRTIM_EnableIT_CMP3
- TIMxDIER CMP3IE LL_HRTIM_EnableIT_CMP3

LL_HRTIM_DisableIT_CMP3

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_CMP3 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the compare 3 interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP3IE LL_HRTIM_DisableIT_CMP3
- TIMxDIER CMP3IE LL_HRTIM_DisableIT_CMP3

LL_HRTIM_IsEnabledIT_CMP3

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_CMP3 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the compare 3 interrupt is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of MCMP3IE/CMP3IE bit in HRTIM_MDIER/HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- MDIER MCMP3IE LL_HRTIM_IsEnabledIT_CMP3
- TIMxDIER CMP3IE LL_HRTIM_IsEnabledIT_CMP3

LL_HRTIM_EnableIT_CMP4

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_CMP4 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the compare 4 interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP4IE LL_HRTIM_EnableIT_CMP4
- TIMxDIER CMP4IE LL_HRTIM_EnableIT_CMP4

LL_HRTIM_DisableIT_CMP4

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_CMP4 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the compare 4 interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP4IE LL_HRTIM_DisableIT_CMP4
- TIMxDIER CMP4IE LL_HRTIM_DisableIT_CMP4

LL_HRTIM_IsEnabledIT_CMP4

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_CMP4 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the compare 4 interrupt is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of MCMP4IE/CMP4IE bit in HRTIM_MDIER/HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- MDIER MCMP4IE LL_HRTIM_IsEnabledIT_CMP4
- TIMxDIER CMP4IE LL_HRTIM_IsEnabledIT_CMP4

LL_HRTIM_EnableIT_CPT1

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_CPT1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the capture 1 interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER CPT1IE LL_HRTIM_EnableIT_CPT1

LL_HRTIM_DisableIT_CPT1

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_CPT1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the capture 1 interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER CPT1IE LL_HRTIM_DisableIT_CPT1

LL_HRTIM_IsEnabledIT_CPT1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_CPT1 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the capture 1 interrupt is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of CPT1IE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER CPT1IE LL_HRTIM_IsEnabledIT_CPT1

LL_HRTIM_EnableIT_CPT2

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_CPT2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the capture 2 interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER CPT2IE LL_HRTIM_EnableIT_CPT2

LL_HRTIM_DisableIT_CPT2

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_CPT2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the capture 2 interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER CPT2IE LL_HRTIM_DisableIT_CPT2

LL_HRTIM_IsEnabledIT_CPT2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_CPT2 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the capture 2 interrupt is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of CPT2IE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER CPT2IE LL_HRTIM_IsEnabledIT_CPT2

LL_HRTIM_EnableIT_SET1

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_SET1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the output 1 set interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER SET1IE LL_HRTIM_EnableIT_SET1

LL_HRTIM_DisableIT_SET1

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_SET1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the output 1 set interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER SET1IE LL_HRTIM_DisableIT_SET1

LL_HRTIM_IsEnabledIT_SET1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_SET1 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the output 1 set interrupt is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of SET1xIE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER SET1IE LL_HRTIM_IsEnabledIT_SET1

LL_HRTIM_EnableIT_RST1

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_RST1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the output 1 reset interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER RST1IE LL_HRTIM_EnableIT_RST1

LL_HRTIM_DisableIT_RST1

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_RST1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the output 1 reset interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER RST1IE LL_HRTIM_DisableIT_RST1

LL_HRTIM_IsEnabledIT_RST1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_RST1 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the output 1 reset interrupt is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of RST1xIE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER RST1IE LL_HRTIM_IsEnabledIT_RST1

LL_HRTIM_EnableIT_SET2

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_SET2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the output 2 set interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER SET2IE LL_HRTIM_EnableIT_SET2

LL_HRTIM_DisableIT_SET2

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_SET2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the output 2 set interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER SET2IE LL_HRTIM_DisableIT_SET2

LL_HRTIM_IsEnabledIT_SET2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_SET2 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the output 2 set interrupt is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of SET2xIE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER SET2IE LL_HRTIM_IsEnabledIT_SET2

LL_HRTIM_EnableIT_RST2

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_RST2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the output 2 reset interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER RST2IE LL_HRTIM_EnableIT_RST2

LL_HRTIM_DisableIT_RST2

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_RST2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the output 2 reset interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER RST2IE LL_HRTIM_DisableIT_RST2

LL_HRTIM_IsEnabledIT_RST2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_RST2 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the output 2 reset LL_HRTIM_IsEnabledIT_RST2 is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of RST2xIE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER RST2IE LL_HRTIM_DisableIT_RST2

LL_HRTIM_EnableIT_RST

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_RST (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the reset/roll-over interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER RSTIE LL_HRTIM_EnableIT_RST

LL_HRTIM_DisableIT_RST

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_RST (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the reset/roll-over interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER RSTIE LL_HRTIM_DisableIT_RST

LL_HRTIM_IsEnabledIT_RST

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_RST (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the reset/roll-over interrupt is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of RSTIE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER RSTIE LL_HRTIM_IsEnabledIT_RST

LL_HRTIM_EnableIT_DLYPRT

Function name

__STATIC_INLINE void LL_HRTIM_EnableIT_DLYPRT (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the delayed protection interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER DLYPRTIE LL_HRTIM_EnableIT_DLYPRT

LL_HRTIM_DisableIT_DLYPRT

Function name

__STATIC_INLINE void LL_HRTIM_DisableIT_DLYPRT (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the delayed protection interrupt for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER DLYPRTIE LL_HRTIM_DisableIT_DLYPRT

LL_HRTIM_IsEnabledIT_DLYPRT

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledIT_DLYPRT (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the delayed protection interrupt is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of DLYPRTIE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER DLYPRTIE LL_HRTIM_IsEnabledIT_DLYPRT

LL_HRTIM_EnableDMAReq_SYNC

Function name

__STATIC_INLINE void LL_HRTIM_EnableDMAReq_SYNC (HRTIM_TypeDef * HRTIMx)

Function description

Enable the synchronization input DMA request.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER SYNCDE LL_HRTIM_EnableDMAReq_SYNC

LL_HRTIM_DisableDMAReq_SYNC

Function name

__STATIC_INLINE void LL_HRTIM_DisableDMAReq_SYNC (HRTIM_TypeDef * HRTIMx)

Function description

Disable the synchronization input DMA request.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER SYNCDE LL_HRTIM_DisableDMAReq_SYNC

LL_HRTIM_IsEnabledDMAReq_SYNC

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledDMAReq_SYNC (const HRTIM_TypeDef * HRTIMx)

Function description

Indicate whether the synchronization input DMA request is enabled.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **State:** of SYNCDE bit in HRTIM_MDIER register (1 or 0).

Reference Manual to LL API cross reference:

- MDIER SYNCDE LL_HRTIM_IsEnabledDMAReq_SYNC

LL_HRTIM_EnableDMAReq_UPDATE

Function name

__STATIC_INLINE void LL_HRTIM_EnableDMAReq_UPDATE (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the update DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MUPDDE LL_HRTIM_EnableDMAReq_UPDATE
- TIMxDIER UPDDE LL_HRTIM_EnableDMAReq_UPDATE

LL_HRTIM_DisableDMAReq_UPDATE

Function name

__STATIC_INLINE void LL_HRTIM_DisableDMAReq_UPDATE (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the update DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MUPDDE LL_HRTIM_DisableDMAReq_UPDATE
- TIMxDIER UPDDE LL_HRTIM_DisableDMAReq_UPDATE

LL_HRTIM_IsEnabledDMAReq_UPDATE

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledDMAReq_UPDATE (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the update DMA request is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of MUPDDE/UPDDE bit in HRTIM_MDIER/HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- MDIER MUPDDE LL_HRTIM_IsEnabledDMAReq_UPDATE
- TIMxDIER UPDDE LL_HRTIM_IsEnabledDMAReq_UPDATE

LL_HRTIM_EnableDMAReq_REP

Function name

__STATIC_INLINE void LL_HRTIM_EnableDMAReq_REP (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the repetition DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MREPDE LL_HRTIM_EnableDMAReq_REP
- TIMxDIER REPDE LL_HRTIM_EnableDMAReq_REP

LL_HRTIM_DisableDMAReq_REP

Function name

__STATIC_INLINE void LL_HRTIM_DisableDMAReq_REP (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the repetition DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MREPDE LL_HRTIM_DisableDMAReq_REP
- TIMxDIER REPDE LL_HRTIM_DisableDMAReq_REP

LL_HRTIM_IsEnabledDMAReq_REP

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledDMAReq_REP (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the repetition DMA request is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of MREPDE/REPDE bit in HRTIM_MDIER/HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- MDIER MREPDE LL_HRTIM_IsEnabledDMAReq_REP
- TIMxDIER REPDE LL_HRTIM_IsEnabledDMAReq_REP

LL_HRTIM_EnableDMAReq_CMP1

Function name

__STATIC_INLINE void LL_HRTIM_EnableDMAReq_CMP1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the compare 1 DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP1DE LL_HRTIM_EnableDMAReq_CMP1
- TIMxDIER CMP1DE LL_HRTIM_EnableDMAReq_CMP1

LL_HRTIM_DisableDMAReq_CMP1

Function name

__STATIC_INLINE void LL_HRTIM_DisableDMAReq_CMP1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the compare 1 DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP1DE LL_HRTIM_DisableDMAReq_CMP1
- TIMxDIER CMP1DE LL_HRTIM_DisableDMAReq_CMP1

LL_HRTIM_IsEnabledDMAReq_CMP1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledDMAReq_CMP1 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the compare 1 DMA request is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of MCMP1DE/CMP1DE bit in HRTIM_MDIER/HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- MDIER MCMP1DE LL_HRTIM_IsEnabledDMAReq_CMP1
- TIMxDIER CMP1DE LL_HRTIM_IsEnabledDMAReq_CMP1

LL_HRTIM_EnableDMAReq_CMP2

Function name

__STATIC_INLINE void LL_HRTIM_EnableDMAReq_CMP2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the compare 2 DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP2DE LL_HRTIM_EnableDMAReq_CMP2
- TIMxDIER CMP2DE LL_HRTIM_EnableDMAReq_CMP2

LL_HRTIM_DisableDMAReq_CMP2

Function name

__STATIC_INLINE void LL_HRTIM_DisableDMAReq_CMP2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the compare 2 DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP2DE LL_HRTIM_DisableDMAReq_CMP2
- TIMxDIER CMP2DE LL_HRTIM_DisableDMAReq_CMP2

LL_HRTIM_IsEnabledDMAReq_CMP2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledDMAReq_CMP2 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the compare 2 DMA request is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of MCMP2DE/CMP2DE bit in HRTIM_MDIER/HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- MDIER MCMP2DE LL_HRTIM_IsEnabledDMAReq_CMP2
- TIMxDIER CMP2DE LL_HRTIM_IsEnabledDMAReq_CMP2

LL_HRTIM_EnableDMAReq_CMP3

Function name

__STATIC_INLINE void LL_HRTIM_EnableDMAReq_CMP3 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the compare 3 DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP3DE LL_HRTIM_EnableDMAReq_CMP3
- TIMxDIER CMP3DE LL_HRTIM_EnableDMAReq_CMP3

LL_HRTIM_DisableDMAReq_CMP3

Function name

__STATIC_INLINE void LL_HRTIM_DisableDMAReq_CMP3 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the compare 3 DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP3DE LL_HRTIM_DisableDMAReq_CMP3
- TIMxDIER CMP3DE LL_HRTIM_DisableDMAReq_CMP3

LL_HRTIM_IsEnabledDMAReq_CMP3

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledDMAReq_CMP3 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the compare 3 DMA request is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of MCMP3DE/CMP3DE bit in HRTIM_MDIER/HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- MDIER MCMP3DE LL_HRTIM_IsEnabledDMAReq_CMP3
- TIMxDIER CMP3DE LL_HRTIM_IsEnabledDMAReq_CMP3

LL_HRTIM_EnableDMAReq_CMP4

Function name

__STATIC_INLINE void LL_HRTIM_EnableDMAReq_CMP4 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the compare 4 DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP4DE LL_HRTIM_EnableDMAReq_CMP4
- TIMxDIER CMP4DE LL_HRTIM_EnableDMAReq_CMP4

LL_HRTIM_DisableDMAReq_CMP4

Function name

__STATIC_INLINE void LL_HRTIM_DisableDMAReq_CMP4 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the compare 4 DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- MDIER MCMP4DE LL_HRTIM_DisableDMAReq_CMP4
- TIMxDIER CMP4DE LL_HRTIM_DisableDMAReq_CMP4

LL_HRTIM_IsEnabledDMAReq_CMP4

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledDMAReq_CMP4 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the compare 4 DMA request is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_MASTER
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of MCMP4DE/CMP4DE bit in HRTIM_MDIER/HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- MDIER MCMP4DE LL_HRTIM_IsEnabledDMAReq_CMP4
- TIMxDIER CMP4DE LL_HRTIM_IsEnabledDMAReq_CMP4

LL_HRTIM_EnableDMAReq_CPT1

Function name

__STATIC_INLINE void LL_HRTIM_EnableDMAReq_CPT1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the capture 1 DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER CPT1DE LL_HRTIM_EnableDMAReq_CPT1

LL_HRTIM_DisableDMAReq_CPT1

Function name

__STATIC_INLINE void LL_HRTIM_DisableDMAReq_CPT1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the capture 1 DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER CPT1DE LL_HRTIM_DisableDMAReq_CPT1

LL_HRTIM_IsEnabledDMAReq_CPT1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledDMAReq_CPT1 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the capture 1 DMA request is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of CPT1DE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER CPT1DE LL_HRTIM_IsEnabledDMAReq_CPT1

LL_HRTIM_EnableDMAReq_CPT2

Function name

__STATIC_INLINE void LL_HRTIM_EnableDMAReq_CPT2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the capture 2 DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER CPT2DE LL_HRTIM_EnableDMAReq_CPT2

LL_HRTIM_DisableDMAReq_CPT2

Function name

__STATIC_INLINE void LL_HRTIM_DisableDMAReq_CPT2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the capture 2 DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER CPT2DE LL_HRTIM_DisableDMAReq_CPT2

LL_HRTIM_IsEnabledDMAReq_CPT2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledDMAReq_CPT2 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the capture 2 DMA request is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of CPT2DE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER CPT2DE LL_HRTIM_IsEnabledDMAReq_CPT2

LL_HRTIM_EnableDMAReq_SET1

Function name

__STATIC_INLINE void LL_HRTIM_EnableDMAReq_SET1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the output 1 set DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER SET1DE LL_HRTIM_EnableDMAReq_SET1

LL_HRTIM_DisableDMAReq_SET1

Function name

__STATIC_INLINE void LL_HRTIM_DisableDMAReq_SET1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the output 1 set DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER SET1DE LL_HRTIM_DisableDMAReq_SET1

LL_HRTIM_IsEnabledDMAReq_SET1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledDMAReq_SET1 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the output 1 set DMA request is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of SET1xDE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER SET1DE LL_HRTIM_IsEnabledDMAReq_SET1

LL_HRTIM_EnableDMAReq_RST1

Function name

__STATIC_INLINE void LL_HRTIM_EnableDMAReq_RST1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the output 1 reset DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER RST1DE LL_HRTIM_EnableDMAReq_RST1

LL_HRTIM_DisableDMAReq_RST1

Function name

__STATIC_INLINE void LL_HRTIM_DisableDMAReq_RST1 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the output 1 reset DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER RST1DE LL_HRTIM_DisableDMAReq_RST1

LL_HRTIM_IsEnabledDMAReq_RST1

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledDMAReq_RST1 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the output 1 reset interrupt is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of RST1xDE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER RST1DE LL_HRTIM_IsEnabledDMAReq_RST1

LL_HRTIM_EnableDMAReq_SET2

Function name

__STATIC_INLINE void LL_HRTIM_EnableDMAReq_SET2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the output 2 set DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER SET2DE LL_HRTIM_EnableDMAReq_SET2

LL_HRTIM_DisableDMAReq_SET2

Function name

__STATIC_INLINE void LL_HRTIM_DisableDMAReq_SET2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the output 2 set DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER SET2DE LL_HRTIM_DisableDMAReq_SET2

LL_HRTIM_IsEnabledDMAReq_SET2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledDMAReq_SET2 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the output 2 set DMA request is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of SET2xDE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER SET2DE LL_HRTIM_IsEnabledDMAReq_SET2

LL_HRTIM_EnableDMAReq_RST2

Function name

__STATIC_INLINE void LL_HRTIM_EnableDMAReq_RST2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the output 2 reset DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER RST2DE LL_HRTIM_EnableDMAReq_RST2

LL_HRTIM_DisableDMAReq_RST2

Function name

__STATIC_INLINE void LL_HRTIM_DisableDMAReq_RST2 (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the output 2 reset DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER RST2DE LL_HRTIM_DisableDMAReq_RST2

LL_HRTIM_IsEnabledDMAReq_RST2

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledDMAReq_RST2 (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the output 2 reset DMA request is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of RST2xDE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER RST2DE LL_HRTIM_IsEnabledDMAReq_RST2

LL_HRTIM_EnableDMAReq_RST

Function name

__STATIC_INLINE void LL_HRTIM_EnableDMAReq_RST (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the reset/roll-over DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER RSTDE LL_HRTIM_EnableDMAReq_RST

LL_HRTIM_DisableDMAReq_RST

Function name

__STATIC_INLINE void LL_HRTIM_DisableDMAReq_RST (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the reset/roll-over DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER RSTDE LL_HRTIM_DisableDMAReq_RST

LL_HRTIM_IsEnabledDMAReq_RST

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledDMAReq_RST (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the reset/roll-over DMA request is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of RSTDE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER RSTDE LL_HRTIM_IsEnabledDMAReq_RST

LL_HRTIM_EnableDMAReq_DLYPRT

Function name

__STATIC_INLINE void LL_HRTIM_EnableDMAReq_DLYPRT (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Enable the delayed protection DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER DLYPRTDE LL_HRTIM_EnableDMAReq_DLYPRT

LL_HRTIM_DisableDMAReq_DLYPRT

Function name

__STATIC_INLINE void LL_HRTIM_DisableDMAReq_DLYPRT (HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Disable the delayed protection DMA request for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMxDIER DLYPRTDE LL_HRTIM_DisableDMAReq_DLYPRT

LL_HRTIM_IsEnabledDMAReq_DLYPRT

Function name

__STATIC_INLINE uint32_t LL_HRTIM_IsEnabledDMAReq_DLYPRT (const HRTIM_TypeDef * HRTIMx, uint32_t Timer)

Function description

Indicate whether the delayed protection DMA request is enabled for a given timer.

Parameters

- **HRTIMx:** High Resolution Timer instance
- **Timer:** This parameter can be one of the following values:
  - LL_HRTIM_TIMER_A
  - LL_HRTIM_TIMER_B
  - LL_HRTIM_TIMER_C
  - LL_HRTIM_TIMER_D
  - LL_HRTIM_TIMER_E

Return values

- **State:** of DLYPRTDE bit in HRTIM_TIMxDIER register (1 or 0).

Reference Manual to LL API cross reference:

- TIMxDIER DLYPRTDE LL_HRTIM_IsEnabledDMAReq_DLYPRT

LL_HRTIM_DeInit

Function name

ErrorStatus LL_HRTIM_DeInit (HRTIM_TypeDef * HRTIMx)

Function description

Set HRTIM instance registers to their reset values.

Parameters

- **HRTIMx:** High Resolution Timer instance

Return values

- **ErrorStatus:** enumeration value:
  - SUCCESS: HRTIMx registers are de-initialized
  - ERROR: invalid HRTIMx instance

111.2 HRTIM Firmware driver defines

The following section lists the various define and macros of the module.

111.2.1 HRTIM

HRTIM

ADC TRIGGER

LL_HRTIM_ADCTRIG_1

ADC trigger 1 identifier

LL_HRTIM_ADCTRIG_2

ADC trigger 2 identifier

LL_HRTIM_ADCTRIG_3

ADC trigger 3 identifier

LL_HRTIM_ADCTRIG_4

ADC trigger 4 identifier

ADC TRIGGER 1/3 SOURCE

LL_HRTIM_ADCTRIG_SRC13_NONE

No ADC trigger event

LL_HRTIM_ADCTRIG_SRC13_MCMP1

ADC Trigger on master compare 1

LL_HRTIM_ADCTRIG_SRC13_MCMP2

ADC Trigger on master compare 2

LL_HRTIM_ADCTRIG_SRC13_MCMP3

ADC Trigger on master compare 3

LL_HRTIM_ADCTRIG_SRC13_MCMP4

ADC Trigger on master compare 4

LL_HRTIM_ADCTRIG_SRC13_MPER

ADC Trigger on master period

LL_HRTIM_ADCTRIG_SRC13_EEV1

ADC Trigger on external event 1

LL_HRTIM_ADCTRIG_SRC13_EEV2

ADC Trigger on external event 2

LL_HRTIM_ADCTRIG_SRC13_EEV3

ADC Trigger on external event 3

LL_HRTIM_ADCTRIG_SRC13_EEV4

ADC Trigger on external event 4

LL_HRTIM_ADCTRIG_SRC13_EEV5

ADC Trigger on external event 5

LL_HRTIM_ADCTRIG_SRC13_TIMACMP2

ADC Trigger on Timer A compare 2

LL_HRTIM_ADCTRIG_SRC13_TIMACMP3

ADC Trigger on Timer A compare 3

LL_HRTIM_ADCTRIG_SRC13_TIMACMP4

ADC Trigger on Timer A compare 4

LL_HRTIM_ADCTRIG_SRC13_TIMAPER

ADC Trigger on Timer A period

LL_HRTIM_ADCTRIG_SRC13_TIMARST

ADC Trigger on Timer A reset

LL_HRTIM_ADCTRIG_SRC13_TIMBCMP2

ADC Trigger on Timer B compare 2

LL_HRTIM_ADCTRIG_SRC13_TIMBCMP3

ADC Trigger on Timer B compare 3

LL_HRTIM_ADCTRIG_SRC13_TIMBCMP4

ADC Trigger on Timer B compare 4

LL_HRTIM_ADCTRIG_SRC13_TIMBPER

ADC Trigger on Timer B period

LL_HRTIM_ADCTRIG_SRC13_TIMBRST

ADC Trigger on Timer B reset

LL_HRTIM_ADCTRIG_SRC13_TIMCCMP2

ADC Trigger on Timer C compare 2

LL_HRTIM_ADCTRIG_SRC13_TIMCCMP3

ADC Trigger on Timer C compare 3

LL_HRTIM_ADCTRIG_SRC13_TIMCCMP4

ADC Trigger on Timer C compare 4

LL_HRTIM_ADCTRIG_SRC13_TIMCPER

ADC Trigger on Timer C period

LL_HRTIM_ADCTRIG_SRC13_TIMDCMP2

ADC Trigger on Timer D compare 2

LL_HRTIM_ADCTRIG_SRC13_TIMDCMP3

ADC Trigger on Timer D compare 3

LL_HRTIM_ADCTRIG_SRC13_TIMDCMP4

ADC Trigger on Timer D compare 4

LL_HRTIM_ADCTRIG_SRC13_TIMDPER

ADC Trigger on Timer D period

LL_HRTIM_ADCTRIG_SRC13_TIMECMP2

ADC Trigger on Timer E compare 2

LL_HRTIM_ADCTRIG_SRC13_TIMECMP3

ADC Trigger on Timer E compare 3

LL_HRTIM_ADCTRIG_SRC13_TIMECMP4

ADC Trigger on Timer E compare 4

LL_HRTIM_ADCTRIG_SRC13_TIMEPER

ADC Trigger on Timer E period

ADC TRIGGER 2/4 SOURCE

LL_HRTIM_ADCTRIG_SRC24_NONE

No ADC trigger event

LL_HRTIM_ADCTRIG_SRC24_MCMP1

ADC Trigger on master compare 1

LL_HRTIM_ADCTRIG_SRC24_MCMP2

ADC Trigger on master compare 2

LL_HRTIM_ADCTRIG_SRC24_MCMP3

ADC Trigger on master compare 3

LL_HRTIM_ADCTRIG_SRC24_MCMP4

ADC Trigger on master compare 4

LL_HRTIM_ADCTRIG_SRC24_MPER

ADC Trigger on master period

LL_HRTIM_ADCTRIG_SRC24_EEV6

ADC Trigger on external event 6

LL_HRTIM_ADCTRIG_SRC24_EEV7

ADC Trigger on external event 7

LL_HRTIM_ADCTRIG_SRC24_EEV8

ADC Trigger on external event 8

LL_HRTIM_ADCTRIG_SRC24_EEV9

ADC Trigger on external event 9

LL_HRTIM_ADCTRIG_SRC24_EEV10

ADC Trigger on external event 10

LL_HRTIM_ADCTRIG_SRC24_TIMACMP2

ADC Trigger on Timer A compare 2

LL_HRTIM_ADCTRIG_SRC24_TIMACMP3

ADC Trigger on Timer A compare 3

LL_HRTIM_ADCTRIG_SRC24_TIMACMP4

ADC Trigger on Timer A compare 4

LL_HRTIM_ADCTRIG_SRC24_TIMAPER

ADC Trigger on Timer A period

LL_HRTIM_ADCTRIG_SRC24_TIMBCMP2

ADC Trigger on Timer B compare 2

LL_HRTIM_ADCTRIG_SRC24_TIMBCMP3

ADC Trigger on Timer B compare 3

LL_HRTIM_ADCTRIG_SRC24_TIMBCMP4

ADC Trigger on Timer B compare 4

LL_HRTIM_ADCTRIG_SRC24_TIMBPER

ADC Trigger on Timer B period

LL_HRTIM_ADCTRIG_SRC24_TIMCCMP2

ADC Trigger on Timer C compare 2

LL_HRTIM_ADCTRIG_SRC24_TIMCCMP3

ADC Trigger on Timer C compare 3

LL_HRTIM_ADCTRIG_SRC24_TIMCCMP4

ADC Trigger on Timer C compare 4

LL_HRTIM_ADCTRIG_SRC24_TIMCPER

ADC Trigger on Timer C period

LL_HRTIM_ADCTRIG_SRC24_TIMCRST

ADC Trigger on Timer C reset

LL_HRTIM_ADCTRIG_SRC24_TIMDCMP2

ADC Trigger on Timer D compare 2

LL_HRTIM_ADCTRIG_SRC24_TIMDCMP3

ADC Trigger on Timer D compare 3

LL_HRTIM_ADCTRIG_SRC24_TIMDCMP4

ADC Trigger on Timer D compare 4

LL_HRTIM_ADCTRIG_SRC24_TIMDPER

ADC Trigger on Timer D period

LL_HRTIM_ADCTRIG_SRC24_TIMDRST

ADC Trigger on Timer D reset

LL_HRTIM_ADCTRIG_SRC24_TIMECMP2

ADC Trigger on Timer E compare 2

LL_HRTIM_ADCTRIG_SRC24_TIMECMP3

ADC Trigger on Timer E compare 3

LL_HRTIM_ADCTRIG_SRC24_TIMECMP4

ADC Trigger on Timer E compare 4

LL_HRTIM_ADCTRIG_SRC24_TIMERST

ADC Trigger on Timer E reset

ADC TRIGGER UPDATE

LL_HRTIM_ADCTRIG_UPDATE_MASTER

HRTIM_ADCxR register update is triggered by the Master timer

LL_HRTIM_ADCTRIG_UPDATE_TIMER_A

HRTIM_ADCxR register update is triggered by the Timer A

LL_HRTIM_ADCTRIG_UPDATE_TIMER_B

HRTIM_ADCxR register update is triggered by the Timer B

LL_HRTIM_ADCTRIG_UPDATE_TIMER_C

HRTIM_ADCxR register update is triggered by the Timer C

LL_HRTIM_ADCTRIG_UPDATE_TIMER_D

HRTIM_ADCxR register update is triggered by the Timer D

LL_HRTIM_ADCTRIG_UPDATE_TIMER_E

HRTIM_ADCxR register update is triggered by the Timer E

BURST MODE CLOCK SOURCE

LL_HRTIM_BM_CLKSRC_MASTER

Master timer counter reset/roll-over is used as clock source for the burst mode counter

LL_HRTIM_BM_CLKSRC_TIMER_A

Timer A counter reset/roll-over is used as clock source for the burst mode counter

LL_HRTIM_BM_CLKSRC_TIMER_B

Timer B counter reset/roll-over is used as clock source for the burst mode counter

LL_HRTIM_BM_CLKSRC_TIMER_C

Timer C counter reset/roll-over is used as clock source for the burst mode counter

LL_HRTIM_BM_CLKSRC_TIMER_D

Timer D counter reset/roll-over is used as clock source for the burst mode counter

LL_HRTIM_BM_CLKSRC_TIMER_E

Timer E counter reset/roll-over is used as clock source for the burst mode counter

LL_HRTIM_BM_CLKSRC_TIM16_OC

On-chip Event 1 (BMClk[1]), acting as a burst mode counter clock

LL_HRTIM_BM_CLKSRC_TIM17_OC

On-chip Event 2 (BMClk[2]), acting as a burst mode counter clock

LL_HRTIM_BM_CLKSRC_TIM7_TRGO

On-chip Event 3 (BMClk[3]), acting as a burst mode counter clock

LL_HRTIM_BM_CLKSRC_FHRTIM

Prescaled fHRTIM clock is used as clock source for the burst mode counter

BURST MODE OPERATING MODE

LL_HRTIM_BM_MODE_SINGLESHOT

Burst mode operates in single shot mode

LL_HRTIM_BM_MODE_CONTINOUS

Burst mode operates in continuous mode

BURST MODE PRESCALER

LL_HRTIM_BM_PRESCALER_DIV1

fBRST = fHRTIM

LL_HRTIM_BM_PRESCALER_DIV2

fBRST = fHRTIM/2

LL_HRTIM_BM_PRESCALER_DIV4

fBRST = fHRTIM/4

LL_HRTIM_BM_PRESCALER_DIV8

fBRST = fHRTIM/8

LL_HRTIM_BM_PRESCALER_DIV16

fBRST = fHRTIM/16

LL_HRTIM_BM_PRESCALER_DIV32

fBRST = fHRTIM/32

LL_HRTIM_BM_PRESCALER_DIV64

fBRST = fHRTIM/64

LL_HRTIM_BM_PRESCALER_DIV128

fBRST = fHRTIM/128

LL_HRTIM_BM_PRESCALER_DIV256

fBRST = fHRTIM/256

LL_HRTIM_BM_PRESCALER_DIV512

fBRST = fHRTIM/512

LL_HRTIM_BM_PRESCALER_DIV1024

fBRST = fHRTIM/1024

LL_HRTIM_BM_PRESCALER_DIV2048

fBRST = fHRTIM/2048

LL_HRTIM_BM_PRESCALER_DIV4096

fBRST = fHRTIM/4096

LL_HRTIM_BM_PRESCALER_DIV8192

fBRST = fHRTIM/8192

LL_HRTIM_BM_PRESCALER_DIV16384

fBRST = fHRTIM/16384

LL_HRTIM_BM_PRESCALER_DIV32768

fBRST = fHRTIM/32768

HRTIM BURST MODE STATUS

LL_HRTIM_BM_STATUS_NORMAL

Normal operation

LL_HRTIM_BM_STATUS_BURST_ONGOING

Burst operation on-going

HRTIM BURST MODE TRIGGER

LL_HRTIM_BM_TRIG_NONE

No trigger

LL_HRTIM_BM_TRIG_MASTER_RESET

Master timer reset event is starting the burst mode operation

LL_HRTIM_BM_TRIG_MASTER_REPETITION

Master timer repetition event is starting the burst mode operation

LL_HRTIM_BM_TRIG_MASTER_CMP1

Master timer compare 1 event is starting the burst mode operation

LL_HRTIM_BM_TRIG_MASTER_CMP2

Master timer compare 2 event is starting the burst mode operation

LL_HRTIM_BM_TRIG_MASTER_CMP3

Master timer compare 3 event is starting the burst mode operation

LL_HRTIM_BM_TRIG_MASTER_CMP4

Master timer compare 4 event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMA_RESET

Timer A reset event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMA_REPETITION

Timer A repetition event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMA_CMP1

Timer A compare 1 event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMA_CMP2

Timer A compare 2 event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMB_RESET

Timer B reset event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMB_REPETITION

Timer B repetition event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMB_CMP1

Timer B compare 1 event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMB_CMP2

Timer B compare 2 event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMC_RESET

Timer C resetevent is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMC_REPETITION

Timer C repetition event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMC_CMP1

Timer C compare 1 event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMC_CMP2

Timer C compare 2 event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMD_RESET

Timer D reset event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMD_REPETITION

Timer D repetition event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMD_CMP1

Timer D compare 1 event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMD_CMP2

Timer D compare 2 event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIME_RESET

Timer E reset event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIME_REPETITION

Timer E repetition event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIME_CMP1

Timer E compare 1 event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIME_CMP2

Timer E compare 2 event is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMA_EVENT7

Timer A period following an external event 7 (conditioned by TIMA filters) is starting the burst mode operation

LL_HRTIM_BM_TRIG_TIMD_EVENT8

Timer D period following an external event 8 (conditioned by TIMD filters) is starting the burst mode operation

LL_HRTIM_BM_TRIG_EVENT_7

External event 7 conditioned by TIMA filters is starting the burst mode operation

LL_HRTIM_BM_TRIG_EVENT_8

External event 8 conditioned by TIMD filters is starting the burst mode operation

LL_HRTIM_BM_TRIG_EVENT_ONCHIP

A rising edge on an on-chip Event (for instance from GP timer or comparator) triggers the burst mode operation

BURST DMA

LL_HRTIM_BURSTDMA_NONE

No register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_MCR

MCR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_MICR

MICR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_MDIER

MDIER register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_MCNT

MCNTR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_MPER

MPER register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_MREP

MREPR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_MCMP1

MCMP1R register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_MCMP2

MCMP2R register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_MCMP3

MCMP3R register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_MCMP4

MCMP4R register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMMCR

TIMxCR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMICR

TIMxICR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMDIER

TIMxDIER register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMCNT

CNTxCR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMPER

PERxR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMREP

REPxR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMCMP1

CMP1xR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMCMP2

CMP2xR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMCMP3

CMP3xR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMCMP4

CMP4xR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMDTR

DTxR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMSET1R

SET1R register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMRST1R

RST1R register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMSET2R

SET2R register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMRST2R

RST1R register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMEEFR1

EEFxR1 register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMEEFR2

EEFxR2 register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMRSTR

RSTxR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMCHPR

CHPxR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMOUTR

OUTxR register is updated by Burst DMA accesses

LL_HRTIM_BURSTDMA_TIMFLTR

FLTxR register is updated by Burst DMA accesses

BURST MODE

LL_HRTIM_BURSTMODE_MAINTAINCLOCK

Timer counter clock is maintained and the timer operates normally

LL_HRTIM_BURSTMODE_RESETCOUNTER

Timer counter clock is stopped and the counter is reset

CAPTURE TRIGGER

LL_HRTIM_CAPTURETRIG_NONE

Capture trigger is disabled

LL_HRTIM_CAPTURETRIG_UPDATE

The update event triggers the Capture

LL_HRTIM_CAPTURETRIG_EEV_1

The External event 1 triggers the Capture

LL_HRTIM_CAPTURETRIG_EEV_2

The External event 2 triggers the Capture

LL_HRTIM_CAPTURETRIG_EEV_3

The External event 3 triggers the Capture

LL_HRTIM_CAPTURETRIG_EEV_4

The External event 4 triggers the Capture

LL_HRTIM_CAPTURETRIG_EEV_5

The External event 5 triggers the Capture

LL_HRTIM_CAPTURETRIG_EEV_6

The External event 6 triggers the Capture

LL_HRTIM_CAPTURETRIG_EEV_7

The External event 7 triggers the Capture

LL_HRTIM_CAPTURETRIG_EEV_8

The External event 8 triggers the Capture

LL_HRTIM_CAPTURETRIG_EEV_9

The External event 9 triggers the Capture

LL_HRTIM_CAPTURETRIG_EEV_10

The External event 10 triggers the Capture

LL_HRTIM_CAPTURETRIG_TA1_SET

Capture is triggered by TA1 output inactive to active transition

LL_HRTIM_CAPTURETRIG_TA1_RESET

Capture is triggered by TA1 output active to inactive transition

LL_HRTIM_CAPTURETRIG_TIMA_CMP1

Timer A Compare 1 triggers Capture

LL_HRTIM_CAPTURETRIG_TIMA_CMP2

Timer A Compare 2 triggers Capture

LL_HRTIM_CAPTURETRIG_TB1_SET

Capture is triggered by TB1 output inactive to active transition

LL_HRTIM_CAPTURETRIG_TB1_RESET

Capture is triggered by TB1 output active to inactive transition

LL_HRTIM_CAPTURETRIG_TIMB_CMP1

Timer B Compare 1 triggers Capture

LL_HRTIM_CAPTURETRIG_TIMB_CMP2

Timer B Compare 2 triggers Capture

LL_HRTIM_CAPTURETRIG_TC1_SET

Capture is triggered by TC1 output inactive to active transition

LL_HRTIM_CAPTURETRIG_TC1_RESET

Capture is triggered by TC1 output active to inactive transition

LL_HRTIM_CAPTURETRIG_TIMC_CMP1

Timer C Compare 1 triggers Capture

LL_HRTIM_CAPTURETRIG_TIMC_CMP2

Timer C Compare 2 triggers Capture

LL_HRTIM_CAPTURETRIG_TD1_SET

Capture is triggered by TD1 output inactive to active transition

LL_HRTIM_CAPTURETRIG_TD1_RESET

Capture is triggered by TD1 output active to inactive transition

LL_HRTIM_CAPTURETRIG_TIMD_CMP1

Timer D Compare 1 triggers Capture

LL_HRTIM_CAPTURETRIG_TIMD_CMP2

Timer D Compare 2 triggers Capture

LL_HRTIM_CAPTURETRIG_TE1_SET

Capture is triggered by TE1 output inactive to active transition

LL_HRTIM_CAPTURETRIG_TE1_RESET

Capture is triggered by TE1 output active to inactive transition

LL_HRTIM_CAPTURETRIG_TIME_CMP1

Timer E Compare 1 triggers Capture

LL_HRTIM_CAPTURETRIG_TIME_CMP2

Timer E Compare 2 triggers Capture

CAPTURE UNIT ID

LL_HRTIM_CAPTUREUNIT_1

Capture unit 1 identifier

LL_HRTIM_CAPTUREUNIT_2

Capture unit 2 identifier

CHOPPER MODE DUTY CYCLE

LL_HRTIM_CHP_DUTYCYCLE_0

Only 1st pulse is present

LL_HRTIM_CHP_DUTYCYCLE_125

Duty cycle of the carrier signal is 12.5 %

LL_HRTIM_CHP_DUTYCYCLE_250

Duty cycle of the carrier signal is 25 %

LL_HRTIM_CHP_DUTYCYCLE_375

Duty cycle of the carrier signal is 37.5 %

LL_HRTIM_CHP_DUTYCYCLE_500

Duty cycle of the carrier signal is 50 %

LL_HRTIM_CHP_DUTYCYCLE_625

Duty cycle of the carrier signal is 62.5 %

LL_HRTIM_CHP_DUTYCYCLE_750

Duty cycle of the carrier signal is 75 %

LL_HRTIM_CHP_DUTYCYCLE_875

Duty cycle of the carrier signal is 87.5 %

CHOPPER MODE PRESCALER

LL_HRTIM_CHP_PRESCALER_DIV16

fCHPFRQ = fHRTIM / 16

LL_HRTIM_CHP_PRESCALER_DIV32

fCHPFRQ = fHRTIM / 32

LL_HRTIM_CHP_PRESCALER_DIV48

fCHPFRQ = fHRTIM / 48

LL_HRTIM_CHP_PRESCALER_DIV64

fCHPFRQ = fHRTIM / 64

LL_HRTIM_CHP_PRESCALER_DIV80

fCHPFRQ = fHRTIM / 80

LL_HRTIM_CHP_PRESCALER_DIV96

fCHPFRQ = fHRTIM / 96

LL_HRTIM_CHP_PRESCALER_DIV112

fCHPFRQ = fHRTIM / 112

LL_HRTIM_CHP_PRESCALER_DIV128

fCHPFRQ = fHRTIM / 128

LL_HRTIM_CHP_PRESCALER_DIV144

fCHPFRQ = fHRTIM / 144

LL_HRTIM_CHP_PRESCALER_DIV160

fCHPFRQ = fHRTIM / 160

LL_HRTIM_CHP_PRESCALER_DIV176

fCHPFRQ = fHRTIM / 176

LL_HRTIM_CHP_PRESCALER_DIV192

fCHPFRQ = fHRTIM / 192

LL_HRTIM_CHP_PRESCALER_DIV208

fCHPFRQ = fHRTIM / 208

LL_HRTIM_CHP_PRESCALER_DIV224

fCHPFRQ = fHRTIM / 224

LL_HRTIM_CHP_PRESCALER_DIV240

fCHPFRQ = fHRTIM / 240

LL_HRTIM_CHP_PRESCALER_DIV256

fCHPFRQ = fHRTIM / 256

CHOPPER MODE PULSE WIDTH

LL_HRTIM_CHP_PULSEWIDTH_16

tSTPW = tHRTIM x 16

LL_HRTIM_CHP_PULSEWIDTH_32

tSTPW = tHRTIM x 32

LL_HRTIM_CHP_PULSEWIDTH_48

tSTPW = tHRTIM x 48

LL_HRTIM_CHP_PULSEWIDTH_64

tSTPW = tHRTIM x 64

LL_HRTIM_CHP_PULSEWIDTH_80

tSTPW = tHRTIM x 80

LL_HRTIM_CHP_PULSEWIDTH_96

tSTPW = tHRTIM x 96

LL_HRTIM_CHP_PULSEWIDTH_112

tSTPW = tHRTIM x 112

LL_HRTIM_CHP_PULSEWIDTH_128

tSTPW = tHRTIM x 128

LL_HRTIM_CHP_PULSEWIDTH_144

tSTPW = tHRTIM x 144

LL_HRTIM_CHP_PULSEWIDTH_160

tSTPW = tHRTIM x 160

LL_HRTIM_CHP_PULSEWIDTH_176

tSTPW = tHRTIM x 176

LL_HRTIM_CHP_PULSEWIDTH_192

tSTPW = tHRTIM x 192

LL_HRTIM_CHP_PULSEWIDTH_208

tSTPW = tHRTIM x 208

LL_HRTIM_CHP_PULSEWIDTH_224

tSTPW = tHRTIM x 224

LL_HRTIM_CHP_PULSEWIDTH_240

tSTPW = tHRTIM x 240

LL_HRTIM_CHP_PULSEWIDTH_256

tSTPW = tHRTIM x 256

COMPARE MODE

LL_HRTIM_COMPAREMODE_REGULAR

standard compare mode

LL_HRTIM_COMPAREMODE_DELAY_NOTIMEOUT

Compare event generated only if a capture has occurred

LL_HRTIM_COMPAREMODE_DELAY_CMP1

Compare event generated if a capture has occurred or after a Compare 1 match (timeout if capture event is missing)

LL_HRTIM_COMPAREMODE_DELAY_CMP3

Compare event generated if a capture has occurred or after a Compare 3 match (timeout if capture event is missing)

COMPARE UNIT ID

LL_HRTIM_COMPAREUNIT_2

Compare unit 2 identifier

LL_HRTIM_COMPAREUNIT_4

Compare unit 4 identifier

CURRENT PUSH-PULL STATUS

LL_HRTIM_CPPSTAT_OUTPUT1

Signal applied on output 1 and output 2 forced inactive

LL_HRTIM_CPPSTAT_OUTPUT2

Signal applied on output 2 and output 1 forced inactive

CROSSBAR INPUT

LL_HRTIM_CROSSBAR_NONE

Reset the output set crossbar

LL_HRTIM_CROSSBAR_RESYNC

Timer reset event coming solely from software or SYNC input forces an output level transition

LL_HRTIM_CROSSBAR_TIMPER

Timer period event forces an output level transition

LL_HRTIM_CROSSBAR_TIMCMP1

Timer compare 1 event forces an output level transition

LL_HRTIM_CROSSBAR_TIMCMP2

Timer compare 2 event forces an output level transition

LL_HRTIM_CROSSBAR_TIMCMP3

Timer compare 3 event forces an output level transition

LL_HRTIM_CROSSBAR_TIMCMP4

Timer compare 4 event forces an output level transition

LL_HRTIM_CROSSBAR_MASTERPER

The master timer period event forces an output level transition

LL_HRTIM_CROSSBAR_MASTERCMP1

Master Timer compare 1 event forces an output level transition

LL_HRTIM_CROSSBAR_MASTERCMP2

Master Timer compare 2 event forces an output level transition

LL_HRTIM_CROSSBAR_MASTERCMP3

Master Timer compare 3 event forces an output level transition

LL_HRTIM_CROSSBAR_MASTERCMP4

Master Timer compare 4 event forces an output level transition

LL_HRTIM_CROSSBAR_TIMEV_1

Timer event 1 forces an output level transition

LL_HRTIM_CROSSBAR_TIMEV_2

Timer event 2 forces an output level transition

LL_HRTIM_CROSSBAR_TIMEV_3

Timer event 3 forces an output level transition

LL_HRTIM_CROSSBAR_TIMEV_4

Timer event 4 forces an output level transition

LL_HRTIM_CROSSBAR_TIMEV_5

Timer event 5 forces an output level transition

LL_HRTIM_CROSSBAR_TIMEV_6

Timer event 6 forces an output level transition

LL_HRTIM_CROSSBAR_TIMEV_7

Timer event 7 forces an output level transition

LL_HRTIM_CROSSBAR_TIMEV_8

Timer event 8 forces an output level transition

LL_HRTIM_CROSSBAR_TIMEV_9

Timer event 9 forces an output level transition

LL_HRTIM_CROSSBAR_EEV_1

External event 1 forces an output level transition

LL_HRTIM_CROSSBAR_EEV_2

External event 2 forces an output level transition

LL_HRTIM_CROSSBAR_EEV_3

External event 3 forces an output level transition

LL_HRTIM_CROSSBAR_EEV_4

External event 4 forces an output level transition

LL_HRTIM_CROSSBAR_EEV_5

External event 5 forces an output level transition

LL_HRTIM_CROSSBAR_EEV_6

External event 6 forces an output level transition

LL_HRTIM_CROSSBAR_EEV_7

External event 7 forces an output level transition

LL_HRTIM_CROSSBAR_EEV_8

External event 8 forces an output level transition

LL_HRTIM_CROSSBAR_EEV_9

External event 9 forces an output level transition

LL_HRTIM_CROSSBAR_EEV_10

External event 10 forces an output level transition

LL_HRTIM_CROSSBAR_UPDATE

Timer register update event forces an output level transition

DAC TRIGGER

LL_HRTIM_DACTRIG_NONE

No DAC synchronization event generated

LL_HRTIM_DACTRIG_DACTRIGOUT_1

DAC synchronization event generated on DACTrigOut1 output upon timer update

LL_HRTIM_DACTRIG_DACTRIGOUT_2

DAC synchronization event generated on DACTrigOut2 output upon timer update

LL_HRTIM_DACTRIG_DACTRIGOUT_3

DAC synchronization event generated on DACTrigOut3 output upon timer update

DELAYED PROTECTION (DLYPRT) MODE

LL_HRTIM_DLYPRT_DELAYOUT1_EEV6

Timers A, B, C: Output 1 delayed Idle on external Event 6

LL_HRTIM_DLYPRT_DELAYOUT2_EEV6

Timers A, B, C: Output 2 delayed Idle on external Event 6

LL_HRTIM_DLYPRT_DELAYBOTH_EEV6

Timers A, B, C: Output 1 and output 2 delayed Idle on external Event 6

LL_HRTIM_DLYPRT_BALANCED_EEV6

Timers A, B, C: Balanced Idle on external Event 6

LL_HRTIM_DLYPRT_DELAYOUT1_EEV7

Timers A, B, C: Output 1 delayed Idle on external Event 7

LL_HRTIM_DLYPRT_DELAYOUT2_EEV7

Timers A, B, C: Output 2 delayed Idle on external Event 7

LL_HRTIM_DLYPRT_DELAYBOTH_EEV7

Timers A, B, C: Output 1 and output2 delayed Idle on external Event 7

LL_HRTIM_DLYPRT_BALANCED_EEV7

Timers A, B, C: Balanced Idle on external Event 7

LL_HRTIM_DLYPRT_DELAYOUT1_EEV8

Timers D, E: Output 1 delayed Idle on external Event 8

LL_HRTIM_DLYPRT_DELAYOUT2_EEV8

Timers D, E: Output 2 delayed Idle on external Event 8

LL_HRTIM_DLYPRT_DELAYBOTH_EEV8

Timers D, E: Output 1 and output 2 delayed Idle on external Event 8

LL_HRTIM_DLYPRT_BALANCED_EEV8

Timers D, E: Balanced Idle on external Event 8

LL_HRTIM_DLYPRT_DELAYOUT1_EEV9

Timers D, E: Output 1 delayed Idle on external Event 9

LL_HRTIM_DLYPRT_DELAYOUT2_EEV9

Timers D, E: Output 2 delayed Idle on external Event 9

LL_HRTIM_DLYPRT_DELAYBOTH_EEV9

Timers D, E: Output 1 and output2 delayed Idle on external Event 9

LL_HRTIM_DLYPRT_BALANCED_EEV9

Timers D, E: Balanced Idle on external Event 9

DEADTIME FALLING SIGN

LL_HRTIM_DT_FALLING_POSITIVE

Positive deadtime on falling edge

LL_HRTIM_DT_FALLING_NEGATIVE

Negative deadtime on falling edge

DEADTIME PRESCALER

LL_HRTIM_DT_PRESCALER_MUL8

fDTG = fHRTIM * 8

LL_HRTIM_DT_PRESCALER_MUL4

fDTG = fHRTIM * 4

LL_HRTIM_DT_PRESCALER_MUL2

fDTG = fHRTIM * 2

LL_HRTIM_DT_PRESCALER_DIV1

fDTG = fHRTIM

LL_HRTIM_DT_PRESCALER_DIV2

fDTG = fHRTIM / 2

LL_HRTIM_DT_PRESCALER_DIV4

fDTG = fHRTIM / 4

LL_HRTIM_DT_PRESCALER_DIV8

fDTG = fHRTIM / 8

LL_HRTIM_DT_PRESCALER_DIV16

fDTG = fHRTIM / 16

DEADTIME RISING SIGN

LL_HRTIM_DT_RISING_POSITIVE

Positive deadtime on rising edge

LL_HRTIM_DT_RISING_NEGATIVE

Negative deadtime on rising edge

EXTERNAL EVENT FAST MODE

LL_HRTIM_EE_FASTMODE_DISABLE

External Event is re-synchronized by the HRTIM logic before acting on outputs

LL_HRTIM_EE_FASTMODE_ENABLE

External Event is acting asynchronously on outputs (low latency mode)

EXTERNAL EVENT DIGITAL FILTER

LL_HRTIM_EE_FILTER_NONE

Filter disabled

LL_HRTIM_EE_FILTER_1

fSAMPLING = fHRTIM, N=2

LL_HRTIM_EE_FILTER_2

fSAMPLING = fHRTIM, N=4

LL_HRTIM_EE_FILTER_3

fSAMPLING = fHRTIM, N=8

LL_HRTIM_EE_FILTER_4

fSAMPLING = fEEVS/2, N=6

LL_HRTIM_EE_FILTER_5

fSAMPLING = fEEVS/2, N=8

LL_HRTIM_EE_FILTER_6

fSAMPLING = fEEVS/4, N=6

LL_HRTIM_EE_FILTER_7

fSAMPLING = fEEVS/4, N=8

LL_HRTIM_EE_FILTER_8

fSAMPLING = fEEVS/8, N=6

LL_HRTIM_EE_FILTER_9

fSAMPLING = fEEVS/8, N=8

LL_HRTIM_EE_FILTER_10

fSAMPLING = fEEVS/16, N=5

LL_HRTIM_EE_FILTER_11

fSAMPLING = fEEVS/16, N=6

LL_HRTIM_EE_FILTER_12

fSAMPLING = fEEVS/16, N=8

LL_HRTIM_EE_FILTER_13

fSAMPLING = fEEVS/32, N=5

LL_HRTIM_EE_FILTER_14

fSAMPLING = fEEVS/32, N=6

LL_HRTIM_EE_FILTER_15

fSAMPLING = fEEVS/32, N=8

EXTERNAL EVENT POLARITY

LL_HRTIM_EE_POLARITY_HIGH

External event is active high

LL_HRTIM_EE_POLARITY_LOW

External event is active low

EXTERNAL EVENT PRESCALER

LL_HRTIM_EE_PRESCALER_DIV1

fEEVS = fHRTIM

LL_HRTIM_EE_PRESCALER_DIV2

fEEVS = fHRTIM / 2

LL_HRTIM_EE_PRESCALER_DIV4

fEEVS = fHRTIM / 4

LL_HRTIM_EE_PRESCALER_DIV8

fEEVS = fHRTIM / 8

EXTERNAL EVENT SENSITIVITY

LL_HRTIM_EE_SENSITIVITY_LEVEL

External event is active on level

LL_HRTIM_EE_SENSITIVITY_RISINGEDGE

External event is active on Rising edge

LL_HRTIM_EE_SENSITIVITY_FALLINGEDGE

External event is active on Falling edge

LL_HRTIM_EE_SENSITIVITY_BOTHEDGES

External event is active on Rising and Falling edges

EXTERNAL EVENT SOURCE

LL_HRTIM_EE_SRC_1

External event source 1 (EExSrc1)

LL_HRTIM_EE_SRC_2

External event source 2 (EExSrc2)

LL_HRTIM_EE_SRC_3

External event source 3 (EExSrc3)

LL_HRTIM_EE_SRC_4

External event source 4 (EExSrc4)

EXTERNAL EVENT ID

LL_HRTIM_EVENT_1

External event channel 1 identifier

LL_HRTIM_EVENT_2

External event channel 2 identifier

LL_HRTIM_EVENT_3

External event channel 3 identifier

LL_HRTIM_EVENT_4

External event channel 4 identifier

LL_HRTIM_EVENT_5

External event channel 5 identifier

LL_HRTIM_EVENT_6

External event channel 6 identifier

LL_HRTIM_EVENT_7

External event channel 7 identifier

LL_HRTIM_EVENT_8

External event channel 8 identifier

LL_HRTIM_EVENT_9

External event channel 9 identifier

LL_HRTIM_EVENT_10

External event channel 10 identifier

FAULT ID

LL_HRTIM_FAULT_1

Fault channel 1 identifier

LL_HRTIM_FAULT_2

Fault channel 2 identifier

LL_HRTIM_FAULT_3

Fault channel 3 identifier

LL_HRTIM_FAULT_4

Fault channel 4 identifier

LL_HRTIM_FAULT_5

Fault channel 5 identifier

FAULT DIGITAL FILTER

LL_HRTIM_FLT_FILTER_NONE

Filter disabled

LL_HRTIM_FLT_FILTER_1

fSAMPLING= fHRTIM, N=2

LL_HRTIM_FLT_FILTER_2

fSAMPLING= fHRTIM, N=4

LL_HRTIM_FLT_FILTER_3

fSAMPLING= fHRTIM, N=8

LL_HRTIM_FLT_FILTER_4

fSAMPLING= fFLTS/2, N=6

LL_HRTIM_FLT_FILTER_5

fSAMPLING= fFLTS/2, N=8

LL_HRTIM_FLT_FILTER_6

fSAMPLING= fFLTS/4, N=6

LL_HRTIM_FLT_FILTER_7

fSAMPLING= fFLTS/4, N=8

LL_HRTIM_FLT_FILTER_8

fSAMPLING= fFLTS/8, N=6

LL_HRTIM_FLT_FILTER_9

fSAMPLING= fFLTS/8, N=8

LL_HRTIM_FLT_FILTER_10

fSAMPLING= fFLTS/16, N=5

LL_HRTIM_FLT_FILTER_11

fSAMPLING= fFLTS/16, N=6

LL_HRTIM_FLT_FILTER_12

fSAMPLING= fFLTS/16, N=8

LL_HRTIM_FLT_FILTER_13

fSAMPLING= fFLTS/32, N=5

LL_HRTIM_FLT_FILTER_14

fSAMPLING= fFLTS/32, N=6

LL_HRTIM_FLT_FILTER_15

fSAMPLING= fFLTS/32, N=8

FAULT POLARITY

LL_HRTIM_FLT_POLARITY_LOW

Fault input is active low

LL_HRTIM_FLT_POLARITY_HIGH

Fault input is active high

BURST FAULT PRESCALER

LL_HRTIM_FLT_PRESCALER_DIV1

fFLTS = fHRTIM

LL_HRTIM_FLT_PRESCALER_DIV2

fFLTS = fHRTIM / 2

LL_HRTIM_FLT_PRESCALER_DIV4

fFLTS = fHRTIM / 4

LL_HRTIM_FLT_PRESCALER_DIV8

fFLTS = fHRTIM / 8

FAULT SOURCE

LL_HRTIM_FLT_SRC_DIGITALINPUT

Fault input is FLT input pin

LL_HRTIM_FLT_SRC_INTERNAL

Fault input is FLT_Int signal (e.g. internal comparator)

Get Flags Defines

LL_HRTIM_ISR_FLT1

LL_HRTIM_ISR_FLT2

LL_HRTIM_ISR_FLT3

LL_HRTIM_ISR_FLT4

LL_HRTIM_ISR_FLT5

LL_HRTIM_ISR_SYSFLT

LL_HRTIM_ISR_BMPER

LL_HRTIM_MISR_MCMP1

LL_HRTIM_MISR_MCMP2

LL_HRTIM_MISR_MCMP3

LL_HRTIM_MISR_MCMP4

LL_HRTIM_MISR_MREP

LL_HRTIM_MISR_SYNC

LL_HRTIM_MISR_MUPD

LL_HRTIM_TIMISR_CMP1

LL_HRTIM_TIMISR_CMP2

LL_HRTIM_TIMISR_CMP3

LL_HRTIM_TIMISR_CMP4

LL_HRTIM_TIMISR_REP

LL_HRTIM_TIMISR_UPD

LL_HRTIM_TIMISR_CPT1

LL_HRTIM_TIMISR_CPT2

LL_HRTIM_TIMISR_SET1

LL_HRTIM_TIMISR_RST1

LL_HRTIM_TIMISR_SET2

LL_HRTIM_TIMISR_RST2

LL_HRTIM_TIMISR_RST

LL_HRTIM_TIMISR_DLYPRT

HALF MODE

LL_HRTIM_HALF_MODE_DISABLED

HRTIM Half Mode is disabled

LL_HRTIM_HALF_MODE_ENABLE

HRTIM Half Mode is Half

IDLE PUSH-PULL STATUS

LL_HRTIM_IPPSTAT_OUTPUT1

Protection occurred when the output 1 was active and output 2 forced inactive

LL_HRTIM_IPPSTAT_OUTPUT2

Protection occurred when the output 2 was active and output 1 forced inactive

IT Defines

LL_HRTIM_IER_FLT1IE

LL_HRTIM_IER_FLT2IE

LL_HRTIM_IER_FLT3IE

LL_HRTIM_IER_FLT4IE

LL_HRTIM_IER_FLT5IE

LL_HRTIM_IER_SYSFLTIE

LL_HRTIM_IER_BMPERIE

LL_HRTIM_MDIER_MCMP1IE

LL_HRTIM_MDIER_MCMP2IE

LL_HRTIM_MDIER_MCMP3IE

LL_HRTIM_MDIER_MCMP4IE

LL_HRTIM_MDIER_MREPIE

LL_HRTIM_MDIER_SYNCIE

LL_HRTIM_MDIER_MUPDIE

LL_HRTIM_TIMDIER_CMP1IE

LL_HRTIM_TIMDIER_CMP2IE

LL_HRTIM_TIMDIER_CMP3IE

LL_HRTIM_TIMDIER_CMP4IE

LL_HRTIM_TIMDIER_REPIE

LL_HRTIM_TIMDIER_UPDIE

LL_HRTIM_TIMDIER_CPT1IE

LL_HRTIM_TIMDIER_CPT2IE

LL_HRTIM_TIMDIER_SET1IE

LL_HRTIM_TIMDIER_RST1IE

LL_HRTIM_TIMDIER_SET2IE

LL_HRTIM_TIMDIER_RST2IE

LL_HRTIM_TIMDIER_RSTIE

LL_HRTIM_TIMDIER_DLYPRTIE

COUNTER MODE

LL_HRTIM_MODE_CONTINUOUS

The timer operates in continuous (free-running) mode

LL_HRTIM_MODE_SINGLESHOT

The timer operates in non retriggerable single-shot mode

LL_HRTIM_MODE_RETRIGGERABLE

The timer operates in retriggerable single-shot mode

OUTPUT ID

LL_HRTIM_OUTPUT_TA1

Timer A - Output 1 identifier

LL_HRTIM_OUTPUT_TA2

Timer A - Output 2 identifier

LL_HRTIM_OUTPUT_TB1

Timer B - Output 1 identifier

LL_HRTIM_OUTPUT_TB2

Timer B - Output 2 identifier

LL_HRTIM_OUTPUT_TC1

Timer C - Output 1 identifier

LL_HRTIM_OUTPUT_TC2

Timer C - Output 2 identifier

LL_HRTIM_OUTPUT_TD1

Timer D - Output 1 identifier

LL_HRTIM_OUTPUT_TD2

Timer D - Output 2 identifier

LL_HRTIM_OUTPUT_TE1

Timer E - Output 1 identifier

LL_HRTIM_OUTPUT_TE2

Timer E - Output 2 identifier

OUTPUT STATE

LL_HRTIM_OUTPUTSTATE_IDLE

Main operating mode, where the output can take the active or inactive level as programmed in the crossbar unit

LL_HRTIM_OUTPUTSTATE_RUN

Default operating state (e.g. after an HRTIM reset, when the outputs are disabled by software or during a burst mode operation)

LL_HRTIM_OUTPUTSTATE_FAULT

Safety state, entered in case of a shut-down request on FAULTx inputs

OUTPUT BURST MODE ENTRY MODE

LL_HRTIM_OUT_BM_ENTRYMODE_REGULAR

The programmed Idle state is applied immediately to the Output

LL_HRTIM_OUT_BM_ENTRYMODE_DELAYED

Deadtime is inserted on output before entering the idle mode

OUTPUT CHOPPER MODE

LL_HRTIM_OUT_CHOPPERMODE_DISABLED

Output signal is not altered

LL_HRTIM_OUT_CHOPPERMODE_ENABLED

Output signal is chopped by a carrier signal

OUTPUT FAULT STATE

LL_HRTIM_OUT_FAULTSTATE_NO_ACTION

The output is not affected by the fault input

LL_HRTIM_OUT_FAULTSTATE_ACTIVE

Output at active level when in FAULT state

LL_HRTIM_OUT_FAULTSTATE_INACTIVE

Output at inactive level when in FAULT state

LL_HRTIM_OUT_FAULTSTATE_HIGHZ

Output is tri-stated when in FAULT state

OUTPUT IDLE LEVEL

LL_HRTIM_OUT_IDLELEVEL_INACTIVE

Output at inactive level when in IDLE state

LL_HRTIM_OUT_IDLELEVEL_ACTIVE

Output at active level when in IDLE state

OUTPUT IDLE MODE

LL_HRTIM_OUT_NO_IDLE

The output is not affected by the burst mode operation

LL_HRTIM_OUT_IDLE_WHEN_BURST

The output is in idle state when requested by the burst mode controller

OUTPUT LEVEL

LL_HRTIM_OUT_LEVEL_INACTIVE

Corresponds to a logic level 0 for a positive polarity (High) and to a logic level 1 for a negative polarity (Low)

LL_HRTIM_OUT_LEVEL_ACTIVE

Corresponds to a logic level 1 for a positive polarity (High) and to a logic level 0 for a negative polarity (Low)

OUPUT_POLARITY

LL_HRTIM_OUT_POSITIVE_POLARITY

Output is active HIGH

LL_HRTIM_OUT_NEGATIVE_POLARITY

Output is active LOW

PRESCALER RATIO

LL_HRTIM_PRESCALERRATIO_DIV1

fHRCK: fHRTIM = 144 MHz - Resolution: 6.95 ns - Min PWM frequency: 2.2 kHz (fHRTIM=144MHz)

LL_HRTIM_PRESCALERRATIO_DIV2

fHRCK: fHRTIM / 2 = 72 MHz - Resolution: 13.88 ns- Min PWM frequency: 1.1 kHz (fHRTIM=144MHz)

LL_HRTIM_PRESCALERRATIO_DIV4

fHRCK: fHRTIM / 4 = 36 MHz - Resolution: 27.7 ns- Min PWM frequency: 550Hz (fHRTIM=144MHz)

RESET TRIGGER

LL_HRTIM_RESETTRIG_NONE

No counter reset trigger

LL_HRTIM_RESETTRIG_UPDATE

The timer counter is reset upon update event

LL_HRTIM_RESETTRIG_CMP2

The timer counter is reset upon Timer Compare 2 event

LL_HRTIM_RESETTRIG_CMP4

The timer counter is reset upon Timer Compare 4 event

LL_HRTIM_RESETTRIG_MASTER_PER

The timer counter is reset upon master timer period event

LL_HRTIM_RESETTRIG_MASTER_CMP1

The timer counter is reset upon master timer Compare 1 event

LL_HRTIM_RESETTRIG_MASTER_CMP2

The timer counter is reset upon master timer Compare 2 event

LL_HRTIM_RESETTRIG_MASTER_CMP3

The timer counter is reset upon master timer Compare 3 event

LL_HRTIM_RESETTRIG_MASTER_CMP4

The timer counter is reset upon master timer Compare 4 event

LL_HRTIM_RESETTRIG_EEV_1

The timer counter is reset upon external event 1

LL_HRTIM_RESETTRIG_EEV_2

The timer counter is reset upon external event 2

LL_HRTIM_RESETTRIG_EEV_3

The timer counter is reset upon external event 3

LL_HRTIM_RESETTRIG_EEV_4

The timer counter is reset upon external event 4

LL_HRTIM_RESETTRIG_EEV_5

The timer counter is reset upon external event 5

LL_HRTIM_RESETTRIG_EEV_6

The timer counter is reset upon external event 6

LL_HRTIM_RESETTRIG_EEV_7

The timer counter is reset upon external event 7

LL_HRTIM_RESETTRIG_EEV_8

The timer counter is reset upon external event 8

LL_HRTIM_RESETTRIG_EEV_9

The timer counter is reset upon external event 9

LL_HRTIM_RESETTRIG_EEV_10

The timer counter is reset upon external event 10

LL_HRTIM_RESETTRIG_OTHER1_CMP1

The timer counter is reset upon other timer Compare 1 event

LL_HRTIM_RESETTRIG_OTHER1_CMP2

The timer counter is reset upon other timer Compare 2 event

LL_HRTIM_RESETTRIG_OTHER1_CMP4

The timer counter is reset upon other timer Compare 4 event

LL_HRTIM_RESETTRIG_OTHER2_CMP1

The timer counter is reset upon other timer Compare 1 event

LL_HRTIM_RESETTRIG_OTHER2_CMP2

The timer counter is reset upon other timer Compare 2 event

LL_HRTIM_RESETTRIG_OTHER2_CMP4

The timer counter is reset upon other timer Compare 4 event

LL_HRTIM_RESETTRIG_OTHER3_CMP1

The timer counter is reset upon other timer Compare 1 event

LL_HRTIM_RESETTRIG_OTHER3_CMP2

The timer counter is reset upon other timer Compare 2 event

LL_HRTIM_RESETTRIG_OTHER3_CMP4

The timer counter is reset upon other timer Compare 4 event

LL_HRTIM_RESETTRIG_OTHER4_CMP1

The timer counter is reset upon other timer Compare 1 event

LL_HRTIM_RESETTRIG_OTHER4_CMP2

The timer counter is reset upon other timer Compare 2 event

LL_HRTIM_RESETTRIG_OTHER4_CMP4

The timer counter is reset upon other timer Compare 4 event

SYNCHRONIZATION INPUT SOURCE

LL_HRTIM_SYNCIN_SRC_NONE

HRTIM is not synchronized and runs in standalone mode

LL_HRTIM_SYNCIN_SRC_TIM_EVENT

The HRTIM is synchronized with the on-chip timer

LL_HRTIM_SYNCIN_SRC_EXTERNAL_EVENT

A positive pulse on SYNCIN input triggers the HRTIM

SYNCHRONIZATION OUTPUT POLARITY

LL_HRTIM_SYNCOUT_DISABLED

Synchronization output event is disabled

LL_HRTIM_SYNCOUT_POSITIVE_PULSE

SCOUT pin has a low idle level and issues a positive pulse of 16 fHRTIM clock cycles length for the synchronization

LL_HRTIM_SYNCOUT_NEGATIVE_PULSE

SCOUT pin has a high idle level and issues a negative pulse of 16 fHRTIM clock cycles length for the synchronization

SYNCHRONIZATION OUTPUT SOURCE

LL_HRTIM_SYNCOUT_SRC_MASTER_START

A pulse is sent on HRTIM_SCOUT output and hrtim_out_sync2 upon master timer start event

LL_HRTIM_SYNCOUT_SRC_MASTER_CMP1

A pulse is sent on HRTIM_SCOUT output and hrtim_out_sync2 upon master timer compare 1 event

LL_HRTIM_SYNCOUT_SRC_TIMA_START

A pulse is sent on HRTIM_SCOUT output and hrtim_out_sync2 upon timer A start or reset events

LL_HRTIM_SYNCOUT_SRC_TIMA_CMP1

A pulse is sent on HRTIM_SCOUT output and hrtim_out_sync2 upon timer A compare 1 event

TIMER ID

LL_HRTIM_TIMER_NONE

Master timer identifier

LL_HRTIM_TIMER_MASTER

Master timer identifier

LL_HRTIM_TIMER_A

Timer A identifier

LL_HRTIM_TIMER_B

Timer B identifier

LL_HRTIM_TIMER_C

Timer C identifier

LL_HRTIM_TIMER_D

Timer D identifier

LL_HRTIM_TIMER_E

Timer E identifier

LL_HRTIM_TIMER_X

LL_HRTIM_TIMER_ALL

TIMER EXTERNAL EVENT FILTER

LL_HRTIM_EEFLTR_NONE

LL_HRTIM_EEFLTR_BLANKINGCMP1

Blanking from counter reset/roll-over to Compare 1

LL_HRTIM_EEFLTR_BLANKINGCMP2

Blanking from counter reset/roll-over to Compare 2

LL_HRTIM_EEFLTR_BLANKINGCMP3

Blanking from counter reset/roll-over to Compare 3

LL_HRTIM_EEFLTR_BLANKINGCMP4

Blanking from counter reset/roll-over to Compare 4

LL_HRTIM_EEFLTR_BLANKINGFLTR1

Blanking from another timing unit: TIMFLTR1 source

LL_HRTIM_EEFLTR_BLANKINGFLTR2

Blanking from another timing unit: TIMFLTR2 source

LL_HRTIM_EEFLTR_BLANKINGFLTR3

Blanking from another timing unit: TIMFLTR3 source

LL_HRTIM_EEFLTR_BLANKINGFLTR4

Blanking from another timing unit: TIMFLTR4 source

LL_HRTIM_EEFLTR_BLANKINGFLTR5

Blanking from another timing unit: TIMFLTR5 source

LL_HRTIM_EEFLTR_BLANKINGFLTR6

Blanking from another timing unit: TIMFLTR6 source

LL_HRTIM_EEFLTR_BLANKINGFLTR7

Blanking from another timing unit: TIMFLTR7 source

LL_HRTIM_EEFLTR_BLANKINGFLTR8

Blanking from another timing unit: TIMFLTR8 source

LL_HRTIM_EEFLTR_WINDOWINGCMP2

Windowing from counter reset/roll-over to Compare 2

LL_HRTIM_EEFLTR_WINDOWINGCMP3

Windowing from counter reset/roll-over to Compare 3

LL_HRTIM_EEFLTR_WINDOWINGTIM

Windowing from another timing unit: TIMWIN source

TIMER EXTERNAL EVENT LATCH STATUS

LL_HRTIM_EELATCH_DISABLED

Event is ignored if it happens during a blank, or passed through during a window

LL_HRTIM_EELATCH_ENABLED

Event is latched and delayed till the end of the blanking or windowing period

UPDATE GATING

LL_HRTIM_UPDATEGATING_INDEPENDENT

Update done independently from the DMA burst transfer completion

LL_HRTIM_UPDATEGATING_DMABURST

Update done when the DMA burst transfer is completed

LL_HRTIM_UPDATEGATING_DMABURST_UPDATE

Update done on timer roll-over following a DMA burst transfer completion

LL_HRTIM_UPDATEGATING_UPDEN1

Slave timer only - Update done on a rising edge of HRTIM update enable input 1

LL_HRTIM_UPDATEGATING_UPDEN2

Slave timer only - Update done on a rising edge of HRTIM update enable input 2

LL_HRTIM_UPDATEGATING_UPDEN3

Slave timer only - Update done on a rising edge of HRTIM update enable input 3

LL_HRTIM_UPDATEGATING_UPDEN1_UPDATE

Slave timer only - Update done on the update event following a rising edge of HRTIM update enable input 1

LL_HRTIM_UPDATEGATING_UPDEN2_UPDATE

Slave timer only - Update done on the update event following a rising edge of HRTIM update enable input 2

LL_HRTIM_UPDATEGATING_UPDEN3_UPDATE

Slave timer only - Update done on the update event following a rising edge of HRTIM update enable input 3

UPDATE TRIGGER

LL_HRTIM_UPDATETRIG_NONE

Register update is disabled

LL_HRTIM_UPDATETRIG_MASTER

Register update is triggered by the master timer update

LL_HRTIM_UPDATETRIG_TIMER_A

Register update is triggered by the timer A update

LL_HRTIM_UPDATETRIG_TIMER_B

Register update is triggered by the timer B update

LL_HRTIM_UPDATETRIG_TIMER_C

Register update is triggered by the timer C update

LL_HRTIM_UPDATETRIG_TIMER_D

Register update is triggered by the timer D update

LL_HRTIM_UPDATETRIG_TIMER_E

Register update is triggered by the timer E update

LL_HRTIM_UPDATETRIG_REPETITION

Register update is triggered when the counter rolls over and HRTIM_REPx = 0

LL_HRTIM_UPDATETRIG_RESET

Register update is triggered by counter reset or roll-over to 0 after reaching the period value in continuous mode

Exported_Macros

__LL_HRTIM_GET_OUTPUT_STATE

Description:

- HELPER macro returning the output state from output enable/disable status.

Parameters:

- __OUTPUT_STATUS_EN__: output enable status
- __OUTPUT_STATUS_DIS__: output Disable status

Return value:

- Returned: value can be one of the following values:
  - LL_HRTIM_OUTPUTSTATE_IDLE
  - LL_HRTIM_OUTPUTSTATE_RUN
  - LL_HRTIM_OUTPUTSTATE_FAULT

Common Write and read registers Macros

LL_HRTIM_WriteReg

Description:

- Write a value in HRTIM register.

Parameters:

- __INSTANCE__: HRTIM Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_HRTIM_ReadReg

Description:

- Read a value in HRTIM register.

Parameters:

- __INSTANCE__: HRTIM Instance
- __REG__: Register to be read

Return value:

- Register: value

112 LL HSEM Generic Driver

112.1 HSEM Firmware driver API description

The following section lists the various functions of the HSEM library.

112.1.1 Detailed description of functions

LL_HSEM_IsSemaphoreLocked

Function name

__STATIC_INLINE uint32_t LL_HSEM_IsSemaphoreLocked (HSEM_TypeDef * HSEMx, uint32_t Semaphore)

Function description

Return 1 if the semaphore is locked, else return 0.

Parameters

- **HSEMx:** HSEM Instance.
- **Semaphore:** Semaphore number. Value between Min_Data=0 and Max_Data=31

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- R LOCK LL_HSEM_IsSemaphoreLocked

LL_HSEM_GetCoreId

Function name

__STATIC_INLINE uint32_t LL_HSEM_GetCoreId (HSEM_TypeDef * HSEMx, uint32_t Semaphore)

Function description

Get core id.

Parameters

- **HSEMx:** HSEM Instance.
- **Semaphore:** Semaphore number. Value between Min_Data=0 and Max_Data=31

Return values

- **Returned:** value can be one of the following values:
  - LL_HSEM_COREID_NONE
  - LL_HSEM_COREID_CPU1
  - LL_HSEM_COREID_CPU2

Reference Manual to LL API cross reference:

- R COREID LL_HSEM_GetCoreId

LL_HSEM_GetProcessId

Function name

__STATIC_INLINE uint32_t LL_HSEM_GetProcessId (HSEM_TypeDef * HSEMx, uint32_t Semaphore)

Function description

Get process id.

Parameters

- **HSEMx:** HSEM Instance.
- **Semaphore:** Semaphore number. Value between Min_Data=0 and Max_Data=31

Return values

- **Process:** number. Value between Min_Data=0 and Max_Data=255

Reference Manual to LL API cross reference:

- R PROCID LL_HSEM_GetProcessId

LL_HSEM_SetLock

Function name

__STATIC_INLINE void LL_HSEM_SetLock (HSEM_TypeDef * HSEMx, uint32_t Semaphore, uint32_t process)

Function description

Get the lock by writing in R register.

Parameters

- **HSEMx:** HSEM Instance.
- **Semaphore:** Semaphore number. Value between Min_Data=0 and Max_Data=31
- **process:** Process id. Value between Min_Data=0 and Max_Data=255

Return values

- **None:**

Notes

- The R register has to be read to determined if the lock is taken.

Reference Manual to LL API cross reference:

- R LOCK LL_HSEM_SetLock
- R COREID LL_HSEM_SetLock
- R PROCID LL_HSEM_SetLock

LL_HSEM_2StepLock

Function name

__STATIC_INLINE uint32_t LL_HSEM_2StepLock (HSEM_TypeDef * HSEMx, uint32_t Semaphore, uint32_t process)

Function description

Get the lock with 2-step lock.

Parameters

- **HSEMx:** HSEM Instance.
- **Semaphore:** Semaphore number. Value between Min_Data=0 and Max_Data=31
- **process:** Process id. Value between Min_Data=0 and Max_Data=255

Return values

- **1:** lock fail, 0 lock successful or already locked by same process and core

Reference Manual to LL API cross reference:

- R LOCK LL_HSEM_2StepLock
- R COREID LL_HSEM_2StepLock
- R PROCID LL_HSEM_2StepLock

LL_HSEM_1StepLock

Function name

__STATIC_INLINE uint32_t LL_HSEM_1StepLock (HSEM_TypeDef * HSEMx, uint32_t Semaphore)

Function description

Get the lock with 1-step lock.

Parameters

- **HSEMx:** HSEM Instance.
- **Semaphore:** Semaphore number. Value between Min_Data=0 and Max_Data=31

Return values

- **1:** lock fail, 0 lock successful or already locked by same core

Reference Manual to LL API cross reference:

- RLR LOCK LL_HSEM_1StepLock
- RLR COREID LL_HSEM_1StepLock
- RLR PROCID LL_HSEM_1StepLock

LL_HSEM_ReleaseLock

Function name

__STATIC_INLINE void LL_HSEM_ReleaseLock (HSEM_TypeDef * HSEMx, uint32_t Semaphore, uint32_t process)

Function description

Release the lock of the semaphore.

Parameters

- **HSEMx:** HSEM Instance.
- **Semaphore:** Semaphore number. Value between Min_Data=0 and Max_Data=31
- **process:** Process number. Value between Min_Data=0 and Max_Data=255

Return values

- **None:**

Notes

- In case of LL_HSEM_1StepLock usage to lock a semaphore, the process is 0.

Reference Manual to LL API cross reference:

- R LOCK LL_HSEM_ReleaseLock

LL_HSEM_GetStatus

Function name

__STATIC_INLINE uint32_t LL_HSEM_GetStatus (HSEM_TypeDef * HSEMx, uint32_t Semaphore)

Function description

Get the lock status of the semaphore.

Parameters

- **HSEMx:** HSEM Instance.
- **Semaphore:** Semaphore number. Value between Min_Data=0 and Max_Data=31

Return values

- **0:** semaphore is free, 1 semaphore is locked

Reference Manual to LL API cross reference:

- R LOCK LL_HSEM_GetStatus

LL_HSEM_SetKey

Function name

__STATIC_INLINE void LL_HSEM_SetKey (HSEM_TypeDef * HSEMx, uint32_t key)

Function description

Set the key.

Parameters

- **HSEMx:** HSEM Instance.
- **key:** Key value.

Return values

- **None:**

Reference Manual to LL API cross reference:

- KEYR KEY LL_HSEM_SetKey

LL_HSEM_GetKey

Function name

__STATIC_INLINE uint32_t LL_HSEM_GetKey (HSEM_TypeDef * HSEMx)

Function description

Get the key.

Parameters

- **HSEMx:** HSEM Instance.

Return values

- **key:** to unlock all semaphore from the same core

Reference Manual to LL API cross reference:

- KEYR KEY LL_HSEM_GetKey

LL_HSEM_ResetAllLock

Function name

__STATIC_INLINE void LL_HSEM_ResetAllLock (HSEM_TypeDef * HSEMx, uint32_t key, uint32_t core)

Function description

Release all semaphore with the same core id.

Parameters

- **HSEMx:** HSEM Instance.
- **key:** Key value.
- **core:** This parameter can be one of the following values:
  - LL_HSEM_COREID_CPU1
  - LL_HSEM_COREID_CPU2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR KEY LL_HSEM_ResetAllLock
- CR SEC LL_HSEM_ResetAllLock
- CR PRIV LL_HSEM_ResetAllLock

LL_HSEM_EnableIT_C1IER

Function name

__STATIC_INLINE void LL_HSEM_EnableIT_C1IER (HSEM_TypeDef * HSEMx, uint32_t SemaphoreMask)

Function description

Enable interrupt.

Parameters

- **HSEMx:** HSEM Instance.
- **SemaphoreMask:** This parameter can be a combination of the following values:
  - LL_HSEM_SEMAPHORE_0
  - LL_HSEM_SEMAPHORE_1
  - LL_HSEM_SEMAPHORE_2
  - LL_HSEM_SEMAPHORE_3
  - LL_HSEM_SEMAPHORE_4
  - LL_HSEM_SEMAPHORE_5
  - LL_HSEM_SEMAPHORE_6
  - LL_HSEM_SEMAPHORE_7
  - LL_HSEM_SEMAPHORE_8
  - LL_HSEM_SEMAPHORE_9
  - LL_HSEM_SEMAPHORE_10
  - LL_HSEM_SEMAPHORE_11
  - LL_HSEM_SEMAPHORE_12
  - LL_HSEM_SEMAPHORE_13
  - LL_HSEM_SEMAPHORE_14
  - LL_HSEM_SEMAPHORE_15
  - LL_HSEM_SEMAPHORE_16
  - LL_HSEM_SEMAPHORE_17
  - LL_HSEM_SEMAPHORE_18
  - LL_HSEM_SEMAPHORE_19
  - LL_HSEM_SEMAPHORE_20
  - LL_HSEM_SEMAPHORE_21
  - LL_HSEM_SEMAPHORE_22
  - LL_HSEM_SEMAPHORE_23
  - LL_HSEM_SEMAPHORE_24
  - LL_HSEM_SEMAPHORE_25
  - LL_HSEM_SEMAPHORE_26
  - LL_HSEM_SEMAPHORE_27
  - LL_HSEM_SEMAPHORE_28
  - LL_HSEM_SEMAPHORE_29
  - LL_HSEM_SEMAPHORE_30
  - LL_HSEM_SEMAPHORE_31
  - LL_HSEM_SEMAPHORE_ALL

Return values

- **None:**

Notes

- Availability of flags LL_HSEM_SEMAPHORE_16 to LL_HSEM_SEMAPHORE_31 depends on devices.

Reference Manual to LL API cross reference:

- C1IER ISEM LL_HSEM_EnableIT_C1IER

LL_HSEM_DisableIT_C1IER

Function name

__STATIC_INLINE void LL_HSEM_DisableIT_C1IER (HSEM_TypeDef * HSEMx, uint32_t SemaphoreMask)

Function description

Disable interrupt.

Parameters

- **HSEMx:** HSEM Instance.
- **SemaphoreMask:** This parameter can be a combination of the following values:
  - LL_HSEM_SEMAPHORE_0
  - LL_HSEM_SEMAPHORE_1
  - LL_HSEM_SEMAPHORE_2
  - LL_HSEM_SEMAPHORE_3
  - LL_HSEM_SEMAPHORE_4
  - LL_HSEM_SEMAPHORE_5
  - LL_HSEM_SEMAPHORE_6
  - LL_HSEM_SEMAPHORE_7
  - LL_HSEM_SEMAPHORE_8
  - LL_HSEM_SEMAPHORE_9
  - LL_HSEM_SEMAPHORE_10
  - LL_HSEM_SEMAPHORE_11
  - LL_HSEM_SEMAPHORE_12
  - LL_HSEM_SEMAPHORE_13
  - LL_HSEM_SEMAPHORE_14
  - LL_HSEM_SEMAPHORE_15
  - LL_HSEM_SEMAPHORE_16
  - LL_HSEM_SEMAPHORE_17
  - LL_HSEM_SEMAPHORE_18
  - LL_HSEM_SEMAPHORE_19
  - LL_HSEM_SEMAPHORE_20
  - LL_HSEM_SEMAPHORE_21
  - LL_HSEM_SEMAPHORE_22
  - LL_HSEM_SEMAPHORE_23
  - LL_HSEM_SEMAPHORE_24
  - LL_HSEM_SEMAPHORE_25
  - LL_HSEM_SEMAPHORE_26
  - LL_HSEM_SEMAPHORE_27
  - LL_HSEM_SEMAPHORE_28
  - LL_HSEM_SEMAPHORE_29
  - LL_HSEM_SEMAPHORE_30
  - LL_HSEM_SEMAPHORE_31
  - LL_HSEM_SEMAPHORE_ALL

Return values

- **None:**

Notes

- Availability of flags LL_HSEM_SEMAPHORE_16 to LL_HSEM_SEMAPHORE_31 depends on devices.

Reference Manual to LL API cross reference:

- C1IER ISEM LL_HSEM_DisableIT_C1IER

LL_HSEM_IsEnabledIT_C1IER

Function name

__STATIC_INLINE uint32_t LL_HSEM_IsEnabledIT_C1IER (HSEM_TypeDef * HSEMx, uint32_t SemaphoreMask)

Function description

Check if interrupt is enabled.

Parameters

- **HSEMx:** HSEM Instance.
- **SemaphoreMask:** This parameter can be a combination of the following values:
  - LL_HSEM_SEMAPHORE_0
  - LL_HSEM_SEMAPHORE_1
  - LL_HSEM_SEMAPHORE_2
  - LL_HSEM_SEMAPHORE_3
  - LL_HSEM_SEMAPHORE_4
  - LL_HSEM_SEMAPHORE_5
  - LL_HSEM_SEMAPHORE_6
  - LL_HSEM_SEMAPHORE_7
  - LL_HSEM_SEMAPHORE_8
  - LL_HSEM_SEMAPHORE_9
  - LL_HSEM_SEMAPHORE_10
  - LL_HSEM_SEMAPHORE_11
  - LL_HSEM_SEMAPHORE_12
  - LL_HSEM_SEMAPHORE_13
  - LL_HSEM_SEMAPHORE_14
  - LL_HSEM_SEMAPHORE_15
  - LL_HSEM_SEMAPHORE_16
  - LL_HSEM_SEMAPHORE_17
  - LL_HSEM_SEMAPHORE_18
  - LL_HSEM_SEMAPHORE_19
  - LL_HSEM_SEMAPHORE_20
  - LL_HSEM_SEMAPHORE_21
  - LL_HSEM_SEMAPHORE_22
  - LL_HSEM_SEMAPHORE_23
  - LL_HSEM_SEMAPHORE_24
  - LL_HSEM_SEMAPHORE_25
  - LL_HSEM_SEMAPHORE_26
  - LL_HSEM_SEMAPHORE_27
  - LL_HSEM_SEMAPHORE_28
  - LL_HSEM_SEMAPHORE_29
  - LL_HSEM_SEMAPHORE_30
  - LL_HSEM_SEMAPHORE_31
  - LL_HSEM_SEMAPHORE_ALL

Return values

- **State:** of bit (1 or 0).

Notes

- Availability of flags LL_HSEM_SEMAPHORE_16 to LL_HSEM_SEMAPHORE_31 depends on devices.

Reference Manual to LL API cross reference:

- C1IER ISEM LL_HSEM_IsEnabledIT_C1IER

LL_HSEM_EnableIT_C2IER

Function name

__STATIC_INLINE void LL_HSEM_EnableIT_C2IER (HSEM_TypeDef * HSEMx, uint32_t SemaphoreMask)

Function description

Enable interrupt.

Parameters

- **HSEMx:** HSEM Instance.
- **SemaphoreMask:** This parameter can be a combination of the following values:
  - LL_HSEM_SEMAPHORE_0
  - LL_HSEM_SEMAPHORE_1
  - LL_HSEM_SEMAPHORE_2
  - LL_HSEM_SEMAPHORE_3
  - LL_HSEM_SEMAPHORE_4
  - LL_HSEM_SEMAPHORE_5
  - LL_HSEM_SEMAPHORE_6
  - LL_HSEM_SEMAPHORE_7
  - LL_HSEM_SEMAPHORE_8
  - LL_HSEM_SEMAPHORE_9
  - LL_HSEM_SEMAPHORE_10
  - LL_HSEM_SEMAPHORE_11
  - LL_HSEM_SEMAPHORE_12
  - LL_HSEM_SEMAPHORE_13
  - LL_HSEM_SEMAPHORE_14
  - LL_HSEM_SEMAPHORE_15
  - LL_HSEM_SEMAPHORE_16
  - LL_HSEM_SEMAPHORE_17
  - LL_HSEM_SEMAPHORE_18
  - LL_HSEM_SEMAPHORE_19
  - LL_HSEM_SEMAPHORE_20
  - LL_HSEM_SEMAPHORE_21
  - LL_HSEM_SEMAPHORE_22
  - LL_HSEM_SEMAPHORE_23
  - LL_HSEM_SEMAPHORE_24
  - LL_HSEM_SEMAPHORE_25
  - LL_HSEM_SEMAPHORE_26
  - LL_HSEM_SEMAPHORE_27
  - LL_HSEM_SEMAPHORE_28
  - LL_HSEM_SEMAPHORE_29
  - LL_HSEM_SEMAPHORE_30
  - LL_HSEM_SEMAPHORE_31
  - LL_HSEM_SEMAPHORE_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- C2IER ISEM LL_HSEM_EnableIT_C2IER

LL_HSEM_DisableIT_C2IER

Function name

__STATIC_INLINE void LL_HSEM_DisableIT_C2IER (HSEM_TypeDef * HSEMx, uint32_t SemaphoreMask)

Function description

Disable interrupt.

Parameters

- **HSEMx:** HSEM Instance.
- **SemaphoreMask:** This parameter can be a combination of the following values:
  - LL_HSEM_SEMAPHORE_0
  - LL_HSEM_SEMAPHORE_1
  - LL_HSEM_SEMAPHORE_2
  - LL_HSEM_SEMAPHORE_3
  - LL_HSEM_SEMAPHORE_4
  - LL_HSEM_SEMAPHORE_5
  - LL_HSEM_SEMAPHORE_6
  - LL_HSEM_SEMAPHORE_7
  - LL_HSEM_SEMAPHORE_8
  - LL_HSEM_SEMAPHORE_9
  - LL_HSEM_SEMAPHORE_10
  - LL_HSEM_SEMAPHORE_11
  - LL_HSEM_SEMAPHORE_12
  - LL_HSEM_SEMAPHORE_13
  - LL_HSEM_SEMAPHORE_14
  - LL_HSEM_SEMAPHORE_15
  - LL_HSEM_SEMAPHORE_16
  - LL_HSEM_SEMAPHORE_17
  - LL_HSEM_SEMAPHORE_18
  - LL_HSEM_SEMAPHORE_19
  - LL_HSEM_SEMAPHORE_20
  - LL_HSEM_SEMAPHORE_21
  - LL_HSEM_SEMAPHORE_22
  - LL_HSEM_SEMAPHORE_23
  - LL_HSEM_SEMAPHORE_24
  - LL_HSEM_SEMAPHORE_25
  - LL_HSEM_SEMAPHORE_26
  - LL_HSEM_SEMAPHORE_27
  - LL_HSEM_SEMAPHORE_28
  - LL_HSEM_SEMAPHORE_29
  - LL_HSEM_SEMAPHORE_30
  - LL_HSEM_SEMAPHORE_31
  - LL_HSEM_SEMAPHORE_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- C2IER ISEM LL_HSEM_DisableIT_C2IER

LL_HSEM_IsEnabledIT_C2IER

Function name

__STATIC_INLINE uint32_t LL_HSEM_IsEnabledIT_C2IER (HSEM_TypeDef * HSEMx, uint32_t SemaphoreMask)

Function description

Check if interrupt is enabled.

Parameters

- **HSEMx:** HSEM Instance.
- **SemaphoreMask:** This parameter can be a combination of the following values:
  - LL_HSEM_SEMAPHORE_0
  - LL_HSEM_SEMAPHORE_1
  - LL_HSEM_SEMAPHORE_2
  - LL_HSEM_SEMAPHORE_3
  - LL_HSEM_SEMAPHORE_4
  - LL_HSEM_SEMAPHORE_5
  - LL_HSEM_SEMAPHORE_6
  - LL_HSEM_SEMAPHORE_7
  - LL_HSEM_SEMAPHORE_8
  - LL_HSEM_SEMAPHORE_9
  - LL_HSEM_SEMAPHORE_10
  - LL_HSEM_SEMAPHORE_11
  - LL_HSEM_SEMAPHORE_12
  - LL_HSEM_SEMAPHORE_13
  - LL_HSEM_SEMAPHORE_14
  - LL_HSEM_SEMAPHORE_15
  - LL_HSEM_SEMAPHORE_16
  - LL_HSEM_SEMAPHORE_17
  - LL_HSEM_SEMAPHORE_18
  - LL_HSEM_SEMAPHORE_19
  - LL_HSEM_SEMAPHORE_20
  - LL_HSEM_SEMAPHORE_21
  - LL_HSEM_SEMAPHORE_22
  - LL_HSEM_SEMAPHORE_23
  - LL_HSEM_SEMAPHORE_24
  - LL_HSEM_SEMAPHORE_25
  - LL_HSEM_SEMAPHORE_26
  - LL_HSEM_SEMAPHORE_27
  - LL_HSEM_SEMAPHORE_28
  - LL_HSEM_SEMAPHORE_29
  - LL_HSEM_SEMAPHORE_30
  - LL_HSEM_SEMAPHORE_31
  - LL_HSEM_SEMAPHORE_ALL

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- C2IER ISEM LL_HSEM_IsEnabledIT_C2IER

LL_HSEM_ClearFlag_C1ICR

Function name

__STATIC_INLINE void LL_HSEM_ClearFlag_C1ICR (HSEM_TypeDef * HSEMx, uint32_t SemaphoreMask)

Function description

Clear interrupt status.

Parameters

- **HSEMx:** HSEM Instance.
- **SemaphoreMask:** This parameter can be a combination of the following values:
  - LL_HSEM_SEMAPHORE_0
  - LL_HSEM_SEMAPHORE_1
  - LL_HSEM_SEMAPHORE_2
  - LL_HSEM_SEMAPHORE_3
  - LL_HSEM_SEMAPHORE_4
  - LL_HSEM_SEMAPHORE_5
  - LL_HSEM_SEMAPHORE_6
  - LL_HSEM_SEMAPHORE_7
  - LL_HSEM_SEMAPHORE_8
  - LL_HSEM_SEMAPHORE_9
  - LL_HSEM_SEMAPHORE_10
  - LL_HSEM_SEMAPHORE_11
  - LL_HSEM_SEMAPHORE_12
  - LL_HSEM_SEMAPHORE_13
  - LL_HSEM_SEMAPHORE_14
  - LL_HSEM_SEMAPHORE_15
  - LL_HSEM_SEMAPHORE_16
  - LL_HSEM_SEMAPHORE_17
  - LL_HSEM_SEMAPHORE_18
  - LL_HSEM_SEMAPHORE_19
  - LL_HSEM_SEMAPHORE_20
  - LL_HSEM_SEMAPHORE_21
  - LL_HSEM_SEMAPHORE_22
  - LL_HSEM_SEMAPHORE_23
  - LL_HSEM_SEMAPHORE_24
  - LL_HSEM_SEMAPHORE_25
  - LL_HSEM_SEMAPHORE_26
  - LL_HSEM_SEMAPHORE_27
  - LL_HSEM_SEMAPHORE_28
  - LL_HSEM_SEMAPHORE_29
  - LL_HSEM_SEMAPHORE_30
  - LL_HSEM_SEMAPHORE_31
  - LL_HSEM_SEMAPHORE_ALL

Return values

- **None:**

Notes

- Availability of flags LL_HSEM_SEMAPHORE_16 to LL_HSEM_SEMAPHORE_31 depends on devices.

Reference Manual to LL API cross reference:

- C1ICR ISEM LL_HSEM_ClearFlag_C1ICR

LL_HSEM_IsActiveFlag_C1ISR

Function name

__STATIC_INLINE uint32_t LL_HSEM_IsActiveFlag_C1ISR (HSEM_TypeDef * HSEMx, uint32_t SemaphoreMask)

Function description

Get interrupt status from ISR register.

Parameters

- **HSEMx:** HSEM Instance.
- **SemaphoreMask:** This parameter can be a combination of the following values:
  - LL_HSEM_SEMAPHORE_0
  - LL_HSEM_SEMAPHORE_1
  - LL_HSEM_SEMAPHORE_2
  - LL_HSEM_SEMAPHORE_3
  - LL_HSEM_SEMAPHORE_4
  - LL_HSEM_SEMAPHORE_5
  - LL_HSEM_SEMAPHORE_6
  - LL_HSEM_SEMAPHORE_7
  - LL_HSEM_SEMAPHORE_8
  - LL_HSEM_SEMAPHORE_9
  - LL_HSEM_SEMAPHORE_10
  - LL_HSEM_SEMAPHORE_11
  - LL_HSEM_SEMAPHORE_12
  - LL_HSEM_SEMAPHORE_13
  - LL_HSEM_SEMAPHORE_14
  - LL_HSEM_SEMAPHORE_15
  - LL_HSEM_SEMAPHORE_16
  - LL_HSEM_SEMAPHORE_17
  - LL_HSEM_SEMAPHORE_18
  - LL_HSEM_SEMAPHORE_19
  - LL_HSEM_SEMAPHORE_20
  - LL_HSEM_SEMAPHORE_21
  - LL_HSEM_SEMAPHORE_22
  - LL_HSEM_SEMAPHORE_23
  - LL_HSEM_SEMAPHORE_24
  - LL_HSEM_SEMAPHORE_25
  - LL_HSEM_SEMAPHORE_26
  - LL_HSEM_SEMAPHORE_27
  - LL_HSEM_SEMAPHORE_28
  - LL_HSEM_SEMAPHORE_29
  - LL_HSEM_SEMAPHORE_30
  - LL_HSEM_SEMAPHORE_31
  - LL_HSEM_SEMAPHORE_ALL

Return values

- **State:** of bit (1 or 0).

Notes

- Availability of flags LL_HSEM_SEMAPHORE_16 to LL_HSEM_SEMAPHORE_31 depends on devices.

Reference Manual to LL API cross reference:

- C1ISR ISEM LL_HSEM_IsActiveFlag_C1ISR

LL_HSEM_IsActiveFlag_C1MISR

Function name

__STATIC_INLINE uint32_t LL_HSEM_IsActiveFlag_C1MISR (HSEM_TypeDef * HSEMx, uint32_t SemaphoreMask)

Function description

Get interrupt status from MISR register.

Parameters

- **HSEMx:** HSEM Instance.
- **SemaphoreMask:** This parameter can be a combination of the following values:
  - LL_HSEM_SEMAPHORE_0
  - LL_HSEM_SEMAPHORE_1
  - LL_HSEM_SEMAPHORE_2
  - LL_HSEM_SEMAPHORE_3
  - LL_HSEM_SEMAPHORE_4
  - LL_HSEM_SEMAPHORE_5
  - LL_HSEM_SEMAPHORE_6
  - LL_HSEM_SEMAPHORE_7
  - LL_HSEM_SEMAPHORE_8
  - LL_HSEM_SEMAPHORE_9
  - LL_HSEM_SEMAPHORE_10
  - LL_HSEM_SEMAPHORE_11
  - LL_HSEM_SEMAPHORE_12
  - LL_HSEM_SEMAPHORE_13
  - LL_HSEM_SEMAPHORE_14
  - LL_HSEM_SEMAPHORE_15
  - LL_HSEM_SEMAPHORE_16
  - LL_HSEM_SEMAPHORE_17
  - LL_HSEM_SEMAPHORE_18
  - LL_HSEM_SEMAPHORE_19
  - LL_HSEM_SEMAPHORE_20
  - LL_HSEM_SEMAPHORE_21
  - LL_HSEM_SEMAPHORE_22
  - LL_HSEM_SEMAPHORE_23
  - LL_HSEM_SEMAPHORE_24
  - LL_HSEM_SEMAPHORE_25
  - LL_HSEM_SEMAPHORE_26
  - LL_HSEM_SEMAPHORE_27
  - LL_HSEM_SEMAPHORE_28
  - LL_HSEM_SEMAPHORE_29
  - LL_HSEM_SEMAPHORE_30
  - LL_HSEM_SEMAPHORE_31
  - LL_HSEM_SEMAPHORE_ALL

Return values

- **State:** of bit (1 or 0).

Notes

- Availability of flags LL_HSEM_SEMAPHORE_16 to LL_HSEM_SEMAPHORE_31 depends on devices.

Reference Manual to LL API cross reference:

- C1MISR ISEM LL_HSEM_IsActiveFlag_C1MISR

LL_HSEM_ClearFlag_C2ICR

Function name

__STATIC_INLINE void LL_HSEM_ClearFlag_C2ICR (HSEM_TypeDef * HSEMx, uint32_t SemaphoreMask)

Function description

Clear interrupt status.

Parameters

- **HSEMx:** HSEM Instance.
- **SemaphoreMask:** This parameter can be a combination of the following values:
  - LL_HSEM_SEMAPHORE_0
  - LL_HSEM_SEMAPHORE_1
  - LL_HSEM_SEMAPHORE_2
  - LL_HSEM_SEMAPHORE_3
  - LL_HSEM_SEMAPHORE_4
  - LL_HSEM_SEMAPHORE_5
  - LL_HSEM_SEMAPHORE_6
  - LL_HSEM_SEMAPHORE_7
  - LL_HSEM_SEMAPHORE_8
  - LL_HSEM_SEMAPHORE_9
  - LL_HSEM_SEMAPHORE_10
  - LL_HSEM_SEMAPHORE_11
  - LL_HSEM_SEMAPHORE_12
  - LL_HSEM_SEMAPHORE_13
  - LL_HSEM_SEMAPHORE_14
  - LL_HSEM_SEMAPHORE_15
  - LL_HSEM_SEMAPHORE_16
  - LL_HSEM_SEMAPHORE_17
  - LL_HSEM_SEMAPHORE_18
  - LL_HSEM_SEMAPHORE_19
  - LL_HSEM_SEMAPHORE_20
  - LL_HSEM_SEMAPHORE_21
  - LL_HSEM_SEMAPHORE_22
  - LL_HSEM_SEMAPHORE_23
  - LL_HSEM_SEMAPHORE_24
  - LL_HSEM_SEMAPHORE_25
  - LL_HSEM_SEMAPHORE_26
  - LL_HSEM_SEMAPHORE_27
  - LL_HSEM_SEMAPHORE_28
  - LL_HSEM_SEMAPHORE_29
  - LL_HSEM_SEMAPHORE_30
  - LL_HSEM_SEMAPHORE_31
  - LL_HSEM_SEMAPHORE_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- C2ICR ISEM LL_HSEM_ClearFlag_C2ICR

LL_HSEM_IsActiveFlag_C2ISR

Function name

__STATIC_INLINE uint32_t LL_HSEM_IsActiveFlag_C2ISR (HSEM_TypeDef * HSEMx, uint32_t SemaphoreMask)

Function description

Get interrupt status from ISR register.

Parameters

- **HSEMx:** HSEM Instance.
- **SemaphoreMask:** This parameter can be a combination of the following values:
  - LL_HSEM_SEMAPHORE_0
  - LL_HSEM_SEMAPHORE_1
  - LL_HSEM_SEMAPHORE_2
  - LL_HSEM_SEMAPHORE_3
  - LL_HSEM_SEMAPHORE_4
  - LL_HSEM_SEMAPHORE_5
  - LL_HSEM_SEMAPHORE_6
  - LL_HSEM_SEMAPHORE_7
  - LL_HSEM_SEMAPHORE_8
  - LL_HSEM_SEMAPHORE_9
  - LL_HSEM_SEMAPHORE_10
  - LL_HSEM_SEMAPHORE_11
  - LL_HSEM_SEMAPHORE_12
  - LL_HSEM_SEMAPHORE_13
  - LL_HSEM_SEMAPHORE_14
  - LL_HSEM_SEMAPHORE_15
  - LL_HSEM_SEMAPHORE_16
  - LL_HSEM_SEMAPHORE_17
  - LL_HSEM_SEMAPHORE_18
  - LL_HSEM_SEMAPHORE_19
  - LL_HSEM_SEMAPHORE_20
  - LL_HSEM_SEMAPHORE_21
  - LL_HSEM_SEMAPHORE_22
  - LL_HSEM_SEMAPHORE_23
  - LL_HSEM_SEMAPHORE_24
  - LL_HSEM_SEMAPHORE_25
  - LL_HSEM_SEMAPHORE_26
  - LL_HSEM_SEMAPHORE_27
  - LL_HSEM_SEMAPHORE_28
  - LL_HSEM_SEMAPHORE_29
  - LL_HSEM_SEMAPHORE_30
  - LL_HSEM_SEMAPHORE_31
  - LL_HSEM_SEMAPHORE_ALL

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- C2ISR ISEM LL_HSEM_IsActiveFlag_C2ISR

LL_HSEM_IsActiveFlag_C2MISR

Function name

__STATIC_INLINE uint32_t LL_HSEM_IsActiveFlag_C2MISR (HSEM_TypeDef * HSEMx, uint32_t SemaphoreMask)

Function description

Get interrupt status from MISR register.

Parameters

- **HSEMx:** HSEM Instance.
- **SemaphoreMask:** This parameter can be a combination of the following values:
  - LL_HSEM_SEMAPHORE_0
  - LL_HSEM_SEMAPHORE_1
  - LL_HSEM_SEMAPHORE_2
  - LL_HSEM_SEMAPHORE_3
  - LL_HSEM_SEMAPHORE_4
  - LL_HSEM_SEMAPHORE_5
  - LL_HSEM_SEMAPHORE_6
  - LL_HSEM_SEMAPHORE_7
  - LL_HSEM_SEMAPHORE_8
  - LL_HSEM_SEMAPHORE_9
  - LL_HSEM_SEMAPHORE_10
  - LL_HSEM_SEMAPHORE_11
  - LL_HSEM_SEMAPHORE_12
  - LL_HSEM_SEMAPHORE_13
  - LL_HSEM_SEMAPHORE_14
  - LL_HSEM_SEMAPHORE_15
  - LL_HSEM_SEMAPHORE_16
  - LL_HSEM_SEMAPHORE_17
  - LL_HSEM_SEMAPHORE_18
  - LL_HSEM_SEMAPHORE_19
  - LL_HSEM_SEMAPHORE_20
  - LL_HSEM_SEMAPHORE_21
  - LL_HSEM_SEMAPHORE_22
  - LL_HSEM_SEMAPHORE_23
  - LL_HSEM_SEMAPHORE_24
  - LL_HSEM_SEMAPHORE_25
  - LL_HSEM_SEMAPHORE_26
  - LL_HSEM_SEMAPHORE_27
  - LL_HSEM_SEMAPHORE_28
  - LL_HSEM_SEMAPHORE_29
  - LL_HSEM_SEMAPHORE_30
  - LL_HSEM_SEMAPHORE_31
  - LL_HSEM_SEMAPHORE_ALL

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- C2MISR ISEM LL_HSEM_IsActiveFlag_C2MISR

112.2 HSEM Firmware driver defines

The following section lists the various define and macros of the module.

112.2.1 HSEM

HSEM

COREID Defines

LL_HSEM_COREID_NONE

LL_HSEM_COREID_CPU1

LL_HSEM_COREID_CPU2

LL_HSEM_COREID

Get Flags Defines

LL_HSEM_SEMAPHORE_0

LL_HSEM_SEMAPHORE_1

LL_HSEM_SEMAPHORE_2

LL_HSEM_SEMAPHORE_3

LL_HSEM_SEMAPHORE_4

LL_HSEM_SEMAPHORE_5

LL_HSEM_SEMAPHORE_6

LL_HSEM_SEMAPHORE_7

LL_HSEM_SEMAPHORE_8

LL_HSEM_SEMAPHORE_9

LL_HSEM_SEMAPHORE_10

LL_HSEM_SEMAPHORE_11

LL_HSEM_SEMAPHORE_12

LL_HSEM_SEMAPHORE_13

LL_HSEM_SEMAPHORE_14

LL_HSEM_SEMAPHORE_15

LL_HSEM_SEMAPHORE_16

LL_HSEM_SEMAPHORE_17

LL_HSEM_SEMAPHORE_18

LL_HSEM_SEMAPHORE_19

LL_HSEM_SEMAPHORE_20

LL_HSEM_SEMAPHORE_21

LL_HSEM_SEMAPHORE_22

LL_HSEM_SEMAPHORE_23

LL_HSEM_SEMAPHORE_24

LL_HSEM_SEMAPHORE_25

LL_HSEM_SEMAPHORE_26

LL_HSEM_SEMAPHORE_27

LL_HSEM_SEMAPHORE_28

LL_HSEM_SEMAPHORE_29

LL_HSEM_SEMAPHORE_30

LL_HSEM_SEMAPHORE_31

LL_HSEM_SEMAPHORE_ALL

Common Write and read registers Macros

LL_HSEM_WriteReg

Description:

- Write a value in HSEM register.

Parameters:

- __INSTANCE__: HSEM Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_HSEM_ReadReg

Description:

- Read a value in HSEM register.

Parameters:

- __INSTANCE__: HSEM Instance
- __REG__: Register to be read

Return value:

- Register: value

113 LL I2C Generic Driver

113.1 I2C Firmware driver registers structures

113.1.1 LL_I2C_InitTypeDef

LL_I2C_InitTypeDef is defined in the stm32h7xx_ll_i2c.h

Data Fields

- ***uint32_t PeripheralMode***
- ***uint32_t Timing***
- ***uint32_t AnalogFilter***
- ***uint32_t DigitalFilter***
- ***uint32_t OwnAddress1***
- ***uint32_t TypeAcknowledge***
- ***uint32_t OwnAddrSize***

Field Documentation

- ***uint32_t LL_I2C_InitTypeDef::PeripheralMode***

  Specifies the peripheral mode. This parameter can be a value of ***I2C_LL_EC_PERIPHERAL_MODE***.This feature can be modified afterwards using unitary function **LL_I2C_SetMode()**.
- ***uint32_t LL_I2C_InitTypeDef::Timing***

  Specifies the SDA setup, hold time and the SCL high, low period values. This parameter must be set by referring to the STM32CubeMX Tool and the helper macro **__LL_I2C_CONVERT_TIMINGS()**.This feature can be modified afterwards using unitary function **LL_I2C_SetTiming()**.
- ***uint32_t LL_I2C_InitTypeDef::AnalogFilter***

  Enables or disables analog noise filter. This parameter can be a value of ***I2C_LL_EC_ANALOGFILTER_SELECTION***.This feature can be modified afterwards using unitary functions **LL_I2C_EnableAnalogFilter()** or **LL_I2C_DisableAnalogFilter()**.
- ***uint32_t LL_I2C_InitTypeDef::DigitalFilter***

  Configures the digital noise filter. This parameter can be a number between Min_Data = 0x00 and Max_Data = 0x0F.This feature can be modified afterwards using unitary function **LL_I2C_SetDigitalFilter()**.
- ***uint32_t LL_I2C_InitTypeDef::OwnAddress1***

  Specifies the device own address 1. This parameter must be a value between Min_Data = 0x00 and Max_Data = 0x3FF.This feature can be modified afterwards using unitary function **LL_I2C_SetOwnAddress1()**.
- ***uint32_t LL_I2C_InitTypeDef::TypeAcknowledge***

  Specifies the ACKnowledge or Non ACKnowledge condition after the address receive match code or next received byte. This parameter can be a value of ***I2C_LL_EC_I2C_ACKNOWLEDGE***.This feature can be modified afterwards using unitary function **LL_I2C_AcknowledgeNextData()**.
- ***uint32_t LL_I2C_InitTypeDef::OwnAddrSize***

  Specifies the device own address 1 size (7-bit or 10-bit). This parameter can be a value of ***I2C_LL_EC_OWNADDRESS1***.This feature can be modified afterwards using unitary function **LL_I2C_SetOwnAddress1()**.

113.2 I2C Firmware driver API description

The following section lists the various functions of the I2C library.

113.2.1 Detailed description of functions

LL_I2C_Enable

Function name

__STATIC_INLINE void LL_I2C_Enable (I2C_TypeDef * I2Cx)

Function description

Enable I2C peripheral (PE = 1).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 PE LL_I2C_Enable

LL_I2C_Disable

Function name

__STATIC_INLINE void LL_I2C_Disable (I2C_TypeDef * I2Cx)

Function description

Disable I2C peripheral (PE = 0).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- When PE = 0, the I2C SCL and SDA lines are released. Internal state machines and status bits are put back to their reset value. When cleared, PE must be kept low for at least 3 APB clock cycles.

Reference Manual to LL API cross reference:

- CR1 PE LL_I2C_Disable

LL_I2C_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabled (I2C_TypeDef * I2Cx)

Function description

Check if the I2C peripheral is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 PE LL_I2C_IsEnabled

LL_I2C_ConfigFilters

Function name

__STATIC_INLINE void LL_I2C_ConfigFilters (I2C_TypeDef * I2Cx, uint32_t AnalogFilter, uint32_t DigitalFilter)

Function description

Configure Noise Filters (Analog and Digital).

Parameters

- **I2Cx:** I2C Instance.
- **AnalogFilter:** This parameter can be one of the following values:
  - LL_I2C_ANALOGFILTER_ENABLE
  - LL_I2C_ANALOGFILTER_DISABLE
- **DigitalFilter:** This parameter must be a value between Min_Data=0x00 (Digital filter disabled) and Max_Data=0x0F (Digital filter enabled and filtering capability up to 15*ti2cclk). This parameter is used to configure the digital noise filter on SDA and SCL input. The digital filter will filter spikes with a length of up to DNF[3:0]*ti2cclk.

Return values

- **None:**

Notes

- If the analog filter is also enabled, the digital filter is added to analog filter. The filters can only be programmed when the I2C is disabled (PE = 0).

Reference Manual to LL API cross reference:

- CR1 ANFOFF LL_I2C_ConfigFilters
- CR1 DNF LL_I2C_ConfigFilters

LL_I2C_SetDigitalFilter

Function name

__STATIC_INLINE void LL_I2C_SetDigitalFilter (I2C_TypeDef * I2Cx, uint32_t DigitalFilter)

Function description

Configure Digital Noise Filter.

Parameters

- **I2Cx:** I2C Instance.
- **DigitalFilter:** This parameter must be a value between Min_Data=0x00 (Digital filter disabled) and Max_Data=0x0F (Digital filter enabled and filtering capability up to 15*ti2cclk). This parameter is used to configure the digital noise filter on SDA and SCL input. The digital filter will filter spikes with a length of up to DNF[3:0]*ti2cclk.

Return values

- **None:**

Notes

- If the analog filter is also enabled, the digital filter is added to analog filter. This filter can only be programmed when the I2C is disabled (PE = 0).

Reference Manual to LL API cross reference:

- CR1 DNF LL_I2C_SetDigitalFilter

LL_I2C_GetDigitalFilter

Function name

__STATIC_INLINE uint32_t LL_I2C_GetDigitalFilter (I2C_TypeDef * I2Cx)

Function description

Get the current Digital Noise Filter configuration.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0xF

Reference Manual to LL API cross reference:

- CR1 DNF LL_I2C_GetDigitalFilter

LL_I2C_EnableAnalogFilter

Function name

__STATIC_INLINE void LL_I2C_EnableAnalogFilter (I2C_TypeDef * I2Cx)

Function description

Enable Analog Noise Filter.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- This filter can only be programmed when the I2C is disabled (PE = 0).

Reference Manual to LL API cross reference:

- CR1 ANFOFF LL_I2C_EnableAnalogFilter

LL_I2C_DisableAnalogFilter

Function name

__STATIC_INLINE void LL_I2C_DisableAnalogFilter (I2C_TypeDef * I2Cx)

Function description

Disable Analog Noise Filter.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- This filter can only be programmed when the I2C is disabled (PE = 0).

Reference Manual to LL API cross reference:

- CR1 ANFOFF LL_I2C_DisableAnalogFilter

LL_I2C_IsEnabledAnalogFilter

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledAnalogFilter (I2C_TypeDef * I2Cx)

Function description

Check if Analog Noise Filter is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 ANFOFF LL_I2C_IsEnabledAnalogFilter

LL_I2C_EnableDMAReq_TX

Function name

__STATIC_INLINE void LL_I2C_EnableDMAReq_TX (I2C_TypeDef * I2Cx)

Function description

Enable DMA transmission requests.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXDMAEN LL_I2C_EnableDMAReq_TX

LL_I2C_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_I2C_DisableDMAReq_TX (I2C_TypeDef * I2Cx)

Function description

Disable DMA transmission requests.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXDMAEN LL_I2C_DisableDMAReq_TX

LL_I2C_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledDMAReq_TX (I2C_TypeDef * I2Cx)

Function description

Check if DMA transmission requests are enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TXDMAEN LL_I2C_IsEnabledDMAReq_TX

LL_I2C_EnableDMAReq_RX

Function name

__STATIC_INLINE void LL_I2C_EnableDMAReq_RX (I2C_TypeDef * I2Cx)

Function description

Enable DMA reception requests.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXDMAEN LL_I2C_EnableDMAReq_RX

LL_I2C_DisableDMAReq_RX

Function name

__STATIC_INLINE void LL_I2C_DisableDMAReq_RX (I2C_TypeDef * I2Cx)

Function description

Disable DMA reception requests.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXDMAEN LL_I2C_DisableDMAReq_RX

LL_I2C_IsEnabledDMAReq_RX

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledDMAReq_RX (I2C_TypeDef * I2Cx)

Function description

Check if DMA reception requests are enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 RXDMAEN LL_I2C_IsEnabledDMAReq_RX

LL_I2C_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_I2C_DMA_GetRegAddr (I2C_TypeDef * I2Cx, uint32_t Direction)

Function description

Get the data register address used for DMA transfer.

Parameters

- **I2Cx:** I2C Instance
- **Direction:** This parameter can be one of the following values:
  - LL_I2C_DMA_REG_DATA_TRANSMIT
  - LL_I2C_DMA_REG_DATA_RECEIVE

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- TXDR TXDATA LL_I2C_DMA_GetRegAddr
- RXDR RXDATA LL_I2C_DMA_GetRegAddr

LL_I2C_EnableClockStretching

Function name

__STATIC_INLINE void LL_I2C_EnableClockStretching (I2C_TypeDef * I2Cx)

Function description

Enable Clock stretching.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- This bit can only be programmed when the I2C is disabled (PE = 0).

Reference Manual to LL API cross reference:

- CR1 NOSTRETCH LL_I2C_EnableClockStretching

LL_I2C_DisableClockStretching

Function name

__STATIC_INLINE void LL_I2C_DisableClockStretching (I2C_TypeDef * I2Cx)

Function description

Disable Clock stretching.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- This bit can only be programmed when the I2C is disabled (PE = 0).

Reference Manual to LL API cross reference:

- CR1 NOSTRETCH LL_I2C_DisableClockStretching

LL_I2C_IsEnabledClockStretching

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledClockStretching (I2C_TypeDef * I2Cx)

Function description

Check if Clock stretching is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 NOSTRETCH LL_I2C_IsEnabledClockStretching

LL_I2C_EnableSlaveByteControl

Function name

__STATIC_INLINE void LL_I2C_EnableSlaveByteControl (I2C_TypeDef * I2Cx)

Function description

Enable hardware byte control in slave mode.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 SBC LL_I2C_EnableSlaveByteControl

LL_I2C_DisableSlaveByteControl

Function name

__STATIC_INLINE void LL_I2C_DisableSlaveByteControl (I2C_TypeDef * I2Cx)

Function description

Disable hardware byte control in slave mode.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 SBC LL_I2C_DisableSlaveByteControl

LL_I2C_IsEnabledSlaveByteControl

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledSlaveByteControl (I2C_TypeDef * I2Cx)

Function description

Check if hardware byte control in slave mode is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 SBC LL_I2C_IsEnabledSlaveByteControl

LL_I2C_EnableWakeUpFromStop

Function name

__STATIC_INLINE void LL_I2C_EnableWakeUpFromStop (I2C_TypeDef * I2Cx)

Function description

Enable Wakeup from STOP.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_I2C_WAKEUP_FROMSTOP_INSTANCE(I2Cx) can be used to check whether or not WakeUpFromStop feature is supported by the I2Cx Instance.
- This bit can only be programmed when Digital Filter is disabled.

Reference Manual to LL API cross reference:

- CR1 WUPEN LL_I2C_EnableWakeUpFromStop

LL_I2C_DisableWakeUpFromStop

Function name

__STATIC_INLINE void LL_I2C_DisableWakeUpFromStop (I2C_TypeDef * I2Cx)

Function description

Disable Wakeup from STOP.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_I2C_WAKEUP_FROMSTOP_INSTANCE(I2Cx) can be used to check whether or not WakeUpFromStop feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 WUPEN LL_I2C_DisableWakeUpFromStop

LL_I2C_IsEnabledWakeUpFromStop

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledWakeUpFromStop (I2C_TypeDef * I2Cx)

Function description

Check if Wakeup from STOP is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_I2C_WAKEUP_FROMSTOP_INSTANCE(I2Cx) can be used to check whether or not WakeUpFromStop feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 WUPEN LL_I2C_IsEnabledWakeUpFromStop

LL_I2C_EnableGeneralCall

Function name

__STATIC_INLINE void LL_I2C_EnableGeneralCall (I2C_TypeDef * I2Cx)

Function description

Enable General Call.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- When enabled the Address 0x00 is ACKed.

Reference Manual to LL API cross reference:

- CR1 GCEN LL_I2C_EnableGeneralCall

LL_I2C_DisableGeneralCall

Function name

__STATIC_INLINE void LL_I2C_DisableGeneralCall (I2C_TypeDef * I2Cx)

Function description

Disable General Call.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- When disabled the Address 0x00 is NACKed.

Reference Manual to LL API cross reference:

- CR1 GCEN LL_I2C_DisableGeneralCall

LL_I2C_IsEnabledGeneralCall

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledGeneralCall (I2C_TypeDef * I2Cx)

Function description

Check if General Call is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 GCEN LL_I2C_IsEnabledGeneralCall

LL_I2C_SetMasterAddressingMode

Function name

__STATIC_INLINE void LL_I2C_SetMasterAddressingMode (I2C_TypeDef * I2Cx, uint32_t AddressingMode)

Function description

Configure the Master to operate in 7-bit or 10-bit addressing mode.

Parameters

- **I2Cx:** I2C Instance.
- **AddressingMode:** This parameter can be one of the following values:
  - LL_I2C_ADDRESSING_MODE_7BIT
  - LL_I2C_ADDRESSING_MODE_10BIT

Return values

- **None:**

Notes

- Changing this bit is not allowed, when the START bit is set.

Reference Manual to LL API cross reference:

- CR2 ADD10 LL_I2C_SetMasterAddressingMode

LL_I2C_GetMasterAddressingMode

Function name

__STATIC_INLINE uint32_t LL_I2C_GetMasterAddressingMode (I2C_TypeDef * I2Cx)

Function description

Get the Master addressing mode.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I2C_ADDRESSING_MODE_7BIT
  - LL_I2C_ADDRESSING_MODE_10BIT

Reference Manual to LL API cross reference:

- CR2 ADD10 LL_I2C_GetMasterAddressingMode

LL_I2C_SetOwnAddress1

Function name

__STATIC_INLINE void LL_I2C_SetOwnAddress1 (I2C_TypeDef * I2Cx, uint32_t OwnAddress1, uint32_t OwnAddrSize)

Function description

Set the Own Address1.

Parameters

- **I2Cx:** I2C Instance.
- **OwnAddress1:** This parameter must be a value between Min_Data=0 and Max_Data=0x3FF.
- **OwnAddrSize:** This parameter can be one of the following values:
  - LL_I2C_OWNADDRESS1_7BIT
  - LL_I2C_OWNADDRESS1_10BIT

Return values

- **None:**

Reference Manual to LL API cross reference:

- OAR1 OA1 LL_I2C_SetOwnAddress1
- OAR1 OA1MODE LL_I2C_SetOwnAddress1

LL_I2C_EnableOwnAddress1

Function name

__STATIC_INLINE void LL_I2C_EnableOwnAddress1 (I2C_TypeDef * I2Cx)

Function description

Enable acknowledge on Own Address1 match address.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- OAR1 OA1EN LL_I2C_EnableOwnAddress1

LL_I2C_DisableOwnAddress1

Function name

__STATIC_INLINE void LL_I2C_DisableOwnAddress1 (I2C_TypeDef * I2Cx)

Function description

Disable acknowledge on Own Address1 match address.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- OAR1 OA1EN LL_I2C_DisableOwnAddress1

LL_I2C_IsEnabledOwnAddress1

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledOwnAddress1 (I2C_TypeDef * I2Cx)

Function description

Check if Own Address1 acknowledge is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- OAR1 OA1EN LL_I2C_IsEnabledOwnAddress1

LL_I2C_SetOwnAddress2

Function name

__STATIC_INLINE void LL_I2C_SetOwnAddress2 (I2C_TypeDef * I2Cx, uint32_t OwnAddress2, uint32_t OwnAddrMask)

Function description

Set the 7bits Own Address2.

Parameters

- **I2Cx:** I2C Instance.
- **OwnAddress2:** Value between Min_Data=0 and Max_Data=0x7F.
- **OwnAddrMask:** This parameter can be one of the following values:
  - LL_I2C_OWNADDRESS2_NOMASK
  - LL_I2C_OWNADDRESS2_MASK01
  - LL_I2C_OWNADDRESS2_MASK02
  - LL_I2C_OWNADDRESS2_MASK03
  - LL_I2C_OWNADDRESS2_MASK04
  - LL_I2C_OWNADDRESS2_MASK05
  - LL_I2C_OWNADDRESS2_MASK06
  - LL_I2C_OWNADDRESS2_MASK07

Return values

- **None:**

Notes

- This action has no effect if own address2 is enabled.

Reference Manual to LL API cross reference:

- OAR2 OA2 LL_I2C_SetOwnAddress2
- OAR2 OA2MSK LL_I2C_SetOwnAddress2

LL_I2C_EnableOwnAddress2

Function name

__STATIC_INLINE void LL_I2C_EnableOwnAddress2 (I2C_TypeDef * I2Cx)

Function description

Enable acknowledge on Own Address2 match address.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- OAR2 OA2EN LL_I2C_EnableOwnAddress2

LL_I2C_DisableOwnAddress2

Function name

__STATIC_INLINE void LL_I2C_DisableOwnAddress2 (I2C_TypeDef * I2Cx)

Function description

Disable acknowledge on Own Address2 match address.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- OAR2 OA2EN LL_I2C_DisableOwnAddress2

LL_I2C_IsEnabledOwnAddress2

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledOwnAddress2 (I2C_TypeDef * I2Cx)

Function description

Check if Own Address1 acknowledge is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- OAR2 OA2EN LL_I2C_IsEnabledOwnAddress2

LL_I2C_SetTiming

Function name

__STATIC_INLINE void LL_I2C_SetTiming (I2C_TypeDef * I2Cx, uint32_t Timing)

Function description

Configure the SDA setup, hold time and the SCL high, low period.

Parameters

- **I2Cx:** I2C Instance.
- **Timing:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFFFFFFF.

Return values

- **None:**

Notes

- This bit can only be programmed when the I2C is disabled (PE = 0).
- This parameter is computed with the STM32CubeMX Tool.

Reference Manual to LL API cross reference:

- TIMINGR TIMINGR LL_I2C_SetTiming

LL_I2C_GetTimingPrescaler

Function name

__STATIC_INLINE uint32_t LL_I2C_GetTimingPrescaler (I2C_TypeDef * I2Cx)

Function description

Get the Timing Prescaler setting.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0xF

Reference Manual to LL API cross reference:

- TIMINGR PRESC LL_I2C_GetTimingPrescaler

LL_I2C_GetClockLowPeriod

Function name

__STATIC_INLINE uint32_t LL_I2C_GetClockLowPeriod (I2C_TypeDef * I2Cx)

Function description

Get the SCL low period setting.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- TIMINGR SCLL LL_I2C_GetClockLowPeriod

LL_I2C_GetClockHighPeriod

Function name

__STATIC_INLINE uint32_t LL_I2C_GetClockHighPeriod (I2C_TypeDef * I2Cx)

Function description

Get the SCL high period setting.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- TIMINGR SCLH LL_I2C_GetClockHighPeriod

LL_I2C_GetDataHoldTime

Function name

__STATIC_INLINE uint32_t LL_I2C_GetDataHoldTime (I2C_TypeDef * I2Cx)

Function description

Get the SDA hold time.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0xF

Reference Manual to LL API cross reference:

- TIMINGR SDADEL LL_I2C_GetDataHoldTime

LL_I2C_GetDataSetupTime

Function name

__STATIC_INLINE uint32_t LL_I2C_GetDataSetupTime (I2C_TypeDef * I2Cx)

Function description

Get the SDA setup time.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0xF

Reference Manual to LL API cross reference:

- TIMINGR SCLDEL LL_I2C_GetDataSetupTime

LL_I2C_SetMode

Function name

__STATIC_INLINE void LL_I2C_SetMode (I2C_TypeDef * I2Cx, uint32_t PeripheralMode)

Function description

Configure peripheral mode.

Parameters

- **I2Cx:** I2C Instance.
- **PeripheralMode:** This parameter can be one of the following values:
  - LL_I2C_MODE_I2C
  - LL_I2C_MODE_SMBUS_HOST
  - LL_I2C_MODE_SMBUS_DEVICE
  - LL_I2C_MODE_SMBUS_DEVICE_ARP

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 SMBHEN LL_I2C_SetMode
- CR1 SMBDEN LL_I2C_SetMode

LL_I2C_GetMode

Function name

__STATIC_INLINE uint32_t LL_I2C_GetMode (I2C_TypeDef * I2Cx)

Function description

Get peripheral mode.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I2C_MODE_I2C
  - LL_I2C_MODE_SMBUS_HOST
  - LL_I2C_MODE_SMBUS_DEVICE
  - LL_I2C_MODE_SMBUS_DEVICE_ARP

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 SMBHEN LL_I2C_GetMode
- CR1 SMBDEN LL_I2C_GetMode

LL_I2C_EnableSMBusAlert

Function name

__STATIC_INLINE void LL_I2C_EnableSMBusAlert (I2C_TypeDef * I2Cx)

Function description

Enable SMBus alert (Host or Device mode)

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- SMBus Device mode: SMBus Alert pin is drived low and Alert Response Address Header acknowledge is enabled. SMBus Host mode:SMBus Alert pin management is supported.

Reference Manual to LL API cross reference:

- CR1 ALERTEN LL_I2C_EnableSMBusAlert

LL_I2C_DisableSMBusAlert

Function name

__STATIC_INLINE void LL_I2C_DisableSMBusAlert (I2C_TypeDef * I2Cx)

Function description

Disable SMBus alert (Host or Device mode)

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- SMBus Device mode: SMBus Alert pin is not drived (can be used as a standard GPIO) and Alert Response Address Header acknowledge is disabled. SMBus Host mode:SMBus Alert pin management is not supported.

Reference Manual to LL API cross reference:

- CR1 ALERTEN LL_I2C_DisableSMBusAlert

LL_I2C_IsEnabledSMBusAlert

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusAlert (I2C_TypeDef * I2Cx)

Function description

Check if SMBus alert (Host or Device mode) is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 ALERTEN LL_I2C_IsEnabledSMBusAlert

LL_I2C_EnableSMBusPEC

Function name

__STATIC_INLINE void LL_I2C_EnableSMBusPEC (I2C_TypeDef * I2Cx)

Function description

Enable SMBus Packet Error Calculation (PEC).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 PECEN LL_I2C_EnableSMBusPEC

LL_I2C_DisableSMBusPEC

Function name

__STATIC_INLINE void LL_I2C_DisableSMBusPEC (I2C_TypeDef * I2Cx)

Function description

Disable SMBus Packet Error Calculation (PEC).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 PECEN LL_I2C_DisableSMBusPEC

LL_I2C_IsEnabledSMBusPEC

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusPEC (I2C_TypeDef * I2Cx)

Function description

Check if SMBus Packet Error Calculation (PEC) is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 PECEN LL_I2C_IsEnabledSMBusPEC

LL_I2C_ConfigSMBusTimeout

Function name

__STATIC_INLINE void LL_I2C_ConfigSMBusTimeout (I2C_TypeDef * I2Cx, uint32_t TimeoutA, uint32_t TimeoutAMode, uint32_t TimeoutB)

Function description

Configure the SMBus Clock Timeout.

Parameters

- **I2Cx:** I2C Instance.
- **TimeoutA:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFF.
- **TimeoutAMode:** This parameter can be one of the following values:
  - LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW
  - LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH
- **TimeoutB:**

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- This configuration can only be programmed when associated Timeout is disabled (TimeoutA and/ orTimeoutB).

Reference Manual to LL API cross reference:

- TIMEOUTR TIMEOUTA LL_I2C_ConfigSMBusTimeout
- TIMEOUTR TIDLE LL_I2C_ConfigSMBusTimeout
- TIMEOUTR TIMEOUTB LL_I2C_ConfigSMBusTimeout

LL_I2C_SetSMBusTimeoutA

Function name

__STATIC_INLINE void LL_I2C_SetSMBusTimeoutA (I2C_TypeDef * I2Cx, uint32_t TimeoutA)

Function description

Configure the SMBus Clock TimeoutA (SCL low timeout or SCL and SDA high timeout depends on TimeoutA mode).

Parameters

- **I2Cx:** I2C Instance.
- **TimeoutA:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFF.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- These bits can only be programmed when TimeoutA is disabled.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMEOUTA LL_I2C_SetSMBusTimeoutA

LL_I2C_GetSMBusTimeoutA

Function name

__STATIC_INLINE uint32_t LL_I2C_GetSMBusTimeoutA (I2C_TypeDef * I2Cx)

Function description

Get the SMBus Clock TimeoutA setting.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFF

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMEOUTA LL_I2C_GetSMBusTimeoutA

LL_I2C_SetSMBusTimeoutAMode

Function name

__STATIC_INLINE void LL_I2C_SetSMBusTimeoutAMode (I2C_TypeDef * I2Cx, uint32_t TimeoutAMode)

Function description

Set the SMBus Clock TimeoutA mode.

Parameters

- **I2Cx:** I2C Instance.
- **TimeoutAMode:** This parameter can be one of the following values:
  - LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW
  - LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- This bit can only be programmed when TimeoutA is disabled.

Reference Manual to LL API cross reference:

- TIMEOUTR TIDLE LL_I2C_SetSMBusTimeoutAMode

LL_I2C_GetSMBusTimeoutAMode

Function name

__STATIC_INLINE uint32_t LL_I2C_GetSMBusTimeoutAMode (I2C_TypeDef * I2Cx)

Function description

Get the SMBus Clock TimeoutA mode.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW
  - LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- TIMEOUTR TIDLE LL_I2C_GetSMBusTimeoutAMode

LL_I2C_SetSMBusTimeoutB

Function name

__STATIC_INLINE void LL_I2C_SetSMBusTimeoutB (I2C_TypeDef * I2Cx, uint32_t TimeoutB)

Function description

Configure the SMBus Extended Cumulative Clock TimeoutB (Master or Slave mode).

Parameters

- **I2Cx:** I2C Instance.
- **TimeoutB:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFF.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- These bits can only be programmed when TimeoutB is disabled.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMEOUTB LL_I2C_SetSMBusTimeoutB

LL_I2C_GetSMBusTimeoutB

Function name

__STATIC_INLINE uint32_t LL_I2C_GetSMBusTimeoutB (I2C_TypeDef * I2Cx)

Function description

Get the SMBus Extended Cumulative Clock TimeoutB setting.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFF

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMEOUTB LL_I2C_GetSMBusTimeoutB

LL_I2C_EnableSMBusTimeout

Function name

__STATIC_INLINE void LL_I2C_EnableSMBusTimeout (I2C_TypeDef * I2Cx, uint32_t ClockTimeout)

Function description

Enable the SMBus Clock Timeout.

Parameters

- **I2Cx:** I2C Instance.
- **ClockTimeout:** This parameter can be one of the following values:
  - LL_I2C_SMBUS_TIMEOUTA
  - LL_I2C_SMBUS_TIMEOUTB
  - LL_I2C_SMBUS_ALL_TIMEOUT

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMOUTEN LL_I2C_EnableSMBusTimeout
- TIMEOUTR TEXTEN LL_I2C_EnableSMBusTimeout

LL_I2C_DisableSMBusTimeout

Function name

__STATIC_INLINE void LL_I2C_DisableSMBusTimeout (I2C_TypeDef * I2Cx, uint32_t ClockTimeout)

Function description

Disable the SMBus Clock Timeout.

Parameters

- **I2Cx:** I2C Instance.
- **ClockTimeout:** This parameter can be one of the following values:
  - LL_I2C_SMBUS_TIMEOUTA
  - LL_I2C_SMBUS_TIMEOUTB
  - LL_I2C_SMBUS_ALL_TIMEOUT

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMOUTEN LL_I2C_DisableSMBusTimeout
- TIMEOUTR TEXTEN LL_I2C_DisableSMBusTimeout

LL_I2C_IsEnabledSMBusTimeout

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusTimeout (I2C_TypeDef * I2Cx, uint32_t ClockTimeout)

Function description

Check if the SMBus Clock Timeout is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.
- **ClockTimeout:** This parameter can be one of the following values:
  - LL_I2C_SMBUS_TIMEOUTA
  - LL_I2C_SMBUS_TIMEOUTB
  - LL_I2C_SMBUS_ALL_TIMEOUT

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMOUTEN LL_I2C_IsEnabledSMBusTimeout
- TIMEOUTR TEXTEN LL_I2C_IsEnabledSMBusTimeout

LL_I2C_EnableIT_TX

Function name

__STATIC_INLINE void LL_I2C_EnableIT_TX (I2C_TypeDef * I2Cx)

Function description

Enable TXIS interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXIE LL_I2C_EnableIT_TX

LL_I2C_DisableIT_TX

Function name

__STATIC_INLINE void LL_I2C_DisableIT_TX (I2C_TypeDef * I2Cx)

Function description

Disable TXIS interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXIE LL_I2C_DisableIT_TX

LL_I2C_IsEnabledIT_TX

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_TX (I2C_TypeDef * I2Cx)

Function description

Check if the TXIS Interrupt is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TXIE LL_I2C_IsEnabledIT_TX

LL_I2C_EnableIT_RX

Function name

__STATIC_INLINE void LL_I2C_EnableIT_RX (I2C_TypeDef * I2Cx)

Function description

Enable RXNE interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXIE LL_I2C_EnableIT_RX

LL_I2C_DisableIT_RX

Function name

__STATIC_INLINE void LL_I2C_DisableIT_RX (I2C_TypeDef * I2Cx)

Function description

Disable RXNE interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXIE LL_I2C_DisableIT_RX

LL_I2C_IsEnabledIT_RX

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_RX (I2C_TypeDef * I2Cx)

Function description

Check if the RXNE Interrupt is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 RXIE LL_I2C_IsEnabledIT_RX

LL_I2C_EnableIT_ADDR

Function name

__STATIC_INLINE void LL_I2C_EnableIT_ADDR (I2C_TypeDef * I2Cx)

Function description

Enable Address match interrupt (slave mode only).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 ADDRIE LL_I2C_EnableIT_ADDR

LL_I2C_DisableIT_ADDR

Function name

__STATIC_INLINE void LL_I2C_DisableIT_ADDR (I2C_TypeDef * I2Cx)

Function description

Disable Address match interrupt (slave mode only).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 ADDRIE LL_I2C_DisableIT_ADDR

LL_I2C_IsEnabledIT_ADDR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_ADDR (I2C_TypeDef * I2Cx)

Function description

Check if Address match interrupt is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 ADDRIE LL_I2C_IsEnabledIT_ADDR

LL_I2C_EnableIT_NACK

Function name

__STATIC_INLINE void LL_I2C_EnableIT_NACK (I2C_TypeDef * I2Cx)

Function description

Enable Not acknowledge received interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 NACKIE LL_I2C_EnableIT_NACK

LL_I2C_DisableIT_NACK

Function name

__STATIC_INLINE void LL_I2C_DisableIT_NACK (I2C_TypeDef * I2Cx)

Function description

Disable Not acknowledge received interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 NACKIE LL_I2C_DisableIT_NACK

LL_I2C_IsEnabledIT_NACK

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_NACK (I2C_TypeDef * I2Cx)

Function description

Check if Not acknowledge received interrupt is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 NACKIE LL_I2C_IsEnabledIT_NACK

LL_I2C_EnableIT_STOP

Function name

__STATIC_INLINE void LL_I2C_EnableIT_STOP (I2C_TypeDef * I2Cx)

Function description

Enable STOP detection interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 STOPIE LL_I2C_EnableIT_STOP

LL_I2C_DisableIT_STOP

Function name

__STATIC_INLINE void LL_I2C_DisableIT_STOP (I2C_TypeDef * I2Cx)

Function description

Disable STOP detection interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 STOPIE LL_I2C_DisableIT_STOP

LL_I2C_IsEnabledIT_STOP

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_STOP (I2C_TypeDef * I2Cx)

Function description

Check if STOP detection interrupt is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 STOPIE LL_I2C_IsEnabledIT_STOP

LL_I2C_EnableIT_TC

Function name

__STATIC_INLINE void LL_I2C_EnableIT_TC (I2C_TypeDef * I2Cx)

Function description

Enable Transfer Complete interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Any of these events will generate interrupt : Transfer Complete (TC) Transfer Complete Reload (TCR)

Reference Manual to LL API cross reference:

- CR1 TCIE LL_I2C_EnableIT_TC

LL_I2C_DisableIT_TC

Function name

__STATIC_INLINE void LL_I2C_DisableIT_TC (I2C_TypeDef * I2Cx)

Function description

Disable Transfer Complete interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Any of these events will generate interrupt : Transfer Complete (TC) Transfer Complete Reload (TCR)

Reference Manual to LL API cross reference:

- CR1 TCIE LL_I2C_DisableIT_TC

LL_I2C_IsEnabledIT_TC

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_TC (I2C_TypeDef * I2Cx)

Function description

Check if Transfer Complete interrupt is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TCIE LL_I2C_IsEnabledIT_TC

LL_I2C_EnableIT_ERR

Function name

__STATIC_INLINE void LL_I2C_EnableIT_ERR (I2C_TypeDef * I2Cx)

Function description

Enable Error interrupts.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- Any of these errors will generate interrupt : Arbitration Loss (ARLO) Bus Error detection (BERR) Overrun/ Underrun (OVR) SMBus Timeout detection (TIMEOUT) SMBus PEC error detection (PECERR) SMBus Alert pin event detection (ALERT)

Reference Manual to LL API cross reference:

- CR1 ERRIE LL_I2C_EnableIT_ERR

LL_I2C_DisableIT_ERR

Function name

__STATIC_INLINE void LL_I2C_DisableIT_ERR (I2C_TypeDef * I2Cx)

Function description

Disable Error interrupts.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- Any of these errors will generate interrupt : Arbitration Loss (ARLO) Bus Error detection (BERR) Overrun/ Underrun (OVR) SMBus Timeout detection (TIMEOUT) SMBus PEC error detection (PECERR) SMBus Alert pin event detection (ALERT)

Reference Manual to LL API cross reference:

- CR1 ERRIE LL_I2C_DisableIT_ERR

LL_I2C_IsEnabledIT_ERR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_ERR (I2C_TypeDef * I2Cx)

Function description

Check if Error interrupts are enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 ERRIE LL_I2C_IsEnabledIT_ERR

LL_I2C_IsActiveFlag_TXE

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TXE (I2C_TypeDef * I2Cx)

Function description

Indicate the status of Transmit data register empty flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: When next data is written in Transmit data register. SET: When Transmit data register is empty.

Reference Manual to LL API cross reference:

- ISR TXE LL_I2C_IsActiveFlag_TXE

LL_I2C_IsActiveFlag_TXIS

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TXIS (I2C_TypeDef * I2Cx)

Function description

Indicate the status of Transmit interrupt flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: When next data is written in Transmit data register. SET: When Transmit data register is empty.

Reference Manual to LL API cross reference:

- ISR TXIS LL_I2C_IsActiveFlag_TXIS

LL_I2C_IsActiveFlag_RXNE

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_RXNE (I2C_TypeDef * I2Cx)

Function description

Indicate the status of Receive data register not empty flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: When Receive data register is read. SET: When the received data is copied in Receive data register.

Reference Manual to LL API cross reference:

- ISR RXNE LL_I2C_IsActiveFlag_RXNE

LL_I2C_IsActiveFlag_ADDR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_ADDR (I2C_TypeDef * I2Cx)

Function description

Indicate the status of Address matched flag (slave mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When the received slave address matched with one of the enabled slave address.

Reference Manual to LL API cross reference:

- ISR ADDR LL_I2C_IsActiveFlag_ADDR

LL_I2C_IsActiveFlag_NACK

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_NACK (I2C_TypeDef * I2Cx)

Function description

Indicate the status of Not Acknowledge received flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When a NACK is received after a byte transmission.

Reference Manual to LL API cross reference:

- ISR NACKF LL_I2C_IsActiveFlag_NACK

LL_I2C_IsActiveFlag_STOP

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_STOP (I2C_TypeDef * I2Cx)

Function description

Indicate the status of Stop detection flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When a Stop condition is detected.

Reference Manual to LL API cross reference:

- ISR STOPF LL_I2C_IsActiveFlag_STOP

LL_I2C_IsActiveFlag_TC

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TC (I2C_TypeDef * I2Cx)

Function description

Indicate the status of Transfer complete flag (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When RELOAD=0, AUTOEND=0 and NBYTES date have been transferred.

Reference Manual to LL API cross reference:

- ISR TC LL_I2C_IsActiveFlag_TC

LL_I2C_IsActiveFlag_TCR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TCR (I2C_TypeDef * I2Cx)

Function description

Indicate the status of Transfer complete flag (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When RELOAD=1 and NBYTES date have been transferred.

Reference Manual to LL API cross reference:

- ISR TCR LL_I2C_IsActiveFlag_TCR

LL_I2C_IsActiveFlag_BERR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_BERR (I2C_TypeDef * I2Cx)

Function description

Indicate the status of Bus error flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When a misplaced Start or Stop condition is detected.

Reference Manual to LL API cross reference:

- ISR BERR LL_I2C_IsActiveFlag_BERR

LL_I2C_IsActiveFlag_ARLO

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_ARLO (I2C_TypeDef * I2Cx)

Function description

Indicate the status of Arbitration lost flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When arbitration lost.

Reference Manual to LL API cross reference:

- ISR ARLO LL_I2C_IsActiveFlag_ARLO

LL_I2C_IsActiveFlag_OVR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_OVR (I2C_TypeDef * I2Cx)

Function description

Indicate the status of Overrun/Underrun flag (slave mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When an overrun/underrun error occurs (Clock Stretching Disabled).

Reference Manual to LL API cross reference:

- ISR OVR LL_I2C_IsActiveFlag_OVR

LL_I2C_IsActiveSMBusFlag_PECERR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_PECERR (I2C_TypeDef * I2Cx)

Function description

Indicate the status of SMBus PEC error flag in reception.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- RESET: Clear default value. SET: When the received PEC does not match with the PEC register content.

Reference Manual to LL API cross reference:

- ISR PECERR LL_I2C_IsActiveSMBusFlag_PECERR

LL_I2C_IsActiveSMBusFlag_TIMEOUT

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_TIMEOUT (I2C_TypeDef * I2Cx)

Function description

Indicate the status of SMBus Timeout detection flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- RESET: Clear default value. SET: When a timeout or extended clock timeout occurs.

Reference Manual to LL API cross reference:

- ISR TIMEOUT LL_I2C_IsActiveSMBusFlag_TIMEOUT

LL_I2C_IsActiveSMBusFlag_ALERT

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_ALERT (I2C_TypeDef * I2Cx)

Function description

Indicate the status of SMBus alert flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- RESET: Clear default value. SET: When SMBus host configuration, SMBus alert enabled and a falling edge event occurs on SMBA pin.

Reference Manual to LL API cross reference:

- ISR ALERT LL_I2C_IsActiveSMBusFlag_ALERT

LL_I2C_IsActiveFlag_BUSY

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_BUSY (I2C_TypeDef * I2Cx)

Function description

Indicate the status of Bus Busy flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When a Start condition is detected.

Reference Manual to LL API cross reference:

- ISR BUSY LL_I2C_IsActiveFlag_BUSY

LL_I2C_ClearFlag_ADDR

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_ADDR (I2C_TypeDef * I2Cx)

Function description

Clear Address Matched flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ADDRCF LL_I2C_ClearFlag_ADDR

LL_I2C_ClearFlag_NACK

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_NACK (I2C_TypeDef * I2Cx)

Function description

Clear Not Acknowledge flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR NACKCF LL_I2C_ClearFlag_NACK

LL_I2C_ClearFlag_STOP

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_STOP (I2C_TypeDef * I2Cx)

Function description

Clear Stop detection flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR STOPCF LL_I2C_ClearFlag_STOP

LL_I2C_ClearFlag_TXE

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_TXE (I2C_TypeDef * I2Cx)

Function description

Clear Transmit data register empty flag (TXE).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- This bit can be clear by software in order to flush the transmit data register (TXDR).

Reference Manual to LL API cross reference:

- ISR TXE LL_I2C_ClearFlag_TXE

LL_I2C_ClearFlag_BERR

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_BERR (I2C_TypeDef * I2Cx)

Function description

Clear Bus error flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR BERRCF LL_I2C_ClearFlag_BERR

LL_I2C_ClearFlag_ARLO

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_ARLO (I2C_TypeDef * I2Cx)

Function description

Clear Arbitration lost flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ARLOCF LL_I2C_ClearFlag_ARLO

LL_I2C_ClearFlag_OVR

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_OVR (I2C_TypeDef * I2Cx)

Function description

Clear Overrun/Underrun flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR OVRCF LL_I2C_ClearFlag_OVR

LL_I2C_ClearSMBusFlag_PECERR

Function name

__STATIC_INLINE void LL_I2C_ClearSMBusFlag_PECERR (I2C_TypeDef * I2Cx)

Function description

Clear SMBus PEC error flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- ICR PECCF LL_I2C_ClearSMBusFlag_PECERR

LL_I2C_ClearSMBusFlag_TIMEOUT

Function name

__STATIC_INLINE void LL_I2C_ClearSMBusFlag_TIMEOUT (I2C_TypeDef * I2Cx)

Function description

Clear SMBus Timeout detection flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- ICR TIMOUTCF LL_I2C_ClearSMBusFlag_TIMEOUT

LL_I2C_ClearSMBusFlag_ALERT

Function name

__STATIC_INLINE void LL_I2C_ClearSMBusFlag_ALERT (I2C_TypeDef * I2Cx)

Function description

Clear SMBus Alert flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- ICR ALERTCF LL_I2C_ClearSMBusFlag_ALERT

LL_I2C_EnableAutoEndMode

Function name

__STATIC_INLINE void LL_I2C_EnableAutoEndMode (I2C_TypeDef * I2Cx)

Function description

Enable automatic STOP condition generation (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Automatic end mode : a STOP condition is automatically sent when NBYTES data are transferred. This bit has no effect in slave mode or when RELOAD bit is set.

Reference Manual to LL API cross reference:

- CR2 AUTOEND LL_I2C_EnableAutoEndMode

LL_I2C_DisableAutoEndMode

Function name

__STATIC_INLINE void LL_I2C_DisableAutoEndMode (I2C_TypeDef * I2Cx)

Function description

Disable automatic STOP condition generation (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Software end mode : TC flag is set when NBYTES data are transferre, stretching SCL low.

Reference Manual to LL API cross reference:

- CR2 AUTOEND LL_I2C_DisableAutoEndMode

LL_I2C_IsEnabledAutoEndMode

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledAutoEndMode (I2C_TypeDef * I2Cx)

Function description

Check if automatic STOP condition is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 AUTOEND LL_I2C_IsEnabledAutoEndMode

LL_I2C_EnableReloadMode

Function name

__STATIC_INLINE void LL_I2C_EnableReloadMode (I2C_TypeDef * I2Cx)

Function description

Enable reload mode (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The transfer is not completed after the NBYTES data transfer, NBYTES will be reloaded when TCR flag is set.

Reference Manual to LL API cross reference:

- CR2 RELOAD LL_I2C_EnableReloadMode

LL_I2C_DisableReloadMode

Function name

__STATIC_INLINE void LL_I2C_DisableReloadMode (I2C_TypeDef * I2Cx)

Function description

Disable reload mode (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The transfer is completed after the NBYTES data transfer(STOP or RESTART will follow).

Reference Manual to LL API cross reference:

- CR2 RELOAD LL_I2C_DisableReloadMode

LL_I2C_IsEnabledReloadMode

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledReloadMode (I2C_TypeDef * I2Cx)

Function description

Check if reload mode is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 RELOAD LL_I2C_IsEnabledReloadMode

LL_I2C_SetTransferSize

Function name

__STATIC_INLINE void LL_I2C_SetTransferSize (I2C_TypeDef * I2Cx, uint32_t TransferSize)

Function description

Configure the number of bytes for transfer.

Parameters

- **I2Cx:** I2C Instance.
- **TransferSize:** This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Notes

- Changing these bits when START bit is set is not allowed.

Reference Manual to LL API cross reference:

- CR2 NBYTES LL_I2C_SetTransferSize

LL_I2C_GetTransferSize

Function name

__STATIC_INLINE uint32_t LL_I2C_GetTransferSize (I2C_TypeDef * I2Cx)

Function description

Get the number of bytes configured for transfer.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- CR2 NBYTES LL_I2C_GetTransferSize

LL_I2C_AcknowledgeNextData

Function name

__STATIC_INLINE void LL_I2C_AcknowledgeNextData (I2C_TypeDef * I2Cx, uint32_t TypeAcknowledge)

Function description

Prepare the generation of a ACKnowledge or Non ACKnowledge condition after the address receive match code or next received byte.

Parameters

- **I2Cx:** I2C Instance.
- **TypeAcknowledge:** This parameter can be one of the following values:
  - LL_I2C_ACK
  - LL_I2C_NACK

Return values

- **None:**

Notes

- Usage in Slave mode only.

Reference Manual to LL API cross reference:

- CR2 NACK LL_I2C_AcknowledgeNextData

LL_I2C_GenerateStartCondition

Function name

__STATIC_INLINE void LL_I2C_GenerateStartCondition (I2C_TypeDef * I2Cx)

Function description

Generate a START or RESTART condition.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The START bit can be set even if bus is BUSY or I2C is in slave mode. This action has no effect when RELOAD is set.

Reference Manual to LL API cross reference:

- CR2 START LL_I2C_GenerateStartCondition

LL_I2C_GenerateStopCondition

Function name

__STATIC_INLINE void LL_I2C_GenerateStopCondition (I2C_TypeDef * I2Cx)

Function description

Generate a STOP condition after the current byte transfer (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 STOP LL_I2C_GenerateStopCondition

LL_I2C_EnableAuto10BitRead

Function name

__STATIC_INLINE void LL_I2C_EnableAuto10BitRead (I2C_TypeDef * I2Cx)

Function description

Enable automatic RESTART Read request condition for 10bit address header (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The master sends the complete 10bit slave address read sequence : Start + 2 bytes 10bit address in Write direction + Restart + first 7 bits of 10bit address in Read direction.

Reference Manual to LL API cross reference:

- CR2 HEAD10R LL_I2C_EnableAuto10BitRead

LL_I2C_DisableAuto10BitRead

Function name

__STATIC_INLINE void LL_I2C_DisableAuto10BitRead (I2C_TypeDef * I2Cx)

Function description

Disable automatic RESTART Read request condition for 10bit address header (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The master only sends the first 7 bits of 10bit address in Read direction.

Reference Manual to LL API cross reference:

- CR2 HEAD10R LL_I2C_DisableAuto10BitRead

LL_I2C_IsEnabledAuto10BitRead

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledAuto10BitRead (I2C_TypeDef * I2Cx)

Function description

Check if automatic RESTART Read request condition for 10bit address header is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 HEAD10R LL_I2C_IsEnabledAuto10BitRead

LL_I2C_SetTransferRequest

Function name

__STATIC_INLINE void LL_I2C_SetTransferRequest (I2C_TypeDef * I2Cx, uint32_t TransferRequest)

Function description

Configure the transfer direction (master mode).

Parameters

- **I2Cx:** I2C Instance.
- **TransferRequest:** This parameter can be one of the following values:
  - LL_I2C_REQUEST_WRITE
  - LL_I2C_REQUEST_READ

Return values

- **None:**

Notes

- Changing these bits when START bit is set is not allowed.

Reference Manual to LL API cross reference:

- CR2 RD_WRN LL_I2C_SetTransferRequest

LL_I2C_GetTransferRequest

Function name

__STATIC_INLINE uint32_t LL_I2C_GetTransferRequest (I2C_TypeDef * I2Cx)

Function description

Get the transfer direction requested (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I2C_REQUEST_WRITE
  - LL_I2C_REQUEST_READ

Reference Manual to LL API cross reference:

- CR2 RD_WRN LL_I2C_GetTransferRequest

LL_I2C_SetSlaveAddr

Function name

__STATIC_INLINE void LL_I2C_SetSlaveAddr (I2C_TypeDef * I2Cx, uint32_t SlaveAddr)

Function description

Configure the slave address for transfer (master mode).

Parameters

- **I2Cx:** I2C Instance.
- **SlaveAddr:** This parameter must be a value between Min_Data=0x00 and Max_Data=0x3F.

Return values

- **None:**

Notes

- Changing these bits when START bit is set is not allowed.

Reference Manual to LL API cross reference:

- CR2 SADD LL_I2C_SetSlaveAddr

LL_I2C_GetSlaveAddr

Function name

__STATIC_INLINE uint32_t LL_I2C_GetSlaveAddr (I2C_TypeDef * I2Cx)

Function description

Get the slave address programmed for transfer.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0x3F

Reference Manual to LL API cross reference:

- CR2 SADD LL_I2C_GetSlaveAddr

LL_I2C_HandleTransfer

Function name

__STATIC_INLINE void LL_I2C_HandleTransfer (I2C_TypeDef * I2Cx, uint32_t SlaveAddr, uint32_t SlaveAddrSize, uint32_t TransferSize, uint32_t EndMode, uint32_t Request)

Function description

Handles I2Cx communication when starting transfer or during transfer (TC or TCR flag are set).

Parameters

- **I2Cx:** I2C Instance.
- **SlaveAddr:** Specifies the slave address to be programmed.
- **SlaveAddrSize:** This parameter can be one of the following values:
  - LL_I2C_ADDRSLAVE_7BIT
  - LL_I2C_ADDRSLAVE_10BIT
- **TransferSize:** Specifies the number of bytes to be programmed. This parameter must be a value between Min_Data=0 and Max_Data=255.
- **EndMode:** This parameter can be one of the following values:
  - LL_I2C_MODE_RELOAD
  - LL_I2C_MODE_AUTOEND
  - LL_I2C_MODE_SOFTEND
  - LL_I2C_MODE_SMBUS_RELOAD
  - LL_I2C_MODE_SMBUS_AUTOEND_NO_PEC
  - LL_I2C_MODE_SMBUS_SOFTEND_NO_PEC
  - LL_I2C_MODE_SMBUS_AUTOEND_WITH_PEC
  - LL_I2C_MODE_SMBUS_SOFTEND_WITH_PEC
- **Request:** This parameter can be one of the following values:
  - LL_I2C_GENERATE_NOSTARTSTOP
  - LL_I2C_GENERATE_STOP
  - LL_I2C_GENERATE_START_READ
  - LL_I2C_GENERATE_START_WRITE
  - LL_I2C_GENERATE_RESTART_7BIT_READ
  - LL_I2C_GENERATE_RESTART_7BIT_WRITE
  - LL_I2C_GENERATE_RESTART_10BIT_READ
  - LL_I2C_GENERATE_RESTART_10BIT_WRITE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 SADD LL_I2C_HandleTransfer
- CR2 ADD10 LL_I2C_HandleTransfer
- CR2 RD_WRN LL_I2C_HandleTransfer
- CR2 START LL_I2C_HandleTransfer
- CR2 STOP LL_I2C_HandleTransfer
- CR2 RELOAD LL_I2C_HandleTransfer
- CR2 NBYTES LL_I2C_HandleTransfer
- CR2 AUTOEND LL_I2C_HandleTransfer
- CR2 HEAD10R LL_I2C_HandleTransfer

LL_I2C_GetTransferDirection

Function name

__STATIC_INLINE uint32_t LL_I2C_GetTransferDirection (I2C_TypeDef * I2Cx)

Function description

Indicate the value of transfer direction (slave mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I2C_DIRECTION_WRITE
  - LL_I2C_DIRECTION_READ

Notes

- RESET: Write transfer, Slave enters in receiver mode. SET: Read transfer, Slave enters in transmitter mode.

Reference Manual to LL API cross reference:

- ISR DIR LL_I2C_GetTransferDirection

LL_I2C_GetAddressMatchCode

Function name

__STATIC_INLINE uint32_t LL_I2C_GetAddressMatchCode (I2C_TypeDef * I2Cx)

Function description

Return the slave matched address.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x3F

Reference Manual to LL API cross reference:

- ISR ADDCODE LL_I2C_GetAddressMatchCode

LL_I2C_EnableSMBusPECCompare

Function name

__STATIC_INLINE void LL_I2C_EnableSMBusPECCompare (I2C_TypeDef * I2Cx)

Function description

Enable internal comparison of the SMBus Packet Error byte (transmission or reception mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- This feature is cleared by hardware when the PEC byte is transferred, or when a STOP condition or an Address Matched is received. This bit has no effect when RELOAD bit is set. This bit has no effect in device mode when SBC bit is not set.

Reference Manual to LL API cross reference:

- CR2 PECBYTE LL_I2C_EnableSMBusPECCompare

LL_I2C_IsEnabledSMBusPECCompare

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusPECCompare (I2C_TypeDef * I2Cx)

Function description

Check if the SMBus Packet Error byte internal comparison is requested or not.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR2 PECBYTE LL_I2C_IsEnabledSMBusPECCompare

LL_I2C_GetSMBusPEC

Function name

__STATIC_INLINE uint32_t LL_I2C_GetSMBusPEC (I2C_TypeDef * I2Cx)

Function description

Get the SMBus Packet Error byte calculated.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- PECR PEC LL_I2C_GetSMBusPEC

LL_I2C_ReceiveData8

Function name

__STATIC_INLINE uint8_t LL_I2C_ReceiveData8 (I2C_TypeDef * I2Cx)

Function description

Read Receive Data register.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- RXDR RXDATA LL_I2C_ReceiveData8

LL_I2C_TransmitData8

Function name

__STATIC_INLINE void LL_I2C_TransmitData8 (I2C_TypeDef * I2Cx, uint8_t Data)

Function description

Write in Transmit Data Register .

Parameters

- **I2Cx:** I2C Instance.
- **Data:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TXDR TXDATA LL_I2C_TransmitData8

LL_I2C_Init

Function name

ErrorStatus LL_I2C_Init (I2C_TypeDef * I2Cx, LL_I2C_InitTypeDef * I2C_InitStruct)

Function description

Initialize the I2C registers according to the specified parameters in I2C_InitStruct.

Parameters

- **I2Cx:** I2C Instance.
- **I2C_InitStruct:** pointer to a LL_I2C_InitTypeDef structure.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: I2C registers are initialized
  - ERROR: Not applicable

LL_I2C_DeInit

Function name

ErrorStatus LL_I2C_DeInit (I2C_TypeDef * I2Cx)

Function description

De-initialize the I2C registers to their default reset values.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: I2C registers are de-initialized
  - ERROR: I2C registers are not de-initialized

LL_I2C_StructInit

Function name

void LL_I2C_StructInit (LL_I2C_InitTypeDef * I2C_InitStruct)

Function description

Set each LL_I2C_InitTypeDef field to default value.

Parameters

- **I2C_InitStruct:** Pointer to a LL_I2C_InitTypeDef structure.

Return values

- **None:**

113.3 I2C Firmware driver defines

The following section lists the various define and macros of the module.

113.3.1 I2C

I2C

Master Addressing Mode

LL_I2C_ADDRESSING_MODE_7BIT

Master operates in 7-bit addressing mode.

LL_I2C_ADDRESSING_MODE_10BIT

Master operates in 10-bit addressing mode.

Slave Address Length

LL_I2C_ADDRSLAVE_7BIT

Slave Address in 7-bit.

LL_I2C_ADDRSLAVE_10BIT

Slave Address in 10-bit.

Analog Filter Selection

LL_I2C_ANALOGFILTER_ENABLE

Analog filter is enabled.

LL_I2C_ANALOGFILTER_DISABLE

Analog filter is disabled.

Clear Flags Defines

LL_I2C_ICR_ADDRCF

Address Matched flag

LL_I2C_ICR_NACKCF

Not Acknowledge flag

LL_I2C_ICR_STOPCF

Stop detection flag

LL_I2C_ICR_BERRCF

Bus error flag

LL_I2C_ICR_ARLOCF

Arbitration Lost flag

LL_I2C_ICR_OVRCF

Overrun/Underrun flag

LL_I2C_ICR_PECCF

PEC error flag

LL_I2C_ICR_TIMOUTCF

Timeout detection flag

LL_I2C_ICR_ALERTCF

Alert flag

Read Write Direction

LL_I2C_DIRECTION_WRITE

Write transfer request by master, slave enters receiver mode.

LL_I2C_DIRECTION_READ

Read transfer request by master, slave enters transmitter mode.

DMA Register Data

LL_I2C_DMA_REG_DATA_TRANSMIT

Get address of data register used for transmission

LL_I2C_DMA_REG_DATA_RECEIVE

Get address of data register used for reception

Start And Stop Generation

LL_I2C_GENERATE_NOSTARTSTOP

Don't Generate Stop and Start condition.

LL_I2C_GENERATE_STOP

Generate Stop condition (Size should be set to 0).

LL_I2C_GENERATE_START_READ

Generate Start for read request.

LL_I2C_GENERATE_START_WRITE

Generate Start for write request.

LL_I2C_GENERATE_RESTART_7BIT_READ

Generate Restart for read request, slave 7Bit address.

LL_I2C_GENERATE_RESTART_7BIT_WRITE

Generate Restart for write request, slave 7Bit address.

LL_I2C_GENERATE_RESTART_10BIT_READ

Generate Restart for read request, slave 10Bit address.

LL_I2C_GENERATE_RESTART_10BIT_WRITE

Generate Restart for write request, slave 10Bit address.

Get Flags Defines

LL_I2C_ISR_TXE

Transmit data register empty

LL_I2C_ISR_TXIS

Transmit interrupt status

LL_I2C_ISR_RXNE

Receive data register not empty

LL_I2C_ISR_ADDR

Address matched (slave mode)

LL_I2C_ISR_NACKF

Not Acknowledge received flag

LL_I2C_ISR_STOPF

Stop detection flag

LL_I2C_ISR_TC

Transfer Complete (master mode)

LL_I2C_ISR_TCR

Transfer Complete Reload

LL_I2C_ISR_BERR

Bus error

LL_I2C_ISR_ARLO

Arbitration lost

LL_I2C_ISR_OVR

Overrun/Underrun (slave mode)

LL_I2C_ISR_PECERR

PEC Error in reception (SMBus mode)

LL_I2C_ISR_TIMEOUT

Timeout detection flag (SMBus mode)

LL_I2C_ISR_ALERT

SMBus alert (SMBus mode)

LL_I2C_ISR_BUSY

Bus busy

Acknowledge Generation

LL_I2C_ACK

ACK is sent after current received byte.

LL_I2C_NACK

NACK is sent after current received byte.

IT Defines

LL_I2C_CR1_TXIE

TX Interrupt enable

LL_I2C_CR1_RXIE

RX Interrupt enable

LL_I2C_CR1_ADDRIE

Address match Interrupt enable (slave only)

LL_I2C_CR1_NACKIE

Not acknowledge received Interrupt enable

LL_I2C_CR1_STOPIE

STOP detection Interrupt enable

LL_I2C_CR1_TCIE

Transfer Complete interrupt enable

LL_I2C_CR1_ERRIE

Error interrupts enable

Transfer End Mode

LL_I2C_MODE_RELOAD

Enable I2C Reload mode.

LL_I2C_MODE_AUTOEND

Enable I2C Automatic end mode with no HW PEC comparison.

LL_I2C_MODE_SOFTEND

Enable I2C Software end mode with no HW PEC comparison.

LL_I2C_MODE_SMBUS_RELOAD

Enable SMBUS Automatic end mode with HW PEC comparison.

LL_I2C_MODE_SMBUS_AUTOEND_NO_PEC

Enable SMBUS Automatic end mode with HW PEC comparison.

LL_I2C_MODE_SMBUS_SOFTEND_NO_PEC

Enable SMBUS Software end mode with HW PEC comparison.

LL_I2C_MODE_SMBUS_AUTOEND_WITH_PEC

Enable SMBUS Automatic end mode with HW PEC comparison.

LL_I2C_MODE_SMBUS_SOFTEND_WITH_PEC

Enable SMBUS Software end mode with HW PEC comparison.

Own Address 1 Length

LL_I2C_OWNADDRESS1_7BIT

Own address 1 is a 7-bit address.

LL_I2C_OWNADDRESS1_10BIT

Own address 1 is a 10-bit address.

Own Address 2 Masks

LL_I2C_OWNADDRESS2_NOMASK

Own Address2 No mask.

LL_I2C_OWNADDRESS2_MASK01

Only Address2 bits[7:2] are compared.

LL_I2C_OWNADDRESS2_MASK02

Only Address2 bits[7:3] are compared.

LL_I2C_OWNADDRESS2_MASK03

Only Address2 bits[7:4] are compared.

LL_I2C_OWNADDRESS2_MASK04

Only Address2 bits[7:5] are compared.

LL_I2C_OWNADDRESS2_MASK05

Only Address2 bits[7:6] are compared.

LL_I2C_OWNADDRESS2_MASK06

Only Address2 bits[7] are compared.

LL_I2C_OWNADDRESS2_MASK07

No comparison is done. All Address2 are acknowledged.

Peripheral Mode

LL_I2C_MODE_I2C

I2C Master or Slave mode

LL_I2C_MODE_SMBUS_HOST

SMBus Host address acknowledge

LL_I2C_MODE_SMBUS_DEVICE

SMBus Device default mode (Default address not acknowledge)

LL_I2C_MODE_SMBUS_DEVICE_ARP

SMBus Device Default address acknowledge

Transfer Request Direction

LL_I2C_REQUEST_WRITE

Master request a write transfer.

LL_I2C_REQUEST_READ

Master request a read transfer.

SMBus TimeoutA Mode SCL SDA Timeout

LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW

TimeoutA is used to detect SCL low level timeout.

LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH

TimeoutA is used to detect both SCL and SDA high level timeout.

SMBus Timeout Selection

LL_I2C_SMBUS_TIMEOUTA

TimeoutA enable bit

LL_I2C_SMBUS_TIMEOUTB

TimeoutB (extended clock) enable bit

LL_I2C_SMBUS_ALL_TIMEOUT

TimeoutA and TimeoutB (extended clock) enable bits

Convert SDA SCL timings

__LL_I2C_CONVERT_TIMINGS

Description:

- Configure the SDA setup, hold time and the SCL high, low period.

Parameters:

- __PRESCALER__: This parameter must be a value between Min_Data=0 and Max_Data=0xF.
- __SETUP_TIME__: This parameter must be a value between Min_Data=0 and Max_Data=0xF. (tscldel = (SCLDEL+1)xtpresc)
- __HOLD_TIME__: This parameter must be a value between Min_Data=0 and Max_Data=0xF. (tsdadel = SDADELxtpresc)
- __SCLH_PERIOD__: This parameter must be a value between Min_Data=0 and Max_Data=0xFF. (tsclh = (SCLH+1)xtpresc)
- __SCLL_PERIOD__: This parameter must be a value between Min_Data=0 and Max_Data=0xFF. (tscll = (SCLL+1)xtpresc)

Return value:

- Value: between Min_Data=0 and Max_Data=0xFFFFFFFF

Common Write and read registers Macros

LL_I2C_WriteReg

Description:

- Write a value in I2C register.

Parameters:

- __INSTANCE__: I2C Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_I2C_ReadReg

Description:

- Read a value in I2C register.

Parameters:

- __INSTANCE__: I2C Instance
- __REG__: Register to be read

Return value:

- Register: value

114 LL IWDG Generic Driver

114.1 IWDG Firmware driver API description

The following section lists the various functions of the IWDG library.

114.1.1 Detailed description of functions

LL_IWDG_Enable

Function name

__STATIC_INLINE void LL_IWDG_Enable (IWDG_TypeDef * IWDGx)

Function description

Start the Independent Watchdog.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Notes

- Except if the hardware watchdog option is selected

Reference Manual to LL API cross reference:

- KR KEY LL_IWDG_Enable

LL_IWDG_ReloadCounter

Function name

__STATIC_INLINE void LL_IWDG_ReloadCounter (IWDG_TypeDef * IWDGx)

Function description

Reloads IWDG counter with value defined in the reload register.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- KR KEY LL_IWDG_ReloadCounter

LL_IWDG_EnableWriteAccess

Function name

__STATIC_INLINE void LL_IWDG_EnableWriteAccess (IWDG_TypeDef * IWDGx)

Function description

Enable write access to IWDG_PR, IWDG_RLR and IWDG_WINR registers.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- KR KEY LL_IWDG_EnableWriteAccess

LL_IWDG_DisableWriteAccess

Function name

__STATIC_INLINE void LL_IWDG_DisableWriteAccess (IWDG_TypeDef * IWDGx)

Function description

Disable write access to IWDG_PR, IWDG_RLR and IWDG_WINR registers.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- KR KEY LL_IWDG_DisableWriteAccess

LL_IWDG_SetPrescaler

Function name

__STATIC_INLINE void LL_IWDG_SetPrescaler (IWDG_TypeDef * IWDGx, uint32_t Prescaler)

Function description

Select the prescaler of the IWDG.

Parameters

- **IWDGx:** IWDG Instance
- **Prescaler:** This parameter can be one of the following values:
  - LL_IWDG_PRESCALER_4
  - LL_IWDG_PRESCALER_8
  - LL_IWDG_PRESCALER_16
  - LL_IWDG_PRESCALER_32
  - LL_IWDG_PRESCALER_64
  - LL_IWDG_PRESCALER_128
  - LL_IWDG_PRESCALER_256

Return values

- **None:**

Reference Manual to LL API cross reference:

- PR PR LL_IWDG_SetPrescaler

LL_IWDG_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_IWDG_GetPrescaler (IWDG_TypeDef * IWDGx)

Function description

Get the selected prescaler of the IWDG.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_IWDG_PRESCALER_4
  - LL_IWDG_PRESCALER_8
  - LL_IWDG_PRESCALER_16
  - LL_IWDG_PRESCALER_32
  - LL_IWDG_PRESCALER_64
  - LL_IWDG_PRESCALER_128
  - LL_IWDG_PRESCALER_256

Reference Manual to LL API cross reference:

- PR PR LL_IWDG_GetPrescaler

LL_IWDG_SetReloadCounter

Function name

__STATIC_INLINE void LL_IWDG_SetReloadCounter (IWDG_TypeDef * IWDGx, uint32_t Counter)

Function description

Specify the IWDG down-counter reload value.

Parameters

- **IWDGx:** IWDG Instance
- **Counter:** Value between Min_Data=0 and Max_Data=0x0FFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RLR RL LL_IWDG_SetReloadCounter

LL_IWDG_GetReloadCounter

Function name

__STATIC_INLINE uint32_t LL_IWDG_GetReloadCounter (IWDG_TypeDef * IWDGx)

Function description

Get the specified IWDG down-counter reload value.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **Value:** between Min_Data=0 and Max_Data=0x0FFF

Reference Manual to LL API cross reference:

- RLR RL LL_IWDG_GetReloadCounter

LL_IWDG_SetWindow

Function name

__STATIC_INLINE void LL_IWDG_SetWindow (IWDG_TypeDef * IWDGx, uint32_t Window)

Function description

Specify high limit of the window value to be compared to the down-counter.

Parameters

- **IWDGx:** IWDG Instance
- **Window:** Value between Min_Data=0 and Max_Data=0x0FFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- WINR WIN LL_IWDG_SetWindow

LL_IWDG_GetWindow

Function name

__STATIC_INLINE uint32_t LL_IWDG_GetWindow (IWDG_TypeDef * IWDGx)

Function description

Get the high limit of the window value specified.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **Value:** between Min_Data=0 and Max_Data=0x0FFF

Reference Manual to LL API cross reference:

- WINR WIN LL_IWDG_GetWindow

LL_IWDG_IsActiveFlag_PVU

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsActiveFlag_PVU (IWDG_TypeDef * IWDGx)

Function description

Check if flag Prescaler Value Update is set or not.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR PVU LL_IWDG_IsActiveFlag_PVU

LL_IWDG_IsActiveFlag_RVU

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsActiveFlag_RVU (IWDG_TypeDef * IWDGx)

Function description

Check if flag Reload Value Update is set or not.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR RVU LL_IWDG_IsActiveFlag_RVU

LL_IWDG_IsActiveFlag_WVU

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsActiveFlag_WVU (IWDG_TypeDef * IWDGx)

Function description

Check if flag Window Value Update is set or not.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR WVU LL_IWDG_IsActiveFlag_WVU

LL_IWDG_IsReady

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsReady (IWDG_TypeDef * IWDGx)

Function description

Check if all flags Prescaler, Reload & Window Value Update are reset or not.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **State:** of bits (1 or 0).

Reference Manual to LL API cross reference:

- SR PVU LL_IWDG_IsReady
- SR RVU LL_IWDG_IsReady
- SR WVU LL_IWDG_IsReady

114.2 IWDG Firmware driver defines

The following section lists the various define and macros of the module.

114.2.1 IWDG

IWDG

Get Flags Defines

LL_IWDG_SR_PVU

Watchdog prescaler value update

LL_IWDG_SR_RVU

Watchdog counter reload value update

LL_IWDG_SR_WVU

Watchdog counter window value update

Prescaler Divider

LL_IWDG_PRESCALER_4

Divider by 4

LL_IWDG_PRESCALER_8

Divider by 8

LL_IWDG_PRESCALER_16

Divider by 16

LL_IWDG_PRESCALER_32

Divider by 32

LL_IWDG_PRESCALER_64

Divider by 64

LL_IWDG_PRESCALER_128

Divider by 128

LL_IWDG_PRESCALER_256

Divider by 256

Common Write and read registers Macros

LL_IWDG_WriteReg

Description:

- Write a value in IWDG register.

Parameters:

- __INSTANCE__: IWDG Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_IWDG_ReadReg

Description:

- Read a value in IWDG register.

Parameters:

- __INSTANCE__: IWDG Instance
- __REG__: Register to be read

Return value:

- Register: value

115 LL LPTIM Generic Driver

115.1 LPTIM Firmware driver registers structures

115.1.1 LL_LPTIM_InitTypeDef

LL_LPTIM_InitTypeDef is defined in the stm32h7xx_ll_lptim.h

Data Fields

- ***uint32_t ClockSource***
- ***uint32_t Prescaler***
- ***uint32_t Waveform***
- ***uint32_t Polarity***

Field Documentation

- ***uint32_t LL_LPTIM_InitTypeDef::ClockSource***

  Specifies the source of the clock used by the LPTIM instance. This parameter can be a value of ***LPTIM_LL_EC_CLK_SOURCE***.This feature can be modified afterwards using unitary function **LL_LPTIM_SetClockSource()**.
- ***uint32_t LL_LPTIM_InitTypeDef::Prescaler***

  Specifies the prescaler division ratio. This parameter can be a value of ***LPTIM_LL_EC_PRESCALER***.This feature can be modified afterwards using using unitary function **LL_LPTIM_SetPrescaler()**.
- ***uint32_t LL_LPTIM_InitTypeDef::Waveform***

  Specifies the waveform shape. This parameter can be a value of ***LPTIM_LL_EC_OUTPUT_WAVEFORM***.This feature can be modified afterwards using unitary function **LL_LPTIM_ConfigOutput()**.
- ***uint32_t LL_LPTIM_InitTypeDef::Polarity***

  Specifies waveform polarity. This parameter can be a value of ***LPTIM_LL_EC_OUTPUT_POLARITY***.This feature can be modified afterwards using unitary function **LL_LPTIM_ConfigOutput()**.

115.2 LPTIM Firmware driver API description

The following section lists the various functions of the LPTIM library.

115.2.1 Detailed description of functions

LL_LPTIM_DeInit

Function name

ErrorStatus LL_LPTIM_DeInit (LPTIM_TypeDef * LPTIMx)

Function description

Set LPTIMx registers to their reset values.

Parameters

- **LPTIMx:** LP Timer instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: LPTIMx registers are de-initialized
  - ERROR: invalid LPTIMx instance

LL_LPTIM_StructInit

Function name

void LL_LPTIM_StructInit (LL_LPTIM_InitTypeDef * LPTIM_InitStruct)

Function description

Set each fields of the LPTIM_InitStruct structure to its default value.

Parameters

- **LPTIM_InitStruct:** pointer to a LL_LPTIM_InitTypeDef structure

Return values

- **None:**

LL_LPTIM_Init

Function name

ErrorStatus LL_LPTIM_Init (LPTIM_TypeDef * LPTIMx, const LL_LPTIM_InitTypeDef * LPTIM_InitStruct)

Function description

Configure the LPTIMx peripheral according to the specified parameters.

Parameters

- **LPTIMx:** LP Timer Instance
- **LPTIM_InitStruct:** pointer to a LL_LPTIM_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: LPTIMx instance has been initialized
  - ERROR: LPTIMx instance hasn't been initialized

Notes

- LL_LPTIM_Init can only be called when the LPTIM instance is disabled.
- LPTIMx can be disabled using unitary function LL_LPTIM_Disable().

LL_LPTIM_Disable

Function name

void LL_LPTIM_Disable (LPTIM_TypeDef * LPTIMx)

Function description

Disable the LPTIM instance.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- The following sequence is required to solve LPTIM disable HW limitation. Please check Errata Sheet ES0335 for more details under "MCU may remain stuck in LPTIM interrupt when entering Stop mode" section.

Reference Manual to LL API cross reference:

- CR ENABLE LL_LPTIM_Disable

LL_LPTIM_Enable

Function name

__STATIC_INLINE void LL_LPTIM_Enable (LPTIM_TypeDef * LPTIMx)

Function description

Enable the LPTIM instance.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- After setting the ENABLE bit, a delay of two counter clock is needed before the LPTIM instance is actually enabled.

Reference Manual to LL API cross reference:

- CR ENABLE LL_LPTIM_Enable

LL_LPTIM_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabled (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the LPTIM instance is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR ENABLE LL_LPTIM_IsEnabled

LL_LPTIM_StartCounter

Function name

__STATIC_INLINE void LL_LPTIM_StartCounter (LPTIM_TypeDef * LPTIMx, uint32_t OperatingMode)

Function description

Starts the LPTIM counter in the desired mode.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **OperatingMode:** This parameter can be one of the following values:
  - LL_LPTIM_OPERATING_MODE_CONTINUOUS
  - LL_LPTIM_OPERATING_MODE_ONESHOT

Return values

- **None:**

Notes

- LPTIM instance must be enabled before starting the counter.
- It is possible to change on the fly from One Shot mode to Continuous mode.

Reference Manual to LL API cross reference:

- CR CNTSTRT LL_LPTIM_StartCounter
- CR SNGSTRT LL_LPTIM_StartCounter

LL_LPTIM_EnableResetAfterRead

Function name

__STATIC_INLINE void LL_LPTIM_EnableResetAfterRead (LPTIM_TypeDef * LPTIMx)

Function description

Enable reset after read.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- After calling this function any read access to LPTIM_CNT register will asynchronously reset the LPTIM_CNT register content.

Reference Manual to LL API cross reference:

- CR RSTARE LL_LPTIM_EnableResetAfterRead

LL_LPTIM_DisableResetAfterRead

Function name

__STATIC_INLINE void LL_LPTIM_DisableResetAfterRead (LPTIM_TypeDef * LPTIMx)

Function description

Disable reset after read.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RSTARE LL_LPTIM_DisableResetAfterRead

LL_LPTIM_IsEnabledResetAfterRead

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledResetAfterRead (const LPTIM_TypeDef * LPTIMx)

Function description

Indicate whether the reset after read feature is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR RSTARE LL_LPTIM_IsEnabledResetAfterRead

LL_LPTIM_ResetCounter

Function name

__STATIC_INLINE void LL_LPTIM_ResetCounter (LPTIM_TypeDef * LPTIMx)

Function description

Reset of the LPTIM_CNT counter register (synchronous).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- Due to the synchronous nature of this reset, it only takes place after a synchronization delay of 3 LPTIM core clock cycles (LPTIM core clock may be different from APB clock).
- COUNTRST is automatically cleared by hardware

Reference Manual to LL API cross reference:

- CR COUNTRST LL_LPTIM_ResetCounter
-

LL_LPTIM_SetUpdateMode

Function name

__STATIC_INLINE void LL_LPTIM_SetUpdateMode (LPTIM_TypeDef * LPTIMx, uint32_t UpdateMode)

Function description

Set the LPTIM registers update mode (enable/disable register preload)

Parameters

- **LPTIMx:** Low-Power Timer instance
- **UpdateMode:** This parameter can be one of the following values:
  - LL_LPTIM_UPDATE_MODE_IMMEDIATE
  - LL_LPTIM_UPDATE_MODE_ENDOFPERIOD

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.

Reference Manual to LL API cross reference:

- CFGR PRELOAD LL_LPTIM_SetUpdateMode

LL_LPTIM_GetUpdateMode

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetUpdateMode (const LPTIM_TypeDef * LPTIMx)

Function description

Get the LPTIM registers update mode.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_UPDATE_MODE_IMMEDIATE
  - LL_LPTIM_UPDATE_MODE_ENDOFPERIOD

Reference Manual to LL API cross reference:

- CFGR PRELOAD LL_LPTIM_GetUpdateMode

LL_LPTIM_SetAutoReload

Function name

__STATIC_INLINE void LL_LPTIM_SetAutoReload (LPTIM_TypeDef * LPTIMx, uint32_t AutoReload)

Function description

Set the auto reload value.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **AutoReload:** Value between Min_Data=0x0001 and Max_Data=0xFFFF

Return values

- **None:**

Notes

- The LPTIMx_ARR register content must only be modified when the LPTIM is enabled
- After a write to the LPTIMx_ARR register a new write operation to the same register can only be performed when the previous write operation is completed. Any successive write before the ARROK flag is set, will lead to unpredictable results.
- autoreload value be strictly greater than the compare value.

Reference Manual to LL API cross reference:

- ARR ARR LL_LPTIM_SetAutoReload

LL_LPTIM_GetAutoReload

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetAutoReload (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual auto reload value.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **AutoReload:** Value between Min_Data=0x0001 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- ARR ARR LL_LPTIM_GetAutoReload

LL_LPTIM_SetCompare

Function name

__STATIC_INLINE void LL_LPTIM_SetCompare (LPTIM_TypeDef * LPTIMx, uint32_t CompareValue)

Function description

Set the compare value.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **CompareValue:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Return values

- **None:**

Notes

- After a write to the LPTIMx_CMP register a new write operation to the same register can only be performed when the previous write operation is completed. Any successive write before the CMPOK flag is set, will lead to unpredictable results.

Reference Manual to LL API cross reference:

- CMP CMP LL_LPTIM_SetCompare

LL_LPTIM_GetCompare

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetCompare (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual compare value.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **CompareValue:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- CMP CMP LL_LPTIM_GetCompare

LL_LPTIM_GetCounter

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetCounter (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual counter value.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Counter:** value

Notes

- When the LPTIM instance is running with an asynchronous clock, reading the LPTIMx_CNT register may return unreliable values. So in this case it is necessary to perform two consecutive read accesses and verify that the two returned values are identical.

Reference Manual to LL API cross reference:

- CNT CNT LL_LPTIM_GetCounter

LL_LPTIM_SetCounterMode

Function name

__STATIC_INLINE void LL_LPTIM_SetCounterMode (LPTIM_TypeDef * LPTIMx, uint32_t CounterMode)

Function description

Set the counter mode (selection of the LPTIM counter clock source).

Parameters

- **LPTIMx:** Low-Power Timer instance
- **CounterMode:** This parameter can be one of the following values:
  - LL_LPTIM_COUNTER_MODE_INTERNAL
  - LL_LPTIM_COUNTER_MODE_EXTERNAL

Return values

- **None:**

Notes

- The counter mode can be set only when the LPTIM instance is disabled.

Reference Manual to LL API cross reference:

- CFGR COUNTMODE LL_LPTIM_SetCounterMode

LL_LPTIM_GetCounterMode

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetCounterMode (const LPTIM_TypeDef * LPTIMx)

Function description

Get the counter mode.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_COUNTER_MODE_INTERNAL
  - LL_LPTIM_COUNTER_MODE_EXTERNAL

Reference Manual to LL API cross reference:

- CFGR COUNTMODE LL_LPTIM_GetCounterMode

LL_LPTIM_ConfigOutput

Function name

__STATIC_INLINE void LL_LPTIM_ConfigOutput (LPTIM_TypeDef * LPTIMx, uint32_t Waveform, uint32_t Polarity)

Function description

Configure the LPTIM instance output (LPTIMx_OUT)

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Waveform:** This parameter can be one of the following values:
  - LL_LPTIM_OUTPUT_WAVEFORM_PWM
  - LL_LPTIM_OUTPUT_WAVEFORM_SETONCE
- **Polarity:** This parameter can be one of the following values:
  - LL_LPTIM_OUTPUT_POLARITY_REGULAR
  - LL_LPTIM_OUTPUT_POLARITY_INVERSE

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- Regarding the LPTIM output polarity the change takes effect immediately, so the output default value will change immediately after the polarity is re-configured, even before the timer is enabled.

Reference Manual to LL API cross reference:

- CFGR WAVE LL_LPTIM_ConfigOutput
- CFGR WAVPOL LL_LPTIM_ConfigOutput

LL_LPTIM_SetWaveform

Function name

__STATIC_INLINE void LL_LPTIM_SetWaveform (LPTIM_TypeDef * LPTIMx, uint32_t Waveform)

Function description

Set waveform shape.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Waveform:** This parameter can be one of the following values:
  - LL_LPTIM_OUTPUT_WAVEFORM_PWM
  - LL_LPTIM_OUTPUT_WAVEFORM_SETONCE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR WAVE LL_LPTIM_SetWaveform

LL_LPTIM_GetWaveform

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetWaveform (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual waveform shape.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_OUTPUT_WAVEFORM_PWM
  - LL_LPTIM_OUTPUT_WAVEFORM_SETONCE

Reference Manual to LL API cross reference:

- CFGR WAVE LL_LPTIM_GetWaveform

LL_LPTIM_SetPolarity

Function name

__STATIC_INLINE void LL_LPTIM_SetPolarity (LPTIM_TypeDef * LPTIMx, uint32_t Polarity)

Function description

Set output polarity.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Polarity:** This parameter can be one of the following values:
  - LL_LPTIM_OUTPUT_POLARITY_REGULAR
  - LL_LPTIM_OUTPUT_POLARITY_INVERSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR WAVPOL LL_LPTIM_SetPolarity

LL_LPTIM_GetPolarity

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetPolarity (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual output polarity.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_OUTPUT_POLARITY_REGULAR
  - LL_LPTIM_OUTPUT_POLARITY_INVERSE

Reference Manual to LL API cross reference:

- CFGR WAVPOL LL_LPTIM_GetPolarity

LL_LPTIM_SetPrescaler

Function name

__STATIC_INLINE void LL_LPTIM_SetPrescaler (LPTIM_TypeDef * LPTIMx, uint32_t Prescaler)

Function description

Set actual prescaler division ratio.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Prescaler:** This parameter can be one of the following values:
  - LL_LPTIM_PRESCALER_DIV1
  - LL_LPTIM_PRESCALER_DIV2
  - LL_LPTIM_PRESCALER_DIV4
  - LL_LPTIM_PRESCALER_DIV8
  - LL_LPTIM_PRESCALER_DIV16
  - LL_LPTIM_PRESCALER_DIV32
  - LL_LPTIM_PRESCALER_DIV64
  - LL_LPTIM_PRESCALER_DIV128

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- When the LPTIM is configured to be clocked by an internal clock source and the LPTIM counter is configured to be updated by active edges detected on the LPTIM external Input1, the internal clock provided to the LPTIM must be not be prescaled.

Reference Manual to LL API cross reference:

- CFGR PRESC LL_LPTIM_SetPrescaler

LL_LPTIM_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetPrescaler (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual prescaler division ratio.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_PRESCALER_DIV1
  - LL_LPTIM_PRESCALER_DIV2
  - LL_LPTIM_PRESCALER_DIV4
  - LL_LPTIM_PRESCALER_DIV8
  - LL_LPTIM_PRESCALER_DIV16
  - LL_LPTIM_PRESCALER_DIV32
  - LL_LPTIM_PRESCALER_DIV64
  - LL_LPTIM_PRESCALER_DIV128

Reference Manual to LL API cross reference:

- CFGR PRESC LL_LPTIM_GetPrescaler

LL_LPTIM_SetInput1Src

Function name

__STATIC_INLINE void LL_LPTIM_SetInput1Src (LPTIM_TypeDef * LPTIMx, uint32_t Src)

Function description

Set LPTIM input 1 source (default GPIO).

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Src:** This parameter can be one of the following values:
  - LL_LPTIM_INPUT1_SRC_GPIO
  - LL_LPTIM_INPUT1_SRC_COMP1
  - LL_LPTIM_INPUT1_SRC_COMP2
  - LL_LPTIM_INPUT1_SRC_COMP1_COMP2
  - LL_LPTIM_INPUT1_SRC_SAI4_FS_A
  - LL_LPTIM_INPUT1_SRC_SAI4_FS_B

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 IN1SEL LL_LPTIM_SetInput1Src

LL_LPTIM_SetInput2Src

Function name

__STATIC_INLINE void LL_LPTIM_SetInput2Src (LPTIM_TypeDef * LPTIMx, uint32_t Src)

Function description

Set LPTIM input 2 source (default GPIO).

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Src:** This parameter can be one of the following values:
  - LL_LPTIM_INPUT2_SRC_GPIO
  - LL_LPTIM_INPUT2_SRC_COMP2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 IN2SEL LL_LPTIM_SetInput2Src

LL_LPTIM_EnableTimeout

Function name

__STATIC_INLINE void LL_LPTIM_EnableTimeout (LPTIM_TypeDef * LPTIMx)

Function description

Enable the timeout function.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- The first trigger event will start the timer, any successive trigger event will reset the counter and the timer will restart.
- The timeout value corresponds to the compare value; if no trigger occurs within the expected time frame, the MCU is waked-up by the compare match event.

Reference Manual to LL API cross reference:

- CFGR TIMOUT LL_LPTIM_EnableTimeout

LL_LPTIM_DisableTimeout

Function name

__STATIC_INLINE void LL_LPTIM_DisableTimeout (LPTIM_TypeDef * LPTIMx)

Function description

Disable the timeout function.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- A trigger event arriving when the timer is already started will be ignored.

Reference Manual to LL API cross reference:

- CFGR TIMOUT LL_LPTIM_DisableTimeout

LL_LPTIM_IsEnabledTimeout

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledTimeout (const LPTIM_TypeDef * LPTIMx)

Function description

Indicate whether the timeout function is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR TIMOUT LL_LPTIM_IsEnabledTimeout

LL_LPTIM_TrigSw

Function name

__STATIC_INLINE void LL_LPTIM_TrigSw (LPTIM_TypeDef * LPTIMx)

Function description

Start the LPTIM counter.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.

Reference Manual to LL API cross reference:

- CFGR TRIGEN LL_LPTIM_TrigSw

LL_LPTIM_ConfigTrigger

Function name

__STATIC_INLINE void LL_LPTIM_ConfigTrigger (LPTIM_TypeDef * LPTIMx, uint32_t Source, uint32_t Filter, uint32_t Polarity)

Function description

Configure the external trigger used as a trigger event for the LPTIM.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Source:** This parameter can be one of the following values:
  - LL_LPTIM_TRIG_SOURCE_GPIO
  - LL_LPTIM_TRIG_SOURCE_RTCALARMA
  - LL_LPTIM_TRIG_SOURCE_RTCALARMB
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP1
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP2
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP3
  - LL_LPTIM_TRIG_SOURCE_COMP1
  - LL_LPTIM_TRIG_SOURCE_COMP2
  - LL_LPTIM_TRIG_SOURCE_LPTIM2 (*)
  - LL_LPTIM_TRIG_SOURCE_LPTIM3 (*)
  - LL_LPTIM_TRIG_SOURCE_LPTIM4 (*)
  - LL_LPTIM_TRIG_SOURCE_LPTIM5 (*)
  - LL_LPTIM_TRIG_SOURCE_SAI1_FS_A (*)
  - LL_LPTIM_TRIG_SOURCE_SAI1_FS_B (*)
  - LL_LPTIM_TRIG_SOURCE_SAI2_FS_A (*)
  - LL_LPTIM_TRIG_SOURCE_SAI2_FS_B (*)
  - LL_LPTIM_TRIG_SOURCE_SAI4_FS_A (*)
  - LL_LPTIM_TRIG_SOURCE_SAI4_FS_B (*)
  - LL_LPTIM_TRIG_SOURCE_DFSDM2_BRK (*)

  (*) Value not defined in all devices.
-
- **Filter:** This parameter can be one of the following values:
  - LL_LPTIM_TRIG_FILTER_NONE
  - LL_LPTIM_TRIG_FILTER_2
  - LL_LPTIM_TRIG_FILTER_4
  - LL_LPTIM_TRIG_FILTER_8
- **Polarity:** This parameter can be one of the following values:
  - LL_LPTIM_TRIG_POLARITY_RISING
  - LL_LPTIM_TRIG_POLARITY_FALLING
  - LL_LPTIM_TRIG_POLARITY_RISING_FALLING

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- An internal clock source must be present when a digital filter is required for the trigger.

Reference Manual to LL API cross reference:

- CFGR TRIGSEL LL_LPTIM_ConfigTrigger
- CFGR TRGFLT LL_LPTIM_ConfigTrigger
- CFGR TRIGEN LL_LPTIM_ConfigTrigger

LL_LPTIM_GetTriggerSource

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetTriggerSource (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual external trigger source.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_TRIG_SOURCE_GPIO
  - LL_LPTIM_TRIG_SOURCE_RTCALARMA
  - LL_LPTIM_TRIG_SOURCE_RTCALARMB
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP1
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP2
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP3
  - LL_LPTIM_TRIG_SOURCE_COMP1
  - LL_LPTIM_TRIG_SOURCE_COMP2
  - LL_LPTIM_TRIG_SOURCE_LPTIM2 (*)
  - LL_LPTIM_TRIG_SOURCE_LPTIM3 (*)
  - LL_LPTIM_TRIG_SOURCE_LPTIM4 (*)
  - LL_LPTIM_TRIG_SOURCE_LPTIM5 (*)
  - LL_LPTIM_TRIG_SOURCE_SAI1_FS_A (*)
  - LL_LPTIM_TRIG_SOURCE_SAI1_FS_B (*)
  - LL_LPTIM_TRIG_SOURCE_SAI2_FS_A (*)
  - LL_LPTIM_TRIG_SOURCE_SAI2_FS_B (*)
  - LL_LPTIM_TRIG_SOURCE_SAI4_FS_A (*)
  - LL_LPTIM_TRIG_SOURCE_SAI4_FS_B (*)
  - LL_LPTIM_TRIG_SOURCE_DFSDM2_BRK (*)

  (*) Value not defined in all devices.
-

Reference Manual to LL API cross reference:

- CFGR TRIGSEL LL_LPTIM_GetTriggerSource

LL_LPTIM_GetTriggerFilter

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetTriggerFilter (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual external trigger filter.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_TRIG_FILTER_NONE
  - LL_LPTIM_TRIG_FILTER_2
  - LL_LPTIM_TRIG_FILTER_4
  - LL_LPTIM_TRIG_FILTER_8

Reference Manual to LL API cross reference:

- CFGR TRGFLT LL_LPTIM_GetTriggerFilter

LL_LPTIM_GetTriggerPolarity

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetTriggerPolarity (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual external trigger polarity.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_TRIG_POLARITY_RISING
  - LL_LPTIM_TRIG_POLARITY_FALLING
  - LL_LPTIM_TRIG_POLARITY_RISING_FALLING

Reference Manual to LL API cross reference:

- CFGR TRIGEN LL_LPTIM_GetTriggerPolarity

LL_LPTIM_SetClockSource

Function name

__STATIC_INLINE void LL_LPTIM_SetClockSource (LPTIM_TypeDef * LPTIMx, uint32_t ClockSource)

Function description

Set the source of the clock used by the LPTIM instance.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **ClockSource:** This parameter can be one of the following values:
  - LL_LPTIM_CLK_SOURCE_INTERNAL
  - LL_LPTIM_CLK_SOURCE_EXTERNAL

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.

Reference Manual to LL API cross reference:

- CFGR CKSEL LL_LPTIM_SetClockSource

LL_LPTIM_GetClockSource

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetClockSource (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual LPTIM instance clock source.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_CLK_SOURCE_INTERNAL
  - LL_LPTIM_CLK_SOURCE_EXTERNAL

Reference Manual to LL API cross reference:

- CFGR CKSEL LL_LPTIM_GetClockSource

LL_LPTIM_ConfigClock

Function name

__STATIC_INLINE void LL_LPTIM_ConfigClock (LPTIM_TypeDef * LPTIMx, uint32_t ClockFilter, uint32_t ClockPolarity)

Function description

Configure the active edge or edges used by the counter when the LPTIM is clocked by an external clock source.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **ClockFilter:** This parameter can be one of the following values:
  - LL_LPTIM_CLK_FILTER_NONE
  - LL_LPTIM_CLK_FILTER_2
  - LL_LPTIM_CLK_FILTER_4
  - LL_LPTIM_CLK_FILTER_8
- **ClockPolarity:** This parameter can be one of the following values:
  - LL_LPTIM_CLK_POLARITY_RISING
  - LL_LPTIM_CLK_POLARITY_FALLING
  - LL_LPTIM_CLK_POLARITY_RISING_FALLING

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- When both external clock signal edges are considered active ones, the LPTIM must also be clocked by an internal clock source with a frequency equal to at least four times the external clock frequency.
- An internal clock source must be present when a digital filter is required for external clock.

Reference Manual to LL API cross reference:

- CFGR CKFLT LL_LPTIM_ConfigClock
- CFGR CKPOL LL_LPTIM_ConfigClock

LL_LPTIM_GetClockPolarity

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetClockPolarity (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual clock polarity.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_CLK_POLARITY_RISING
  - LL_LPTIM_CLK_POLARITY_FALLING
  - LL_LPTIM_CLK_POLARITY_RISING_FALLING

Reference Manual to LL API cross reference:

- CFGR CKPOL LL_LPTIM_GetClockPolarity

LL_LPTIM_GetClockFilter

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetClockFilter (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual clock digital filter.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_CLK_FILTER_NONE
  - LL_LPTIM_CLK_FILTER_2
  - LL_LPTIM_CLK_FILTER_4
  - LL_LPTIM_CLK_FILTER_8

Reference Manual to LL API cross reference:

- CFGR CKFLT LL_LPTIM_GetClockFilter

LL_LPTIM_SetEncoderMode

Function name

__STATIC_INLINE void LL_LPTIM_SetEncoderMode (LPTIM_TypeDef * LPTIMx, uint32_t EncoderMode)

Function description

Configure the encoder mode.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **EncoderMode:** This parameter can be one of the following values:
  - LL_LPTIM_ENCODER_MODE_RISING
  - LL_LPTIM_ENCODER_MODE_FALLING
  - LL_LPTIM_ENCODER_MODE_RISING_FALLING

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.

Reference Manual to LL API cross reference:

- CFGR CKPOL LL_LPTIM_SetEncoderMode

LL_LPTIM_GetEncoderMode

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetEncoderMode (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual encoder mode.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_ENCODER_MODE_RISING
  - LL_LPTIM_ENCODER_MODE_FALLING
  - LL_LPTIM_ENCODER_MODE_RISING_FALLING

Reference Manual to LL API cross reference:

- CFGR CKPOL LL_LPTIM_GetEncoderMode

LL_LPTIM_EnableEncoderMode

Function name

__STATIC_INLINE void LL_LPTIM_EnableEncoderMode (LPTIM_TypeDef * LPTIMx)

Function description

Enable the encoder mode.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- In this mode the LPTIM instance must be clocked by an internal clock source. Also, the prescaler division ratio must be equal to 1.
- LPTIM instance must be configured in continuous mode prior enabling the encoder mode.

Reference Manual to LL API cross reference:

- CFGR ENC LL_LPTIM_EnableEncoderMode

LL_LPTIM_DisableEncoderMode

Function name

__STATIC_INLINE void LL_LPTIM_DisableEncoderMode (LPTIM_TypeDef * LPTIMx)

Function description

Disable the encoder mode.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.

Reference Manual to LL API cross reference:

- CFGR ENC LL_LPTIM_DisableEncoderMode

LL_LPTIM_IsEnabledEncoderMode

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledEncoderMode (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the LPTIM operates in encoder mode.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR ENC LL_LPTIM_IsEnabledEncoderMode

LL_LPTIM_ClearFlag_CMPM

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_CMPM (LPTIM_TypeDef * LPTIMx)

Function description

Clear the compare match flag (CMPMCF)

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CMPMCF LL_LPTIM_ClearFlag_CMPM

LL_LPTIM_IsActiveFlag_CMPM

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_CMPM (const LPTIM_TypeDef * LPTIMx)

Function description

Inform application whether a compare match interrupt has occurred.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CMPM LL_LPTIM_IsActiveFlag_CMPM

LL_LPTIM_ClearFlag_ARRM

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_ARRM (LPTIM_TypeDef * LPTIMx)

Function description

Clear the autoreload match flag (ARRMCF)

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ARRMCF LL_LPTIM_ClearFlag_ARRM

LL_LPTIM_IsActiveFlag_ARRM

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_ARRM (const LPTIM_TypeDef * LPTIMx)

Function description

Inform application whether a autoreload match interrupt has occurred.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ARRM LL_LPTIM_IsActiveFlag_ARRM

LL_LPTIM_ClearFlag_EXTTRIG

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_EXTTRIG (LPTIM_TypeDef * LPTIMx)

Function description

Clear the external trigger valid edge flag(EXTTRIGCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR EXTTRIGCF LL_LPTIM_ClearFlag_EXTTRIG

LL_LPTIM_IsActiveFlag_EXTTRIG

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_EXTTRIG (const LPTIM_TypeDef * LPTIMx)

Function description

Inform application whether a valid edge on the selected external trigger input has occurred.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR EXTTRIG LL_LPTIM_IsActiveFlag_EXTTRIG

LL_LPTIM_ClearFlag_CMPOK

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_CMPOK (LPTIM_TypeDef * LPTIMx)

Function description

Clear the compare register update interrupt flag (CMPOKCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CMPOKCF LL_LPTIM_ClearFlag_CMPOK

LL_LPTIM_IsActiveFlag_CMPOK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_CMPOK (const LPTIM_TypeDef * LPTIMx)

Function description

Informs application whether the APB bus write operation to the LPTIMx_CMP register has been successfully completed.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CMPOK LL_LPTIM_IsActiveFlag_CMPOK

LL_LPTIM_ClearFlag_ARROK

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_ARROK (LPTIM_TypeDef * LPTIMx)

Function description

Clear the autoreload register update interrupt flag (ARROKCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ARROKCF LL_LPTIM_ClearFlag_ARROK

LL_LPTIM_IsActiveFlag_ARROK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_ARROK (const LPTIM_TypeDef * LPTIMx)

Function description

Informs application whether the APB bus write operation to the LPTIMx_ARR register has been successfully completed.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ARROK LL_LPTIM_IsActiveFlag_ARROK

LL_LPTIM_ClearFlag_UP

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_UP (LPTIM_TypeDef * LPTIMx)

Function description

Clear the counter direction change to up interrupt flag (UPCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR UPCF LL_LPTIM_ClearFlag_UP

LL_LPTIM_IsActiveFlag_UP

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_UP (const LPTIM_TypeDef * LPTIMx)

Function description

Informs the application whether the counter direction has changed from down to up (when the LPTIM instance operates in encoder mode).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR UP LL_LPTIM_IsActiveFlag_UP

LL_LPTIM_ClearFlag_DOWN

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_DOWN (LPTIM_TypeDef * LPTIMx)

Function description

Clear the counter direction change to down interrupt flag (DOWNCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR DOWNCF LL_LPTIM_ClearFlag_DOWN

LL_LPTIM_IsActiveFlag_DOWN

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_DOWN (const LPTIM_TypeDef * LPTIMx)

Function description

Informs the application whether the counter direction has changed from up to down (when the LPTIM instance operates in encoder mode).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR DOWN LL_LPTIM_IsActiveFlag_DOWN

LL_LPTIM_EnableIT_CMPM

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_CMPM (LPTIM_TypeDef * LPTIMx)

Function description

Enable compare match interrupt (CMPMIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CMPMIE LL_LPTIM_EnableIT_CMPM

LL_LPTIM_DisableIT_CMPM

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_CMPM (LPTIM_TypeDef * LPTIMx)

Function description

Disable compare match interrupt (CMPMIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CMPMIE LL_LPTIM_DisableIT_CMPM

LL_LPTIM_IsEnabledIT_CMPM

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_CMPM (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the compare match interrupt (CMPMIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER CMPMIE LL_LPTIM_IsEnabledIT_CMPM

LL_LPTIM_EnableIT_ARRM

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_ARRM (LPTIM_TypeDef * LPTIMx)

Function description

Enable autoreload match interrupt (ARRMIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ARRMIE LL_LPTIM_EnableIT_ARRM

LL_LPTIM_DisableIT_ARRM

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_ARRM (LPTIM_TypeDef * LPTIMx)

Function description

Disable autoreload match interrupt (ARRMIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ARRMIE LL_LPTIM_DisableIT_ARRM

LL_LPTIM_IsEnabledIT_ARRM

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_ARRM (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the autoreload match interrupt (ARRMIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER ARRMIE LL_LPTIM_IsEnabledIT_ARRM

LL_LPTIM_EnableIT_EXTTRIG

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_EXTTRIG (LPTIM_TypeDef * LPTIMx)

Function description

Enable external trigger valid edge interrupt (EXTTRIGIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EXTTRIGIE LL_LPTIM_EnableIT_EXTTRIG

LL_LPTIM_DisableIT_EXTTRIG

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_EXTTRIG (LPTIM_TypeDef * LPTIMx)

Function description

Disable external trigger valid edge interrupt (EXTTRIGIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EXTTRIGIE LL_LPTIM_DisableIT_EXTTRIG

LL_LPTIM_IsEnabledIT_EXTTRIG

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_EXTTRIG (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates external trigger valid edge interrupt (EXTTRIGIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER EXTTRIGIE LL_LPTIM_IsEnabledIT_EXTTRIG

LL_LPTIM_EnableIT_CMPOK

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_CMPOK (LPTIM_TypeDef * LPTIMx)

Function description

Enable compare register write completed interrupt (CMPOKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CMPOKIE LL_LPTIM_EnableIT_CMPOK

LL_LPTIM_DisableIT_CMPOK

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_CMPOK (LPTIM_TypeDef * LPTIMx)

Function description

Disable compare register write completed interrupt (CMPOKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CMPOKIE LL_LPTIM_DisableIT_CMPOK

LL_LPTIM_IsEnabledIT_CMPOK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_CMPOK (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the compare register write completed interrupt (CMPOKIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER CMPOKIE LL_LPTIM_IsEnabledIT_CMPOK

LL_LPTIM_EnableIT_ARROK

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_ARROK (LPTIM_TypeDef * LPTIMx)

Function description

Enable autoreload register write completed interrupt (ARROKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ARROKIE LL_LPTIM_EnableIT_ARROK

LL_LPTIM_DisableIT_ARROK

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_ARROK (LPTIM_TypeDef * LPTIMx)

Function description

Disable autoreload register write completed interrupt (ARROKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ARROKIE LL_LPTIM_DisableIT_ARROK

LL_LPTIM_IsEnabledIT_ARROK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_ARROK (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the autoreload register write completed interrupt (ARROKIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit(1 or 0).

Reference Manual to LL API cross reference:

- IER ARROKIE LL_LPTIM_IsEnabledIT_ARROK

LL_LPTIM_EnableIT_UP

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_UP (LPTIM_TypeDef * LPTIMx)

Function description

Enable direction change to up interrupt (UPIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER UPIE LL_LPTIM_EnableIT_UP

LL_LPTIM_DisableIT_UP

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_UP (LPTIM_TypeDef * LPTIMx)

Function description

Disable direction change to up interrupt (UPIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER UPIE LL_LPTIM_DisableIT_UP

LL_LPTIM_IsEnabledIT_UP

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_UP (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the direction change to up interrupt (UPIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit(1 or 0).

Reference Manual to LL API cross reference:

- IER UPIE LL_LPTIM_IsEnabledIT_UP

LL_LPTIM_EnableIT_DOWN

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_DOWN (LPTIM_TypeDef * LPTIMx)

Function description

Enable direction change to down interrupt (DOWNIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER DOWNIE LL_LPTIM_EnableIT_DOWN

LL_LPTIM_DisableIT_DOWN

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_DOWN (LPTIM_TypeDef * LPTIMx)

Function description

Disable direction change to down interrupt (DOWNIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER DOWNIE LL_LPTIM_DisableIT_DOWN

LL_LPTIM_IsEnabledIT_DOWN

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_DOWN (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the direction change to down interrupt (DOWNIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit(1 or 0).

Reference Manual to LL API cross reference:

- IER DOWNIE LL_LPTIM_IsEnabledIT_DOWN

115.3 LPTIM Firmware driver defines

The following section lists the various define and macros of the module.

115.3.1 LPTIM

LPTIM

Input1 Source

LL_LPTIM_INPUT1_SRC_GPIO

For LPTIM1 and LPTIM2

LL_LPTIM_INPUT1_SRC_COMP1

For LPTIM1 and LPTIM2

LL_LPTIM_INPUT1_SRC_COMP2

For LPTIM2

LL_LPTIM_INPUT1_SRC_COMP1_COMP2

For LPTIM2

LL_LPTIM_INPUT1_SRC_SAI4_FS_A

For LPTIM3

LL_LPTIM_INPUT1_SRC_SAI4_FS_B

For LPTIM3

Input2 Source

LL_LPTIM_INPUT2_SRC_GPIO

For LPTIM1

LL_LPTIM_INPUT2_SRC_COMP2

For LPTIM1

Clock Filter

LL_LPTIM_CLK_FILTER_NONE

Any external clock signal level change is considered as a valid transition

LL_LPTIM_CLK_FILTER_2

External clock signal level change must be stable for at least 2 clock periods before it is considered as valid transition

LL_LPTIM_CLK_FILTER_4

External clock signal level change must be stable for at least 4 clock periods before it is considered as valid transition

LL_LPTIM_CLK_FILTER_8

External clock signal level change must be stable for at least 8 clock periods before it is considered as valid transition

Clock Polarity

LL_LPTIM_CLK_POLARITY_RISING

The rising edge is the active edge used for counting

LL_LPTIM_CLK_POLARITY_FALLING

The falling edge is the active edge used for counting

LL_LPTIM_CLK_POLARITY_RISING_FALLING

Both edges are active edges

Clock Source

LL_LPTIM_CLK_SOURCE_INTERNAL

LPTIM is clocked by internal clock source (APB clock or any of the embedded oscillators)

LL_LPTIM_CLK_SOURCE_EXTERNAL

LPTIM is clocked by an external clock source through the LPTIM external Input1

Counter Mode

LL_LPTIM_COUNTER_MODE_INTERNAL

The counter is incremented following each internal clock pulse

LL_LPTIM_COUNTER_MODE_EXTERNAL

The counter is incremented following each valid clock pulse on the LPTIM external Input1

Encoder Mode

LL_LPTIM_ENCODER_MODE_RISING

The rising edge is the active edge used for counting

LL_LPTIM_ENCODER_MODE_FALLING

The falling edge is the active edge used for counting

LL_LPTIM_ENCODER_MODE_RISING_FALLING

Both edges are active edges

Get Flags Defines

LL_LPTIM_ISR_CMPM

Compare match

LL_LPTIM_ISR_CMPOK

Compare register update OK

LL_LPTIM_ISR_ARRM

Autoreload match

LL_LPTIM_ISR_EXTTRIG

External trigger edge event

LL_LPTIM_ISR_ARROK

Autoreload register update OK

LL_LPTIM_ISR_UP

Counter direction change down to up

LL_LPTIM_ISR_DOWN

Counter direction change up to down

IT Defines

LL_LPTIM_IER_CMPMIE

Compare match

LL_LPTIM_IER_CMPOKIE

Compare register update OK

LL_LPTIM_IER_ARRMIE

Autoreload match

LL_LPTIM_IER_EXTTRIGIE

External trigger edge event

LL_LPTIM_IER_ARROKIE

Autoreload register update OK

LL_LPTIM_IER_UPIE

Counter direction change down to up

LL_LPTIM_IER_DOWNIE

Counter direction change up to down

Operating Mode

LL_LPTIM_OPERATING_MODE_CONTINUOUS

LP Timer starts in continuous mode

LL_LPTIM_OPERATING_MODE_ONESHOT

LP Tilmer starts in single mode

Output Polarity

LL_LPTIM_OUTPUT_POLARITY_REGULAR

The LPTIM output reflects the compare results between LPTIMx_ARR and LPTIMx_CMP registers

LL_LPTIM_OUTPUT_POLARITY_INVERSE

The LPTIM output reflects the inverse of the compare results between LPTIMx_ARR and LPTIMx_CMP registers

Output Waveform Type

LL_LPTIM_OUTPUT_WAVEFORM_PWM

LPTIM generates either a PWM waveform or a One pulse waveform depending on chosen operating mode CONTINUOUS or SINGLE

LL_LPTIM_OUTPUT_WAVEFORM_SETONCE

LPTIM generates a Set Once waveform

Prescaler Value

LL_LPTIM_PRESCALER_DIV1

Prescaler division factor is set to 1

LL_LPTIM_PRESCALER_DIV2

Prescaler division factor is set to 2

LL_LPTIM_PRESCALER_DIV4

Prescaler division factor is set to 4

LL_LPTIM_PRESCALER_DIV8

Prescaler division factor is set to 8

LL_LPTIM_PRESCALER_DIV16

Prescaler division factor is set to 16

LL_LPTIM_PRESCALER_DIV32

Prescaler division factor is set to 32

LL_LPTIM_PRESCALER_DIV64

Prescaler division factor is set to 64

LL_LPTIM_PRESCALER_DIV128

Prescaler division factor is set to 128

Trigger Filter

LL_LPTIM_TRIG_FILTER_NONE

Any trigger active level change is considered as a valid trigger

LL_LPTIM_TRIG_FILTER_2

Trigger active level change must be stable for at least 2 clock periods before it is considered as valid trigger

LL_LPTIM_TRIG_FILTER_4

Trigger active level change must be stable for at least 4 clock periods before it is considered as valid trigger

LL_LPTIM_TRIG_FILTER_8

Trigger active level change must be stable for at least 8 clock periods before it is considered as valid trigger

Trigger Polarity

LL_LPTIM_TRIG_POLARITY_RISING

LPTIM counter starts when a rising edge is detected

LL_LPTIM_TRIG_POLARITY_FALLING

LPTIM counter starts when a falling edge is detected

LL_LPTIM_TRIG_POLARITY_RISING_FALLING

LPTIM counter starts when a rising or a falling edge is detected

Trigger Source

LL_LPTIM_TRIG_SOURCE_GPIO

External input trigger is connected to TIMx_ETR input

LL_LPTIM_TRIG_SOURCE_RTCALARMA

External input trigger is connected to RTC Alarm A

LL_LPTIM_TRIG_SOURCE_RTCALARMB

External input trigger is connected to RTC Alarm B

LL_LPTIM_TRIG_SOURCE_RTCTAMP1

External input trigger is connected to RTC Tamper 1

LL_LPTIM_TRIG_SOURCE_RTCTAMP2

External input trigger is connected to RTC Tamper 2

LL_LPTIM_TRIG_SOURCE_RTCTAMP3

External input trigger is connected to RTC Tamper 3

LL_LPTIM_TRIG_SOURCE_COMP1

External input trigger is connected to COMP1 output

LL_LPTIM_TRIG_SOURCE_COMP2

External input trigger is connected to COMP2 output

LL_LPTIM_TRIG_SOURCE_LPTIM2

External input trigger is connected to LPTIM2 output

LL_LPTIM_TRIG_SOURCE_LPTIM3

External input trigger is connected to LPTIM3 output

LL_LPTIM_TRIG_SOURCE_LPTIM4

External input trigger is connected to LPTIM4 output

LL_LPTIM_TRIG_SOURCE_LPTIM5

External input trigger is connected to LPTIM5 output

LL_LPTIM_TRIG_SOURCE_SAI1_FS_A

External input trigger is connected to SAI1 FS A output

LL_LPTIM_TRIG_SOURCE_SAI1_FS_B

External input trigger is connected to SAI1 FS B output

LL_LPTIM_TRIG_SOURCE_SAI2_FS_A

External input trigger is connected to SAI2 FS A output

LL_LPTIM_TRIG_SOURCE_SAI2_FS_B

External input trigger is connected to SAI2 FS B output

LL_LPTIM_TRIG_SOURCE_SAI4_FS_A

External input trigger is connected to SAI4 FS A output

LL_LPTIM_TRIG_SOURCE_SAI4_FS_B

External input trigger is connected to SAI4 FS B output

LL_LPTIM_TRIG_SOURCE_DFSDM2_BRK

External input trigger is connected to DFSDM2_BRK[0]

Update Mode

LL_LPTIM_UPDATE_MODE_IMMEDIATE

Preload is disabled: registers are updated after each APB bus write access

LL_LPTIM_UPDATE_MODE_ENDOFPERIOD

preload is enabled: registers are updated at the end of the current LPTIM period

Common Write and read registers Macros

LL_LPTIM_WriteReg

Description:

- Write a value in LPTIM register.

Parameters:

- __INSTANCE__: LPTIM Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_LPTIM_ReadReg

Description:

- Read a value in LPTIM register.

Parameters:

- __INSTANCE__: LPTIM Instance
- __REG__: Register to be read

Return value:

- Register: value

116 LL LPUART Generic Driver

116.1 LPUART Firmware driver registers structures

116.1.1 LL_LPUART_InitTypeDef

LL_LPUART_InitTypeDef is defined in the stm32h7xx_ll_lpuart.h

Data Fields

- ***uint32_t PrescalerValue***
- ***uint32_t BaudRate***
- ***uint32_t DataWidth***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t TransferDirection***
- ***uint32_t HardwareFlowControl***

Field Documentation

- ***uint32_t LL_LPUART_InitTypeDef::PrescalerValue***

  Specifies the Prescaler to compute the communication baud rate. This parameter can be a value of ***LPUART_LL_EC_PRESCALER***.This feature can be modified afterwards using unitary function **LL_LPUART_SetPrescaler()**.
- ***uint32_t LL_LPUART_InitTypeDef::BaudRate***

  This field defines expected LPUART communication baud rate.This feature can be modified afterwards using unitary function **LL_LPUART_SetBaudRate()**.
- ***uint32_t LL_LPUART_InitTypeDef::DataWidth***

  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***LPUART_LL_EC_DATAWIDTH***.This feature can be modified afterwards using unitary function **LL_LPUART_SetDataWidth()**.
- ***uint32_t LL_LPUART_InitTypeDef::StopBits***

  Specifies the number of stop bits transmitted. This parameter can be a value of ***LPUART_LL_EC_STOPBITS***.This feature can be modified afterwards using unitary function **LL_LPUART_SetStopBitsLength()**.
- ***uint32_t LL_LPUART_InitTypeDef::Parity***

  Specifies the parity mode. This parameter can be a value of ***LPUART_LL_EC_PARITY***.This feature can be modified afterwards using unitary function **LL_LPUART_SetParity()**.
- ***uint32_t LL_LPUART_InitTypeDef::TransferDirection***

  Specifies whether the Receive and/or Transmit mode is enabled or disabled. This parameter can be a value of ***LPUART_LL_EC_DIRECTION***.This feature can be modified afterwards using unitary function **LL_LPUART_SetTransferDirection()**.
- ***uint32_t LL_LPUART_InitTypeDef::HardwareFlowControl***

  Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of ***LPUART_LL_EC_HWCONTROL***.This feature can be modified afterwards using unitary function **LL_LPUART_SetHWFlowCtrl()**.

116.2 LPUART Firmware driver API description

The following section lists the various functions of the LPUART library.

116.2.1 Detailed description of functions

LL_LPUART_Enable

Function name

__STATIC_INLINE void LL_LPUART_Enable (USART_TypeDef * LPUARTx)

Function description

LPUART Enable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 UE LL_LPUART_Enable

LL_LPUART_Disable

Function name

__STATIC_INLINE void LL_LPUART_Disable (USART_TypeDef * LPUARTx)

Function description

LPUART Disable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- When LPUART is disabled, LPUART prescalers and outputs are stopped immediately, and current operations are discarded. The configuration of the LPUART is kept, but all the status flags, in the LPUARTx_ISR are set to their default values.
- In order to go into low-power mode without generating errors on the line, the TE bit must be reset before and the software must wait for the TC bit in the LPUART_ISR to be set before resetting the UE bit. The DMA requests are also reset when UE = 0 so the DMA channel must be disabled before resetting the UE bit.

Reference Manual to LL API cross reference:

- CR1 UE LL_LPUART_Disable

LL_LPUART_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabled (const USART_TypeDef * LPUARTx)

Function description

Indicate if LPUART is enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 UE LL_LPUART_IsEnabled

LL_LPUART_EnableFIFO

Function name

__STATIC_INLINE void LL_LPUART_EnableFIFO (USART_TypeDef * LPUARTx)

Function description

FIFO Mode Enable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_LPUART_EnableFIFO

LL_LPUART_DisableFIFO

Function name

__STATIC_INLINE void LL_LPUART_DisableFIFO (USART_TypeDef * LPUARTx)

Function description

FIFO Mode Disable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_LPUART_DisableFIFO

LL_LPUART_IsEnabledFIFO

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledFIFO (const USART_TypeDef * LPUARTx)

Function description

Indicate if FIFO Mode is enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_LPUART_IsEnabledFIFO

LL_LPUART_SetTXFIFOThreshold

Function name

__STATIC_INLINE void LL_LPUART_SetTXFIFOThreshold (USART_TypeDef * LPUARTx, uint32_t Threshold)

Function description

Configure TX FIFO Threshold.

Parameters

- **LPUARTx:** LPUART Instance
- **Threshold:** This parameter can be one of the following values:
  - LL_LPUART_FIFOTHRESHOLD_1_8
  - LL_LPUART_FIFOTHRESHOLD_1_4
  - LL_LPUART_FIFOTHRESHOLD_1_2
  - LL_LPUART_FIFOTHRESHOLD_3_4
  - LL_LPUART_FIFOTHRESHOLD_7_8
  - LL_LPUART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_LPUART_SetTXFIFOThreshold

LL_LPUART_GetTXFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTXFIFOThreshold (const USART_TypeDef * LPUARTx)

Function description

Return TX FIFO Threshold Configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_FIFOTHRESHOLD_1_8
  - LL_LPUART_FIFOTHRESHOLD_1_4
  - LL_LPUART_FIFOTHRESHOLD_1_2
  - LL_LPUART_FIFOTHRESHOLD_3_4
  - LL_LPUART_FIFOTHRESHOLD_7_8
  - LL_LPUART_FIFOTHRESHOLD_8_8

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_LPUART_GetTXFIFOThreshold

LL_LPUART_SetRXFIFOThreshold

Function name

__STATIC_INLINE void LL_LPUART_SetRXFIFOThreshold (USART_TypeDef * LPUARTx, uint32_t Threshold)

Function description

Configure RX FIFO Threshold.

Parameters

- **LPUARTx:** LPUART Instance
- **Threshold:** This parameter can be one of the following values:
  - LL_LPUART_FIFOTHRESHOLD_1_8
  - LL_LPUART_FIFOTHRESHOLD_1_4
  - LL_LPUART_FIFOTHRESHOLD_1_2
  - LL_LPUART_FIFOTHRESHOLD_3_4
  - LL_LPUART_FIFOTHRESHOLD_7_8
  - LL_LPUART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RXFTCFG LL_LPUART_SetRXFIFOThreshold

LL_LPUART_GetRXFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetRXFIFOThreshold (const USART_TypeDef * LPUARTx)

Function description

Return RX FIFO Threshold Configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_FIFOTHRESHOLD_1_8
  - LL_LPUART_FIFOTHRESHOLD_1_4
  - LL_LPUART_FIFOTHRESHOLD_1_2
  - LL_LPUART_FIFOTHRESHOLD_3_4
  - LL_LPUART_FIFOTHRESHOLD_7_8
  - LL_LPUART_FIFOTHRESHOLD_8_8

Reference Manual to LL API cross reference:

- CR3 RXFTCFG LL_LPUART_GetRXFIFOThreshold

LL_LPUART_ConfigFIFOsThreshold

Function name

__STATIC_INLINE void LL_LPUART_ConfigFIFOsThreshold (USART_TypeDef * LPUARTx, uint32_t TXThreshold, uint32_t RXThreshold)

Function description

Configure TX and RX FIFOs Threshold.

Parameters

- **LPUARTx:** LPUART Instance
- **TXThreshold:** This parameter can be one of the following values:
  - LL_LPUART_FIFOTHRESHOLD_1_8
  - LL_LPUART_FIFOTHRESHOLD_1_4
  - LL_LPUART_FIFOTHRESHOLD_1_2
  - LL_LPUART_FIFOTHRESHOLD_3_4
  - LL_LPUART_FIFOTHRESHOLD_7_8
  - LL_LPUART_FIFOTHRESHOLD_8_8
- **RXThreshold:** This parameter can be one of the following values:
  - LL_LPUART_FIFOTHRESHOLD_1_8
  - LL_LPUART_FIFOTHRESHOLD_1_4
  - LL_LPUART_FIFOTHRESHOLD_1_2
  - LL_LPUART_FIFOTHRESHOLD_3_4
  - LL_LPUART_FIFOTHRESHOLD_7_8
  - LL_LPUART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_LPUART_ConfigFIFOsThreshold
- CR3 RXFTCFG LL_LPUART_ConfigFIFOsThreshold

LL_LPUART_EnableInStopMode

Function name

__STATIC_INLINE void LL_LPUART_EnableInStopMode (USART_TypeDef * LPUARTx)

Function description

LPUART enabled in STOP Mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- When this function is enabled, LPUART is able to wake up the MCU from Stop mode, provided that LPUART clock selection is HSI or LSE in RCC.

Reference Manual to LL API cross reference:

- CR1 UESM LL_LPUART_EnableInStopMode

LL_LPUART_DisableInStopMode

Function name

__STATIC_INLINE void LL_LPUART_DisableInStopMode (USART_TypeDef * LPUARTx)

Function description

LPUART disabled in STOP Mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- When this function is disabled, LPUART is not able to wake up the MCU from Stop mode

Reference Manual to LL API cross reference:

- CR1 UESM LL_LPUART_DisableInStopMode

LL_LPUART_IsEnabledInStopMode

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledInStopMode (const USART_TypeDef * LPUARTx)

Function description

Indicate if LPUART is enabled in STOP Mode (able to wake up MCU from Stop mode or not)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 UESM LL_LPUART_IsEnabledInStopMode

LL_LPUART_EnableDirectionRx

Function name

__STATIC_INLINE void LL_LPUART_EnableDirectionRx (USART_TypeDef * LPUARTx)

Function description

Receiver Enable (Receiver is enabled and begins searching for a start bit)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_LPUART_EnableDirectionRx

LL_LPUART_DisableDirectionRx

Function name

__STATIC_INLINE void LL_LPUART_DisableDirectionRx (USART_TypeDef * LPUARTx)

Function description

Receiver Disable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_LPUART_DisableDirectionRx

LL_LPUART_EnableDirectionTx

Function name

__STATIC_INLINE void LL_LPUART_EnableDirectionTx (USART_TypeDef * LPUARTx)

Function description

Transmitter Enable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TE LL_LPUART_EnableDirectionTx

LL_LPUART_DisableDirectionTx

Function name

__STATIC_INLINE void LL_LPUART_DisableDirectionTx (USART_TypeDef * LPUARTx)

Function description

Transmitter Disable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TE LL_LPUART_DisableDirectionTx

LL_LPUART_SetTransferDirection

Function name

__STATIC_INLINE void LL_LPUART_SetTransferDirection (USART_TypeDef * LPUARTx, uint32_t TransferDirection)

Function description

Configure simultaneously enabled/disabled states of Transmitter and Receiver.

Parameters

- **LPUARTx:** LPUART Instance
- **TransferDirection:** This parameter can be one of the following values:
  - LL_LPUART_DIRECTION_NONE
  - LL_LPUART_DIRECTION_RX
  - LL_LPUART_DIRECTION_TX
  - LL_LPUART_DIRECTION_TX_RX

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_LPUART_SetTransferDirection
- CR1 TE LL_LPUART_SetTransferDirection

LL_LPUART_GetTransferDirection

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTransferDirection (const USART_TypeDef * LPUARTx)

Function description

Return enabled/disabled states of Transmitter and Receiver.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_DIRECTION_NONE
  - LL_LPUART_DIRECTION_RX
  - LL_LPUART_DIRECTION_TX
  - LL_LPUART_DIRECTION_TX_RX

Reference Manual to LL API cross reference:

- CR1 RE LL_LPUART_GetTransferDirection
- CR1 TE LL_LPUART_GetTransferDirection

LL_LPUART_SetParity

Function name

__STATIC_INLINE void LL_LPUART_SetParity (USART_TypeDef * LPUARTx, uint32_t Parity)

Function description

Configure Parity (enabled/disabled and parity mode if enabled)

Parameters

- **LPUARTx:** LPUART Instance
- **Parity:** This parameter can be one of the following values:
  - LL_LPUART_PARITY_NONE
  - LL_LPUART_PARITY_EVEN
  - LL_LPUART_PARITY_ODD

Return values

- **None:**

Notes

- This function selects if hardware parity control (generation and detection) is enabled or disabled. When the parity control is enabled (Odd or Even), computed parity bit is inserted at the MSB position (depending on data width) and parity is checked on the received data.

Reference Manual to LL API cross reference:

- CR1 PS LL_LPUART_SetParity
- CR1 PCE LL_LPUART_SetParity

LL_LPUART_GetParity

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetParity (const USART_TypeDef * LPUARTx)

Function description

Return Parity configuration (enabled/disabled and parity mode if enabled)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_PARITY_NONE
  - LL_LPUART_PARITY_EVEN
  - LL_LPUART_PARITY_ODD

Reference Manual to LL API cross reference:

- CR1 PS LL_LPUART_GetParity
- CR1 PCE LL_LPUART_GetParity

LL_LPUART_SetWakeUpMethod

Function name

__STATIC_INLINE void LL_LPUART_SetWakeUpMethod (USART_TypeDef * LPUARTx, uint32_t Method)

Function description

Set Receiver Wake Up method from Mute mode.

Parameters

- **LPUARTx:** LPUART Instance
- **Method:** This parameter can be one of the following values:
  - LL_LPUART_WAKEUP_IDLELINE
  - LL_LPUART_WAKEUP_ADDRESSMARK

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 WAKE LL_LPUART_SetWakeUpMethod

LL_LPUART_GetWakeUpMethod

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetWakeUpMethod (const USART_TypeDef * LPUARTx)

Function description

Return Receiver Wake Up method from Mute mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_WAKEUP_IDLELINE
  - LL_LPUART_WAKEUP_ADDRESSMARK

Reference Manual to LL API cross reference:

- CR1 WAKE LL_LPUART_GetWakeUpMethod

LL_LPUART_SetDataWidth

Function name

__STATIC_INLINE void LL_LPUART_SetDataWidth (USART_TypeDef * LPUARTx, uint32_t DataWidth)

Function description

Set Word length (nb of data bits, excluding start and stop bits)

Parameters

- **LPUARTx:** LPUART Instance
- **DataWidth:** This parameter can be one of the following values:
  - LL_LPUART_DATAWIDTH_7B
  - LL_LPUART_DATAWIDTH_8B
  - LL_LPUART_DATAWIDTH_9B

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 M LL_LPUART_SetDataWidth

LL_LPUART_GetDataWidth

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetDataWidth (const USART_TypeDef * LPUARTx)

Function description

Return Word length (i.e.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_DATAWIDTH_7B
  - LL_LPUART_DATAWIDTH_8B
  - LL_LPUART_DATAWIDTH_9B

Reference Manual to LL API cross reference:

- CR1 M LL_LPUART_GetDataWidth

LL_LPUART_EnableMuteMode

Function name

__STATIC_INLINE void LL_LPUART_EnableMuteMode (USART_TypeDef * LPUARTx)

Function description

Allow switch between Mute Mode and Active mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 MME LL_LPUART_EnableMuteMode

LL_LPUART_DisableMuteMode

Function name

__STATIC_INLINE void LL_LPUART_DisableMuteMode (USART_TypeDef * LPUARTx)

Function description

Prevent Mute Mode use.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 MME LL_LPUART_DisableMuteMode

LL_LPUART_IsEnabledMuteMode

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledMuteMode (const USART_TypeDef * LPUARTx)

Function description

Indicate if switch between Mute Mode and Active mode is allowed.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 MME LL_LPUART_IsEnabledMuteMode

LL_LPUART_SetPrescaler

Function name

__STATIC_INLINE void LL_LPUART_SetPrescaler (USART_TypeDef * LPUARTx, uint32_t PrescalerValue)

Function description

Configure Clock source prescaler for baudrate generator and oversampling.

Parameters

- **LPUARTx:** LPUART Instance
- **PrescalerValue:** This parameter can be one of the following values:
  - LL_LPUART_PRESCALER_DIV1
  - LL_LPUART_PRESCALER_DIV2
  - LL_LPUART_PRESCALER_DIV4
  - LL_LPUART_PRESCALER_DIV6
  - LL_LPUART_PRESCALER_DIV8
  - LL_LPUART_PRESCALER_DIV10
  - LL_LPUART_PRESCALER_DIV12
  - LL_LPUART_PRESCALER_DIV16
  - LL_LPUART_PRESCALER_DIV32
  - LL_LPUART_PRESCALER_DIV64
  - LL_LPUART_PRESCALER_DIV128
  - LL_LPUART_PRESCALER_DIV256

Return values

- **None:**

Reference Manual to LL API cross reference:

- PRESC PRESCALER LL_LPUART_SetPrescaler

LL_LPUART_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetPrescaler (const USART_TypeDef * LPUARTx)

Function description

Retrieve the Clock source prescaler for baudrate generator and oversampling.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_PRESCALER_DIV1
  - LL_LPUART_PRESCALER_DIV2
  - LL_LPUART_PRESCALER_DIV4
  - LL_LPUART_PRESCALER_DIV6
  - LL_LPUART_PRESCALER_DIV8
  - LL_LPUART_PRESCALER_DIV10
  - LL_LPUART_PRESCALER_DIV12
  - LL_LPUART_PRESCALER_DIV16
  - LL_LPUART_PRESCALER_DIV32
  - LL_LPUART_PRESCALER_DIV64
  - LL_LPUART_PRESCALER_DIV128
  - LL_LPUART_PRESCALER_DIV256

Reference Manual to LL API cross reference:

- PRESC PRESCALER LL_LPUART_GetPrescaler

LL_LPUART_SetStopBitsLength

Function name

__STATIC_INLINE void LL_LPUART_SetStopBitsLength (USART_TypeDef * LPUARTx, uint32_t StopBits)

Function description

Set the length of the stop bits.

Parameters

- **LPUARTx:** LPUART Instance
- **StopBits:** This parameter can be one of the following values:
  - LL_LPUART_STOPBITS_1
  - LL_LPUART_STOPBITS_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 STOP LL_LPUART_SetStopBitsLength

LL_LPUART_GetStopBitsLength

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetStopBitsLength (const USART_TypeDef * LPUARTx)

Function description

Retrieve the length of the stop bits.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_STOPBITS_1
  - LL_LPUART_STOPBITS_2

Reference Manual to LL API cross reference:

- CR2 STOP LL_LPUART_GetStopBitsLength

LL_LPUART_ConfigCharacter

Function name

__STATIC_INLINE void LL_LPUART_ConfigCharacter (USART_TypeDef * LPUARTx, uint32_t DataWidth, uint32_t Parity, uint32_t StopBits)

Function description

Configure Character frame format (Datawidth, Parity control, Stop Bits)

Parameters

- **LPUARTx:** LPUART Instance
- **DataWidth:** This parameter can be one of the following values:
  - LL_LPUART_DATAWIDTH_7B
  - LL_LPUART_DATAWIDTH_8B
  - LL_LPUART_DATAWIDTH_9B
- **Parity:** This parameter can be one of the following values:
  - LL_LPUART_PARITY_NONE
  - LL_LPUART_PARITY_EVEN
  - LL_LPUART_PARITY_ODD
- **StopBits:** This parameter can be one of the following values:
  - LL_LPUART_STOPBITS_1
  - LL_LPUART_STOPBITS_2

Return values

- **None:**

Notes

- Call of this function is equivalent to following function call sequence : Data Width configuration using LL_LPUART_SetDataWidth() functionParity Control and mode configuration using LL_LPUART_SetParity() functionStop bits configuration using LL_LPUART_SetStopBitsLength() function

Reference Manual to LL API cross reference:

- CR1 PS LL_LPUART_ConfigCharacter
- CR1 PCE LL_LPUART_ConfigCharacter
- CR1 M LL_LPUART_ConfigCharacter
- CR2 STOP LL_LPUART_ConfigCharacter

LL_LPUART_SetTXRXSwap

Function name

__STATIC_INLINE void LL_LPUART_SetTXRXSwap (USART_TypeDef * LPUARTx, uint32_t SwapConfig)

Function description

Configure TX/RX pins swapping setting.

Parameters

- **LPUARTx:** LPUART Instance
- **SwapConfig:** This parameter can be one of the following values:
  - LL_LPUART_TXRX_STANDARD
  - LL_LPUART_TXRX_SWAPPED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 SWAP LL_LPUART_SetTXRXSwap

LL_LPUART_GetTXRXSwap

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTXRXSwap (const USART_TypeDef * LPUARTx)

Function description

Retrieve TX/RX pins swapping configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_TXRX_STANDARD
  - LL_LPUART_TXRX_SWAPPED

Reference Manual to LL API cross reference:

- CR2 SWAP LL_LPUART_GetTXRXSwap

LL_LPUART_SetRXPinLevel

Function name

__STATIC_INLINE void LL_LPUART_SetRXPinLevel (USART_TypeDef * LPUARTx, uint32_t PinInvMethod)

Function description

Configure RX pin active level logic.

Parameters

- **LPUARTx:** LPUART Instance
- **PinInvMethod:** This parameter can be one of the following values:
  - LL_LPUART_RXPIN_LEVEL_STANDARD
  - LL_LPUART_RXPIN_LEVEL_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RXINV LL_LPUART_SetRXPinLevel

LL_LPUART_GetRXPinLevel

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetRXPinLevel (const USART_TypeDef * LPUARTx)

Function description

Retrieve RX pin active level logic configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_RXPIN_LEVEL_STANDARD
  - LL_LPUART_RXPIN_LEVEL_INVERTED

Reference Manual to LL API cross reference:

- CR2 RXINV LL_LPUART_GetRXPinLevel

LL_LPUART_SetTXPinLevel

Function name

__STATIC_INLINE void LL_LPUART_SetTXPinLevel (USART_TypeDef * LPUARTx, uint32_t PinInvMethod)

Function description

Configure TX pin active level logic.

Parameters

- **LPUARTx:** LPUART Instance
- **PinInvMethod:** This parameter can be one of the following values:
  - LL_LPUART_TXPIN_LEVEL_STANDARD
  - LL_LPUART_TXPIN_LEVEL_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 TXINV LL_LPUART_SetTXPinLevel

LL_LPUART_GetTXPinLevel

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTXPinLevel (const USART_TypeDef * LPUARTx)

Function description

Retrieve TX pin active level logic configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_TXPIN_LEVEL_STANDARD
  - LL_LPUART_TXPIN_LEVEL_INVERTED

Reference Manual to LL API cross reference:

- CR2 TXINV LL_LPUART_GetTXPinLevel

LL_LPUART_SetBinaryDataLogic

Function name

__STATIC_INLINE void LL_LPUART_SetBinaryDataLogic (USART_TypeDef * LPUARTx, uint32_t DataLogic)

Function description

Configure Binary data logic.

Parameters

- **LPUARTx:** LPUART Instance
- **DataLogic:** This parameter can be one of the following values:
  - LL_LPUART_BINARY_LOGIC_POSITIVE
  - LL_LPUART_BINARY_LOGIC_NEGATIVE

Return values

- **None:**

Notes

- Allow to define how Logical data from the data register are send/received : either in positive/direct logic (1=H, 0=L) or in negative/inverse logic (1=L, 0=H)

Reference Manual to LL API cross reference:

- CR2 DATAINV LL_LPUART_SetBinaryDataLogic

LL_LPUART_GetBinaryDataLogic

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetBinaryDataLogic (const USART_TypeDef * LPUARTx)

Function description

Retrieve Binary data configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_BINARY_LOGIC_POSITIVE
  - LL_LPUART_BINARY_LOGIC_NEGATIVE

Reference Manual to LL API cross reference:

- CR2 DATAINV LL_LPUART_GetBinaryDataLogic

LL_LPUART_SetTransferBitOrder

Function name

__STATIC_INLINE void LL_LPUART_SetTransferBitOrder (USART_TypeDef * LPUARTx, uint32_t BitOrder)

Function description

Configure transfer bit order (either Less or Most Significant Bit First)

Parameters

- **LPUARTx:** LPUART Instance
- **BitOrder:** This parameter can be one of the following values:
  - LL_LPUART_BITORDER_LSBFIRST
  - LL_LPUART_BITORDER_MSBFIRST

Return values

- **None:**

Notes

- MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.

Reference Manual to LL API cross reference:

- CR2 MSBFIRST LL_LPUART_SetTransferBitOrder

LL_LPUART_GetTransferBitOrder

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTransferBitOrder (const USART_TypeDef * LPUARTx)

Function description

Return transfer bit order (either Less or Most Significant Bit First)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_BITORDER_LSBFIRST
  - LL_LPUART_BITORDER_MSBFIRST

Notes

- MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.

Reference Manual to LL API cross reference:

- CR2 MSBFIRST LL_LPUART_GetTransferBitOrder

LL_LPUART_ConfigNodeAddress

Function name

__STATIC_INLINE void LL_LPUART_ConfigNodeAddress (USART_TypeDef * LPUARTx, uint32_t AddressLen, uint32_t NodeAddress)

Function description

Set Address of the LPUART node.

Parameters

- **LPUARTx:** LPUART Instance
- **AddressLen:** This parameter can be one of the following values:
  - LL_LPUART_ADDRESS_DETECT_4B
  - LL_LPUART_ADDRESS_DETECT_7B
- **NodeAddress:** 4 or 7 bit Address of the LPUART node.

Return values

- **None:**

Notes

- This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with address mark detection.
- 4bits address node is used when 4-bit Address Detection is selected in ADDM7. (b7-b4 should be set to 0) 8bits address node is used when 7-bit Address Detection is selected in ADDM7. (This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with 7-bit address mark detection. The MSB of the character sent by the transmitter should be equal to 1. It may also be used for character detection during normal reception, Mute mode inactive (for example, end of block detection in ModBus protocol). In this case, the whole received character (8-bit) is compared to the ADD[7:0] value and CMF flag is set on match)

Reference Manual to LL API cross reference:

- CR2 ADD LL_LPUART_ConfigNodeAddress
- CR2 ADDM7 LL_LPUART_ConfigNodeAddress

LL_LPUART_GetNodeAddress

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetNodeAddress (const USART_TypeDef * LPUARTx)

Function description

Return 8 bit Address of the LPUART node as set in ADD field of CR2.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Address:** of the LPUART node (Value between Min_Data=0 and Max_Data=255)

Notes

- If 4-bit Address Detection is selected in ADDM7, only 4bits (b3-b0) of returned value are relevant (b31-b4 are not relevant) If 7-bit Address Detection is selected in ADDM7, only 8bits (b7-b0) of returned value are relevant (b31-b8 are not relevant)

Reference Manual to LL API cross reference:

- CR2 ADD LL_LPUART_GetNodeAddress

LL_LPUART_GetNodeAddressLen

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetNodeAddressLen (const USART_TypeDef * LPUARTx)

Function description

Return Length of Node Address used in Address Detection mode (7-bit or 4-bit)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_ADDRESS_DETECT_4B
  - LL_LPUART_ADDRESS_DETECT_7B

Reference Manual to LL API cross reference:

- CR2 ADDM7 LL_LPUART_GetNodeAddressLen

LL_LPUART_EnableRTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_LPUART_EnableRTSHWFlowCtrl (USART_TypeDef * LPUARTx)

Function description

Enable RTS HW Flow Control.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RTSE LL_LPUART_EnableRTSHWFlowCtrl

LL_LPUART_DisableRTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_LPUART_DisableRTSHWFlowCtrl (USART_TypeDef * LPUARTx)

Function description

Disable RTS HW Flow Control.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RTSE LL_LPUART_DisableRTSHWFlowCtrl

LL_LPUART_EnableCTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_LPUART_EnableCTSHWFlowCtrl (USART_TypeDef * LPUARTx)

Function description

Enable CTS HW Flow Control.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 CTSE LL_LPUART_EnableCTSHWFlowCtrl

LL_LPUART_DisableCTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_LPUART_DisableCTSHWFlowCtrl (USART_TypeDef * LPUARTx)

Function description

Disable CTS HW Flow Control.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 CTSE LL_LPUART_DisableCTSHWFlowCtrl

LL_LPUART_SetHWFlowCtrl

Function name

__STATIC_INLINE void LL_LPUART_SetHWFlowCtrl (USART_TypeDef * LPUARTx, uint32_t HardwareFlowControl)

Function description

Configure HW Flow Control mode (both CTS and RTS)

Parameters

- **LPUARTx:** LPUART Instance
- **HardwareFlowControl:** This parameter can be one of the following values:
  - LL_LPUART_HWCONTROL_NONE
  - LL_LPUART_HWCONTROL_RTS
  - LL_LPUART_HWCONTROL_CTS
  - LL_LPUART_HWCONTROL_RTS_CTS

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RTSE LL_LPUART_SetHWFlowCtrl
- CR3 CTSE LL_LPUART_SetHWFlowCtrl

LL_LPUART_GetHWFlowCtrl

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetHWFlowCtrl (const USART_TypeDef * LPUARTx)

Function description

Return HW Flow Control configuration (both CTS and RTS)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_HWCONTROL_NONE
  - LL_LPUART_HWCONTROL_RTS
  - LL_LPUART_HWCONTROL_CTS
  - LL_LPUART_HWCONTROL_RTS_CTS

Reference Manual to LL API cross reference:

- CR3 RTSE LL_LPUART_GetHWFlowCtrl
- CR3 CTSE LL_LPUART_GetHWFlowCtrl

LL_LPUART_EnableOverrunDetect

Function name

__STATIC_INLINE void LL_LPUART_EnableOverrunDetect (USART_TypeDef * LPUARTx)

Function description

Enable Overrun detection.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_LPUART_EnableOverrunDetect

LL_LPUART_DisableOverrunDetect

Function name

__STATIC_INLINE void LL_LPUART_DisableOverrunDetect (USART_TypeDef * LPUARTx)

Function description

Disable Overrun detection.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_LPUART_DisableOverrunDetect

LL_LPUART_IsEnabledOverrunDetect

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledOverrunDetect (const USART_TypeDef * LPUARTx)

Function description

Indicate if Overrun detection is enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_LPUART_IsEnabledOverrunDetect

LL_LPUART_SetWKUPType

Function name

__STATIC_INLINE void LL_LPUART_SetWKUPType (USART_TypeDef * LPUARTx, uint32_t Type)

Function description

Select event type for Wake UP Interrupt Flag (WUS[1:0] bits)

Parameters

- **LPUARTx:** LPUART Instance
- **Type:** This parameter can be one of the following values:
  - LL_LPUART_WAKEUP_ON_ADDRESS
  - LL_LPUART_WAKEUP_ON_STARTBIT
  - LL_LPUART_WAKEUP_ON_RXNE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 WUS LL_LPUART_SetWKUPType

LL_LPUART_GetWKUPType

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetWKUPType (const USART_TypeDef * LPUARTx)

Function description

Return event type for Wake UP Interrupt Flag (WUS[1:0] bits)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_WAKEUP_ON_ADDRESS
  - LL_LPUART_WAKEUP_ON_STARTBIT
  - LL_LPUART_WAKEUP_ON_RXNE

Reference Manual to LL API cross reference:

- CR3 WUS LL_LPUART_GetWKUPType

LL_LPUART_SetBaudRate

Function name

__STATIC_INLINE void LL_LPUART_SetBaudRate (USART_TypeDef * LPUARTx, uint32_t PeriphClk, uint32_t PrescalerValue, uint32_t BaudRate)

Function description

Configure LPUART BRR register for achieving expected Baud Rate value.

Parameters

- **LPUARTx:** LPUART Instance
- **PeriphClk:** Peripheral Clock
- **PrescalerValue:** This parameter can be one of the following values:
  - LL_LPUART_PRESCALER_DIV1
  - LL_LPUART_PRESCALER_DIV2
  - LL_LPUART_PRESCALER_DIV4
  - LL_LPUART_PRESCALER_DIV6
  - LL_LPUART_PRESCALER_DIV8
  - LL_LPUART_PRESCALER_DIV10
  - LL_LPUART_PRESCALER_DIV12
  - LL_LPUART_PRESCALER_DIV16
  - LL_LPUART_PRESCALER_DIV32
  - LL_LPUART_PRESCALER_DIV64
  - LL_LPUART_PRESCALER_DIV128
  - LL_LPUART_PRESCALER_DIV256
- **BaudRate:** Baud Rate

Return values

- **None:**

Notes

- Compute and set LPUARTDIV value in BRR Register (full BRR content) according to used Peripheral Clock and expected Baud Rate values
- Peripheral clock and Baud Rate values provided as function parameters should be valid (Baud rate value != 0).
- Provided that LPUARTx_BRR must be > = 0x300 and LPUART_BRR is 20-bit, a care should be taken when generating high baud rates using high PeriphClk values. PeriphClk must be in the range [3 x BaudRate, 4096 x BaudRate].

Reference Manual to LL API cross reference:

- BRR BRR LL_LPUART_SetBaudRate

LL_LPUART_GetBaudRate

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetBaudRate (const USART_TypeDef * LPUARTx, uint32_t PeriphClk, uint32_t PrescalerValue)

Function description

Return current Baud Rate value, according to LPUARTDIV present in BRR register (full BRR content), and to used Peripheral Clock values.

Parameters

- **LPUARTx:** LPUART Instance
- **PeriphClk:** Peripheral Clock
- **PrescalerValue:** This parameter can be one of the following values:
  - LL_LPUART_PRESCALER_DIV1
  - LL_LPUART_PRESCALER_DIV2
  - LL_LPUART_PRESCALER_DIV4
  - LL_LPUART_PRESCALER_DIV6
  - LL_LPUART_PRESCALER_DIV8
  - LL_LPUART_PRESCALER_DIV10
  - LL_LPUART_PRESCALER_DIV12
  - LL_LPUART_PRESCALER_DIV16
  - LL_LPUART_PRESCALER_DIV32
  - LL_LPUART_PRESCALER_DIV64
  - LL_LPUART_PRESCALER_DIV128
  - LL_LPUART_PRESCALER_DIV256

Return values

- **Baud:** Rate

Notes

- In case of non-initialized or invalid value stored in BRR register, value 0 will be returned.

Reference Manual to LL API cross reference:

- BRR BRR LL_LPUART_GetBaudRate

LL_LPUART_EnableHalfDuplex

Function name

__STATIC_INLINE void LL_LPUART_EnableHalfDuplex (USART_TypeDef * LPUARTx)

Function description

Enable Single Wire Half-Duplex mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_LPUART_EnableHalfDuplex

LL_LPUART_DisableHalfDuplex

Function name

__STATIC_INLINE void LL_LPUART_DisableHalfDuplex (USART_TypeDef * LPUARTx)

Function description

Disable Single Wire Half-Duplex mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_LPUART_DisableHalfDuplex

LL_LPUART_IsEnabledHalfDuplex

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledHalfDuplex (const USART_TypeDef * LPUARTx)

Function description

Indicate if Single Wire Half-Duplex mode is enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_LPUART_IsEnabledHalfDuplex

LL_LPUART_SetDEDeassertionTime

Function name

__STATIC_INLINE void LL_LPUART_SetDEDeassertionTime (USART_TypeDef * LPUARTx, uint32_t Time)

Function description

Set DEDT (Driver Enable De-Assertion Time), Time value expressed on 5 bits ([4:0] bits).

Parameters

- **LPUARTx:** LPUART Instance
- **Time:** Value between Min_Data=0 and Max_Data=31

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 DEDT LL_LPUART_SetDEDeassertionTime

LL_LPUART_GetDEDeassertionTime

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetDEDeassertionTime (const USART_TypeDef * LPUARTx)

Function description

Return DEDT (Driver Enable De-Assertion Time)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Time:** value expressed on 5 bits ([4:0] bits) : c

Reference Manual to LL API cross reference:

- CR1 DEDT LL_LPUART_GetDEDeassertionTime

LL_LPUART_SetDEAssertionTime

Function name

__STATIC_INLINE void LL_LPUART_SetDEAssertionTime (USART_TypeDef * LPUARTx, uint32_t Time)

Function description

Set DEAT (Driver Enable Assertion Time), Time value expressed on 5 bits ([4:0] bits).

Parameters

- **LPUARTx:** LPUART Instance
- **Time:** Value between Min_Data=0 and Max_Data=31

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 DEAT LL_LPUART_SetDEAssertionTime

LL_LPUART_GetDEAssertionTime

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetDEAssertionTime (const USART_TypeDef * LPUARTx)

Function description

Return DEAT (Driver Enable Assertion Time)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Time:** value expressed on 5 bits ([4:0] bits) : Time Value between Min_Data=0 and Max_Data=31

Reference Manual to LL API cross reference:

- CR1 DEAT LL_LPUART_GetDEAssertionTime

LL_LPUART_EnableDEMode

Function name

__STATIC_INLINE void LL_LPUART_EnableDEMode (USART_TypeDef * LPUARTx)

Function description

Enable Driver Enable (DE) Mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DEM LL_LPUART_EnableDEMode

LL_LPUART_DisableDEMode

Function name

__STATIC_INLINE void LL_LPUART_DisableDEMode (USART_TypeDef * LPUARTx)

Function description

Disable Driver Enable (DE) Mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DEM LL_LPUART_DisableDEMode

LL_LPUART_IsEnabledDEMode

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDEMode (const USART_TypeDef * LPUARTx)

Function description

Indicate if Driver Enable (DE) Mode is enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DEM LL_LPUART_IsEnabledDEMode

LL_LPUART_SetDESignalPolarity

Function name

__STATIC_INLINE void LL_LPUART_SetDESignalPolarity (USART_TypeDef * LPUARTx, uint32_t Polarity)

Function description

Select Driver Enable Polarity.

Parameters

- **LPUARTx:** LPUART Instance
- **Polarity:** This parameter can be one of the following values:
  - LL_LPUART_DE_POLARITY_HIGH
  - LL_LPUART_DE_POLARITY_LOW

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DEP LL_LPUART_SetDESignalPolarity

LL_LPUART_GetDESignalPolarity

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetDESignalPolarity (const USART_TypeDef * LPUARTx)

Function description

Return Driver Enable Polarity.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_DE_POLARITY_HIGH
  - LL_LPUART_DE_POLARITY_LOW

Reference Manual to LL API cross reference:

- CR3 DEP LL_LPUART_GetDESignalPolarity

LL_LPUART_IsActiveFlag_PE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_PE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Parity Error Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR PE LL_LPUART_IsActiveFlag_PE

LL_LPUART_IsActiveFlag_FE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_FE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Framing Error Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR FE LL_LPUART_IsActiveFlag_FE

LL_LPUART_IsActiveFlag_NE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_NE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Noise error detected Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR NE LL_LPUART_IsActiveFlag_NE

LL_LPUART_IsActiveFlag_ORE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_ORE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART OverRun Error Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ORE LL_LPUART_IsActiveFlag_ORE

LL_LPUART_IsActiveFlag_IDLE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_IDLE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART IDLE line detected Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR IDLE LL_LPUART_IsActiveFlag_IDLE

LL_LPUART_IsActiveFlag_RXNE_RXFNE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RXNE_RXFNE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Read Data Register or LPUART RX FIFO Not Empty Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RXNE_RXFNE LL_LPUART_IsActiveFlag_RXNE_RXFNE

LL_LPUART_IsActiveFlag_TC

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TC (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Transmission Complete Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TC LL_LPUART_IsActiveFlag_TC

LL_LPUART_IsActiveFlag_TXE_TXFNF

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TXE_TXFNF (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Transmit Data Register Empty or LPUART TX FIFO Not Full Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TXE_TXFNF LL_LPUART_IsActiveFlag_TXE_TXFNF

LL_LPUART_IsActiveFlag_nCTS

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_nCTS (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART CTS interrupt Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CTSIF LL_LPUART_IsActiveFlag_nCTS

LL_LPUART_IsActiveFlag_CTS

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_CTS (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART CTS Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CTS LL_LPUART_IsActiveFlag_CTS

LL_LPUART_IsActiveFlag_BUSY

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_BUSY (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Busy Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR BUSY LL_LPUART_IsActiveFlag_BUSY

LL_LPUART_IsActiveFlag_CM

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_CM (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Character Match Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CMF LL_LPUART_IsActiveFlag_CM

LL_LPUART_IsActiveFlag_SBK

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_SBK (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Send Break Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR SBKF LL_LPUART_IsActiveFlag_SBK

LL_LPUART_IsActiveFlag_RWU

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RWU (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Receive Wake Up from mute mode Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RWU LL_LPUART_IsActiveFlag_RWU

LL_LPUART_IsActiveFlag_WKUP

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_WKUP (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Wake Up from stop mode Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR WUF LL_LPUART_IsActiveFlag_WKUP

LL_LPUART_IsActiveFlag_TEACK

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TEACK (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Transmit Enable Acknowledge Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEACK LL_LPUART_IsActiveFlag_TEACK

LL_LPUART_IsActiveFlag_REACK

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_REACK (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Receive Enable Acknowledge Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR REACK LL_LPUART_IsActiveFlag_REACK

LL_LPUART_IsActiveFlag_TXFE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TXFE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART TX FIFO Empty Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TXFE LL_LPUART_IsActiveFlag_TXFE

LL_LPUART_IsActiveFlag_RXFF

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RXFF (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART RX FIFO Full Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RXFF LL_LPUART_IsActiveFlag_RXFF

LL_LPUART_IsActiveFlag_TXFT

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TXFT (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART TX FIFO Threshold Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TXFT LL_LPUART_IsActiveFlag_TXFT

LL_LPUART_IsActiveFlag_RXFT

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RXFT (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART RX FIFO Threshold Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RXFT LL_LPUART_IsActiveFlag_RXFT

LL_LPUART_ClearFlag_PE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_PE (USART_TypeDef * LPUARTx)

Function description

Clear Parity Error Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR PECF LL_LPUART_ClearFlag_PE

LL_LPUART_ClearFlag_FE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_FE (USART_TypeDef * LPUARTx)

Function description

Clear Framing Error Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR FECF LL_LPUART_ClearFlag_FE

LL_LPUART_ClearFlag_NE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_NE (USART_TypeDef * LPUARTx)

Function description

Clear Noise detected Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR NECF LL_LPUART_ClearFlag_NE

LL_LPUART_ClearFlag_ORE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_ORE (USART_TypeDef * LPUARTx)

Function description

Clear OverRun Error Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ORECF LL_LPUART_ClearFlag_ORE

LL_LPUART_ClearFlag_IDLE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_IDLE (USART_TypeDef * LPUARTx)

Function description

Clear IDLE line detected Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR IDLECF LL_LPUART_ClearFlag_IDLE

LL_LPUART_ClearFlag_TC

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_TC (USART_TypeDef * LPUARTx)

Function description

Clear Transmission Complete Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TCCF LL_LPUART_ClearFlag_TC

LL_LPUART_ClearFlag_nCTS

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_nCTS (USART_TypeDef * LPUARTx)

Function description

Clear CTS Interrupt Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CTSCF LL_LPUART_ClearFlag_nCTS

LL_LPUART_ClearFlag_CM

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_CM (USART_TypeDef * LPUARTx)

Function description

Clear Character Match Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CMCF LL_LPUART_ClearFlag_CM

LL_LPUART_ClearFlag_WKUP

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_WKUP (USART_TypeDef * LPUARTx)

Function description

Clear Wake Up from stop mode Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR WUCF LL_LPUART_ClearFlag_WKUP

LL_LPUART_EnableIT_IDLE

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_IDLE (USART_TypeDef * LPUARTx)

Function description

Enable IDLE Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_LPUART_EnableIT_IDLE

LL_LPUART_EnableIT_RXNE_RXFNE

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_RXNE_RXFNE (USART_TypeDef * LPUARTx)

Function description

Enable RX Not Empty and RX FIFO Not Empty Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_LPUART_EnableIT_RXNE_RXFNE

LL_LPUART_EnableIT_TC

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_TC (USART_TypeDef * LPUARTx)

Function description

Enable Transmission Complete Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TCIE LL_LPUART_EnableIT_TC

LL_LPUART_EnableIT_TXE_TXFNF

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_TXE_TXFNF (USART_TypeDef * LPUARTx)

Function description

Enable TX Empty and TX FIFO Not Full Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_LPUART_EnableIT_TXE_TXFNF

LL_LPUART_EnableIT_PE

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_PE (USART_TypeDef * LPUARTx)

Function description

Enable Parity Error Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 PEIE LL_LPUART_EnableIT_PE

LL_LPUART_EnableIT_CM

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_CM (USART_TypeDef * LPUARTx)

Function description

Enable Character Match Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CMIE LL_LPUART_EnableIT_CM

LL_LPUART_EnableIT_TXFE

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_TXFE (USART_TypeDef * LPUARTx)

Function description

Enable TX FIFO Empty Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_LPUART_EnableIT_TXFE

LL_LPUART_EnableIT_RXFF

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_RXFF (USART_TypeDef * LPUARTx)

Function description

Enable RX FIFO Full Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_LPUART_EnableIT_RXFF

LL_LPUART_EnableIT_ERROR

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_ERROR (USART_TypeDef * LPUARTx)

Function description

Enable Error Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register). 0: Interrupt is inhibited1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register.

Reference Manual to LL API cross reference:

- CR3 EIE LL_LPUART_EnableIT_ERROR

LL_LPUART_EnableIT_CTS

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_CTS (USART_TypeDef * LPUARTx)

Function description

Enable CTS Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_LPUART_EnableIT_CTS

LL_LPUART_EnableIT_WKUP

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_WKUP (USART_TypeDef * LPUARTx)

Function description

Enable Wake Up from Stop Mode Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 WUFIE LL_LPUART_EnableIT_WKUP

LL_LPUART_EnableIT_TXFT

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_TXFT (USART_TypeDef * LPUARTx)

Function description

Enable TX FIFO Threshold Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_LPUART_EnableIT_TXFT

LL_LPUART_EnableIT_RXFT

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_RXFT (USART_TypeDef * LPUARTx)

Function description

Enable RX FIFO Threshold Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_LPUART_EnableIT_RXFT

LL_LPUART_DisableIT_IDLE

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_IDLE (USART_TypeDef * LPUARTx)

Function description

Disable IDLE Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_LPUART_DisableIT_IDLE

LL_LPUART_DisableIT_RXNE_RXFNE

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_RXNE_RXFNE (USART_TypeDef * LPUARTx)

Function description

Disable RX Not Empty and RX FIFO Not Empty Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_LPUART_DisableIT_RXNE_RXFNE

LL_LPUART_DisableIT_TC

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_TC (USART_TypeDef * LPUARTx)

Function description

Disable Transmission Complete Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TCIE LL_LPUART_DisableIT_TC

LL_LPUART_DisableIT_TXE_TXFNF

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_TXE_TXFNF (USART_TypeDef * LPUARTx)

Function description

Disable TX Empty and TX FIFO Not Full Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_LPUART_DisableIT_TXE_TXFNF

LL_LPUART_DisableIT_PE

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_PE (USART_TypeDef * LPUARTx)

Function description

Disable Parity Error Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 PEIE LL_LPUART_DisableIT_PE

LL_LPUART_DisableIT_CM

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_CM (USART_TypeDef * LPUARTx)

Function description

Disable Character Match Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CMIE LL_LPUART_DisableIT_CM

LL_LPUART_DisableIT_TXFE

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_TXFE (USART_TypeDef * LPUARTx)

Function description

Disable TX FIFO Empty Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_LPUART_DisableIT_TXFE

LL_LPUART_DisableIT_RXFF

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_RXFF (USART_TypeDef * LPUARTx)

Function description

Disable RX FIFO Full Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_LPUART_DisableIT_RXFF

LL_LPUART_DisableIT_ERROR

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_ERROR (USART_TypeDef * LPUARTx)

Function description

Disable Error Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register). 0: Interrupt is inhibited1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register.

Reference Manual to LL API cross reference:

- CR3 EIE LL_LPUART_DisableIT_ERROR

LL_LPUART_DisableIT_CTS

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_CTS (USART_TypeDef * LPUARTx)

Function description

Disable CTS Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_LPUART_DisableIT_CTS

LL_LPUART_DisableIT_WKUP

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_WKUP (USART_TypeDef * LPUARTx)

Function description

Disable Wake Up from Stop Mode Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 WUFIE LL_LPUART_DisableIT_WKUP

LL_LPUART_DisableIT_TXFT

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_TXFT (USART_TypeDef * LPUARTx)

Function description

Disable TX FIFO Threshold Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_LPUART_DisableIT_TXFT

LL_LPUART_DisableIT_RXFT

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_RXFT (USART_TypeDef * LPUARTx)

Function description

Disable RX FIFO Threshold Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_LPUART_DisableIT_RXFT

LL_LPUART_IsEnabledIT_IDLE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_IDLE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART IDLE Interrupt source is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_LPUART_IsEnabledIT_IDLE

LL_LPUART_IsEnabledIT_RXNE_RXFNE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_RXNE_RXFNE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART RX Not Empty and LPUART RX FIFO Not Empty Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_LPUART_IsEnabledIT_RXNE_RXFNE

LL_LPUART_IsEnabledIT_TC

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TC (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Transmission Complete Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TCIE LL_LPUART_IsEnabledIT_TC

LL_LPUART_IsEnabledIT_TXE_TXFNF

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TXE_TXFNF (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART TX Empty and LPUART TX FIFO Not Full Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_LPUART_IsEnabledIT_TXE_TXFNF

LL_LPUART_IsEnabledIT_PE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_PE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Parity Error Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 PEIE LL_LPUART_IsEnabledIT_PE

LL_LPUART_IsEnabledIT_CM

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_CM (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Character Match Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 CMIE LL_LPUART_IsEnabledIT_CM

LL_LPUART_IsEnabledIT_TXFE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TXFE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART TX FIFO Empty Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_LPUART_IsEnabledIT_TXFE

LL_LPUART_IsEnabledIT_RXFF

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_RXFF (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART RX FIFO Full Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_LPUART_IsEnabledIT_RXFF

LL_LPUART_IsEnabledIT_ERROR

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_ERROR (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Error Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 EIE LL_LPUART_IsEnabledIT_ERROR

LL_LPUART_IsEnabledIT_CTS

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_CTS (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART CTS Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_LPUART_IsEnabledIT_CTS

LL_LPUART_IsEnabledIT_WKUP

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_WKUP (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Wake Up from Stop Mode Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 WUFIE LL_LPUART_IsEnabledIT_WKUP

LL_LPUART_IsEnabledIT_TXFT

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TXFT (const USART_TypeDef * LPUARTx)

Function description

Check if LPUART TX FIFO Threshold Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_LPUART_IsEnabledIT_TXFT

LL_LPUART_IsEnabledIT_RXFT

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_RXFT (const USART_TypeDef * LPUARTx)

Function description

Check if LPUART RX FIFO Threshold Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_LPUART_IsEnabledIT_RXFT

LL_LPUART_EnableDMAReq_RX

Function name

__STATIC_INLINE void LL_LPUART_EnableDMAReq_RX (USART_TypeDef * LPUARTx)

Function description

Enable DMA Mode for reception.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAR LL_LPUART_EnableDMAReq_RX

LL_LPUART_DisableDMAReq_RX

Function name

__STATIC_INLINE void LL_LPUART_DisableDMAReq_RX (USART_TypeDef * LPUARTx)

Function description

Disable DMA Mode for reception.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAR LL_LPUART_DisableDMAReq_RX

LL_LPUART_IsEnabledDMAReq_RX

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDMAReq_RX (const USART_TypeDef * LPUARTx)

Function description

Check if DMA Mode is enabled for reception.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DMAR LL_LPUART_IsEnabledDMAReq_RX

LL_LPUART_EnableDMAReq_TX

Function name

__STATIC_INLINE void LL_LPUART_EnableDMAReq_TX (USART_TypeDef * LPUARTx)

Function description

Enable DMA Mode for transmission.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAT LL_LPUART_EnableDMAReq_TX

LL_LPUART_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_LPUART_DisableDMAReq_TX (USART_TypeDef * LPUARTx)

Function description

Disable DMA Mode for transmission.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAT LL_LPUART_DisableDMAReq_TX

LL_LPUART_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDMAReq_TX (const USART_TypeDef * LPUARTx)

Function description

Check if DMA Mode is enabled for transmission.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DMAT LL_LPUART_IsEnabledDMAReq_TX

LL_LPUART_EnableDMADeactOnRxErr

Function name

__STATIC_INLINE void LL_LPUART_EnableDMADeactOnRxErr (USART_TypeDef * LPUARTx)

Function description

Enable DMA Disabling on Reception Error.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DDRE LL_LPUART_EnableDMADeactOnRxErr

LL_LPUART_DisableDMADeactOnRxErr

Function name

__STATIC_INLINE void LL_LPUART_DisableDMADeactOnRxErr (USART_TypeDef * LPUARTx)

Function description

Disable DMA Disabling on Reception Error.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DDRE LL_LPUART_DisableDMADeactOnRxErr

LL_LPUART_IsEnabledDMADeactOnRxErr

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDMADeactOnRxErr (const USART_TypeDef * LPUARTx)

Function description

Indicate if DMA Disabling on Reception Error is disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DDRE LL_LPUART_IsEnabledDMADeactOnRxErr

LL_LPUART_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_LPUART_DMA_GetRegAddr (const USART_TypeDef * LPUARTx, uint32_t Direction)

Function description

Get the LPUART data register address used for DMA transfer.

Parameters

- **LPUARTx:** LPUART Instance
- **Direction:** This parameter can be one of the following values:
  - LL_LPUART_DMA_REG_DATA_TRANSMIT
  - LL_LPUART_DMA_REG_DATA_RECEIVE

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- RDR RDR LL_LPUART_DMA_GetRegAddr
-
- TDR TDR LL_LPUART_DMA_GetRegAddr

LL_LPUART_ReceiveData8

Function name

__STATIC_INLINE uint8_t LL_LPUART_ReceiveData8 (const USART_TypeDef * LPUARTx)

Function description

Read Receiver Data register (Receive Data value, 8 bits)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Time:** Value between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- RDR RDR LL_LPUART_ReceiveData8

LL_LPUART_ReceiveData9

Function name

__STATIC_INLINE uint16_t LL_LPUART_ReceiveData9 (const USART_TypeDef * LPUARTx)

Function description

Read Receiver Data register (Receive Data value, 9 bits)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Time:** Value between Min_Data=0x00 and Max_Data=0x1FF

Reference Manual to LL API cross reference:

- RDR RDR LL_LPUART_ReceiveData9

LL_LPUART_TransmitData8

Function name

__STATIC_INLINE void LL_LPUART_TransmitData8 (USART_TypeDef * LPUARTx, uint8_t Value)

Function description

Write in Transmitter Data Register (Transmit Data value, 8 bits)

Parameters

- **LPUARTx:** LPUART Instance
- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TDR TDR LL_LPUART_TransmitData8

LL_LPUART_TransmitData9

Function name

__STATIC_INLINE void LL_LPUART_TransmitData9 (USART_TypeDef * LPUARTx, uint16_t Value)

Function description

Write in Transmitter Data Register (Transmit Data value, 9 bits)

Parameters

- **LPUARTx:** LPUART Instance
- **Value:** between Min_Data=0x00 and Max_Data=0x1FF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TDR TDR LL_LPUART_TransmitData9

LL_LPUART_RequestBreakSending

Function name

__STATIC_INLINE void LL_LPUART_RequestBreakSending (USART_TypeDef * LPUARTx)

Function description

Request Break sending.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RQR SBKRQ LL_LPUART_RequestBreakSending

LL_LPUART_RequestEnterMuteMode

Function name

__STATIC_INLINE void LL_LPUART_RequestEnterMuteMode (USART_TypeDef * LPUARTx)

Function description

Put LPUART in mute mode and set the RWU flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RQR MMRQ LL_LPUART_RequestEnterMuteMode

LL_LPUART_RequestRxDataFlush

Function name

__STATIC_INLINE void LL_LPUART_RequestRxDataFlush (USART_TypeDef * LPUARTx)

Function description

Request a Receive Data and FIFO flush.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- Allows to discard the received data without reading them, and avoid an overrun condition.

Reference Manual to LL API cross reference:

- RQR RXFRQ LL_LPUART_RequestRxDataFlush

LL_LPUART_DeInit

Function name

ErrorStatus LL_LPUART_DeInit (const USART_TypeDef * LPUARTx)

Function description

De-initialize LPUART registers (Registers restored to their default values).

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: LPUART registers are de-initialized
  - ERROR: not applicable

LL_LPUART_Init

Function name

ErrorStatus LL_LPUART_Init (USART_TypeDef * LPUARTx, const LL_LPUART_InitTypeDef * LPUART_InitStruct)

Function description

Initialize LPUART registers according to the specified parameters in LPUART_InitStruct.

Parameters

- **LPUARTx:** LPUART Instance
- **LPUART_InitStruct:** pointer to a LL_LPUART_InitTypeDef structure that contains the configuration information for the specified LPUART peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: LPUART registers are initialized according to LPUART_InitStruct content
  - ERROR: Problem occurred during LPUART Registers initialization

Notes

- As some bits in LPUART configuration registers can only be written when the LPUART is disabled (USART_CR1_UE bit =0), LPUART Peripheral should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.
- Baud rate value stored in LPUART_InitStruct BaudRate field, should be valid (different from 0).

LL_LPUART_StructInit

Function name

void LL_LPUART_StructInit (LL_LPUART_InitTypeDef * LPUART_InitStruct)

Function description

Set each LL_LPUART_InitTypeDef field to default value.

Parameters

- **LPUART_InitStruct:** pointer to a LL_LPUART_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

116.3 LPUART Firmware driver defines

The following section lists the various define and macros of the module.

116.3.1 LPUART

LPUART

Address Length Detection

LL_LPUART_ADDRESS_DETECT_4B

4-bit address detection method selected

LL_LPUART_ADDRESS_DETECT_7B

7-bit address detection (in 8-bit data mode) method selected

Binary Data Inversion

LL_LPUART_BINARY_LOGIC_POSITIVE

Logical data from the data register are send/received in positive/direct logic. (1=H, 0=L)

LL_LPUART_BINARY_LOGIC_NEGATIVE

Logical data from the data register are send/received in negative/inverse logic. (1=L, 0=H). The parity bit is also inverted.

Bit Order

LL_LPUART_BITORDER_LSBFIRST

data is transmitted/received with data bit 0 first, following the start bit

LL_LPUART_BITORDER_MSBFIRST

data is transmitted/received with the MSB first, following the start bit

Clear Flags Defines

LL_LPUART_ICR_PECF

Parity error clear flag

LL_LPUART_ICR_FECF

Framing error clear flag

LL_LPUART_ICR_NCF

Noise error detected clear flag

LL_LPUART_ICR_ORECF

Overrun error clear flag

LL_LPUART_ICR_IDLECF

Idle line detected clear flag

LL_LPUART_ICR_TCCF

Transmission complete clear flag

LL_LPUART_ICR_CTSCF

CTS clear flag

LL_LPUART_ICR_CMCF

Character match clear flag

LL_LPUART_ICR_WUCF

Wakeup from Stop mode clear flag

Datawidth

LL_LPUART_DATAWIDTH_7B

7 bits word length : Start bit, 7 data bits, n stop bits

LL_LPUART_DATAWIDTH_8B

8 bits word length : Start bit, 8 data bits, n stop bits

LL_LPUART_DATAWIDTH_9B

9 bits word length : Start bit, 9 data bits, n stop bits

Driver Enable Polarity

LL_LPUART_DE_POLARITY_HIGH

DE signal is active high

LL_LPUART_DE_POLARITY_LOW

DE signal is active low

Direction

LL_LPUART_DIRECTION_NONE

Transmitter and Receiver are disabled

LL_LPUART_DIRECTION_RX

Transmitter is disabled and Receiver is enabled

LL_LPUART_DIRECTION_TX

Transmitter is enabled and Receiver is disabled

LL_LPUART_DIRECTION_TX_RX

Transmitter and Receiver are enabled

DMA Register Data

LL_LPUART_DMA_REG_DATA_TRANSMIT

Get address of data register used for transmission

LL_LPUART_DMA_REG_DATA_RECEIVE

Get address of data register used for reception

FIFO Threshold

LL_LPUART_FIFOTHRESHOLD_1_8

FIFO reaches 1/8 of its depth

LL_LPUART_FIFOTHRESHOLD_1_4

FIFO reaches 1/4 of its depth

LL_LPUART_FIFOTHRESHOLD_1_2

FIFO reaches 1/2 of its depth

LL_LPUART_FIFOTHRESHOLD_3_4

FIFO reaches 3/4 of its depth

LL_LPUART_FIFOTHRESHOLD_7_8

FIFO reaches 7/8 of its depth

LL_LPUART_FIFOTHRESHOLD_8_8

FIFO becomes empty for TX and full for RX

Get Flags Defines

LL_LPUART_ISR_PE

Parity error flag

LL_LPUART_ISR_FE

Framing error flag

LL_LPUART_ISR_NE

Noise detected flag

LL_LPUART_ISR_ORE

Overrun error flag

LL_LPUART_ISR_IDLE

Idle line detected flag

LL_LPUART_ISR_RXNE_RXFNE

Read data register or RX FIFO not empty flag

LL_LPUART_ISR_TC

Transmission complete flag

LL_LPUART_ISR_TXE_TXFNF

Transmit data register empty or TX FIFO Not Full flag

LL_LPUART_ISR_CTSIF

CTS interrupt flag

LL_LPUART_ISR_CTS

CTS flag

LL_LPUART_ISR_BUSY

Busy flag

LL_LPUART_ISR_CMF

Character match flag

LL_LPUART_ISR_SBKF

Send break flag

LL_LPUART_ISR_RWU

Receiver wakeup from Mute mode flag

LL_LPUART_ISR_WUF

Wakeup from Stop mode flag

LL_LPUART_ISR_TEACK

Transmit enable acknowledge flag

LL_LPUART_ISR_REACK

Receive enable acknowledge flag

LL_LPUART_ISR_TXFE

TX FIFO empty flag

LL_LPUART_ISR_RXFF

RX FIFO full flag

LL_LPUART_ISR_RXFT

RX FIFO threshold flag

LL_LPUART_ISR_TXFT

TX FIFO threshold flag

Hardware Control

LL_LPUART_HWCONTROL_NONE

CTS and RTS hardware flow control disabled

LL_LPUART_HWCONTROL_RTS

RTS output enabled, data is only requested when there is space in the receive buffer

LL_LPUART_HWCONTROL_CTS

CTS mode enabled, data is only transmitted when the nCTS input is asserted (tied to 0)

LL_LPUART_HWCONTROL_RTS_CTS

CTS and RTS hardware flow control enabled

IT Defines

LL_LPUART_CR1_IDLEIE

IDLE interrupt enable

LL_LPUART_CR1_RXNEIE_RXFNEIE

Read data register and RXFIFO not empty interrupt enable

LL_LPUART_CR1_TCIE

Transmission complete interrupt enable

LL_LPUART_CR1_TXEIE_TXFNFIE

Transmit data register empty and TX FIFO not full interrupt enable

LL_LPUART_CR1_PEIE

Parity error

LL_LPUART_CR1_CMIE

Character match interrupt enable

LL_LPUART_CR1_TXFEIE

TX FIFO empty interrupt enable

LL_LPUART_CR1_RXFFIE

RX FIFO full interrupt enable

LL_LPUART_CR3_EIE

Error interrupt enable

LL_LPUART_CR3_CTSIE

CTS interrupt enable

LL_LPUART_CR3_WUFIE

Wakeup from Stop mode interrupt enable

LL_LPUART_CR3_TXFTIE

TX FIFO threshold interrupt enable

LL_LPUART_CR3_RXFTIE

RX FIFO threshold interrupt enable

Parity Control

LL_LPUART_PARITY_NONE

Parity control disabled

LL_LPUART_PARITY_EVEN

Parity control enabled and Even Parity is selected

LL_LPUART_PARITY_ODD

Parity control enabled and Odd Parity is selected

Clock Source Prescaler

LL_LPUART_PRESCALER_DIV1

Input clock not divided

LL_LPUART_PRESCALER_DIV2

Input clock divided by 2

LL_LPUART_PRESCALER_DIV4

Input clock divided by 4

LL_LPUART_PRESCALER_DIV6

Input clock divided by 6

LL_LPUART_PRESCALER_DIV8

Input clock divided by 8

LL_LPUART_PRESCALER_DIV10

Input clock divided by 10

LL_LPUART_PRESCALER_DIV12

Input clock divided by 12

LL_LPUART_PRESCALER_DIV16

Input clock divided by 16

LL_LPUART_PRESCALER_DIV32

Input clock divided by 32

LL_LPUART_PRESCALER_DIV64

Input clock divided by 64

LL_LPUART_PRESCALER_DIV128

Input clock divided by 128

LL_LPUART_PRESCALER_DIV256

Input clock divided by 256

RX Pin Active Level Inversion

LL_LPUART_RXPIN_LEVEL_STANDARD

RX pin signal works using the standard logic levels

LL_LPUART_RXPIN_LEVEL_INVERTED

RX pin signal values are inverted.

Stop Bits

LL_LPUART_STOPBITS_1

1 stop bit

LL_LPUART_STOPBITS_2

2 stop bits

TX Pin Active Level Inversion

LL_LPUART_TXPIN_LEVEL_STANDARD

TX pin signal works using the standard logic levels

LL_LPUART_TXPIN_LEVEL_INVERTED

TX pin signal values are inverted.

TX RX Pins Swap

LL_LPUART_TXRX_STANDARD

TX/RX pins are used as defined in standard pinout

LL_LPUART_TXRX_SWAPPED

TX and RX pins functions are swapped.

Wakeup

LL_LPUART_WAKEUP_IDLELINE

LPUART wake up from Mute mode on Idle Line

LL_LPUART_WAKEUP_ADDRESSMARK

LPUART wake up from Mute mode on Address Mark

Wakeup Activation

LL_LPUART_WAKEUP_ON_ADDRESS

Wake up active on address match

LL_LPUART_WAKEUP_ON_STARTBIT

Wake up active on Start bit detection

LL_LPUART_WAKEUP_ON_RXNE

Wake up active on RXNE

FLAG_Management

LL_LPUART_IsActiveFlag_RXNE

LL_LPUART_IsActiveFlag_TXE

IT_Management

LL_LPUART_EnableIT_RXNE

LL_LPUART_EnableIT_TXE

LL_LPUART_DisableIT_RXNE

LL_LPUART_DisableIT_TXE

LL_LPUART_IsEnabledIT_RXNE

LL_LPUART_IsEnabledIT_TXE

Helper Macros

__LL_LPUART_DIV

Description:

- Compute LPUARTDIV value according to Peripheral Clock and expected Baud Rate (20-bit value of LPUARTDIV is returned)

Parameters:

- __PERIPHCLK__: Peripheral Clock frequency used for LPUART Instance
- __PRESCALER__: This parameter can be one of the following values:
  - LL_LPUART_PRESCALER_DIV1
  - LL_LPUART_PRESCALER_DIV2
  - LL_LPUART_PRESCALER_DIV4
  - LL_LPUART_PRESCALER_DIV6
  - LL_LPUART_PRESCALER_DIV8
  - LL_LPUART_PRESCALER_DIV10
  - LL_LPUART_PRESCALER_DIV12
  - LL_LPUART_PRESCALER_DIV16
  - LL_LPUART_PRESCALER_DIV32
  - LL_LPUART_PRESCALER_DIV64
  - LL_LPUART_PRESCALER_DIV128
  - LL_LPUART_PRESCALER_DIV256
- __BAUDRATE__: Baud Rate value to achieve

Return value:

- LPUARTDIV: value to be used for BRR register filling

Common Write and read registers Macros

LL_LPUART_WriteReg

Description:

- Write a value in LPUART register.

Parameters:

- __INSTANCE__: LPUART Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_LPUART_ReadReg

Description:

- Read a value in LPUART register.

Parameters:

- __INSTANCE__: LPUART Instance
- __REG__: Register to be read

Return value:

- Register: value

117 LL MDMA Generic Driver

117.1 MDMA Firmware driver registers structures

117.1.1 LL_MDMA_InitTypeDef

LL_MDMA_InitTypeDef is defined in the stm32h7xx_ll_mdma.h

Data Fields

- ***uint32_t SrcAddress***
- ***uint32_t DstAddress***
- ***uint32_t RequestMode***
- ***uint32_t TriggerMode***
- ***uint32_t HWTrigger***
- ***uint32_t BlockDataLength***
- ***uint32_t BlockRepeatCount***
- ***uint32_t BlockRepeatDestAddrUpdateMode***
- ***uint32_t BlockRepeatSrcAddrUpdateMode***
- ***uint32_t BlockRepeatDestAddrUpdateVal***
- ***uint32_t BlockRepeatSrcAddrUpdateVal***
- ***uint32_t LinkAddress***
- ***uint32_t WordEndianess***
- ***uint32_t HalfWordEndianess***
- ***uint32_t ByteEndianess***
- ***uint32_t Priority***
- ***uint32_t BufferableWriteMode***
- ***uint32_t PaddingAlignment***
- ***uint32_t PackMode***
- ***uint32_t BufferTransferLength***
- ***uint32_t DestBurst***
- ***uint32_t SrctBurst***
- ***uint32_t DestIncSize***
- ***uint32_t SrcIncSize***
- ***uint32_t DestDataSize***
- ***uint32_t SrcDataSize***
- ***uint32_t DestIncMode***
- ***uint32_t SrcIncMode***
- ***uint32_t DestBus***
- ***uint32_t SrcBus***
- ***uint32_t MaskAddress***
- ***uint32_t MaskData***

Field Documentation

- ***uint32_t LL_MDMA_InitTypeDef::SrcAddress***

  Specifies the transfer source address This parameter must be a value between Min_Data = 0 and Max_Data = 0xFFFFFFFF. This feature can be modified afterwards using unitary function **LL_MDMA_SetSourceAddress()**

- ***uint32_t LL_MDMA_InitTypeDef::DstAddress***

  Specifies the transfer destination address This parameter must be a value between Min_Data = 0 and Max_Data = 0xFFFFFFFF. This feature can be modified afterwards using unitary function **LL_MDMA_SetDestinationAddress()**
- ***uint32_t LL_MDMA_InitTypeDef::RequestMode***

  Specifies the request mode Hardware or Software. This parameter can be a value of ***MDMA_LL_EC_REQUEST_MODE*** This feature can be modified afterwards using unitary function **LL_MDMA_SetRequestMode()**
- ***uint32_t LL_MDMA_InitTypeDef::TriggerMode***

  Specifies the transfer trigger mode. This parameter can be a value of ***MDMA_LL_EC_TRIGGER_MODE*** This feature can be modified afterwards using unitary function **LL_MDMA_SetTriggerMode()**
- ***uint32_t LL_MDMA_InitTypeDef::HWTrigger***

  Specifies the HW transfer trigger used when RequestMode is HW. This parameter can be a value of ***MDMA_LL_EC_HW_TRIGGER_SELCTION*** This feature can be modified afterwards using unitary function **LL_MDMA_SetHWTrigger()**
- ***uint32_t LL_MDMA_InitTypeDef::BlockDataLength***

  Specifies the length of a block transfer in bytes This parameter must be a value between Min_Data = 0 and Max_Data = 0x00010000. This feature can be modified afterwards using unitary function **LL_MDMA_SetBlkDataLength()**
- ***uint32_t LL_MDMA_InitTypeDef::BlockRepeatCount***

  Specifies the Block Repeat Count This parameter must be a value between Min_Data = 0 and Max_Data = 0x00000FFF. This feature can be modified afterwards using unitary function **LL_MDMA_SetBlkRepeatCount()**
- ***uint32_t LL_MDMA_InitTypeDef::BlockRepeatDestAddrUpdateMode***

  Specifies the block repeat destination address update mode. This parameter can be a value of ***MDMA_LL_EC_BLK_RPT_DEST_ADDR_UPDATE_MODE*** This feature can be modified afterwards using unitary function **LL_MDMA_SetBlkRepeatDestAddrUpdate()**
- ***uint32_t LL_MDMA_InitTypeDef::BlockRepeatSrcAddrUpdateMode***

  Specifies the block repeat source address update mode. This parameter can be a value of ***MDMA_LL_EC_SRC_BLK_RPT_ADDR_UPDATE_MODE*** This feature can be modified afterwards using unitary function **LL_MDMA_SetBlkRepeatSrcAddrUpdate()**
- ***uint32_t LL_MDMA_InitTypeDef::BlockRepeatDestAddrUpdateVal***

  Specifies the block repeat destination address update value. This parameter can be a value Between 0 to 0x0000FFFF This feature can be modified afterwards using unitary function **LL_MDMA_SetBlkRptDestAddrUpdateValue()**
- ***uint32_t LL_MDMA_InitTypeDef::BlockRepeatSrcAddrUpdateVal***

  Specifies the block repeat source address update value. This parameter can be a value Between 0 to 0x0000FFFF This feature can be modified afterwards using unitary function **LL_MDMA_SetBlkRptSrcAddrUpdateValue()**
- ***uint32_t LL_MDMA_InitTypeDef::LinkAddress***

  Specifies the linked list next transfer node address. This parameter can be a value Between 0 to 0xFFFFFFFF This feature can be modified afterwards using unitary function **LL_MDMA_SetLinkAddress()**
- ***uint32_t LL_MDMA_InitTypeDef::WordEndianess***

  Specifies the Word transfer endianness This parameter can be a value of ***MDMA_LL_EC_WORD_ENDIANNESS***. This feature can be modified afterwards using unitary function **LL_MDMA_SetWordEndianness()**
- ***uint32_t LL_MDMA_InitTypeDef::HalfWordEndianess***

  Specifies the Half Word transfer endianness This parameter can be a value of ***MDMA_LL_EC_HALFWORD_ENDIANNESS***. This feature can be modified afterwards using unitary function **LL_MDMA_SetHalfWordEndianness()**
- ***uint32_t LL_MDMA_InitTypeDef::ByteEndianess***

  Specifies the Byte transfer endianness This parameter can be a value of ***MDMA_LL_EC_BYTE_ENDIANNESS***. This feature can be modified afterwards using unitary function **LL_MDMA_SetByteEndianness()**

- ***uint32_t LL_MDMA_InitTypeDef::Priority***

  Specifies the channel priority level. This parameter can be a value of ***MDMA_LL_EC_PRIORITY*** This feature can be modified afterwards using unitary function **LL_MDMA_SetChannelPriorityLevel()**
- ***uint32_t LL_MDMA_InitTypeDef::BufferableWriteMode***

  Specifies the transfer Bufferable Write Mode. This parameter can be a value of ***MDMA_LL_EC_BUFF_WRITE_MODE*** This feature can be modified afterwards using unitary function **LL_MDMA_EnableBufferableWrMode()** and LL_MDMA_DisableBufferableWrMode
- ***uint32_t LL_MDMA_InitTypeDef::PaddingAlignment***

  Specifies the transfer Padding and Alignment. This parameter can be a value of ***MDMA_LL_EC_PADDING_ALIGNMENT_MODE*** This feature can be modified afterwards using unitary function **LL_MDMA_SetPaddingAlignment()**
- ***uint32_t LL_MDMA_InitTypeDef::PackMode***

  Specifies the transfer Packing enabled or disabled. This parameter can be a value of ***MDMA_LL_EC_PACKING_MODE*** This feature can be modified afterwards using unitary function **LL_MDMA_EnablePacking()** and **LL_MDMA_DisablePacking()**
- ***uint32_t LL_MDMA_InitTypeDef::BufferTransferLength***

  Specifies the length of a buffer transfer in bytes This parameter must be a value between Min_Data = 0 and Max_Data = 0x0000007F. This feature can be modified afterwards using unitary function **LL_MDMA_SetBufferTransferLength()**
- ***uint32_t LL_MDMA_InitTypeDef::DestBurst***

  Specifies the destination burst size. This parameter can be a value of ***MDMA_LL_EC_DEST_BURST*** This feature can be modified afterwards using unitary function **LL_MDMA_SetDestinationBurstSize()**
- ***uint32_t LL_MDMA_InitTypeDef::SrctBurst***

  Specifies the source burst size. This parameter can be a value of ***MDMA_LL_EC_SRC_BURST*** This feature can be modified afterwards using unitary function **LL_MDMA_SetSourceBurstSize()**
- ***uint32_t LL_MDMA_InitTypeDef::DestIncSize***

  Specifies the destination increment size. This parameter can be a value of ***MDMA_LL_EC_DEST_INC_OFFSET_SIZE*** This feature can be modified afterwards using unitary function **LL_MDMA_SetDestinationIncSize()**
- ***uint32_t LL_MDMA_InitTypeDef::SrcIncSize***

  Specifies the source increment size. This parameter can be a value of ***MDMA_LL_EC_SRC_INC_OFFSET_SIZE*** This feature can be modified afterwards using unitary function **LL_MDMA_SetSourceIncSize()**
- ***uint32_t LL_MDMA_InitTypeDef::DestDataSize***

  Specifies the destination data size. This parameter can be a value of ***MDMA_LL_EC_DEST_DATA_SIZE*** This feature can be modified afterwards using unitary function **LL_MDMA_SetDestinationDataSize()**
- ***uint32_t LL_MDMA_InitTypeDef::SrcDataSize***

  Specifies the source data size. This parameter can be a value of ***MDMA_LL_EC_SRC_DATA_SIZE*** This feature can be modified afterwards using unitary function **LL_MDMA_SetSourceDataSize()**
- ***uint32_t LL_MDMA_InitTypeDef::DestIncMode***

  Specifies the destination increment mode. This parameter can be a value of ***MDMA_LL_EC_DEST_INC_MODE*** This feature can be modified afterwards using unitary function **LL_MDMA_SetDestinationIncMode()**
- ***uint32_t LL_MDMA_InitTypeDef::SrcIncMode***

  Specifies the source increment mode. This parameter can be a value of ***MDMA_LL_EC_SRC_INC_MODE*** This feature can be modified afterwards using unitary function **LL_MDMA_SetSourceIncMode()**
- ***uint32_t LL_MDMA_InitTypeDef::DestBus***

  Specifies the destination transfer bus, System AXI or AHB/TCM bus. This parameter can be a value of ***MDMA_LL_EC_DEST_BUS*** This feature can be modified afterwards using unitary function **LL_MDMA_SetDestBusSelection()**
- ***uint32_t LL_MDMA_InitTypeDef::SrcBus***

  Specifies the source transfer bus, System AXI or AHB/TCM bus. This parameter can be a value of ***MDMA_LL_EC_SRC_BUS*** This feature can be modified afterwards using unitary function **LL_MDMA_SetSrcBusSelection()**

- ***uint32_t LL_MDMA_InitTypeDef::MaskAddress***

  Specifies the address to be updated (written) with MaskData after a request is served. MaskAddress and MaskData could be used to automatically clear a peripheral flag when the request is served This parameter can be a value Between 0 to 0xFFFFFFFF This feature can be modified afterwards using unitary function **LL_MDMA_SetMaskAddress()**
- ***uint32_t LL_MDMA_InitTypeDef::MaskData***

  Specifies the value to be written to MaskAddress after a request is served. MaskAddress and MaskData could be used to automatically clear a peripheral flag when the request is served This parameter can be a value Between 0 to 0xFFFFFFFF This feature can be modified afterwards using unitary function **LL_MDMA_SetMaskData()**

117.1.2 LL_MDMA_LinkNodeTypeDef

LL_MDMA_LinkNodeTypeDef is defined in the stm32h7xx_ll_mdma.h

Data Fields

- ***__IO uint32_t CTCR***
- ***__IO uint32_t CBNDTR***
- ***__IO uint32_t CSAR***
- ***__IO uint32_t CDAR***
- ***__IO uint32_t CBRUR***
- ***__IO uint32_t CLAR***
- ***__IO uint32_t CTBR***
- ***__IO uint32_t Reserved***
- ***__IO uint32_t CMAR***
- ***__IO uint32_t CMDR***

Field Documentation

- ***__IO uint32_t LL_MDMA_LinkNodeTypeDef::CTCR***

  New CTCR register configuration for the given MDMA linked list node
- ***__IO uint32_t LL_MDMA_LinkNodeTypeDef::CBNDTR***

  New CBNDTR register configuration for the given MDMA linked list node
- ***__IO uint32_t LL_MDMA_LinkNodeTypeDef::CSAR***

  New CSAR register configuration for the given MDMA linked list node
- ***__IO uint32_t LL_MDMA_LinkNodeTypeDef::CDAR***

  New CDAR register configuration for the given MDMA linked list node
- ***__IO uint32_t LL_MDMA_LinkNodeTypeDef::CBRUR***

  New CBRUR register configuration for the given MDMA linked list node
- ***__IO uint32_t LL_MDMA_LinkNodeTypeDef::CLAR***

  New CLAR register configuration for the given MDMA linked list node
- ***__IO uint32_t LL_MDMA_LinkNodeTypeDef::CTBR***

  New CTBR register configuration for the given MDMA linked list node
- ***__IO uint32_t LL_MDMA_LinkNodeTypeDef::Reserved***

  Reserved register
- ***__IO uint32_t LL_MDMA_LinkNodeTypeDef::CMAR***

  New CMAR register configuration for the given MDMA linked list node
- ***__IO uint32_t LL_MDMA_LinkNodeTypeDef::CMDR***

  New CMDR register configuration for the given MDMA linked list node

117.2 MDMA Firmware driver API description

The following section lists the various functions of the MDMA library.

117.2.1 Detailed description of functions

LL_MDMA_EnableChannel

Function name

__STATIC_INLINE void LL_MDMA_EnableChannel (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Enable MDMA channel.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR EN LL_MDMA_EnableChannel

LL_MDMA_DisableChannel

Function name

__STATIC_INLINE void LL_MDMA_DisableChannel (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Disable MDMA channel.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR EN LL_MDMA_DisableChannel

LL_MDMA_IsEnabledChannel

Function name

__STATIC_INLINE uint32_t LL_MDMA_IsEnabledChannel (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Check if MDMA channel is enabled or disabled.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCR EN LL_MDMA_IsEnabledChannel

LL_MDMA_GenerateSWRequest

Function name

__STATIC_INLINE void LL_MDMA_GenerateSWRequest (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Generate a SW transfer request on the MDMA channel.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR SWRQ LL_MDMA_GenerateSWRequest

LL_MDMA_ConfigXferEndianness

Function name

__STATIC_INLINE void LL_MDMA_ConfigXferEndianness (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t Configuration)

Function description

Configure Transfer endianness parameters : Word, Half word and Bytes Endianness.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_MDMA_WORD_ENDIANNESS_PRESERVE or LL_MDMA_WORD_ENDIANNESS_EXCHANGE
  - LL_MDMA_HALFWORD_ENDIANNESS_PRESERVE or LL_MDMA_HALFWORD_ENDIANNESS_EXCHANGE
  - LL_MDMA_BYTE_ENDIANNESS_PRESERVE or LL_MDMA_BYTE_ENDIANNESS_EXCHANGE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR WEX LL_MDMA_ConfigXferEndianness
-
- CCR HEX LL_MDMA_ConfigXferEndianness
-
- CCR BEX LL_MDMA_ConfigXferEndianness

LL_MDMA_SetWordEndianness

Function name

__STATIC_INLINE void LL_MDMA_SetWordEndianness (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t Endianness)

Function description

Set Words Endianness.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **Endianness:** This parameter can be one of the following values:
  - LL_MDMA_WORD_ENDIANNESS_PRESERVE
  - LL_MDMA_WORD_ENDIANNESS_EXCHANGE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR WEX LL_MDMA_SetWordEndianness

LL_MDMA_GetWordEndianness

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetWordEndianness (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get Words Endianness.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_MDMA_WORD_ENDIANNESS_PRESERVE
  - LL_MDMA_WORD_ENDIANNESS_EXCHANGE
- **None:**

Reference Manual to LL API cross reference:

- CCR WEX LL_MDMA_GetWordEndianness

LL_MDMA_SetHalfWordEndianness

Function name

__STATIC_INLINE void LL_MDMA_SetHalfWordEndianness (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t Endianness)

Function description

Set Half Words Endianness.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **Endianness:** This parameter can be one of the following values:
  - LL_MDMA_HALFWORD_ENDIANNESS_PRESERVE
  - LL_MDMA_HALFWORD_ENDIANNESS_EXCHANGE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR HEX LL_MDMA_SetHalfWordEndianness

LL_MDMA_GetHalfWordEndianness

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetHalfWordEndianness (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get Half Words Endianness.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_MDMA_HALFWORD_ENDIANNESS_PRESERVE
  - LL_MDMA_HALFWORD_ENDIANNESS_EXCHANGE
- **None:**

Reference Manual to LL API cross reference:

- CCR HEX LL_MDMA_GetHalfWordEndianness

LL_MDMA_SetByteEndianness

Function name

__STATIC_INLINE void LL_MDMA_SetByteEndianness (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t Endianness)

Function description

Set Bytes Endianness.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **Endianness:** This parameter can be one of the following values:
  - LL_MDMA_BYTE_ENDIANNESS_PRESERVE
  - LL_MDMA_BYTE_ENDIANNESS_EXCHANGE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR BEX LL_MDMA_SetByteEndianness

LL_MDMA_GetByteEndianness

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetByteEndianness (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get Bytes Endianness.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_MDMA_BYTE_ENDIANNESS_PRESERVE
  - LL_MDMA_BYTE_ENDIANNESS_EXCHANGE
- **None:**

Reference Manual to LL API cross reference:

- CCR BEX LL_MDMA_GetByteEndianness

LL_MDMA_SetChannelPriorityLevel

Function name

__STATIC_INLINE void LL_MDMA_SetChannelPriorityLevel (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t Priority)

Function description

Set Channel priority level.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **Priority:** This parameter can be one of the following values:
  - LL_MDMA_PRIORITY_LOW
  - LL_MDMA_PRIORITY_MEDIUM
  - LL_MDMA_PRIORITY_HIGH
  - LL_MDMA_PRIORITY_VERYHIGH

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR PL LL_MDMA_SetChannelPriorityLevel

LL_MDMA_GetChannelPriorityLevel

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetChannelPriorityLevel (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get Channel priority level.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_MDMA_PRIORITY_LOW
  - LL_MDMA_PRIORITY_MEDIUM
  - LL_MDMA_PRIORITY_HIGH
  - LL_MDMA_PRIORITY_VERYHIGH
- **None:**

Reference Manual to LL API cross reference:

- CCR PL LL_MDMA_GetChannelPriorityLevel

LL_MDMA_ConfigTransfer

Function name

__STATIC_INLINE void LL_MDMA_ConfigTransfer (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t Configuration, uint32_t BufferXferLength)

Function description

Configure MDMA transfer parameters.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_MDMA_BUFF_WRITE_DISABLE or LL_MDMA_BUFF_WRITE_ENABLE
  - LL_MDMA_REQUEST_MODE_HW or LL_MDMA_REQUEST_MODE_SW
  - LL_MDMA_BUFFER_TRANSFER or LL_MDMA_BLOCK_TRANSFER or LL_MDMA_REPEAT_BLOCK_TRANSFER or LL_MDMA_FULL_TRANSFER
  - LL_MDMA_DATAALIGN_RIGHT or LL_MDMA_DATAALIGN_RIGHT_SIGNED or LL_MDMA_DATAALIGN_LEFT
  - LL_MDMA_PACK_DISABLE or LL_MDMA_PACK_ENABLE
  - LL_MDMA_DEST_BURST_SINGLE or LL_MDMA_DEST_BURST_2BEATS or LL_MDMA_DEST_BURST_4BEATS or LL_MDMA_DEST_BURST_8BEATS or LL_MDMA_DEST_BURST_16BEATS or LL_MDMA_DEST_BURST_32BEATS or LL_MDMA_DEST_BURST_64BEATS or LL_MDMA_DEST_BURST_128BEATS
  - LL_MDMA_SRC_BURST_SINGLE or LL_MDMA_SRC_BURST_2BEATS or LL_MDMA_SRC_BURST_4BEATS or LL_MDMA_SRC_BURST_8BEATS or LL_MDMA_SRC_BURST_16BEATS or LL_MDMA_SRC_BURST_32BEATS or LL_MDMA_SRC_BURST_64BEATS or LL_MDMA_SRC_BURST_128BEATS
  - LL_MDMA_DEST_INC_OFFSET_BYTE or LL_MDMA_DEST_INC_OFFSET_HALFWORD or LL_MDMA_DEST_INC_OFFSET_WORD or LL_MDMA_DEST_INC_OFFSET_DOUBLEWORD
  - LL_MDMA_SRC_INC_OFFSET_BYTE or LL_MDMA_SRC_INC_OFFSET_HALFWORD or LL_MDMA_SRC_INC_OFFSET_WORD or LL_MDMA_SRC_INC_OFFSET_DOUBLEWORD
  - LL_MDMA_DEST_DATA_SIZE_BYTE or LL_MDMA_DEST_DATA_SIZE_HALFWORD or LL_MDMA_DEST_DATA_SIZE_WORD or LL_MDMA_DEST_DATA_SIZE_DOUBLEWORD
  - LL_MDMA_SRC_DATA_SIZE_BYTE or LL_MDMA_SRC_DATA_SIZE_HALFWORD or LL_MDMA_SRC_DATA_SIZE_WORD or LL_MDMA_SRC_DATA_SIZE_DOUBLEWORD
  - LL_MDMA_DEST_FIXED or LL_MDMA_DEST_INCREMENT or LL_MDMA_DEST_DECREMENT
  - LL_MDMA_SRC_FIXED or LL_MDMA_SRC_INCREMENT or LL_MDMA_SRC_DECREMENT
- **BufferXferLength:** This parameter can be a value Between 0 to 0x0000007F

Return values

- **None:**

Reference Manual to LL API cross reference:

- CTCR BWM LL_MDMA_ConfigTransfer
- CTCR SWRM LL_MDMA_ConfigTransfer
- CTCR TRGM LL_MDMA_ConfigTransfer
- CTCR PAM LL_MDMA_ConfigTransfer
- CTCR PKE LL_MDMA_ConfigTransfer
- CTCR TLEN LL_MDMA_ConfigTransfer
- CTCR DBURST LL_MDMA_ConfigTransfer
- CTCR SBURST LL_MDMA_ConfigTransfer
- CTCR DINCOS LL_MDMA_ConfigTransfer
- CTCR SINCOS LL_MDMA_ConfigTransfer
- CTCR DSIZE LL_MDMA_ConfigTransfer
- CTCR SSIZE LL_MDMA_ConfigTransfer
- CTCR DINC LL_MDMA_ConfigTransfer
- CTCR SINC LL_MDMA_ConfigTransfer

LL_MDMA_EnableBufferableWrMode

Function name

__STATIC_INLINE void LL_MDMA_EnableBufferableWrMode (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Enable Bufferable Write Mode.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CTCR BWM LL_MDMA_EnableBufferableWrMode

LL_MDMA_DisableBufferableWrMode

Function name

__STATIC_INLINE void LL_MDMA_DisableBufferableWrMode (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Disable Bufferable Write Mode.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CTCR BWM LL_MDMA_DisableBufferableWrMode

LL_MDMA_IsEnabledBufferableWrMode

Function name

__STATIC_INLINE uint32_t LL_MDMA_IsEnabledBufferableWrMode (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Check if Bufferable Write Mode is enabled or disabled.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CTCR BWM LL_MDMA_IsEnabledBufferableWrMode

LL_MDMA_SetRequestMode

Function name

__STATIC_INLINE void LL_MDMA_SetRequestMode (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t RequestMode)

Function description

Set Request Mode.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **RequestMode:** This parameter can be one of the following values:
  - LL_MDMA_REQUEST_MODE_HW
  - LL_MDMA_REQUEST_MODE_SW

Return values

- **None:**

Reference Manual to LL API cross reference:

- CTCR SWRM LL_MDMA_SetRequestMode

LL_MDMA_GetRequestMode

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetRequestMode (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get Request Mode.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_MDMA_REQUEST_MODE_HW
  - LL_MDMA_REQUEST_MODE_SW
- **None:**

Reference Manual to LL API cross reference:

- CTCR SWRM LL_MDMA_GetRequestMode

LL_MDMA_SetTriggerMode

Function name

__STATIC_INLINE void LL_MDMA_SetTriggerMode (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t TriggerMode)

Function description

Set Trigger Mode.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **TriggerMode:** This parameter can be one of the following values:
  - LL_MDMA_BUFFER_TRANSFER
  - LL_MDMA_BLOCK_TRANSFER
  - LL_MDMA_REPEAT_BLOCK_TRANSFER
  - LL_MDMA_FULL_TRANSFER

Return values

- **None:**

Reference Manual to LL API cross reference:

- CTCR TRGM LL_MDMA_SetTriggerMode

LL_MDMA_GetTriggerMode

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetTriggerMode (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get Trigger Mode.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_MDMA_BUFFER_TRANSFER
  - LL_MDMA_BLOCK_TRANSFER
  - LL_MDMA_REPEAT_BLOCK_TRANSFER
  - LL_MDMA_FULL_TRANSFER
- **None:**

Reference Manual to LL API cross reference:

- CTCR TRGM LL_MDMA_GetTriggerMode

LL_MDMA_SetPaddingAlignment

Function name

__STATIC_INLINE void LL_MDMA_SetPaddingAlignment (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t PaddingAlignment)

Function description

Set Padding Alignment.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **PaddingAlignment:** This parameter can be one of the following values:
  - LL_MDMA_DATAALIGN_RIGHT
  - LL_MDMA_DATAALIGN_RIGHT_SIGNED
  - LL_MDMA_DATAALIGN_LEFT

Return values

- **None:**

Reference Manual to LL API cross reference:

- CTCR PAM LL_MDMA_SetPaddingAlignment

LL_MDMA_GetPaddingAlignment

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetPaddingAlignment (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get Padding Alignment.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_MDMA_DATAALIGN_RIGHT
  - LL_MDMA_DATAALIGN_RIGHT_SIGNED
  - LL_MDMA_DATAALIGN_LEFT
- **None:**

Reference Manual to LL API cross reference:

- CTCR PAM LL_MDMA_GetPaddingAlignment

LL_MDMA_EnablePacking

Function name

__STATIC_INLINE void LL_MDMA_EnablePacking (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Enable Packing.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CTCR PKE LL_MDMA_EnablePacking

LL_MDMA_DisablePacking

Function name

__STATIC_INLINE void LL_MDMA_DisablePacking (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Disable Packing.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CTCR PKE LL_MDMA_DisablePacking

LL_MDMA_IsEnabledPacking

Function name

__STATIC_INLINE uint32_t LL_MDMA_IsEnabledPacking (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Check if packing is enabled or disabled.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CTCR PKE LL_MDMA_IsEnabledPacking

LL_MDMA_SetBufferTransferLength

Function name

__STATIC_INLINE void LL_MDMA_SetBufferTransferLength (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t Length)

Function description

Set Buffer Transfer Length.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **Length:** Between 0 to 0x0000007F

Return values

- **None:**

Reference Manual to LL API cross reference:

- CTCR TLEN LL_MDMA_SetBufferTransferLength

LL_MDMA_GetBufferTransferLength

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetBufferTransferLength (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get Buffer Transfer Length.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Between:** 0 to 0x0000007F
- **None:**

Reference Manual to LL API cross reference:

- CTCR TLEN LL_MDMA_GetBufferTransferLength

LL_MDMA_SetDestinationBurstSize

Function name

__STATIC_INLINE void LL_MDMA_SetDestinationBurstSize (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t Dburst)

Function description

Set Destination burst transfer.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **Dburst:** This parameter can be one of the following values:
  - LL_MDMA_DEST_BURST_SINGLE
  - LL_MDMA_DEST_BURST_2BEATS
  - LL_MDMA_DEST_BURST_4BEATS
  - LL_MDMA_DEST_BURST_8BEATS
  - LL_MDMA_DEST_BURST_16BEATS
  - LL_MDMA_DEST_BURST_32BEATS
  - LL_MDMA_DEST_BURST_64BEATS
  - LL_MDMA_DEST_BURST_128BEATS

Return values

- **None:**

Reference Manual to LL API cross reference:

- CTCR DBURST LL_MDMA_SetDestinationBurstSize

LL_MDMA_GetDestinationBurstSize

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetDestinationBurstSize (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get Destination burst transfer.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_MDMA_DEST_BURST_SINGLE
  - LL_MDMA_DEST_BURST_2BEATS
  - LL_MDMA_DEST_BURST_4BEATS
  - LL_MDMA_DEST_BURST_8BEATS
  - LL_MDMA_DEST_BURST_16BEATS
  - LL_MDMA_DEST_BURST_32BEATS
  - LL_MDMA_DEST_BURST_64BEATS
  - LL_MDMA_DEST_BURST_128BEATS
- **None:**

Reference Manual to LL API cross reference:

- CTCR DBURST LL_MDMA_GetDestinationBurstSize

LL_MDMA_SetSourceBurstSize

Function name

__STATIC_INLINE void LL_MDMA_SetSourceBurstSize (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t Sburst)

Function description

Set Source burst transfer.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **Sburst:** This parameter can be one of the following values:
  - LL_MDMA_SRC_BURST_SINGLE
  - LL_MDMA_SRC_BURST_2BEATS
  - LL_MDMA_SRC_BURST_4BEATS
  - LL_MDMA_SRC_BURST_8BEATS
  - LL_MDMA_SRC_BURST_16BEATS
  - LL_MDMA_SRC_BURST_32BEATS
  - LL_MDMA_SRC_BURST_64BEATS
  - LL_MDMA_SRC_BURST_128BEATS

Return values

- **None:**

Reference Manual to LL API cross reference:

- CTCR SBURST LL_MDMA_SetSourceBurstSize

LL_MDMA_GetSourceBurstSize

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetSourceBurstSize (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get Source burst transfer.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_MDMA_SRC_BURST_SINGLE
  - LL_MDMA_SRC_BURST_2BEATS
  - LL_MDMA_SRC_BURST_4BEATS
  - LL_MDMA_SRC_BURST_8BEATS
  - LL_MDMA_SRC_BURST_16BEATS
  - LL_MDMA_SRC_BURST_32BEATS
  - LL_MDMA_SRC_BURST_64BEATS
  - LL_MDMA_SRC_BURST_128BEATS
- **None:**

Reference Manual to LL API cross reference:

- CTCR SBURST LL_MDMA_GetSourceBurstSize

LL_MDMA_SetDestinationIncSize

Function name

__STATIC_INLINE void LL_MDMA_SetDestinationIncSize (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t IncSize)

Function description

Set Destination Increment Offset Size.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **IncSize:** This parameter can be one of the following values:
  - LL_MDMA_DEST_INC_OFFSET_BYTE
  - LL_MDMA_DEST_INC_OFFSET_HALFWORD
  - LL_MDMA_DEST_INC_OFFSET_WORD
  - LL_MDMA_DEST_INC_OFFSET_DOUBLEWORD

Return values

- **None:**

Reference Manual to LL API cross reference:

- CTCR DINCOS LL_MDMA_SetDestinationIncSize

LL_MDMA_GetDestinationIncSize

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetDestinationIncSize (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get Destination Increment Offset Size.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_MDMA_DEST_INC_OFFSET_BYTE
  - LL_MDMA_DEST_INC_OFFSET_HALFWORD
  - LL_MDMA_DEST_INC_OFFSET_WORD
  - LL_MDMA_DEST_INC_OFFSET_DOUBLEWORD
- **None:**

Reference Manual to LL API cross reference:

- CTCR DINCOS LL_MDMA_GetDestinationIncSize

LL_MDMA_SetSourceIncSize

Function name

__STATIC_INLINE void LL_MDMA_SetSourceIncSize (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t IncSize)

Function description

Set Source Increment Offset Size.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **IncSize:** This parameter can be one of the following values:
  - LL_MDMA_SRC_INC_OFFSET_BYTE
  - LL_MDMA_SRC_INC_OFFSET_HALFWORD
  - LL_MDMA_SRC_INC_OFFSET_WORD
  - LL_MDMA_SRC_INC_OFFSET_DOUBLEWORD

Return values

- **None:**

Reference Manual to LL API cross reference:

- CTCR SINCOS LL_MDMA_SetSourceIncSize

LL_MDMA_GetSourceIncSize

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetSourceIncSize (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get Source Increment Offset Size.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_MDMA_SRC_INC_OFFSET_BYTE
  - LL_MDMA_SRC_INC_OFFSET_HALFWORD
  - LL_MDMA_SRC_INC_OFFSET_WORD
  - LL_MDMA_SRC_INC_OFFSET_DOUBLEWORD
- **None:**

Reference Manual to LL API cross reference:

- CTCR SINCOS LL_MDMA_GetSourceIncSize

LL_MDMA_SetDestinationDataSize

Function name

__STATIC_INLINE void LL_MDMA_SetDestinationDataSize (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t DestDataSize)

Function description

Set Destination Data Size.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **DestDataSize:** This parameter can be one of the following values:
  - LL_MDMA_DEST_DATA_SIZE_BYTE
  - LL_MDMA_DEST_DATA_SIZE_HALFWORD
  - LL_MDMA_DEST_DATA_SIZE_WORD
  - LL_MDMA_DEST_DATA_SIZE_DOUBLEWORD

Return values

- **None:**

Reference Manual to LL API cross reference:

- CTCR DSIZE LL_MDMA_SetDestinationDataSize

LL_MDMA_GetDestinationDataSize

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetDestinationDataSize (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get Destination Data Size.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_MDMA_DEST_DATA_SIZE_BYTE
  - LL_MDMA_DEST_DATA_SIZE_HALFWORD
  - LL_MDMA_DEST_DATA_SIZE_WORD
  - LL_MDMA_DEST_DATA_SIZE_DOUBLEWORD
- **None:**

Reference Manual to LL API cross reference:

- CTCR DSIZE LL_MDMA_GetDestinationDataSize

LL_MDMA_SetSourceDataSize

Function name

__STATIC_INLINE void LL_MDMA_SetSourceDataSize (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t SrcDataSize)

Function description

Set Source Data Size.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **SrcDataSize:** This parameter can be one of the following values:
  - LL_MDMA_SRC_DATA_SIZE_BYTE
  - LL_MDMA_SRC_DATA_SIZE_HALFWORD
  - LL_MDMA_SRC_DATA_SIZE_WORD
  - LL_MDMA_SRC_DATA_SIZE_DOUBLEWORD

Return values

- **None:**

Reference Manual to LL API cross reference:

- CTCR SSIZE LL_MDMA_SetSourceDataSize

LL_MDMA_GetSourceDataSize

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetSourceDataSize (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get Source Data Size.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_MDMA_SRC_DATA_SIZE_BYTE
  - LL_MDMA_SRC_DATA_SIZE_HALFWORD
  - LL_MDMA_SRC_DATA_SIZE_WORD
  - LL_MDMA_SRC_DATA_SIZE_DOUBLEWORD
- **None:**

Reference Manual to LL API cross reference:

- CTCR SSIZE LL_MDMA_GetSourceDataSize

LL_MDMA_SetDestinationIncMode

Function name

__STATIC_INLINE void LL_MDMA_SetDestinationIncMode (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t DestIncMode)

Function description

Set Destination Increment Mode.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **DestIncMode:** This parameter can be one of the following values:
  - LL_MDMA_DEST_FIXED
  - LL_MDMA_DEST_INCREMENT
  - LL_MDMA_DEST_DECREMENT

Return values

- **None:**

Reference Manual to LL API cross reference:

- CTCR DINC LL_MDMA_SetDestinationIncMode

LL_MDMA_GetDestinationIncMode

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetDestinationIncMode (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get Destination Increment Mode.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_MDMA_DEST_FIXED
  - LL_MDMA_DEST_INCREMENT
  - LL_MDMA_DEST_DECREMENT
- **None:**

Reference Manual to LL API cross reference:

- CTCR DINC LL_MDMA_GetDestinationIncMode

LL_MDMA_SetSourceIncMode

Function name

__STATIC_INLINE void LL_MDMA_SetSourceIncMode (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t SrcIncMode)

Function description

Set Source Increment Mode.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **SrcIncMode:** This parameter can be one of the following values:
  - LL_MDMA_SRC_FIXED
  - LL_MDMA_SRC_INCREMENT
  - LL_MDMA_SRC_DECREMENT

Return values

- **None:**

Reference Manual to LL API cross reference:

- CTCR SINC LL_MDMA_SetSourceIncMode

LL_MDMA_GetSourceIncMode

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetSourceIncMode (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get Source Increment Mode.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_MDMA_SRC_FIXED
  - LL_MDMA_SRC_INCREMENT
  - LL_MDMA_SRC_DECREMENT
- **None:**

Reference Manual to LL API cross reference:

- CTCR SINC LL_MDMA_GetSourceIncMode

LL_MDMA_ConfigBlkCounters

Function name

__STATIC_INLINE void LL_MDMA_ConfigBlkCounters (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t BlockRepeatCount, uint32_t BlkDataLength)

Function description

Configure MDMA Block number of data and repeat Count.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **BlockRepeatCount:** Between 0 to 0x00000FFF
- **BlkDataLength:** Between 0 to 0x00010000

Return values

- **None:**

Reference Manual to LL API cross reference:

- CBNDTR BRC LL_MDMA_ConfigBlkCounters
-
- CBNDTR BNDT LL_MDMA_ConfigBlkCounters

LL_MDMA_SetBlkDataLength

Function name

__STATIC_INLINE void LL_MDMA_SetBlkDataLength (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t BlkDataLength)

Function description

Set Block Number of data bytes to transfer.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **BlkDataLength:** Between 0 to 0x00010000

Return values

- **None:**

Reference Manual to LL API cross reference:

- CBNDTR BNDT LL_MDMA_SetBlkDataLength

LL_MDMA_GetBlkDataLength

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetBlkDataLength (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get Block Number of data bytes to transfer.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Between:** 0 to 0x00010000
- **None:**

Reference Manual to LL API cross reference:

- CBNDTR BNDT LL_MDMA_GetBlkDataLength

LL_MDMA_SetBlkRepeatCount

Function name

__STATIC_INLINE void LL_MDMA_SetBlkRepeatCount (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t BlockRepeatCount)

Function description

Set Block Repeat Count.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **BlockRepeatCount:** Between 0 to 0x00000FFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- CBNDTR BRC LL_MDMA_SetBlkRepeatCount

LL_MDMA_GetBlkRepeatCount

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetBlkRepeatCount (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get Block Repeat Count.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Between:** 0 to 0x00000FFF
- **None:**

Reference Manual to LL API cross reference:

- CBNDTR BRC LL_MDMA_GetBlkRepeatCount

LL_MDMA_ConfigBlkRepeatAddrUpdate

Function name

__STATIC_INLINE void LL_MDMA_ConfigBlkRepeatAddrUpdate (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t Configuration)

Function description

Configure MDMA block repeat address update mode.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_MDMA_BLK_RPT_DEST_ADDR_INCREMENT or LL_MDMA_BLK_RPT_DEST_ADDR_DECREMENT
  - LL_MDMA_BLK_RPT_SRC_ADDR_INCREMENT or LL_MDMA_BLK_RPT_SRC_ADDR_DECREMENT

Return values

- **None:**

Reference Manual to LL API cross reference:

- CBNDTR BRDUM LL_MDMA_ConfigBlkRepeatAddrUpdate
-
- CBNDTR BRSUM LL_MDMA_ConfigBlkRepeatAddrUpdate

LL_MDMA_SetBlkRepeatDestAddrUpdate

Function name

__STATIC_INLINE void LL_MDMA_SetBlkRepeatDestAddrUpdate (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t DestAdrUpdateMode)

Function description

Set Block Repeat Destination address Update Mode.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **DestAdrUpdateMode:** This parameter can be one of the following values:
  - LL_MDMA_BLK_RPT_DEST_ADDR_INCREMENT
  - LL_MDMA_BLK_RPT_DEST_ADDR_DECREMENT

Return values

- **None:**

Reference Manual to LL API cross reference:

- CBNDTR BRDUM LL_MDMA_SetBlkRepeatDestAddrUpdate

LL_MDMA_GetBlkRepeatDestAddrUpdate

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetBlkRepeatDestAddrUpdate (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get Block Repeat Destination address Update Mode.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_MDMA_BLK_RPT_DEST_ADDR_INCREMENT
  - LL_MDMA_BLK_RPT_DEST_ADDR_DECREMENT
- **None:**

Reference Manual to LL API cross reference:

- CBNDTR BRDUM LL_MDMA_GetBlkRepeatDestAddrUpdate

LL_MDMA_SetBlkRepeatSrcAddrUpdate

Function name

__STATIC_INLINE void LL_MDMA_SetBlkRepeatSrcAddrUpdate (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t SrcAdrUpdateMode)

Function description

Set Block Repeat Source address Update Mode.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **SrcAdrUpdateMode:** This parameter can be one of the following values:
  - LL_MDMA_BLK_RPT_SRC_ADDR_INCREMENT
  - LL_MDMA_BLK_RPT_SRC_ADDR_DECREMENT

Return values

- **None:**

Reference Manual to LL API cross reference:

- CBNDTR BRSUM LL_MDMA_SetBlkRepeatSrcAddrUpdate

LL_MDMA_GetBlkRepeatSrcAddrUpdate

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetBlkRepeatSrcAddrUpdate (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get Block Repeat Source address Update Mode.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_MDMA_BLK_RPT_SRC_ADDR_INCREMENT
  - LL_MDMA_BLK_RPT_SRC_ADDR_DECREMENT
- **None:**

Reference Manual to LL API cross reference:

- CBNDTR BRSUM LL_MDMA_GetBlkRepeatSrcAddrUpdate

LL_MDMA_ConfigAddresses

Function name

__STATIC_INLINE void LL_MDMA_ConfigAddresses (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t SrcAddress, uint32_t DstAddress)

Function description

Configure the Source and Destination addresses.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **SrcAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF
- **DstAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF

Return values

- **None:**

Notes

- This API must not be called when the MDMA channel is enabled.

Reference Manual to LL API cross reference:

- CSAR SAR LL_MDMA_ConfigAddresses
-
- CDAR DAR LL_MDMA_ConfigAddresses

LL_MDMA_SetSourceAddress

Function name

__STATIC_INLINE void LL_MDMA_SetSourceAddress (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t SrcAddress)

Function description

Set transfer Source address.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **SrcAddress:** Between 0 to 0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSAR SAR LL_MDMA_SetSourceAddress

LL_MDMA_GetSourceAddress

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetSourceAddress (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get transfer Source address.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Between:** 0 to 0xFFFFFFFF
- **None:**

Reference Manual to LL API cross reference:

- CSAR SAR LL_MDMA_GetSourceAddress

LL_MDMA_SetDestinationAddress

Function name

__STATIC_INLINE void LL_MDMA_SetDestinationAddress (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t DestAddress)

Function description

Set transfer Destination address.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **DestAddress:** Between 0 to 0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- CDAR DAR LL_MDMA_SetDestinationAddress

LL_MDMA_GetDestinationAddress

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetDestinationAddress (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get transfer Destination address.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Between:** 0 to 0xFFFFFFFF
- **None:**

Reference Manual to LL API cross reference:

- CDAR DAR LL_MDMA_GetDestinationAddress

LL_MDMA_ConfigBlkRptAddrUpdateValue

Function name

__STATIC_INLINE void LL_MDMA_ConfigBlkRptAddrUpdateValue (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t SrctAdrUpdateValue, uint32_t DestAdrUpdateValue)

Function description

Configure the Source and Destination Block repeat addresses Update value.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **SrctAdrUpdateValue:** Min_Data = 0 and Max_Data = 0x0000FFFF
- **DestAdrUpdateValue:** Between Min_Data = 0 and Max_Data = 0x0000FFFF

Return values

- **None:**

Notes

- This API must not be called when the MDMA channel is enabled.

Reference Manual to LL API cross reference:

- CBRUR DUV LL_MDMA_ConfigBlkRptAddrUpdateValue
-
- CBRUR SUV LL_MDMA_ConfigBlkRptAddrUpdateValue

LL_MDMA_SetBlkRptDestAddrUpdateValue

Function name

__STATIC_INLINE void LL_MDMA_SetBlkRptDestAddrUpdateValue (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t DestAdrUpdateValue)

Function description

Set transfer Destination address Update Value.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **DestAdrUpdateValue:** Between 0 to 0x0000FFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- CBRUR DUV LL_MDMA_SetBlkRptDestAddrUpdateValue

LL_MDMA_GetBlkRptDestAddrUpdateValue

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetBlkRptDestAddrUpdateValue (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get transfer Destination address Update Value.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Between:** 0 to 0x0000FFFF
- **None:**

Reference Manual to LL API cross reference:

- CBRUR DUV LL_MDMA_GetBlkRptDestAddrUpdateValue

LL_MDMA_SetBlkRptSrcAddrUpdateValue

Function name

__STATIC_INLINE void LL_MDMA_SetBlkRptSrcAddrUpdateValue (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t SrcAdrUpdateValue)

Function description

Set transfer Source address Update Value.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **SrcAdrUpdateValue:** Between 0 to 0x0000FFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- CBRUR SUV LL_MDMA_SetBlkRptSrcAddrUpdateValue

LL_MDMA_GetBlkRptSrcAddrUpdateValue

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetBlkRptSrcAddrUpdateValue (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get transfer Source address Update Value.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Between:** 0 to 0x0000FFFF
- **None:**

Reference Manual to LL API cross reference:

- CBRUR SUV LL_MDMA_GetBlkRptSrcAddrUpdateValue

LL_MDMA_SetLinkAddress

Function name

__STATIC_INLINE void LL_MDMA_SetLinkAddress (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t LinkAddress)

Function description

Set transfer Link Address.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **LinkAddress:** Between 0 to 0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- CLAR LAR LL_MDMA_SetLinkAddress

LL_MDMA_GetLinkAddress

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetLinkAddress (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get transfer Link Address.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Between:** 0 to 0xFFFFFFFF
- **None:**

Reference Manual to LL API cross reference:

- CLAR LAR LL_MDMA_GetLinkAddress

LL_MDMA_ConfigBusSelection

Function name

__STATIC_INLINE void LL_MDMA_ConfigBusSelection (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t Configuration)

Function description

Configure MDMA source and destination bus selection.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_MDMA_DEST_BUS_SYSTEM_AXI or LL_MDMA_DEST_BUS_AHB_TCM
  - LL_MDMA_SRC_BUS_SYSTEM_AXI or LL_MDMA_SRC_BUS_AHB_TCM

Return values

- **None:**

Reference Manual to LL API cross reference:

- CTBR DBUS LL_MDMA_ConfigBusSelection
-
- CTBR SBUS LL_MDMA_ConfigBusSelection

LL_MDMA_SetDestBusSelection

Function name

__STATIC_INLINE void LL_MDMA_SetDestBusSelection (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t DestBus)

Function description

Set Destination Bus Selection.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **DestBus:** This parameter can be one of the following values:
  - LL_MDMA_DEST_BUS_SYSTEM_AXI
  - LL_MDMA_DEST_BUS_AHB_TCM

Return values

- **None:**

Reference Manual to LL API cross reference:

- CTBR DBUS LL_MDMA_SetDestBusSelection

LL_MDMA_GetDestBusSelection

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetDestBusSelection (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get Destination Bus Selection.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_MDMA_DEST_BUS_SYSTEM_AXI
  - LL_MDMA_DEST_BUS_AHB_TCM
- **None:**

Reference Manual to LL API cross reference:

- CTBR DBUS LL_MDMA_GetDestBusSelection

LL_MDMA_SetSrcBusSelection

Function name

__STATIC_INLINE void LL_MDMA_SetSrcBusSelection (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t SrcBus)

Function description

Set Source Bus Selection.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **SrcBus:** This parameter can be one of the following values:
  - LL_MDMA_SRC_BUS_SYSTEM_AXI
  - LL_MDMA_SRC_BUS_AHB_TCM

Return values

- **None:**

Reference Manual to LL API cross reference:

- CTBR SBUS LL_MDMA_SetSrcBusSelection

LL_MDMA_GetSrcBusSelection

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetSrcBusSelection (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get Source Bus Selection.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_MDMA_SRC_BUS_SYSTEM_AXI
  - LL_MDMA_SRC_BUS_AHB_TCM
- **None:**

Reference Manual to LL API cross reference:

- CTBR SBUS LL_MDMA_GetSrcBusSelection

LL_MDMA_SetHWTrigger

Function name

__STATIC_INLINE void LL_MDMA_SetHWTrigger (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t HWRequest)

Function description

Set Transfer hardware trigger (Request).

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

- **HWRequest:** This parameter can be one of the following values:
  - LL_MDMA_REQ_DMA1_STREAM0_TC
  - LL_MDMA_REQ_DMA1_STREAM1_TC
  - LL_MDMA_REQ_DMA1_STREAM2_TC
  - LL_MDMA_REQ_DMA1_STREAM3_TC
  - LL_MDMA_REQ_DMA1_STREAM4_TC
  - LL_MDMA_REQ_DMA1_STREAM5_TC
  - LL_MDMA_REQ_DMA1_STREAM6_TC
  - LL_MDMA_REQ_DMA1_STREAM7_TC
  - LL_MDMA_REQ_DMA2_STREAM0_TC
  - LL_MDMA_REQ_DMA2_STREAM1_TC
  - LL_MDMA_REQ_DMA2_STREAM2_TC
  - LL_MDMA_REQ_DMA2_STREAM3_TC
  - LL_MDMA_REQ_DMA2_STREAM4_TC
  - LL_MDMA_REQ_DMA2_STREAM5_TC
  - LL_MDMA_REQ_DMA2_STREAM6_TC
  - LL_MDMA_REQ_DMA2_STREAM7_TC
  - LL_MDMA_REQ_LTDC_LINE_IT (*)
  - LL_MDMA_REQ_JPEG_INFIFO_TH (*)
  - LL_MDMA_REQ_JPEG_INFIFO_NF (*)
  - LL_MDMA_REQ_JPEG_OUTFIFO_TH (*)
  - LL_MDMA_REQ_JPEG_OUTFIFO_NE (*)
  - LL_MDMA_REQ_JPEG_END_CONVERSION (*)
  - LL_MDMA_REQ_QUADSPI_FIFO_TH (*)
  - LL_MDMA_REQ_QUADSPI_TC (*)
  - LL_MDMA_REQ_OCTOSPI1_FIFO_TH (*)
  - LL_MDMA_REQ_OCTOSPI1_TC (*)
  - LL_MDMA_REQ_DMA2D_CLUT_TC
  - LL_MDMA_REQ_DMA2D_TC
  - LL_MDMA_REQ_DMA2D_TW
  - LL_MDMA_REQ_DSI_TEARING_EFFECT (*)
  - LL_MDMA_REQ_DSI_END_REFRESH (*)
  - LL_MDMA_REQ_SDMMC1_END_DATA
  - LL_MDMA_REQ_SDMMC1_DMA_ENDBUFFER (*)
  - LL_MDMA_REQ_SDMMC1_COMMAND_END (*)
  - LL_MDMA_REQ_OCTOSPI2_FIFO_TH (*)
  - LL_MDMA_REQ_OCTOSPI2_TC (*)

Return values

- **None:**

Notes

- (*) Availability depends on devices.

Reference Manual to LL API cross reference:

- CTBR TSEL LL_MDMA_SetHWTrigger

LL_MDMA_GetHWTrigger

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetHWTrigger (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get Transfer hardware trigger (Request).

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_MDMA_REQ_DMA1_STREAM0_TC
  - LL_MDMA_REQ_DMA1_STREAM1_TC
  - LL_MDMA_REQ_DMA1_STREAM2_TC
  - LL_MDMA_REQ_DMA1_STREAM3_TC
  - LL_MDMA_REQ_DMA1_STREAM4_TC
  - LL_MDMA_REQ_DMA1_STREAM5_TC
  - LL_MDMA_REQ_DMA1_STREAM6_TC
  - LL_MDMA_REQ_DMA1_STREAM7_TC
  - LL_MDMA_REQ_DMA2_STREAM0_TC
  - LL_MDMA_REQ_DMA2_STREAM1_TC
  - LL_MDMA_REQ_DMA2_STREAM2_TC
  - LL_MDMA_REQ_DMA2_STREAM3_TC
  - LL_MDMA_REQ_DMA2_STREAM4_TC
  - LL_MDMA_REQ_DMA2_STREAM5_TC
  - LL_MDMA_REQ_DMA2_STREAM6_TC
  - LL_MDMA_REQ_DMA2_STREAM7_TC
  - LL_MDMA_REQ_LTDC_LINE_IT (*)
  - LL_MDMA_REQ_JPEG_INFIFO_TH (*)
  - LL_MDMA_REQ_JPEG_INFIFO_NF (*)
  - LL_MDMA_REQ_JPEG_OUTFIFO_TH (*)
  - LL_MDMA_REQ_JPEG_OUTFIFO_NE (*)
  - LL_MDMA_REQ_JPEG_END_CONVERSION (*)
  - LL_MDMA_REQ_QUADSPI_FIFO_TH (*)
  - LL_MDMA_REQ_QUADSPI_TC (*)
  - LL_MDMA_REQ_OCTOSPI1_FIFO_TH (*)
  - LL_MDMA_REQ_OCTOSPI1_TC (*)
  - LL_MDMA_REQ_DMA2D_CLUT_TC
  - LL_MDMA_REQ_DMA2D_TC
  - LL_MDMA_REQ_DMA2D_TW
  - LL_MDMA_REQ_DSI_TEARING_EFFECT (*)
  - LL_MDMA_REQ_DSI_END_REFRESH (*)
  - LL_MDMA_REQ_SDMMC1_END_DATA
  - LL_MDMA_REQ_SDMMC1_DMA_ENDBUFFER (*)
  - LL_MDMA_REQ_SDMMC1_COMMAND_END (*)
  - LL_MDMA_REQ_OCTOSPI2_FIFO_TH (*)
  - LL_MDMA_REQ_OCTOSPI2_TC (*)
- **None:**

Notes

- (*) Availability depends on devices.

Reference Manual to LL API cross reference:

- CTBR TSEL LL_MDMA_GetHWTrigger

LL_MDMA_SetMaskAddress

Function name

__STATIC_INLINE void LL_MDMA_SetMaskAddress (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t MaskAddress)

Function description

Set Mask Address.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **MaskAddress:** Between 0 to 0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- CMAR MAR LL_MDMA_SetMaskAddress

LL_MDMA_GetMaskAddress

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetMaskAddress (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get Mask Address.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Between:** 0 to 0xFFFFFFFF
- **None:**

Reference Manual to LL API cross reference:

- CMAR MAR LL_MDMA_GetMaskAddress

LL_MDMA_SetMaskData

Function name

__STATIC_INLINE void LL_MDMA_SetMaskData (MDMA_TypeDef * MDMAx, uint32_t Channel, uint32_t MaskData)

Function description

Set Mask Data.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **MaskData:** Between 0 to 0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- CMDR MDR LL_MDMA_SetMaskData

LL_MDMA_GetMaskData

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetMaskData (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get Mask Data.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Between:** 0 to 0xFFFFFFFF
- **None:**

Reference Manual to LL API cross reference:

- CMDR MDR LL_MDMA_GetMaskData

LL_MDMA_GetXferErrorDirection

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetXferErrorDirection (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get Transfer Error Direction.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_MDMA_READ_ERROR
  - LL_MDMA_WRITE_ERROR
- **None:**

Reference Manual to LL API cross reference:

- CESR TED LL_MDMA_GetXferErrorDirection

LL_MDMA_GetXferErrorLSBAddress

Function name

__STATIC_INLINE uint32_t LL_MDMA_GetXferErrorLSBAddress (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get Transfer Error LSB Address.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **Between:** 0 to 0x0000007F
- **None:**

Reference Manual to LL API cross reference:

- CESR TEA LL_MDMA_GetXferErrorLSBAddress

LL_MDMA_IsActiveFlag_GI

Function name

__STATIC_INLINE uint32_t LL_MDMA_IsActiveFlag_GI (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get MDMA Channel x Global Interrupt flag.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- GISR0 GIFx LL_MDMA_IsActiveFlag_GI

LL_MDMA_IsActiveFlag_TE

Function name

__STATIC_INLINE uint32_t LL_MDMA_IsActiveFlag_TE (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get MDMA Channel x Transfer Error interrupt flag.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CISR TEIF LL_MDMA_IsActiveFlag_TE

LL_MDMA_IsActiveFlag_CTC

Function name

__STATIC_INLINE uint32_t LL_MDMA_IsActiveFlag_CTC (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get MDMA Channel x Channel Transfer Complete interrupt flag.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CISR CTCIF LL_MDMA_IsActiveFlag_CTC

LL_MDMA_IsActiveFlag_BRT

Function name

__STATIC_INLINE uint32_t LL_MDMA_IsActiveFlag_BRT (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get MDMA Channel x Block Repeat Transfer complete interrupt flag.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CISR BRTIF LL_MDMA_IsActiveFlag_BRT

LL_MDMA_IsActiveFlag_BT

Function name

__STATIC_INLINE uint32_t LL_MDMA_IsActiveFlag_BT (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get MDMA Channel x Block Transfer complete interrupt flag.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CISR BTIF LL_MDMA_IsActiveFlag_BT

LL_MDMA_IsActiveFlag_TC

Function name

__STATIC_INLINE uint32_t LL_MDMA_IsActiveFlag_TC (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get MDMA Channel x buffer transfer complete interrupt flag.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CISR TCIF LL_MDMA_IsActiveFlag_TC

LL_MDMA_IsActiveFlag_CRQA

Function name

__STATIC_INLINE uint32_t LL_MDMA_IsActiveFlag_CRQA (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get MDMA Channel x ReQuest Active flag.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CISR CRQA LL_MDMA_IsActiveFlag_CRQA

LL_MDMA_IsActiveFlag_BSE

Function name

__STATIC_INLINE uint32_t LL_MDMA_IsActiveFlag_BSE (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get MDMA Channel x Block Size Error flag.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CESR BSE LL_MDMA_IsActiveFlag_BSE

LL_MDMA_IsActiveFlag_ASE

Function name

__STATIC_INLINE uint32_t LL_MDMA_IsActiveFlag_ASE (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get MDMA Channel x Address/Size Error flag.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CESR ASE LL_MDMA_IsActiveFlag_ASE

LL_MDMA_IsActiveFlag_TEMD

Function name

__STATIC_INLINE uint32_t LL_MDMA_IsActiveFlag_TEMD (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get MDMA Channel x Transfer Error Mask Data flag.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CESR TEMD LL_MDMA_IsActiveFlag_TEMD

LL_MDMA_IsActiveFlag_TELD

Function name

__STATIC_INLINE uint32_t LL_MDMA_IsActiveFlag_TELD (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Get MDMA Channel x Transfer Error Link Data flag.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CESR TELD LL_MDMA_IsActiveFlag_TELD

LL_MDMA_ClearFlag_TE

Function name

__STATIC_INLINE void LL_MDMA_ClearFlag_TE (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Clear MDMA Channel x Transfer Error interrupt flag.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIFCR CTEIF LL_MDMA_ClearFlag_TE

LL_MDMA_ClearFlag_CTC

Function name

__STATIC_INLINE void LL_MDMA_ClearFlag_CTC (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Clear MDMA Channel x Channel Transfer Complete interrupt flag.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIFCR CCTCIF LL_MDMA_ClearFlag_CTC

LL_MDMA_ClearFlag_BRT

Function name

__STATIC_INLINE void LL_MDMA_ClearFlag_BRT (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Clear MDMA Channel x Block Repeat Transfer complete interrupt flag.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIFCR CBRTIF LL_MDMA_ClearFlag_BRT

LL_MDMA_ClearFlag_BT

Function name

__STATIC_INLINE void LL_MDMA_ClearFlag_BT (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Clear MDMA Channel x Block Transfer complete interrupt flag.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIFCR CBTIF LL_MDMA_ClearFlag_BT

LL_MDMA_ClearFlag_TC

Function name

__STATIC_INLINE void LL_MDMA_ClearFlag_TC (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Clear MDMA Channel x buffer transfer Complete Interrupt Flag.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIFCR CLTCIF LL_MDMA_ClearFlag_TC

LL_MDMA_EnableIT_TE

Function name

__STATIC_INLINE void LL_MDMA_EnableIT_TE (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Enable MDMA Channel x Transfer Error interrupt.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR TEIE LL_MDMA_EnableIT_TE

LL_MDMA_EnableIT_CTC

Function name

__STATIC_INLINE void LL_MDMA_EnableIT_CTC (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Enable MDMA Channel x Channel Transfer Complete interrupt.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR CTCIE LL_MDMA_EnableIT_CTC

LL_MDMA_EnableIT_BRT

Function name

__STATIC_INLINE void LL_MDMA_EnableIT_BRT (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Enable MDMA Channel x Block Repeat Transfer interrupt.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR BRTIE LL_MDMA_EnableIT_BRT

LL_MDMA_EnableIT_BT

Function name

__STATIC_INLINE void LL_MDMA_EnableIT_BT (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Enable MDMA Channel x Block Transfer interrupt.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR BTIE LL_MDMA_EnableIT_BT

LL_MDMA_EnableIT_TC

Function name

__STATIC_INLINE void LL_MDMA_EnableIT_TC (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Enable MDMA Channel x buffer transfer complete interrupt.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR TCIE LL_MDMA_EnableIT_TC

LL_MDMA_DisableIT_TE

Function name

__STATIC_INLINE void LL_MDMA_DisableIT_TE (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Disable MDMA Channel x Transfer Error interrupt.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR TEIE LL_MDMA_DisableIT_TE

LL_MDMA_DisableIT_CTC

Function name

__STATIC_INLINE void LL_MDMA_DisableIT_CTC (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Disable MDMA Channel x Channel Transfer Complete interrupt.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR CTCIE LL_MDMA_DisableIT_CTC

LL_MDMA_DisableIT_BRT

Function name

__STATIC_INLINE void LL_MDMA_DisableIT_BRT (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Disable MDMA Channel x Block Repeat Transfer interrupt.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR BRTIE LL_MDMA_DisableIT_BRT

LL_MDMA_DisableIT_BT

Function name

__STATIC_INLINE void LL_MDMA_DisableIT_BT (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Disable MDMA Channel x Block Transfer interrupt.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR BTIE LL_MDMA_DisableIT_BT

LL_MDMA_DisableIT_TC

Function name

__STATIC_INLINE void LL_MDMA_DisableIT_TC (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Disable MDMA Channel x buffer transfer complete interrupt.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR TCIE LL_MDMA_DisableIT_TC

LL_MDMA_IsEnabledIT_TE

Function name

__STATIC_INLINE uint32_t LL_MDMA_IsEnabledIT_TE (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Check if MDMA Channel x Transfer Error interrupt is enabled.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCR TEIE LL_MDMA_IsEnabledIT_TE

LL_MDMA_IsEnabledIT_CTC

Function name

__STATIC_INLINE uint32_t LL_MDMA_IsEnabledIT_CTC (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Check if MDMA Channel x Channel Transfer Complete interrupt is enabled.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCR CTCIE LL_MDMA_IsEnabledIT_CTC

LL_MDMA_IsEnabledIT_BRT

Function name

__STATIC_INLINE uint32_t LL_MDMA_IsEnabledIT_BRT (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Check if MDMA Channel x Block Repeat Transfer complete interrupt is enabled.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCR BRTIE LL_MDMA_IsEnabledIT_BRT

LL_MDMA_IsEnabledIT_BT

Function name

__STATIC_INLINE uint32_t LL_MDMA_IsEnabledIT_BT (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Check if MDMA Channel x Block Transfer interrupt is enabled.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCR BTIE LL_MDMA_IsEnabledIT_BT

LL_MDMA_IsEnabledIT_TC

Function name

__STATIC_INLINE uint32_t LL_MDMA_IsEnabledIT_TC (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

Check if MDMA Channel x buffer transfer complete interrupt is enabled.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCR TCIE LL_MDMA_IsEnabledIT_TC

LL_MDMA_Init

Function name

uint32_t LL_MDMA_Init (MDMA_TypeDef * MDMAx, uint32_t Channel, LL_MDMA_InitTypeDef * MDMA_InitStruct)

Function description

Initialize the MDMA registers according to the specified parameters in MDMA_InitStruct.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
- **MDMA_InitStruct:** pointer to a LL_MDMA_InitTypeDef structure.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: MDMA registers are initialized
  - ERROR: Not applicable

Notes

- To convert MDMAx_Channely Instance to MDMAx Instance and Channely, use helper macros : LL_MDMA_GET_INSTANCE LL_MDMA_GET_CHANNEL

LL_MDMA_DeInit

Function name

uint32_t LL_MDMA_DeInit (MDMA_TypeDef * MDMAx, uint32_t Channel)

Function description

De-initialize the MDMA registers to their default reset values.

Parameters

- **MDMAx:** MDMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_MDMA_CHANNEL_0
  - LL_MDMA_CHANNEL_1
  - LL_MDMA_CHANNEL_2
  - LL_MDMA_CHANNEL_3
  - LL_MDMA_CHANNEL_4
  - LL_MDMA_CHANNEL_5
  - LL_MDMA_CHANNEL_6
  - LL_MDMA_CHANNEL_7
  - LL_MDMA_CHANNEL_8
  - LL_MDMA_CHANNEL_9
  - LL_MDMA_CHANNEL_10
  - LL_MDMA_CHANNEL_11
  - LL_MDMA_CHANNEL_12
  - LL_MDMA_CHANNEL_13
  - LL_MDMA_CHANNEL_14
  - LL_MDMA_CHANNEL_15
  - LL_MDMA_CHANNEL_ALL

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: MDMA registers are de-initialized
  - ERROR: Not applicable

LL_MDMA_StructInit

Function name

void LL_MDMA_StructInit (LL_MDMA_InitTypeDef * MDMA_InitStruct)

Function description

Set each LL_MDMA_InitTypeDef field to default value.

Parameters

- **MDMA_InitStruct:** Pointer to a LL_MDMA_InitTypeDef structure.

Return values

- **None:**

LL_MDMA_CreateLinkNode

Function name

void LL_MDMA_CreateLinkNode (LL_MDMA_InitTypeDef * MDMA_InitStruct, LL_MDMA_LinkNodeTypeDef * pNode)

Function description

Initializes MDMA linked list node according to the specified parameters in the MDMA_InitStruct.

Parameters

- **MDMA_InitStruct:** Pointer to a LL_MDMA_InitTypeDef structure that contains linked list node registers configurations.
- **pNode:** Pointer to linked list node to fill according to MDMA_InitStruct parameters.

Return values

- **None:**

LL_MDMA_ConnectLinkNode

Function name

void LL_MDMA_ConnectLinkNode (LL_MDMA_LinkNodeTypeDef * pPrevLinkNode, LL_MDMA_LinkNodeTypeDef * pNewLinkNode)

Function description

Connect Linked list Nodes.

Parameters

- **pPrevLinkNode:** Pointer to previous linked list node to be connected to new Lined list node.
- **pNewLinkNode:** Pointer to new Linked list.

Return values

- **None:**

LL_MDMA_DisconnectNextLinkNode

Function name

void LL_MDMA_DisconnectNextLinkNode (LL_MDMA_LinkNodeTypeDef * pLinkNode)

Function description

Disconnect the next linked list node.

Parameters

- **pLinkNode:** Pointer to linked list node to be disconnected from the next one.

Return values

- **None:**

117.3 MDMA Firmware driver defines

The following section lists the various define and macros of the module.

117.3.1 MDMA

MDMA

Block Repeat Destination address Update Mode

LL_MDMA_BLK_RPT_DEST_ADDR_INCREMENT

Destination address pointer is incremented after each block transfer by Destination Update Value

LL_MDMA_BLK_RPT_DEST_ADDR_DECREMENT

Destination address pointer is decremented after each block transfer by Destination Update Value

Bufferable Write Mode

LL_MDMA_BUFF_WRITE_DISABLE

destination write operation is non-bufferable

LL_MDMA_BUFF_WRITE_ENABLE

destination write operation is bufferable

Byte Endianness

LL_MDMA_BYTE_ENDIANNESS_PRESERVE

Little endianness preserved for bytes

LL_MDMA_BYTE_ENDIANNESS_EXCHANGE

byte order exchanged when destination data size is half word , word or double word

CHANNEL

LL_MDMA_CHANNEL_0

LL_MDMA_CHANNEL_1

LL_MDMA_CHANNEL_2

LL_MDMA_CHANNEL_3

LL_MDMA_CHANNEL_4

LL_MDMA_CHANNEL_5

LL_MDMA_CHANNEL_6

LL_MDMA_CHANNEL_7

LL_MDMA_CHANNEL_8

LL_MDMA_CHANNEL_9

LL_MDMA_CHANNEL_10

LL_MDMA_CHANNEL_11

LL_MDMA_CHANNEL_12

LL_MDMA_CHANNEL_13

LL_MDMA_CHANNEL_14

LL_MDMA_CHANNEL_15

LL_MDMA_CHANNEL_ALL

Transfer Destination Burst

LL_MDMA_DEST_BURST_SINGLE

Single transfer

LL_MDMA_DEST_BURST_2BEATS

Burst 2 beats

LL_MDMA_DEST_BURST_4BEATS

Burst 4 beats

LL_MDMA_DEST_BURST_8BEATS

Burst 8 beats

LL_MDMA_DEST_BURST_16BEATS

Burst 16 beats

LL_MDMA_DEST_BURST_32BEATS

Burst 32 beats

LL_MDMA_DEST_BURST_64BEATS

Burst 64 beats

LL_MDMA_DEST_BURST_128BEATS

Burst 128 beats

Destination BUS Selection

LL_MDMA_DEST_BUS_SYSTEM_AXI

System/AXI bus is used as destination

LL_MDMA_DEST_BUS_AHB_TCM

AHB bus/TCM is used as destination

Destination Data Size

LL_MDMA_DEST_DATA_SIZE_BYTE

Destination data size is Byte

LL_MDMA_DEST_DATA_SIZE_HALFWORD

Destination data size is half word

LL_MDMA_DEST_DATA_SIZE_WORD

Destination data size is word

LL_MDMA_DEST_DATA_SIZE_DOUBLEWORD

Destination data size is double word

Destination Increment Mode

LL_MDMA_DEST_FIXED

Destination address pointer is fixed

LL_MDMA_DEST_INCREMENT

Destination address pointer is incremented after each data transfer

LL_MDMA_DEST_DECREMENT

Destination address pointer is decremented after each data transfer

Destination Increment Offset Size

LL_MDMA_DEST_INC_OFFSET_BYTE

offset is Byte (8-bit)

LL_MDMA_DEST_INC_OFFSET_HALFWORD

offset is Half Word (16-bit)

LL_MDMA_DEST_INC_OFFSET_WORD

offset is Word (32-bit)

LL_MDMA_DEST_INC_OFFSET_DOUBLEWORD

offset is Double Word (64-bit)

Half Word Endianness

LL_MDMA_HALFWORD_ENDIANNESS_PRESERVE

Little endianness preserved for half words

LL_MDMA_HALFWORD_ENDIANNESS_EXCHANGE

half word order exchanged when destination data size is word or double word

HW Trigger Selection

LL_MDMA_REQ_DMA1_STREAM0_TC

MDMA HW Trigger (request) is DMA1 Stream 0 Transfer Complete Flag

LL_MDMA_REQ_DMA1_STREAM1_TC

MDMA HW Trigger (request) is DMA1 Stream 1 Transfer Complete Flag

LL_MDMA_REQ_DMA1_STREAM2_TC

MDMA HW Trigger (request) is DMA1 Stream 2 Transfer Complete Flag

LL_MDMA_REQ_DMA1_STREAM3_TC

MDMA HW Trigger (request) is DMA1 Stream 3 Transfer Complete Flag

LL_MDMA_REQ_DMA1_STREAM4_TC

MDMA HW Trigger (request) is DMA1 Stream 4 Transfer Complete Flag

LL_MDMA_REQ_DMA1_STREAM5_TC

MDMA HW Trigger (request) is DMA1 Stream 5 Transfer Complete Flag

LL_MDMA_REQ_DMA1_STREAM6_TC

MDMA HW Trigger (request) is DMA1 Stream 6 Transfer Complete Flag

LL_MDMA_REQ_DMA1_STREAM7_TC

MDMA HW Trigger (request) is DMA1 Stream 7 Transfer Complete Flag

LL_MDMA_REQ_DMA2_STREAM0_TC

MDMA HW Trigger (request) is DMA2 Stream 0 Transfer Complete Flag

LL_MDMA_REQ_DMA2_STREAM1_TC

MDMA HW Trigger (request) is DMA2 Stream 1 Transfer Complete Flag

LL_MDMA_REQ_DMA2_STREAM2_TC

MDMA HW Trigger (request) is DMA2 Stream 2 Transfer Complete Flag

LL_MDMA_REQ_DMA2_STREAM3_TC

MDMA HW Trigger (request) is DMA2 Stream 3 Transfer Complete Flag

LL_MDMA_REQ_DMA2_STREAM4_TC

MDMA HW Trigger (request) is DMA2 Stream 4 Transfer Complete Flag

LL_MDMA_REQ_DMA2_STREAM5_TC

MDMA HW Trigger (request) is DMA2 Stream 5 Transfer Complete Flag

LL_MDMA_REQ_DMA2_STREAM6_TC

MDMA HW Trigger (request) is DMA2 Stream 6 Transfer Complete Flag

LL_MDMA_REQ_DMA2_STREAM7_TC

MDMA HW Trigger (request) is DMA2 Stream 7 Transfer Complete Flag

LL_MDMA_REQ_LTDC_LINE_IT

MDMA HW Trigger (request) is LTDC Line interrupt Flag

LL_MDMA_REQ_JPEG_INFIFO_TH

MDMA HW Trigger (request) is JPEG Input FIFO threshold Flag

LL_MDMA_REQ_JPEG_INFIFO_NF

MDMA HW Trigger (request) is JPEG Input FIFO not full Flag

LL_MDMA_REQ_JPEG_OUTFIFO_TH

MDMA HW Trigger (request) is JPEG Output FIFO threshold Flag

LL_MDMA_REQ_JPEG_OUTFIFO_NE

MDMA HW Trigger (request) is JPEG Output FIFO not empty Flag

LL_MDMA_REQ_JPEG_END_CONVERSION

MDMA HW Trigger (request) is JPEG End of conversion Flag

LL_MDMA_REQ_QUADSPI_FIFO_TH

MDMA HW Trigger (request) is QSPI FIFO threshold Flag

LL_MDMA_REQ_QUADSPI_TC

MDMA HW Trigger (request) is QSPI Transfer complete Flag

LL_MDMA_REQ_DMA2D_CLUT_TC

MDMA HW Trigger (request) is DMA2D CLUT Transfer Complete Flag

LL_MDMA_REQ_DMA2D_TC

MDMA HW Trigger (request) is DMA2D Transfer Complete Flag

LL_MDMA_REQ_DMA2D_TW

MDMA HW Trigger (request) is DMA2D Transfer Watermark Flag

LL_MDMA_REQ_SDMMC1_END_DATA

MDMA HW Trigger (request) is SDMMC1 End of Data Flag

LL_MDMA_REQ_SDMMC1_DMA_ENDBUFFER

MDMA HW Trigger (request) is SDMMC1 Internal DMA buffer End Flag : This trigger is available starting from STM32H7 Rev.B devices

LL_MDMA_REQ_SDMMC1_COMMAND_END

MDMA HW Trigger (request) is SDMMC1 Command End Flag : This trigger is available starting from STM32H7 Rev.B devices

Transfer Packing

LL_MDMA_PACK_DISABLE

Packing disabled

LL_MDMA_PACK_ENABLE

Packing enabled

Padding Alignment Mode

LL_MDMA_DATAALIGN_RIGHT

Right Aligned, padded w/ 0s (default)

LL_MDMA_DATAALIGN_RIGHT_SIGNED

Right Aligned, Sign extended , Note : this mode is allowed only if the Source data size smaller than Destination data size

LL_MDMA_DATAALIGN_LEFT

Left Aligned (padded with 0s)

Transfer Priority level

LL_MDMA_PRIORITY_LOW

Priority level : Low

LL_MDMA_PRIORITY_MEDIUM

Priority level : Medium

LL_MDMA_PRIORITY_HIGH

Priority level : High

LL_MDMA_PRIORITY_VERYHIGH

Priority level : Very_High

Request Mode

LL_MDMA_REQUEST_MODE_HW

Request mode is Hardware

LL_MDMA_REQUEST_MODE_SW

Request mode is Software

Source Block Repeat address Update Mode

LL_MDMA_BLK_RPT_SRC_ADDR_INCREMENT

Source address pointer is incremented after each block transfer by Source Update Value

LL_MDMA_BLK_RPT_SRC_ADDR_DECREMENT

Source address pointer is decremented after each block transfer by Source Update Value

Transfer Source Burst

LL_MDMA_SRC_BURST_SINGLE

Single transfer

LL_MDMA_SRC_BURST_2BEATS

Burst 2 beats

LL_MDMA_SRC_BURST_4BEATS

Burst 4 beats

LL_MDMA_SRC_BURST_8BEATS

Burst 8 beats

LL_MDMA_SRC_BURST_16BEATS

Burst 16 beats

LL_MDMA_SRC_BURST_32BEATS

Burst 32 beats

LL_MDMA_SRC_BURST_64BEATS

Burst 64 beats

LL_MDMA_SRC_BURST_128BEATS

Burst 128 beats

Source BUS Selection

LL_MDMA_SRC_BUS_SYSTEM_AXI

System/AXI bus is used as source

LL_MDMA_SRC_BUS_AHB_TCM

AHB bus/TCM is used as source

Source Data Size

LL_MDMA_SRC_DATA_SIZE_BYTE

Source data size is Byte

LL_MDMA_SRC_DATA_SIZE_HALFWORD

Source data size is half word

LL_MDMA_SRC_DATA_SIZE_WORD

Source data size is word

LL_MDMA_SRC_DATA_SIZE_DOUBLEWORD

Source data size is double word

Source Increment Mode

LL_MDMA_SRC_FIXED

Destination address pointer is fixed

LL_MDMA_SRC_INCREMENT

Destination address pointer is incremented after each data transfer

LL_MDMA_SRC_DECREMENT

Destination address pointer is decremented after each data transfer

Source Increment Offset Size

LL_MDMA_SRC_INC_OFFSET_BYTE

offset is Byte (8-bit)

LL_MDMA_SRC_INC_OFFSET_HALFWORD

offset is Half Word (16-bit)

LL_MDMA_SRC_INC_OFFSET_WORD

offset is Word (32-bit)

LL_MDMA_SRC_INC_OFFSET_DOUBLEWORD

offset is Double Word (64-bit)

Trigger Mode

LL_MDMA_BUFFER_TRANSFER

Each MDMA request (SW or HW) triggers a buffer transfer

LL_MDMA_BLOCK_TRANSFER

Each MDMA request (SW or HW) triggers a block transfer

LL_MDMA_REPEAT_BLOCK_TRANSFER

Each MDMA request (SW or HW) triggers a repeated block transfer

LL_MDMA_FULL_TRANSFER

Each MDMA request (SW or HW) triggers a Full transfer or a linked list transfer if any

Word Endianness

LL_MDMA_WORD_ENDIANNESS_PRESERVE

Little endianness preserved for words

LL_MDMA_WORD_ENDIANNESS_EXCHANGE

word order exchanged when destination data size is double word

Transfer Error Direction

LL_MDMA_READ_ERROR

Last transfer error on the channel was a related to a read access

LL_MDMA_WRITE_ERROR

Last transfer error on the channel was a related to a write access

Convert MDMAxChannely

LL_MDMA_GET_INSTANCE

Description:

- Convert MDMAx_Channely into MDMAx.

Parameters:

- __CHANNEL_INSTANCE__: MDMAx_Channely

Return value:

- MDMAx

LL_MDMA_GET_CHANNEL

Description:

- Convert MDMAx_Channely into LL_MDMA_CHANNEL_y.

Parameters:

- __CHANNEL_INSTANCE__: MDMAx_Channely

Return value:

- LL_MDMA_CHANNEL_y

LL_MDMA_GET_CHANNEL_INSTANCE

Description:

- Convert MDMA Instance MDMAx and LL_MDMA_CHANNEL_y into MDMAx_Channely.

Parameters:

- __MDMA_INSTANCE__: MDMAx
- __CHANNEL__: LL_MDMA_CHANNEL_y

Return value:

- MDMAx_Channely

Common Write and read registers macros

LL_MDMA_WriteReg

Description:

- Write a value in MDMA register.

Parameters:

- __INSTANCE__: MDMA Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_MDMA_ReadReg

Description:

- Read a value in MDMA register.

Parameters:

- __INSTANCE__: MDMA Instance
- __REG__: Register to be read

Return value:

- Register: value

118 LL OPAMP Generic Driver

118.1 OPAMP Firmware driver registers structures

118.1.1 LL_OPAMP_InitTypeDef

LL_OPAMP_InitTypeDef is defined in the stm32h7xx_ll_opamp.h

Data Fields

- ***uint32_t PowerMode***
- ***uint32_t FunctionalMode***
- ***uint32_t InputNonInverting***
- ***uint32_t InputInverting***

Field Documentation

- ***uint32_t LL_OPAMP_InitTypeDef::PowerMode***

  Set OPAMP power mode. This parameter can be a value of ***OPAMP_LL_EC_POWER_MODE***This feature can be modified afterwards using unitary function **LL_OPAMP_SetPowerMode()**.
- ***uint32_t LL_OPAMP_InitTypeDef::FunctionalMode***

  Set OPAMP functional mode by setting internal connections: OPAMP operation in standalone, follower, ... This parameter can be a value of ***OPAMP_LL_EC_FUNCTIONAL_MODE***

  Note:

  – If OPAMP is configured in mode PGA, the gain can be configured using function **LL_OPAMP_SetPGAGain()**.

  This feature can be modified afterwards using unitary function **LL_OPAMP_SetFunctionalMode()**.
- ***uint32_t LL_OPAMP_InitTypeDef::InputNonInverting***

  Set OPAMP input non-inverting connection. This parameter can be a value of ***OPAMP_LL_EC_INPUT_NONINVERTING***This feature can be modified afterwards using unitary function **LL_OPAMP_SetInputNonInverting()**.
- ***uint32_t LL_OPAMP_InitTypeDef::InputInverting***

  Set OPAMP inverting input connection. This parameter can be a value of ***OPAMP_LL_EC_INPUT_INVERTING***

  Note:

  – OPAMP inverting input is used with OPAMP in mode standalone or PGA with external capacitors for filtering circuit. Otherwise (OPAMP in mode follower), OPAMP inverting input is not used (not connected to GPIO pin), this parameter is discarded.

  This feature can be modified afterwards using unitary function **LL_OPAMP_SetInputInverting()**.

118.2 OPAMP Firmware driver API description

The following section lists the various functions of the OPAMP library.

118.2.1 Detailed description of functions

LL_OPAMP_SetMode

Function name

__STATIC_INLINE void LL_OPAMP_SetMode (OPAMP_TypeDef * OPAMPx, uint32_t Mode)

Function description

Set OPAMP mode calibration or functional.

Parameters

- **OPAMPx:** OPAMP instance
- **Mode:** This parameter can be one of the following values:
  - LL_OPAMP_MODE_FUNCTIONAL
  - LL_OPAMP_MODE_CALIBRATION

Return values

- **None:**

Notes

- OPAMP mode corresponds to functional or calibration mode: functional mode: OPAMP operation in standalone, follower, ... Set functional mode using function LL_OPAMP_SetFunctionalMode().calibration mode: offset calibration of the selected transistors differential pair NMOS or PMOS.

Reference Manual to LL API cross reference:

- CSR CALON LL_OPAMP_SetMode

LL_OPAMP_GetMode

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetMode (OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP mode calibration or functional.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_MODE_FUNCTIONAL
  - LL_OPAMP_MODE_CALIBRATION

Notes

- OPAMP mode corresponds to functional or calibration mode: functional mode: OPAMP operation in standalone, follower, ... Set functional mode using function LL_OPAMP_SetFunctionalMode().calibration mode: offset calibration of the selected transistors differential pair NMOS or PMOS.

Reference Manual to LL API cross reference:

- CSR CALON LL_OPAMP_GetMode

LL_OPAMP_SetFunctionalMode

Function name

__STATIC_INLINE void LL_OPAMP_SetFunctionalMode (OPAMP_TypeDef * OPAMPx, uint32_t FunctionalMode)

Function description

Set OPAMP functional mode by setting internal connections.

Parameters

- **OPAMPx:** OPAMP instance
- **FunctionalMode:** This parameter can be one of the following values:
  - LL_OPAMP_MODE_STANDALONE
  - LL_OPAMP_MODE_FOLLOWER
  - LL_OPAMP_MODE_PGA
  - LL_OPAMP_MODE_PGA_IO0
  - LL_OPAMP_MODE_PGA_IO0_BIAS
  - LL_OPAMP_MODE_PGA_IO0_IO1_BIAS

Return values

- **None:**

Notes

- This function reset bit of calibration mode to ensure to be in functional mode, in order to have OPAMP parameters (inputs selection, ...) set with the corresponding OPAMP mode to be effective.

Reference Manual to LL API cross reference:

- CSR VMSEL LL_OPAMP_SetFunctionalMode

LL_OPAMP_GetFunctionalMode

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetFunctionalMode (OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP functional mode from setting of internal connections.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_MODE_STANDALONE
  - LL_OPAMP_MODE_FOLLOWER
  - LL_OPAMP_MODE_PGA
  - LL_OPAMP_MODE_PGA_IO0
  - LL_OPAMP_MODE_PGA_IO0_BIAS
  - LL_OPAMP_MODE_PGA_IO0_IO1_BIAS

Reference Manual to LL API cross reference:

- CSR VMSEL LL_OPAMP_GetFunctionalMode

LL_OPAMP_SetPGAGain

Function name

__STATIC_INLINE void LL_OPAMP_SetPGAGain (OPAMP_TypeDef * OPAMPx, uint32_t PGAGain)

Function description

Set OPAMP PGA gain.

Parameters

- **OPAMPx:** OPAMP instance
- **PGAGain:** This parameter can be one of the following values:
  - LL_OPAMP_PGA_GAIN_2_OR_MINUS_1
  - LL_OPAMP_PGA_GAIN_4_OR_MINUS_3
  - LL_OPAMP_PGA_GAIN_8_OR_MINUS_7
  - LL_OPAMP_PGA_GAIN_16_OR_MINUS_15

Return values

- **None:**

Notes

- Preliminarily, OPAMP must be set in mode PGA using function LL_OPAMP_SetFunctionalMode().

Reference Manual to LL API cross reference:

- CSR PGGAIN LL_OPAMP_SetPGAGain

LL_OPAMP_GetPGAGain

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetPGAGain (OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP PGA gain.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_PGA_GAIN_2_OR_MINUS_1
  - LL_OPAMP_PGA_GAIN_4_OR_MINUS_3
  - LL_OPAMP_PGA_GAIN_8_OR_MINUS_7
  - LL_OPAMP_PGA_GAIN_16_OR_MINUS_15

Notes

- Preliminarily, OPAMP must be set in mode PGA using function LL_OPAMP_SetFunctionalMode().

Reference Manual to LL API cross reference:

- CSR PGGAIN LL_OPAMP_GetPGAGain

LL_OPAMP_SetPowerMode

Function name

__STATIC_INLINE void LL_OPAMP_SetPowerMode (OPAMP_TypeDef * OPAMPx, uint32_t PowerMode)

Function description

Set OPAMP power mode normal or highspeed.

Parameters

- **OPAMPx:** OPAMP instance
- **PowerMode:** This parameter can be one of the following values:
  - LL_OPAMP_POWERMODE_NORMAL
  - LL_OPAMP_POWERMODE_HIGHSPEED

Return values

- **None:**

Notes

- OPAMP highspeed mode allows output stage to have a better slew rate.

Reference Manual to LL API cross reference:

- CSR OPAHSM LL_OPAMP_SetPowerMode

LL_OPAMP_GetPowerMode

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetPowerMode (OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP power mode normal or highspeed.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_POWERMODE_NORMAL
  - LL_OPAMP_POWERMODE_HIGHSPEED

Notes

- OPAMP highspeed mode allows output stage to have a better slew rate.

Reference Manual to LL API cross reference:

- CSR OPAHSM LL_OPAMP_GetPowerMode

LL_OPAMP_SetInputNonInverting

Function name

__STATIC_INLINE void LL_OPAMP_SetInputNonInverting (OPAMP_TypeDef * OPAMPx, uint32_t InputNonInverting)

Function description

Set OPAMP non-inverting input connection.

Parameters

- **OPAMPx:** OPAMP instance
- **InputNonInverting:** This parameter can be one of the following values:
  - LL_OPAMP_INPUT_NONINVERT_IO0
  - LL_OPAMP_INPUT_NONINVERT_DAC
  - LL_OPAMP_INPUT_NONINVERT_DAC2 (Only for OPAMP2)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR VPSEL LL_OPAMP_SetInputNonInverting

LL_OPAMP_GetInputNonInverting

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetInputNonInverting (OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP non-inverting input connection.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_INPUT_NONINVERT_IO0
  - LL_OPAMP_INPUT_NONINVERT_DAC
  - LL_OPAMP_INPUT_NONINVERT_DAC2 (Only for OPAMP2)

Reference Manual to LL API cross reference:

- CSR VPSEL LL_OPAMP_GetInputNonInverting

LL_OPAMP_SetInputInverting

Function name

__STATIC_INLINE void LL_OPAMP_SetInputInverting (OPAMP_TypeDef * OPAMPx, uint32_t InputInverting)

Function description

Set OPAMP inverting input connection.

Parameters

- **OPAMPx:** OPAMP instance
- **InputInverting:** This parameter can be one of the following values:
  - LL_OPAMP_INPUT_INVERT_IO0
  - LL_OPAMP_INPUT_INVERT_IO1
  - LL_OPAMP_INPUT_INVERT_CONNECT_NO

Return values

- **None:**

Notes

- OPAMP inverting input is used with OPAMP in mode standalone or PGA with external capacitors for filtering circuit. Otherwise (OPAMP in mode follower), OPAMP inverting input is not used (not connected to GPIO pin).

Reference Manual to LL API cross reference:

- CSR VMSEL LL_OPAMP_SetInputInverting

LL_OPAMP_GetInputInverting

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetInputInverting (OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP inverting input connection.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_INPUT_INVERT_IO0
  - LL_OPAMP_INPUT_INVERT_IO1
  - LL_OPAMP_INPUT_INVERT_CONNECT_NO

Reference Manual to LL API cross reference:

- CSR VMSEL LL_OPAMP_GetInputInverting

LL_OPAMP_SetTrimmingMode

Function name

__STATIC_INLINE void LL_OPAMP_SetTrimmingMode (OPAMP_TypeDef * OPAMPx, uint32_t TrimmingMode)

Function description

Set OPAMP trimming mode.

Parameters

- **OPAMPx:** OPAMP instance
- **TrimmingMode:** This parameter can be one of the following values:
  - LL_OPAMP_TRIMMING_FACTORY
  - LL_OPAMP_TRIMMING_USER

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR USERTRIM LL_OPAMP_SetTrimmingMode

LL_OPAMP_GetTrimmingMode

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetTrimmingMode (OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP trimming mode.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_TRIMMING_FACTORY
  - LL_OPAMP_TRIMMING_USER

Reference Manual to LL API cross reference:

- CSR USERTRIM LL_OPAMP_GetTrimmingMode

LL_OPAMP_SetCalibrationSelection

Function name

__STATIC_INLINE void LL_OPAMP_SetCalibrationSelection (OPAMP_TypeDef * OPAMPx, uint32_t TransistorsDiffPair)

Function description

Set OPAMP offset to calibrate the selected transistors differential pair NMOS or PMOS.

Parameters

- **OPAMPx:** OPAMP instance
- **TransistorsDiffPair:** This parameter can be one of the following values:
  - LL_OPAMP_TRIMMING_NMOS (1)
  - LL_OPAMP_TRIMMING_PMOS (1)
  - LL_OPAMP_TRIMMING_NMOS_VREF_50PC_VDDA
  - LL_OPAMP_TRIMMING_PMOS_VREF_3_3PC_VDDA

  (1) Default parameters to be used for calibration using two trimming steps (one with each transistors differential pair NMOS and PMOS)

Return values

- **None:**

Notes

- Preliminarily, OPAMP must be set in mode calibration using function LL_OPAMP_SetMode().

Reference Manual to LL API cross reference:

- CSR CALSEL LL_OPAMP_SetCalibrationSelection

LL_OPAMP_GetCalibrationSelection

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetCalibrationSelection (OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP offset to calibrate the selected transistors differential pair NMOS or PMOS.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_TRIMMING_NMOS (1)
  - LL_OPAMP_TRIMMING_PMOS (1)
  - LL_OPAMP_TRIMMING_NMOS_VREF_50PC_VDDA
  - LL_OPAMP_TRIMMING_PMOS_VREF_3_3PC_VDDA

  (1) Default parameters to be used for calibration using two trimming steps (one with each transistors differential pair NMOS and PMOS)

Notes

- Preliminarily, OPAMP must be set in mode calibration using function LL_OPAMP_SetMode().

Reference Manual to LL API cross reference:

- CSR CALSEL LL_OPAMP_GetCalibrationSelection

LL_OPAMP_IsCalibrationOutputSet

Function name

__STATIC_INLINE uint32_t LL_OPAMP_IsCalibrationOutputSet (OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP calibration result of toggling output.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **State:** of bit (1 or 0).

Notes

- This functions returns: 0 if OPAMP calibration output is reset 1 if OPAMP calibration output is set

Reference Manual to LL API cross reference:

- CSR OUTCAL LL_OPAMP_IsCalibrationOutputSet

LL_OPAMP_SetTrimmingValue

Function name

__STATIC_INLINE void LL_OPAMP_SetTrimmingValue (OPAMP_TypeDef * OPAMPx, uint32_t PowerMode, uint32_t TransistorsDiffPair, uint32_t TrimmingValue)

Function description

Set OPAMP trimming factor for the selected transistors differential pair NMOS or PMOS, corresponding to the selected power mode.

Parameters

- **OPAMPx:** OPAMP instance
- **PowerMode:** This parameter can be one of the following values:
  - LL_OPAMP_POWERMODE_NORMAL
  - LL_OPAMP_POWERMODE_HIGHSPEED
- **TransistorsDiffPair:** This parameter can be one of the following values:
  - LL_OPAMP_TRIMMING_NMOS
  - LL_OPAMP_TRIMMING_PMOS
- **TrimmingValue:** 0x00...0x1F

Return values

- **None:**

Reference Manual to LL API cross reference:

- OTR TRIMOFFSETN LL_OPAMP_SetTrimmingValue
- OTR TRIMOFFSETP LL_OPAMP_SetTrimmingValue
- HSOTR TRIMHSOFFSETN LL_OPAMP_SetTrimmingValue
- HSOTR TRIMHSOFFSETP LL_OPAMP_SetTrimmingValue

LL_OPAMP_GetTrimmingValue

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetTrimmingValue (OPAMP_TypeDef * OPAMPx, uint32_t PowerMode, uint32_t TransistorsDiffPair)

Function description

Get OPAMP trimming factor for the selected transistors differential pair NMOS or PMOS, corresponding to the selected power mode.

Parameters

- **OPAMPx:** OPAMP instance
- **PowerMode:** This parameter can be one of the following values:
  - LL_OPAMP_POWERMODE_NORMAL
  - LL_OPAMP_POWERMODE_HIGHSPEED
- **TransistorsDiffPair:** This parameter can be one of the following values:
  - LL_OPAMP_TRIMMING_NMOS
  - LL_OPAMP_TRIMMING_PMOS

Return values

- **0x0...0x1F:**

Reference Manual to LL API cross reference:

- OTR TRIMOFFSETN LL_OPAMP_GetTrimmingValue
- OTR TRIMOFFSETP LL_OPAMP_GetTrimmingValue
- HSOTR TRIMHSOFFSETN LL_OPAMP_GetTrimmingValue
- HSOTR TRIMHSOFFSETP LL_OPAMP_GetTrimmingValue

LL_OPAMP_Enable

Function name

__STATIC_INLINE void LL_OPAMP_Enable (OPAMP_TypeDef * OPAMPx)

Function description

Enable OPAMP instance.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **None:**

Notes

- After enable from off state, OPAMP requires a delay to fulfill wake up time specification. Refer to device datasheet, parameter "tWAKEUP".

Reference Manual to LL API cross reference:

- CSR OPAMPXEN LL_OPAMP_Enable

LL_OPAMP_Disable

Function name

__STATIC_INLINE void LL_OPAMP_Disable (OPAMP_TypeDef * OPAMPx)

Function description

Disable OPAMP instance.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR OPAMPXEN LL_OPAMP_Disable

LL_OPAMP_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_OPAMP_IsEnabled (OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP instance enable state (0: OPAMP is disabled, 1: OPAMP is enabled)

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR OPAMPXEN LL_OPAMP_IsEnabled

LL_OPAMP_DeInit

Function name

ErrorStatus LL_OPAMP_DeInit (OPAMP_TypeDef * OPAMPx)

Function description

De-initialize registers of the selected OPAMP instance to their default reset values.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: OPAMP registers are de-initialized
  - ERROR: OPAMP registers are not de-initialized

Notes

- If comparator is locked, de-initialization by software is not possible. The only way to unlock the comparator is a device hardware reset.

LL_OPAMP_Init

Function name

ErrorStatus LL_OPAMP_Init (OPAMP_TypeDef * OPAMPx, LL_OPAMP_InitTypeDef * OPAMP_InitStruct)

Function description

Initialize some features of OPAMP instance.

Parameters

- **OPAMPx:** OPAMP instance
- **OPAMP_InitStruct:** Pointer to a LL_OPAMP_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: OPAMP registers are initialized
  - ERROR: OPAMP registers are not initialized

Notes

- This function reset bit of calibration mode to ensure to be in functional mode, in order to have OPAMP parameters (inputs selection, ...) set with the corresponding OPAMP mode to be effective.

LL_OPAMP_StructInit

Function name

void LL_OPAMP_StructInit (LL_OPAMP_InitTypeDef * OPAMP_InitStruct)

Function description

Set each LL_OPAMP_InitTypeDef field to default value.

Parameters

- **OPAMP_InitStruct:** pointer to a LL_OPAMP_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

118.3 OPAMP Firmware driver defines

The following section lists the various define and macros of the module.

118.3.1 OPAMP

OPAMP

OPAMP functional mode

LL_OPAMP_MODE_STANDALONE

OPAMP functional mode, OPAMP operation in standalone

LL_OPAMP_MODE_FOLLOWER

OPAMP functional mode, OPAMP operation in follower

LL_OPAMP_MODE_PGA

OPAMP functional mode, OPAMP operation in PGA

LL_OPAMP_MODE_PGA_IO0

In PGA mode, the inverting input is connected to VINM0 for filtering

LL_OPAMP_MODE_PGA_IO0_BIAS

In PGA mode, the inverting input is connected to VINM0

LL_OPAMP_MODE_PGA_IO0_IO1_BIAS

In PGA mode, the inverting input is connected to VINM0

Definitions of OPAMP hardware constraints delays

LL_OPAMP_DELAY_STARTUP_US

Delay for OPAMP startup time

OPAMP input inverting

LL_OPAMP_INPUT_INVERT_IO0

OPAMP inverting input connected to I/O VINM0 (PC5 for OPAMP1, PE8 for OPAMP2) Note: On this STM32 series, all OPAMPx are not available on all devices. Refer to device datasheet for more details

LL_OPAMP_INPUT_INVERT_IO1

OPAMP inverting input connected to I/0 VINM1 (PA7 for OPAMP1, PG1 for OPAMP2) Note: On this STM32 series, all OPAMPx are not available on all devices. Refer to device datasheet for more details

LL_OPAMP_INPUT_INVERT_CONNECT_NO

OPAMP inverting input not externally connected (intended for OPAMP in mode follower or PGA with positive gain without bias). Note: On this STM32 series, this literal include cases of value 0x11 for mode follower and value 0x10 for mode PGA.

OPAMP input non-inverting

LL_OPAMP_INPUT_NONINVERT_IO0

OPAMP non inverting input connected to I/O VINP0 (PB0 for OPAMP1, PE9 for OPAMP2) Note: On this STM32 series, all OPAMPx are not available on all devices. Refer to device datasheet for more details

LL_OPAMP_INPUT_NONINVERT_DAC

OPAMP non inverting input connected internally to DAC channel (DAC1_CH1 for OPAMP1, DAC1_CH2 for OPAMP2) Note: On this STM32 series, all OPAMPx are not available on all devices. Refer to device datasheet for more details

OPAMP mode calibration or functional.

LL_OPAMP_MODE_FUNCTIONAL

OPAMP functional mode

LL_OPAMP_MODE_CALIBRATION

OPAMP calibration mode

OPAMP PGA gain (relevant when OPAMP is in functional mode PGA)

LL_OPAMP_PGA_GAIN_2_OR_MINUS_1

OPAMP PGA gain 2 or -1

LL_OPAMP_PGA_GAIN_4_OR_MINUS_3

OPAMP PGA gain 4 or -3

LL_OPAMP_PGA_GAIN_8_OR_MINUS_7

OPAMP PGA gain 8 or -7

LL_OPAMP_PGA_GAIN_16_OR_MINUS_15

OPAMP PGA gain 16 or -15

OPAMP PowerMode

LL_OPAMP_POWERMODE_NORMAL

OPAMP output in normal mode

LL_OPAMP_POWERMODE_HIGHSPEED

OPAMP output in highspeed mode

OPAMP trimming mode

LL_OPAMP_TRIMMING_FACTORY

OPAMP trimming factors set to factory values

LL_OPAMP_TRIMMING_USER

OPAMP trimming factors set to user values

OPAMP trimming of transistors differential pair NMOS or PMOS

LL_OPAMP_TRIMMING_NMOS_VREF_90PC_VDDA

OPAMP trimming of transistors differential pair NMOS (internal reference voltage set to 0.9*Vdda). Default parameters to be used for calibration using two trimming steps (one with each transistors differential pair NMOS and PMOS).

LL_OPAMP_TRIMMING_NMOS_VREF_50PC_VDDA

OPAMP trimming of transistors differential pair NMOS (internal reference voltage set to 0.5*Vdda).

LL_OPAMP_TRIMMING_PMOS_VREF_10PC_VDDA

OPAMP trimming of transistors differential pair PMOS (internal reference voltage set to 0.1*Vdda). Default parameters to be used for calibration using two trimming steps (one with each transistors differential pair NMOS and PMOS).

LL_OPAMP_TRIMMING_PMOS_VREF_3_3PC_VDDA

OPAMP trimming of transistors differential pair PMOS (internal reference voltage set to 0.33*Vdda).

LL_OPAMP_TRIMMING_NMOS

OPAMP trimming of transistors differential pair NMOS (internal reference voltage set to 0.9*Vdda). Default parameters to be used for calibration using two trimming steps (one with each transistors differential pair NMOS and PMOS).

LL_OPAMP_TRIMMING_PMOS

OPAMP trimming of transistors differential pair PMOS (internal reference voltage set to 0.1*Vdda). Default parameters to be used for calibration using two trimming steps (one with each transistors differential pair NMOS and PMOS).

OPAMP helper macro

__LL_OPAMP_COMMON_INSTANCE

Description:

- Helper macro to select the OPAMP common instance to which is belonging the selected OPAMP instance.

Parameters:

- __OPAMPx__: OPAMP instance

Return value:

- OPAMP: common instance

Notes:

- OPAMP common register instance can be used to set parameters common to several OPAMP instances. Refer to functions having argument "OPAMPxy_COMMON" as parameter.

__LL_OPAMP_IS_ENABLED_ALL_COMMON_INSTANCE

Description:

- Helper macro to check if all OPAMP instances sharing the same OPAMP common instance are disabled.

Return value:

- 0: All OPAMP instances sharing the same OPAMP common instance are disabled. 1: At least one OPAMP instance sharing the same OPAMP common instance is enabled

Notes:

- This check is required by functions with setting conditioned to OPAMP state: All OPAMP instances of the OPAMP common group must be disabled. Refer to functions having argument "OPAMPxy_COMMON" as parameter.

Common write and read registers macro

LL_OPAMP_WriteReg

Description:

- Write a value in OPAMP LL_OPAMP_GetPowerModeregister.

Parameters:

- __INSTANCE__: OPAMP Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_OPAMP_ReadReg

Description:

- Read a value in OPAMP register.

Parameters:

- __INSTANCE__: OPAMP Instance
- __REG__: Register to be read

Return value:

- Register: value

119 LL PWR Generic Driver

119.1 PWR Firmware driver API description

The following section lists the various functions of the PWR library.

119.1.1 Detailed description of functions

LL_PWR_SetRegulModeDS

Function name

__STATIC_INLINE void LL_PWR_SetRegulModeDS (uint32_t RegulMode)

Function description

Set the voltage Regulator mode during deep sleep mode.

Parameters

- **RegulMode:** This parameter can be one of the following values:
  - LL_PWR_REGU_DSMODE_MAIN
  - LL_PWR_REGU_DSMODE_LOW_POWER

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 LPDS LL_PWR_SetRegulModeDS

LL_PWR_GetRegulModeDS

Function name

__STATIC_INLINE uint32_t LL_PWR_GetRegulModeDS (void)

Function description

Get the voltage Regulator mode during deep sleep mode.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_REGU_DSMODE_MAIN
  - LL_PWR_REGU_DSMODE_LOW_POWER

Reference Manual to LL API cross reference:

- CR1 LPDS LL_PWR_GetRegulModeDS

LL_PWR_EnablePVD

Function name

__STATIC_INLINE void LL_PWR_EnablePVD (void)

Function description

Enable Power Voltage Detector.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 PVDEN LL_PWR_EnablePVD

LL_PWR_DisablePVD

Function name

__STATIC_INLINE void LL_PWR_DisablePVD (void)

Function description

Disable Power Voltage Detector.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 PVDEN LL_PWR_DisablePVD

LL_PWR_IsEnabledPVD

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledPVD (void)

Function description

Check if Power Voltage Detector is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 PVDEN LL_PWR_IsEnabledPVD

LL_PWR_SetPVDLevel

Function name

__STATIC_INLINE void LL_PWR_SetPVDLevel (uint32_t PVDLevel)

Function description

Configure the voltage threshold detected by the Power Voltage Detector.

Parameters

- **PVDLevel:** This parameter can be one of the following values:
  - LL_PWR_PVDLEVEL_0
  - LL_PWR_PVDLEVEL_1
  - LL_PWR_PVDLEVEL_2
  - LL_PWR_PVDLEVEL_3
  - LL_PWR_PVDLEVEL_4
  - LL_PWR_PVDLEVEL_5
  - LL_PWR_PVDLEVEL_6
  - LL_PWR_PVDLEVEL_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 PLS LL_PWR_SetPVDLevel

LL_PWR_GetPVDLevel

Function name

__STATIC_INLINE uint32_t LL_PWR_GetPVDLevel (void)

Function description

Get the voltage threshold detection.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_PVDLEVEL_0
  - LL_PWR_PVDLEVEL_1
  - LL_PWR_PVDLEVEL_2
  - LL_PWR_PVDLEVEL_3
  - LL_PWR_PVDLEVEL_4
  - LL_PWR_PVDLEVEL_5
  - LL_PWR_PVDLEVEL_6
  - LL_PWR_PVDLEVEL_7

Reference Manual to LL API cross reference:

- CR1 PLS LL_PWR_GetPVDLevel

LL_PWR_EnableBkUpAccess

Function name

__STATIC_INLINE void LL_PWR_EnableBkUpAccess (void)

Function description

Enable access to the backup domain.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 DBP LL_PWR_EnableBkUpAccess

LL_PWR_DisableBkUpAccess

Function name

__STATIC_INLINE void LL_PWR_DisableBkUpAccess (void)

Function description

Disable access to the backup domain.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 DBP LL_PWR_DisableBkUpAccess

LL_PWR_IsEnabledBkUpAccess

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledBkUpAccess (void)

Function description

Check if the backup domain is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 DBP LL_PWR_IsEnabledBkUpAccess

LL_PWR_EnableFlashPowerDown

Function name

__STATIC_INLINE void LL_PWR_EnableFlashPowerDown (void)

Function description

Enable the Flash Power Down in Stop Mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 FLPS LL_PWR_EnableFlashPowerDown

LL_PWR_DisableFlashPowerDown

Function name

__STATIC_INLINE void LL_PWR_DisableFlashPowerDown (void)

Function description

Disable the Flash Power Down in Stop Mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 FLPS LL_PWR_DisableFlashPowerDown

LL_PWR_IsEnabledFlashPowerDown

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledFlashPowerDown (void)

Function description

Check if the Flash Power Down in Stop Mode is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 FLPS LL_PWR_IsEnabledFlashPowerDown

LL_PWR_SetStopModeRegulVoltageScaling

Function name

__STATIC_INLINE void LL_PWR_SetStopModeRegulVoltageScaling (uint32_t VoltageScaling)

Function description

Set the internal Regulator output voltage in STOP mode.

Parameters

- **VoltageScaling:** This parameter can be one of the following values:
  - LL_PWR_REGU_VOLTAGE_SVOS_SCALE3
  - LL_PWR_REGU_VOLTAGE_SVOS_SCALE4
  - LL_PWR_REGU_VOLTAGE_SVOS_SCALE5

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 SVOS LL_PWR_SetStopModeRegulVoltageScaling

LL_PWR_GetStopModeRegulVoltageScaling

Function name

__STATIC_INLINE uint32_t LL_PWR_GetStopModeRegulVoltageScaling (void)

Function description

Get the internal Regulator output voltage in STOP mode.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_REGU_VOLTAGE_SVOS_SCALE3
  - LL_PWR_REGU_VOLTAGE_SVOS_SCALE4
  - LL_PWR_REGU_VOLTAGE_SVOS_SCALE5

Reference Manual to LL API cross reference:

- CR1 SVOS LL_PWR_GetStopModeRegulVoltageScaling

LL_PWR_EnableAVD

Function name

__STATIC_INLINE void LL_PWR_EnableAVD (void)

Function description

Enable Analog Power Voltage Detector.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 AVDEN LL_PWR_EnableAVD

LL_PWR_DisableAVD

Function name

__STATIC_INLINE void LL_PWR_DisableAVD (void)

Function description

Disable Analog Power Voltage Detector.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 AVDEN LL_PWR_DisableAVD

LL_PWR_IsEnabledAVD

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledAVD (void)

Function description

Check if Analog Power Voltage Detector is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 AVDEN LL_PWR_IsEnabledAVD

LL_PWR_SetAVDLevel

Function name

__STATIC_INLINE void LL_PWR_SetAVDLevel (uint32_t AVDLevel)

Function description

Configure the voltage threshold to be detected by the Analog Power Voltage Detector.

Parameters

- **AVDLevel:** This parameter can be one of the following values:
  - LL_PWR_AVDLEVEL_0
  - LL_PWR_AVDLEVEL_1
  - LL_PWR_AVDLEVEL_2
  - LL_PWR_AVDLEVEL_3

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 ALS LL_PWR_SetAVDLevel

LL_PWR_GetAVDLevel

Function name

__STATIC_INLINE uint32_t LL_PWR_GetAVDLevel (void)

Function description

Get the Analog Voltage threshold to be detected by the Analog Power Voltage Detector.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_AVDLEVEL_0
  - LL_PWR_AVDLEVEL_1
  - LL_PWR_AVDLEVEL_2
  - LL_PWR_AVDLEVEL_3

Reference Manual to LL API cross reference:

- CR1 ALS LL_PWR_GetAVDLevel

LL_PWR_EnableBkUpRegulator

Function name

__STATIC_INLINE void LL_PWR_EnableBkUpRegulator (void)

Function description

Enable Backup Regulator.

Return values

- **None:**

Notes

- When set, the Backup Regulator (used to maintain backup SRAM content in Standby and VBAT modes) is enabled. If BRE is reset, the backup Regulator is switched off. The backup SRAM can still be used but its content will be lost in the Standby and VBAT modes. Once set, the application must wait that the Backup Regulator Ready flag (BRR) is set to indicate that the data written into the RAM will be maintained in the Standby and VBAT modes.

Reference Manual to LL API cross reference:

- CR2 BREN LL_PWR_EnableBkUpRegulator

LL_PWR_DisableBkUpRegulator

Function name

__STATIC_INLINE void LL_PWR_DisableBkUpRegulator (void)

Function description

Disable Backup Regulator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 BREN LL_PWR_DisableBkUpRegulator

LL_PWR_IsEnabledBkUpRegulator

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledBkUpRegulator (void)

Function description

Check if the backup Regulator is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 BREN LL_PWR_IsEnabledBkUpRegulator

LL_PWR_EnableMonitoring

Function name

__STATIC_INLINE void LL_PWR_EnableMonitoring (void)

Function description

Enable VBAT and Temperature monitoring.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 MONEN LL_PWR_EnableMonitoring

LL_PWR_DisableMonitoring

Function name

__STATIC_INLINE void LL_PWR_DisableMonitoring (void)

Function description

Disable VBAT and Temperature monitoring.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 MONEN LL_PWR_DisableMonitoring

LL_PWR_IsEnabledMonitoring

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledMonitoring (void)

Function description

Check if the VBAT and Temperature monitoring is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 MONEN LL_PWR_IsEnabledMonitoring

LL_PWR_ConfigSupply

Function name

__STATIC_INLINE void LL_PWR_ConfigSupply (uint32_t SupplySource)

Function description

Configure the PWR supply.

Parameters

- **SupplySource:** This parameter can be one of the following values:
  - LL_PWR_LDO_SUPPLY
  - LL_PWR_DIRECT_SMPS_SUPPLY
  - LL_PWR_SMPS_1V8_SUPPLIES_LDO
  - LL_PWR_SMPS_2V5_SUPPLIES_LDO
  - LL_PWR_SMPS_1V8_SUPPLIES_EXT_AND_LDO
  - LL_PWR_SMPS_2V5_SUPPLIES_EXT_AND_LDO
  - LL_PWR_SMPS_1V8_SUPPLIES_EXT
  - LL_PWR_SMPS_2V5_SUPPLIES_EXT
  - LL_PWR_EXTERNAL_SOURCE_SUPPLY

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 BYPASS LL_PWR_ConfigSupply
- CR3 LDOEN LL_PWR_ConfigSupply
- CR3 SMPSEN LL_PWR_ConfigSupply
- CR3 SMPSEXTHP LL_PWR_ConfigSupply
- CR3 SMPSLEVEL LL_PWR_ConfigSupply

LL_PWR_GetSupply

Function name

__STATIC_INLINE uint32_t LL_PWR_GetSupply (void)

Function description

Get the PWR supply.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_LDO_SUPPLY
  - LL_PWR_DIRECT_SMPS_SUPPLY
  - LL_PWR_SMPS_1V8_SUPPLIES_LDO
  - LL_PWR_SMPS_2V5_SUPPLIES_LDO
  - LL_PWR_SMPS_1V8_SUPPLIES_EXT_AND_LDO
  - LL_PWR_SMPS_2V5_SUPPLIES_EXT_AND_LDO
  - LL_PWR_SMPS_1V8_SUPPLIES_EXT
  - LL_PWR_SMPS_2V5_SUPPLIES_EXT
  - LL_PWR_EXTERNAL_SOURCE_SUPPLY

Reference Manual to LL API cross reference:

- CR3 BYPASS LL_PWR_GetSupply
- CR3 LDOEN LL_PWR_GetSupply
- CR3 SMPSEN LL_PWR_GetSupply
- CR3 SMPSEXTHP LL_PWR_GetSupply
- CR3 SMPSLEVEL LL_PWR_GetSupply

LL_PWR_EnableBatteryCharging

Function name

__STATIC_INLINE void LL_PWR_EnableBatteryCharging (void)

Function description

Enable battery charging.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 VBE LL_PWR_EnableBatteryCharging

LL_PWR_DisableBatteryCharging

Function name

__STATIC_INLINE void LL_PWR_DisableBatteryCharging (void)

Function description

Disable battery charging.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 VBE LL_PWR_DisableBatteryCharging

LL_PWR_IsEnabledBatteryCharging

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledBatteryCharging (void)

Function description

Check if battery charging is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 VBE LL_PWR_IsEnabledBatteryCharging

LL_PWR_SetBattChargResistor

Function name

__STATIC_INLINE void LL_PWR_SetBattChargResistor (uint32_t Resistor)

Function description

Set the Battery charge resistor impedance.

Parameters

- **Resistor:** This parameter can be one of the following values:
  - LL_PWR_BATT_CHARG_RESISTOR_5K
  - LL_PWR_BATT_CHARGRESISTOR_1_5K

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 VBRS LL_PWR_SetBattChargResistor

LL_PWR_GetBattChargResistor

Function name

__STATIC_INLINE uint32_t LL_PWR_GetBattChargResistor (void)

Function description

Get the Battery charge resistor impedance.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_BATT_CHARG_RESISTOR_5K
  - LL_PWR_BATT_CHARGRESISTOR_1_5K

Reference Manual to LL API cross reference:

- CR3 VBRS LL_PWR_GetBattChargResistor

LL_PWR_EnableUSBReg

Function name

__STATIC_INLINE void LL_PWR_EnableUSBReg (void)

Function description

Enable the USB regulator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 USBREGEN LL_PWR_EnableUSBReg

LL_PWR_DisableUSBReg

Function name

__STATIC_INLINE void LL_PWR_DisableUSBReg (void)

Function description

Disable the USB regulator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 USBREGEN LL_PWR_DisableUSBReg

LL_PWR_IsEnabledUSBReg

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledUSBReg (void)

Function description

Check if the USB regulator is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 USBREGEN LL_PWR_IsEnabledUSBReg

LL_PWR_EnableUSBVoltageDetector

Function name

__STATIC_INLINE void LL_PWR_EnableUSBVoltageDetector (void)

Function description

Enable the USB voltage detector.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 USB33DEN LL_PWR_EnableUSBVoltageDetector

LL_PWR_DisableUSBVoltageDetector

Function name

__STATIC_INLINE void LL_PWR_DisableUSBVoltageDetector (void)

Function description

Disable the USB voltage detector.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 USB33DEN LL_PWR_DisableUSBVoltageDetector

LL_PWR_IsEnabledUSBVoltageDetector

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledUSBVoltageDetector (void)

Function description

Check if the USB voltage detector is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 USB33DEN LL_PWR_IsEnabledUSBVoltageDetector

LL_PWR_CPU_SetD1PowerMode

Function name

__STATIC_INLINE void LL_PWR_CPU_SetD1PowerMode (uint32_t PDMode)

Function description

Set the D1 domain Power Down mode when the CPU enters deepsleep.

Parameters

- **PDMode:** This parameter can be one of the following values:
  - LL_PWR_CPU_MODE_D1STOP
  - LL_PWR_CPU_MODE_D1STANDBY

Return values

- **None:**

Reference Manual to LL API cross reference:

- CPUCR PDDS_D1 LL_PWR_CPU_SetD1PowerMode

LL_PWR_CPU2_SetD1PowerMode

Function name

__STATIC_INLINE void LL_PWR_CPU2_SetD1PowerMode (uint32_t PDMode)

Function description

Set the D1 domain Power Down mode when the CPU2 enters deepsleep.

Parameters

- **PDMode:** This parameter can be one of the following values:
  - LL_PWR_CPU2_MODE_D1STOP
  - LL_PWR_CPU2_MODE_D1STANDBY

Return values

- **None:**

Reference Manual to LL API cross reference:

- CPU2CR PDDS_D1 LL_PWR_CPU2_SetD1PowerMode

LL_PWR_CPU_GetD1PowerMode

Function name

__STATIC_INLINE uint32_t LL_PWR_CPU_GetD1PowerMode (void)

Function description

Get the D1 Domain Power Down mode when the CPU enters deepsleep.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_CPU_MODE_D1STOP
  - LL_PWR_CPU_MODE_D1STANDBY

Reference Manual to LL API cross reference:

- CPUCR PDDS_D1 LL_PWR_CPU_GetD1PowerMode

LL_PWR_CPU2_GetD1PowerMode

Function name

__STATIC_INLINE uint32_t LL_PWR_CPU2_GetD1PowerMode (void)

Function description

Get the D1 Domain Power Down mode when the CPU2 enters deepsleep.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_CPU2_MODE_D1STOP
  - LL_PWR_CPU2_MODE_D1STANDBY

Reference Manual to LL API cross reference:

- CPU2CR PDDS_D1 LL_PWR_CPU2_GetD1PowerMode

LL_PWR_CPU_SetD2PowerMode

Function name

__STATIC_INLINE void LL_PWR_CPU_SetD2PowerMode (uint32_t PDMode)

Function description

Set the D2 domain Power Down mode when the CPU enters deepsleep.

Parameters

- **PDMode:** This parameter can be one of the following values:
  - LL_PWR_CPU_MODE_D2STOP
  - LL_PWR_CPU_MODE_D2STANDBY

Return values

- **None:**

Reference Manual to LL API cross reference:

- CPUCR PDDS_D2 LL_PWR_CPU_SetD2PowerMode

LL_PWR_CPU2_SetD2PowerMode

Function name

__STATIC_INLINE void LL_PWR_CPU2_SetD2PowerMode (uint32_t PDMode)

Function description

Set the D2 domain Power Down mode when the CPU2 enters deepsleep.

Parameters

- **PDMode:** This parameter can be one of the following values:
  - LL_PWR_CPU2_MODE_D2STOP
  - LL_PWR_CPU2_MODE_D2STANDBY

Return values

- **None:**

Reference Manual to LL API cross reference:

- CPU2CR PDDS_D2 LL_PWR_CPU2_SetD2PowerMode

LL_PWR_CPU_GetD2PowerMode

Function name

__STATIC_INLINE uint32_t LL_PWR_CPU_GetD2PowerMode (void)

Function description

Get the D2 Domain Power Down mode when the CPU enters deepsleep.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_CPU_MODE_D2STOP
  - LL_PWR_CPU_MODE_D2STANDBY

Reference Manual to LL API cross reference:

- CPUCR PDDS_D2 LL_PWR_CPU_GetD2PowerMode

LL_PWR_CPU2_GetD2PowerMode

Function name

__STATIC_INLINE uint32_t LL_PWR_CPU2_GetD2PowerMode (void)

Function description

Get the D2 Domain Power Down mode when the CPU2 enters deepsleep.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_CPU2_MODE_D2STOP
  - LL_PWR_CPU2_MODE_D2STANDBY

Reference Manual to LL API cross reference:

- CPU2CR PDDS_D2 LL_PWR_CPU2_GetD2PowerMode

LL_PWR_CPU_SetD3PowerMode

Function name

__STATIC_INLINE void LL_PWR_CPU_SetD3PowerMode (uint32_t PDMode)

Function description

Set the D3 domain Power Down mode when the CPU enters deepsleep.

Parameters

- **PDMode:** This parameter can be one of the following values:
  - LL_PWR_CPU_MODE_D3STOP
  - LL_PWR_CPU_MODE_D3STANDBY

Return values

- **None:**

Reference Manual to LL API cross reference:

- CPUCR PDDS_D3 LL_PWR_CPU_SetD3PowerMode

LL_PWR_CPU2_SetD3PowerMode

Function name

__STATIC_INLINE void LL_PWR_CPU2_SetD3PowerMode (uint32_t PDMode)

Function description

Set the D3 domain Power Down mode when the CPU2 enters deepsleep.

Parameters

- **PDMode:** This parameter can be one of the following values:
  - LL_PWR_CPU2_MODE_D3STOP
  - LL_PWR_CPU2_MODE_D3STANDBY

Return values

- **None:**

Reference Manual to LL API cross reference:

- CPU2CR PDDS_D3 LL_PWR_CPU2_SetD3PowerMode

LL_PWR_CPU_GetD3PowerMode

Function name

__STATIC_INLINE uint32_t LL_PWR_CPU_GetD3PowerMode (void)

Function description

Get the D3 Domain Power Down mode when the CPU enters deepsleep.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_CPU_MODE_D3STOP
  - LL_PWR_CPU_MODE_D3STANDBY

Reference Manual to LL API cross reference:

- CPUCR PDDS_D3 LL_PWR_CPU_GetD3PowerMode

LL_PWR_CPU2_GetD3PowerMode

Function name

__STATIC_INLINE uint32_t LL_PWR_CPU2_GetD3PowerMode (void)

Function description

Get the D3 Domain Power Down mode when the CPU2 enters deepsleep.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_CPU2_MODE_D3STOP
  - LL_PWR_CPU2_MODE_D3STANDBY

Reference Manual to LL API cross reference:

- CPU2CR PDDS_D3 LL_PWR_CPU2_GetD3PowerMode

LL_PWR_HoldCPU1

Function name

__STATIC_INLINE void LL_PWR_HoldCPU1 (void)

Function description

Hold the CPU1 and allocated peripherals when exiting from STOP mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CPU2CR HOLD1 LL_PWR_HoldCPU1

LL_PWR_ReleaseCPU1

Function name

__STATIC_INLINE void LL_PWR_ReleaseCPU1 (void)

Function description

Release the CPU1 and allocated peripherals.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CPU2CR HOLD1 LL_PWR_ReleaseCPU1

LL_PWR_IsCPU1Held

Function name

__STATIC_INLINE uint32_t LL_PWR_IsCPU1Held (void)

Function description

Ckeck if the CPU1 and allocated peripherals are held.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CPU2CR HOLD1 LL_PWR_IsCPU1Held

LL_PWR_HoldCPU2

Function name

__STATIC_INLINE void LL_PWR_HoldCPU2 (void)

Function description

Hold the CPU2 and allocated peripherals when exiting from STOP mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CPUCR HOLD2 LL_PWR_HoldCPU2

LL_PWR_ReleaseCPU2

Function name

__STATIC_INLINE void LL_PWR_ReleaseCPU2 (void)

Function description

Release the CPU2 and allocated peripherals.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CPUCR HOLD2 LL_PWR_ReleaseCPU2

LL_PWR_IsCPU2Held

Function name

__STATIC_INLINE uint32_t LL_PWR_IsCPU2Held (void)

Function description

Ckeck if the CPU2 and allocated peripherals are held.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CPUCR HOLD2 LL_PWR_IsCPU2Held

LL_PWR_CPU_EnableD3RunInLowPowerMode

Function name

__STATIC_INLINE void LL_PWR_CPU_EnableD3RunInLowPowerMode (void)

Function description

D3 domain remains in Run mode regardless of CPU subsystem modes.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CPUCR RUN_D3 LL_PWR_CPU_EnableD3RunInLowPowerMode

LL_PWR_CPU2_EnableD3RunInLowPowerMode

Function name

__STATIC_INLINE void LL_PWR_CPU2_EnableD3RunInLowPowerMode (void)

Function description

D3 domain remains in Run mode regardless of CPU2 subsystem modes.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CPU2CR RUN_D3 LL_PWR_CPU2_EnableD3RunInLowPowerMode

LL_PWR_CPU_DisableD3RunInLowPowerMode

Function name

__STATIC_INLINE void LL_PWR_CPU_DisableD3RunInLowPowerMode (void)

Function description

D3 domain follows CPU subsystem modes.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CPUCR RUN_D3 LL_PWR_CPU_DisableD3RunInLowPowerMode

LL_PWR_CPU2_DisableD3RunInLowPowerMode

Function name

__STATIC_INLINE void LL_PWR_CPU2_DisableD3RunInLowPowerMode (void)

Function description

D3 domain follows CPU2 subsystem modes.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CPU2CR RUN_D3 LL_PWR_CPU2_DisableD3RunInLowPowerMode

LL_PWR_CPU_IsEnabledD3RunInLowPowerMode

Function name

__STATIC_INLINE uint32_t LL_PWR_CPU_IsEnabledD3RunInLowPowerMode (void)

Function description

Check if D3 is kept in Run mode when CPU enters low power mode.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CPUCR RUN_D3 LL_PWR_CPU_IsEnabledD3RunInLowPowerMode

LL_PWR_CPU2_IsEnabledD3RunInLowPowerMode

Function name

__STATIC_INLINE uint32_t LL_PWR_CPU2_IsEnabledD3RunInLowPowerMode (void)

Function description

Check if D3 is kept in Run mode when CPU2 enters low power mode.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CPU2CR RUN_D3 LL_PWR_CPU2_IsEnabledD3RunInLowPowerMode

LL_PWR_SetRegulVoltageScaling

Function name

__STATIC_INLINE void LL_PWR_SetRegulVoltageScaling (uint32_t VoltageScaling)

Function description

Set the main internal Regulator output voltage.

Parameters

- **VoltageScaling:** This parameter can be one of the following values:
  - LL_PWR_REGU_VOLTAGE_SCALE0
  - LL_PWR_REGU_VOLTAGE_SCALE1
  - LL_PWR_REGU_VOLTAGE_SCALE2
  - LL_PWR_REGU_VOLTAGE_SCALE3

Return values

- **None:**

Notes

- For all H7 lines except STM32H7Axxx and STM32H7Bxxx lines, VOS0 is applied when PWR_D3CR_VOS[1:0] = 0b11 and SYSCFG_PWRCR_ODEN = 0b1.

Reference Manual to LL API cross reference:

- D3CR VOS LL_PWR_SetRegulVoltageScaling

LL_PWR_GetRegulVoltageScaling

Function name

__STATIC_INLINE uint32_t LL_PWR_GetRegulVoltageScaling (void)

Function description

Get the main internal Regulator output voltage.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_REGU_VOLTAGE_SCALE0
  - LL_PWR_REGU_VOLTAGE_SCALE1
  - LL_PWR_REGU_VOLTAGE_SCALE2
  - LL_PWR_REGU_VOLTAGE_SCALE3

Notes

- For all H7 lines except STM32H7Axxx and STM32H7Bxxx lines, checking VOS0 need the check of PWR_D3CR_VOS[1:0] field and SYSCFG_PWRCR_ODEN bit.

Reference Manual to LL API cross reference:

- D3CR VOS LL_PWR_GetRegulVoltageScaling

LL_PWR_EnableWakeUpPin

Function name

__STATIC_INLINE void LL_PWR_EnableWakeUpPin (uint32_t WakeUpPin)

Function description

Enable the WakeUp PINx functionality.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3 (*)
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5 (*)
  - LL_PWR_WAKEUP_PIN6

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WKUPEPR WKUPEN1 LL_PWR_EnableWakeUpPin
- WKUPEPR WKUPEN2 LL_PWR_EnableWakeUpPin
- WKUPEPR WKUPEN3 LL_PWR_EnableWakeUpPin
- WKUPEPR WKUPEN4 LL_PWR_EnableWakeUpPin
- WKUPEPR WKUPEN5 LL_PWR_EnableWakeUpPin
- WKUPEPR WKUPEN6 LL_PWR_EnableWakeUpPin

LL_PWR_DisableWakeUpPin

Function name

__STATIC_INLINE void LL_PWR_DisableWakeUpPin (uint32_t WakeUpPin)

Function description

Disable the WakeUp PINx functionality.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3 (*)
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5 (*)
  - LL_PWR_WAKEUP_PIN6

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WKUPEPR WKUPEN1 LL_PWR_DisableWakeUpPin
- WKUPEPR WKUPEN2 LL_PWR_DisableWakeUpPin
- WKUPEPR WKUPEN3 LL_PWR_DisableWakeUpPin
- WKUPEPR WKUPEN4 LL_PWR_DisableWakeUpPin
- WKUPEPR WKUPEN5 LL_PWR_DisableWakeUpPin
- WKUPEPR WKUPEN6 LL_PWR_DisableWakeUpPin

LL_PWR_IsEnabledWakeUpPin

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledWakeUpPin (uint32_t WakeUpPin)

Function description

Check if the WakeUp PINx functionality is enabled.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3 (*)
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5 (*)
  - LL_PWR_WAKEUP_PIN6

  (*) value not defined in all devices.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- WKUPEPR WKUPEN1 LL_PWR_IsEnabledWakeUpPin
- WKUPEPR WKUPEN2 LL_PWR_IsEnabledWakeUpPin
- WKUPEPR WKUPEN3 LL_PWR_IsEnabledWakeUpPin
- WKUPEPR WKUPEN4 LL_PWR_IsEnabledWakeUpPin
- WKUPEPR WKUPEN5 LL_PWR_IsEnabledWakeUpPin
- WKUPEPR WKUPEN6 LL_PWR_IsEnabledWakeUpPin

LL_PWR_SetWakeUpPinPolarityLow

Function name

__STATIC_INLINE void LL_PWR_SetWakeUpPinPolarityLow (uint32_t WakeUpPin)

Function description

Set the Wake-Up pin polarity low for the event detection.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3 (*)
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5 (*)
  - LL_PWR_WAKEUP_PIN6

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WKUPEPR WKUPP1 LL_PWR_SetWakeUpPinPolarityLow
- WKUPEPR WKUPP2 LL_PWR_SetWakeUpPinPolarityLow
- WKUPEPR WKUPP3 LL_PWR_SetWakeUpPinPolarityLow
- WKUPEPR WKUPP4 LL_PWR_SetWakeUpPinPolarityLow
- WKUPEPR WKUPP5 LL_PWR_SetWakeUpPinPolarityLow
- WKUPEPR WKUPP6 LL_PWR_SetWakeUpPinPolarityLow

LL_PWR_SetWakeUpPinPolarityHigh

Function name

__STATIC_INLINE void LL_PWR_SetWakeUpPinPolarityHigh (uint32_t WakeUpPin)

Function description

Set the Wake-Up pin polarity high for the event detection.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3 (*)
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5 (*)
  - LL_PWR_WAKEUP_PIN6

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WKUPEPR WKUPP1 LL_PWR_SetWakeUpPinPolarityHigh
- WKUPEPR WKUPP2 LL_PWR_SetWakeUpPinPolarityHigh
- WKUPEPR WKUPP3 LL_PWR_SetWakeUpPinPolarityHigh
- WKUPEPR WKUPP4 LL_PWR_SetWakeUpPinPolarityHigh
- WKUPEPR WKUPP5 LL_PWR_SetWakeUpPinPolarityHigh
- WKUPEPR WKUPP6 LL_PWR_SetWakeUpPinPolarityHigh

LL_PWR_IsWakeUpPinPolarityLow

Function name

__STATIC_INLINE uint32_t LL_PWR_IsWakeUpPinPolarityLow (uint32_t WakeUpPin)

Function description

Get the Wake-Up pin polarity for the event detection.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3 (*)
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5 (*)
  - LL_PWR_WAKEUP_PIN6

  (*) value not defined in all devices.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- WKUPEPR WKUPP1 LL_PWR_IsWakeUpPinPolarityLow
- WKUPEPR WKUPP2 LL_PWR_IsWakeUpPinPolarityLow
- WKUPEPR WKUPP3 LL_PWR_IsWakeUpPinPolarityLow
- WKUPEPR WKUPP4 LL_PWR_IsWakeUpPinPolarityLow
- WKUPEPR WKUPP5 LL_PWR_IsWakeUpPinPolarityLow
- WKUPEPR WKUPP6 LL_PWR_IsWakeUpPinPolarityLow

LL_PWR_SetWakeUpPinPullNone

Function name

__STATIC_INLINE void LL_PWR_SetWakeUpPinPullNone (uint32_t WakeUpPin)

Function description

Set the Wake-Up pin Pull None.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3 (*)
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5 (*)
  - LL_PWR_WAKEUP_PIN6

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WKUPEPR WKUPPUPD1 LL_PWR_SetWakeUpPinPullNone
- WKUPEPR WKUPPUPD2 LL_PWR_SetWakeUpPinPullNone
- WKUPEPR WKUPPUPD3 LL_PWR_SetWakeUpPinPullNone
- WKUPEPR WKUPPUPD4 LL_PWR_SetWakeUpPinPullNone
- WKUPEPR WKUPPUPD5 LL_PWR_SetWakeUpPinPullNone
- WKUPEPR WKUPPUPD6 LL_PWR_SetWakeUpPinPullNone

LL_PWR_SetWakeUpPinPullUp

Function name

__STATIC_INLINE void LL_PWR_SetWakeUpPinPullUp (uint32_t WakeUpPin)

Function description

Set the Wake-Up pin Pull Up.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3 (*)
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5 (*)
  - LL_PWR_WAKEUP_PIN6

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WKUPEPR WKUPPUPD1 LL_PWR_SetWakeUpPinPullUp
- WKUPEPR WKUPPUPD2 LL_PWR_SetWakeUpPinPullUp
- WKUPEPR WKUPPUPD3 LL_PWR_SetWakeUpPinPullUp
- WKUPEPR WKUPPUPD4 LL_PWR_SetWakeUpPinPullUp
- WKUPEPR WKUPPUPD5 LL_PWR_SetWakeUpPinPullUp
- WKUPEPR WKUPPUPD6 LL_PWR_SetWakeUpPinPullUp

LL_PWR_SetWakeUpPinPullDown

Function name

__STATIC_INLINE void LL_PWR_SetWakeUpPinPullDown (uint32_t WakeUpPin)

Function description

Set the Wake-Up pin Pull Down.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3 (*)
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5 (*)
  - LL_PWR_WAKEUP_PIN6

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WKUPEPR WKUPPUPD1 LL_PWR_SetWakeUpPinPullDown
- WKUPEPR WKUPPUPD2 LL_PWR_SetWakeUpPinPullDown
- WKUPEPR WKUPPUPD3 LL_PWR_SetWakeUpPinPullDown
- WKUPEPR WKUPPUPD4 LL_PWR_SetWakeUpPinPullDown
- WKUPEPR WKUPPUPD5 LL_PWR_SetWakeUpPinPullDown
- WKUPEPR WKUPPUPD6 LL_PWR_SetWakeUpPinPullDown

LL_PWR_GetWakeUpPinPull

Function name

__STATIC_INLINE uint32_t LL_PWR_GetWakeUpPinPull (uint32_t WakeUpPin)

Function description

Get the Wake-Up pin pull.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3 (*)
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5 (*)
  - LL_PWR_WAKEUP_PIN6

  (*) value not defined in all devices.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_WAKEUP_PIN_NOPULL
  - LL_PWR_WAKEUP_PIN_PULLUP
  - LL_PWR_WAKEUP_PIN_PULLDOWN

Reference Manual to LL API cross reference:

- WKUPEPR WKUPPUPD1 LL_PWR_GetWakeUpPinPull
- WKUPEPR WKUPPUPD2 LL_PWR_GetWakeUpPinPull
- WKUPEPR WKUPPUPD3 LL_PWR_GetWakeUpPinPull
- WKUPEPR WKUPPUPD4 LL_PWR_GetWakeUpPinPull
- WKUPEPR WKUPPUPD5 LL_PWR_GetWakeUpPinPull
- WKUPEPR WKUPPUPD6 LL_PWR_GetWakeUpPinPull

LL_PWR_IsActiveFlag_PVDO

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_PVDO (void)

Function description

Indicate whether VDD voltage is below the selected PVD threshold.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR1 PVDO LL_PWR_IsActiveFlag_PVDO

LL_PWR_IsActiveFlag_ACTVOS

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_ACTVOS (void)

Function description

Indicate whether the voltage level is ready for current actual used VOS.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR1 ACTVOSRDY LL_PWR_IsActiveFlag_ACTVOS

LL_PWR_IsActiveFlag_AVDO

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_AVDO (void)

Function description

Indicate whether VDDA voltage is below the selected AVD threshold.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR1 AVDO LL_PWR_IsActiveFlag_AVDO

LL_PWR_IsActiveFlag_BRR

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_BRR (void)

Function description

Get Backup Regulator ready Flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 BRRDY LL_PWR_IsActiveFlag_BRR

LL_PWR_IsActiveFlag_VBATL

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_VBATL (void)

Function description

Indicate whether the VBAT level is above or below low threshold.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 VBATL LL_PWR_IsActiveFlag_VBATL

LL_PWR_IsActiveFlag_VBATH

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_VBATH (void)

Function description

Indicate whether the VBAT level is above or below high threshold.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 VBATH LL_PWR_IsActiveFlag_VBATH

LL_PWR_IsActiveFlag_TEMPL

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_TEMPL (void)

Function description

Indicate whether the CPU temperature level is above or below low threshold.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 TEMPL LL_PWR_IsActiveFlag_TEMPL

LL_PWR_IsActiveFlag_TEMPH

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_TEMPH (void)

Function description

Indicate whether the CPU temperature level is above or below high threshold.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 TEMPH LL_PWR_IsActiveFlag_TEMPH

LL_PWR_IsActiveFlag_SMPSEXT

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_SMPSEXT (void)

Function description

Indicate whether the SMPS external supply is ready or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 SMPSEXTRDY LL_PWR_IsActiveFlag_SMPSEXT

LL_PWR_IsActiveFlag_USB

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_USB (void)

Function description

Indicate whether the USB supply is ready or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 USBRDY LL_PWR_IsActiveFlag_USB

LL_PWR_IsActiveFlag_HOLD2

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_HOLD2 (void)

Function description

Get HOLD2 Flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CPUCR HOLD2F LL_PWR_IsActiveFlag_HOLD2

LL_PWR_IsActiveFlag_HOLD1

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_HOLD1 (void)

Function description

Get HOLD1 Flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CPU2CR HOLD1F LL_PWR_IsActiveFlag_HOLD1

LL_PWR_CPU_IsActiveFlag_STOP

Function name

__STATIC_INLINE uint32_t LL_PWR_CPU_IsActiveFlag_STOP (void)

Function description

Get CPU System Stop Flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CPUCR STOPF LL_PWR_CPU_IsActiveFlag_STOP

LL_PWR_CPU2_IsActiveFlag_STOP

Function name

__STATIC_INLINE uint32_t LL_PWR_CPU2_IsActiveFlag_STOP (void)

Function description

Get CPU2 System Stop Flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CPU2CR STOPF LL_PWR_CPU2_IsActiveFlag_STOP

LL_PWR_CPU_IsActiveFlag_SB

Function name

__STATIC_INLINE uint32_t LL_PWR_CPU_IsActiveFlag_SB (void)

Function description

Get CPU System Standby Flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CPUCR SBF LL_PWR_CPU_IsActiveFlag_SB

LL_PWR_CPU2_IsActiveFlag_SB

Function name

__STATIC_INLINE uint32_t LL_PWR_CPU2_IsActiveFlag_SB (void)

Function description

Get CPU2 System Standby Flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CPU2CR SBF LL_PWR_CPU2_IsActiveFlag_SB

LL_PWR_CPU_IsActiveFlag_SB_D1

Function name

__STATIC_INLINE uint32_t LL_PWR_CPU_IsActiveFlag_SB_D1 (void)

Function description

Get CPU D1 Domain Standby Flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CPUCR SBF_D1 LL_PWR_CPU_IsActiveFlag_SB_D1

LL_PWR_CPU2_IsActiveFlag_SB_D1

Function name

__STATIC_INLINE uint32_t LL_PWR_CPU2_IsActiveFlag_SB_D1 (void)

Function description

Get CPU2 D1 Domain Standby Flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CPU2CR SBF_D1 LL_PWR_CPU2_IsActiveFlag_SB_D1

LL_PWR_CPU_IsActiveFlag_SB_D2

Function name

__STATIC_INLINE uint32_t LL_PWR_CPU_IsActiveFlag_SB_D2 (void)

Function description

Get CPU D2 Domain Standby Flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CPUCR SBF_D2 LL_PWR_CPU_IsActiveFlag_SB_D2

LL_PWR_CPU2_IsActiveFlag_SB_D2

Function name

__STATIC_INLINE uint32_t LL_PWR_CPU2_IsActiveFlag_SB_D2 (void)

Function description

Get CPU2 D2 Domain Standby Flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CPU2CR SBF_D2 LL_PWR_CPU2_IsActiveFlag_SB_D2

LL_PWR_IsActiveFlag_VOS

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_VOS (void)

Function description

Indicate whether the Regulator is ready in the selected voltage range or if its output voltage is still changing to the required voltage level.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- D3CR VOSRDY LL_PWR_IsActiveFlag_VOS

LL_PWR_IsActiveFlag_WU6

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU6 (void)

Function description

Get Wake-up Flag 6.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- WKUPFR WKUPF6 LL_PWR_IsActiveFlag_WU6

LL_PWR_IsActiveFlag_WU5

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU5 (void)

Function description

Get Wake-up Flag 5.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- WKUPFR WKUPF5 LL_PWR_IsActiveFlag_WU5

LL_PWR_IsActiveFlag_WU4

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU4 (void)

Function description

Get Wake-up Flag 4.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- WKUPFR WKUPF4 LL_PWR_IsActiveFlag_WU4

LL_PWR_IsActiveFlag_WU3

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU3 (void)

Function description

Get Wake-up Flag 3.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- WKUPFR WKUPF3 LL_PWR_IsActiveFlag_WU3

LL_PWR_IsActiveFlag_WU2

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU2 (void)

Function description

Get Wake-up Flag 2.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- WKUPFR WKUPF2 LL_PWR_IsActiveFlag_WU2

LL_PWR_IsActiveFlag_WU1

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU1 (void)

Function description

Get Wake-up Flag 1.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- WKUPFR WKUPF1 LL_PWR_IsActiveFlag_WU1

LL_PWR_ClearFlag_CPU

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_CPU (void)

Function description

Clear CPU STANDBY, STOP and HOLD flags.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CPUCR CSSF LL_PWR_ClearFlag_CPU

LL_PWR_ClearFlag_CPU2

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_CPU2 (void)

Function description

Clear CPU2 STANDBY, STOP and HOLD flags.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CPU2CR CSSF LL_PWR_ClearFlag_CPU2

LL_PWR_ClearFlag_WU6

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU6 (void)

Function description

Clear Wake-up Flag 6.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WKUPCR WKUPC6 LL_PWR_ClearFlag_WU6

LL_PWR_ClearFlag_WU5

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU5 (void)

Function description

Clear Wake-up Flag 5.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WKUPCR WKUPC5 LL_PWR_ClearFlag_WU5

LL_PWR_ClearFlag_WU4

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU4 (void)

Function description

Clear Wake-up Flag 4.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WKUPCR WKUPC4 LL_PWR_ClearFlag_WU4

LL_PWR_ClearFlag_WU3

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU3 (void)

Function description

Clear Wake-up Flag 3.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WKUPCR WKUPC3 LL_PWR_ClearFlag_WU3

LL_PWR_ClearFlag_WU2

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU2 (void)

Function description

Clear Wake-up Flag 2.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WKUPCR WKUPC2 LL_PWR_ClearFlag_WU2

LL_PWR_ClearFlag_WU1

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU1 (void)

Function description

Clear Wake-up Flag 1.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WKUPCR WKUPC1 LL_PWR_ClearFlag_WU1

LL_PWR_DeInit

Function name

ErrorStatus LL_PWR_DeInit (void)

Function description

De-initialize the PWR registers to their default reset values.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: PWR registers are de-initialized
  - ERROR: not applicable

119.2 PWR Firmware driver defines

The following section lists the various define and macros of the module.

119.2.1 PWR

PWR

Power Analog Voltage Level Detector

LL_PWR_AVDLEVEL_0

Analog Voltage threshold detected by AVD 1.7 V

LL_PWR_AVDLEVEL_1

Analog Voltage threshold detected by AVD 2.1 V

LL_PWR_AVDLEVEL_2

Analog Voltage threshold detected by AVD 2.5 V

LL_PWR_AVDLEVEL_3

Analog Voltage threshold detected by AVD 2.8 V

Battery Charge Resistor

LL_PWR_BATT_CHARG_RESISTOR_5K

Charge the Battery through a 5 kO resistor

LL_PWR_BATT_CHARGRESISTOR_1_5K

Charge the Battery through a 1.5 kO resistor

Clear Flags Defines

LL_PWR_FLAG_CPU_CSSF

Clear flags for CPU

LL_PWR_FLAG_CPU2_CSSF

Clear flags for CPU2

LL_PWR_FLAG_WKUPCR_WKUPC6

Clear PC1 WKUP flag

LL_PWR_FLAG_WKUPCR_WKUPC5

Clear PI11 WKUP flag

LL_PWR_FLAG_WKUPCR_WKUPC4

Clear PC13 WKUP flag

LL_PWR_FLAG_WKUPCR_WKUPC3

Clear PI8 WKUP flag

LL_PWR_FLAG_WKUPCR_WKUPC2

Clear PA2 WKUP flag

LL_PWR_FLAG_WKUPCR_WKUPC1

Clear PA0 WKUP flag

Get Flags Defines

LL_PWR_FLAG_AVDO

Analog voltage detector output on VDDA flag

LL_PWR_FLAG_PVDO

Programmable voltage detect output flag

LL_PWR_FLAG_ACTVOS

Current VOS applied for VCORE voltage scaling flag

LL_PWR_FLAG_ACTVOSRDY

Ready bit for current actual used VOS for VCORE voltage scaling flag

LL_PWR_FLAG_TEMPH

Temperature high threshold flag

LL_PWR_FLAG_TEMPL

Temperature low threshold flag

LL_PWR_FLAG_VBATH

VBAT high threshold flag

LL_PWR_FLAG_VBATL

VBAT low threshold flag

LL_PWR_FLAG_BRRDY

Backup Regulator ready flag

LL_PWR_FLAG_USBRDY

USB supply ready flag

LL_PWR_FLAG_SMPSEXTRDY

SMPS External supply ready flag

LL_PWR_FLAG_CPU_SBF_D2

D2 domain DSTANDBY Flag

LL_PWR_FLAG_CPU_SBF_D1

D1 domain DSTANDBY Flag

LL_PWR_FLAG_CPU_SBF

System STANDBY Flag

LL_PWR_FLAG_CPU_STOPF

STOP Flag

LL_PWR_FLAG_CPU_HOLD2F

CPU2 in hold wakeup flag

LL_PWR_FLAG_CPU2_SBF_D2

D2 domain DSTANDBY Flag

LL_PWR_FLAG_CPU2_SBF_D1

D1 domain DSTANDBY Flag

LL_PWR_FLAG_CPU2_SBF

System STANDBY Flag

LL_PWR_FLAG_CPU2_STOPF

STOP Flag

LL_PWR_FLAG_CPU2_HOLD1F

CPU1 in hold wakeup flag

LL_PWR_D3CR_VOSRDY

Voltage scaling ready flag

LL_PWR_WKUPFR_WKUPF6

Wakeup flag on PC1

LL_PWR_WKUPFR_WKUPF5

Wakeup flag on PI11

LL_PWR_WKUPFR_WKUPF4

Wakeup flag on PC13

LL_PWR_WKUPFR_WKUPF3

Wakeup flag on PI8

LL_PWR_WKUPFR_WKUPF2

Wakeup flag on PA2

LL_PWR_WKUPFR_WKUPF1

Wakeup flag on PA0

Power mode

LL_PWR_CPU_MODE_D1STOP

Enter D1 domain to Stop mode when the CPU enters deepsleep

LL_PWR_CPU_MODE_D1STANDBY

Enter D1 domain to Standby mode when the CPU enters deepsleep

LL_PWR_CPU_MODE_D2STOP

Enter D2 domain to Stop mode when the CPU enters deepsleep

LL_PWR_CPU_MODE_D2STANDBY

Enter D2 domain to Standby mode when the CPU enters deepsleep

LL_PWR_CPU_MODE_D3RUN

Keep system D3 domain in Run mode when the CPU enter deepsleep

LL_PWR_CPU_MODE_D3STOP

Enter D3 domain to Stop mode when the CPU enters deepsleep

LL_PWR_CPU_MODE_D3STANDBY

Enter D3 domain to Standby mode when the CPU enters deepsleep

LL_PWR_CPU2_MODE_D1STOP

Enter D1 domain to Stop mode when the CPU2 enters deepsleep

LL_PWR_CPU2_MODE_D1STANDBY

Enter D1 domain to Standby mode when the CPU2 enters deepsleep

LL_PWR_CPU2_MODE_D2STOP

Enter D2 domain to Stop mode when the CPU2 enters deepsleep

LL_PWR_CPU2_MODE_D2STANDBY

Enter D2 domain to Standby mode when the CPU2 enters deepsleep

LL_PWR_CPU2_MODE_D3RUN

Keep system D3 domain in RUN mode when the CPU2 enter deepsleep

LL_PWR_CPU2_MODE_D3STOP

Enter D3 domain to Stop mode when the CPU2 enters deepsleep

LL_PWR_CPU2_MODE_D3STANDBY

Enter D3 domain to Standby mode when the CPU2 enter deepsleep

Power Digital Voltage Level Detector

LL_PWR_PVDLEVEL_0

Voltage threshold detected by PVD 1.95 V

LL_PWR_PVDLEVEL_1

Voltage threshold detected by PVD 2.1 V

LL_PWR_PVDLEVEL_2

Voltage threshold detected by PVD 2.25 V

LL_PWR_PVDLEVEL_3

Voltage threshold detected by PVD 2.4 V

LL_PWR_PVDLEVEL_4

Voltage threshold detected by PVD 2.55 V

LL_PWR_PVDLEVEL_5

Voltage threshold detected by PVD 2.7 V

LL_PWR_PVDLEVEL_6

Voltage threshold detected by PVD 2.85 V

LL_PWR_PVDLEVEL_7

External voltage level on PVD_IN pin, compared to internal VREFINT level.

Regulator Mode In Deep Sleep Mode

LL_PWR_REGU_DSMODE_MAIN

Voltage Regulator in main mode during deepsleep mode

LL_PWR_REGU_DSMODE_LOW_POWER

Voltage Regulator in low-power mode during deepsleep mode

Run mode Regulator Voltage Scaling

LL_PWR_REGU_VOLTAGE_SCALE3

Select voltage scale 3

LL_PWR_REGU_VOLTAGE_SCALE2

Select voltage scale 2

LL_PWR_REGU_VOLTAGE_SCALE1

Select voltage scale 1

LL_PWR_REGU_VOLTAGE_SCALE0

Select voltage scale 0

Stop mode Regulator Voltage Scaling

LL_PWR_REGU_VOLTAGE_SVOS_SCALE5

Select voltage scale 5 when system enters STOP mode

LL_PWR_REGU_VOLTAGE_SVOS_SCALE4

Select voltage scale 4 when system enters STOP mode

LL_PWR_REGU_VOLTAGE_SVOS_SCALE3

Select voltage scale 3 when system enters STOP mode

Power supply source configuration

LL_PWR_LDO_SUPPLY

Core domains are supplied from the LDO

LL_PWR_DIRECT_SMPS_SUPPLY

Core domains are supplied from the SMPS

LL_PWR_SMPS_1V8_SUPPLIES_LDO

The SMPS 1.8V output supplies the LDO which supplies the Core domains

LL_PWR_SMPS_2V5_SUPPLIES_LDO

The SMPS 2.5V output supplies the LDO which supplies the Core domains

LL_PWR_SMPS_1V8_SUPPLIES_EXT_AND_LDO

The SMPS 1.8V output supplies an external circuits and the LDO. The Core domains are supplied from the LDO

LL_PWR_SMPS_2V5_SUPPLIES_EXT_AND_LDO

The SMPS 2.5V output supplies an external circuits and the LDO. The Core domains are supplied from the LDO

LL_PWR_SMPS_1V8_SUPPLIES_EXT

The SMPS 1.8V output supplies an external source which supplies the Core domains

LL_PWR_SMPS_2V5_SUPPLIES_EXT

The SMPS 2.5V output supplies an external source which supplies the Core domains

LL_PWR_EXTERNAL_SOURCE_SUPPLY

The SMPS and the LDO are Bypassed. The Core domains are supplied from an external source

Wakeup Pins

LL_PWR_WAKEUP_PIN1

Wake-Up pin 1 : PA0

LL_PWR_WAKEUP_PIN2

Wake-Up pin 2 : PA2

LL_PWR_WAKEUP_PIN3

Wake-Up pin 3 : PI8

LL_PWR_WAKEUP_PIN4

Wake-Up pin 4 : PC13

LL_PWR_WAKEUP_PIN5

Wake-Up pin 5 : PI11

LL_PWR_WAKEUP_PIN6

Wake-Up pin 6 : PC1

Wakeup Pins pull configuration

LL_PWR_WAKEUP_PIN_NOPULL

Configure Wake-Up pin in no pull

LL_PWR_WAKEUP_PIN_PULLUP

Configure Wake-Up pin in pull Up

LL_PWR_WAKEUP_PIN_PULLDOWN

Configure Wake-Up pin in pull Down

Common write and read registers Macros

LL_PWR_WriteReg

Description:

- Write a value in PWR register.

Parameters:

- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_PWR_ReadReg

Description:

- Read a value in PWR register.

Parameters:

- __REG__: Register to be read

Return value:

- Register: value

Wake-Up Pins register offsets Defines

LL_PWR_WAKEUP_PINS_PULL_SHIFT_OFFSET

LL_PWR_WAKEUP_PINS_MAX_SHIFT_MASK

120 LL RCC Generic Driver

120.1 RCC Firmware driver registers structures

120.1.1 LL_RCC_ClocksTypeDef

LL_RCC_ClocksTypeDef is defined in the stm32h7xx_ll_rcc.h

Data Fields

- ***uint32_t SYSCLK_Frequency***
- ***uint32_t CPUCLK_Frequency***
- ***uint32_t HCLK_Frequency***
- ***uint32_t PCLK1_Frequency***
- ***uint32_t PCLK2_Frequency***
- ***uint32_t PCLK3_Frequency***
- ***uint32_t PCLK4_Frequency***

Field Documentation

- ***uint32_t LL_RCC_ClocksTypeDef::SYSCLK_Frequency***
- ***uint32_t LL_RCC_ClocksTypeDef::CPUCLK_Frequency***
- ***uint32_t LL_RCC_ClocksTypeDef::HCLK_Frequency***
- ***uint32_t LL_RCC_ClocksTypeDef::PCLK1_Frequency***
- ***uint32_t LL_RCC_ClocksTypeDef::PCLK2_Frequency***
- ***uint32_t LL_RCC_ClocksTypeDef::PCLK3_Frequency***
- ***uint32_t LL_RCC_ClocksTypeDef::PCLK4_Frequency***

120.1.2 LL_PLL_ClocksTypeDef

LL_PLL_ClocksTypeDef is defined in the stm32h7xx_ll_rcc.h

Data Fields

- ***uint32_t PLL_P_Frequency***
- ***uint32_t PLL_Q_Frequency***
- ***uint32_t PLL_R_Frequency***

Field Documentation

- ***uint32_t LL_PLL_ClocksTypeDef::PLL_P_Frequency***
- ***uint32_t LL_PLL_ClocksTypeDef::PLL_Q_Frequency***
- ***uint32_t LL_PLL_ClocksTypeDef::PLL_R_Frequency***

120.2 RCC Firmware driver API description

The following section lists the various functions of the RCC library.

120.2.1 Detailed description of functions

LL_RCC_HSE_EnableCSS

Function name

__STATIC_INLINE void LL_RCC_HSE_EnableCSS (void)

Function description

Enable the Clock Security System.

Return values

- **None:**

Notes

- Once HSE Clock Security System is enabled it cannot be changed anymore unless a reset occurs or system enter in standby mode.

Reference Manual to LL API cross reference:

- CR CSSHSEON LL_RCC_HSE_EnableCSS

LL_RCC_HSE_EnableBypass

Function name

__STATIC_INLINE void LL_RCC_HSE_EnableBypass (void)

Function description

Enable HSE external oscillator (HSE Bypass)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSEBYP LL_RCC_HSE_EnableBypass

LL_RCC_HSE_DisableBypass

Function name

__STATIC_INLINE void LL_RCC_HSE_DisableBypass (void)

Function description

Disable HSE external oscillator (HSE Bypass)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSEBYP LL_RCC_HSE_DisableBypass

LL_RCC_HSE_Enable

Function name

__STATIC_INLINE void LL_RCC_HSE_Enable (void)

Function description

Enable HSE crystal oscillator (HSE ON)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSEON LL_RCC_HSE_Enable

LL_RCC_HSE_Disable

Function name

__STATIC_INLINE void LL_RCC_HSE_Disable (void)

Function description

Disable HSE crystal oscillator (HSE ON)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSEON LL_RCC_HSE_Disable

LL_RCC_HSE_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_HSE_IsReady (void)

Function description

Check if HSE oscillator Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR HSERDY LL_RCC_HSE_IsReady

LL_RCC_HSI_Enable

Function name

__STATIC_INLINE void LL_RCC_HSI_Enable (void)

Function description

Enable HSI oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSION LL_RCC_HSI_Enable

LL_RCC_HSI_Disable

Function name

__STATIC_INLINE void LL_RCC_HSI_Disable (void)

Function description

Disable HSI oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSION LL_RCC_HSI_Disable

LL_RCC_HSI_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI_IsReady (void)

Function description

Check if HSI clock is ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR HSIRDY LL_RCC_HSI_IsReady

LL_RCC_HSI_IsDividerReady

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI_IsDividerReady (void)

Function description

Check if HSI new divider applied and ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR HSIDIVF LL_RCC_HSI_IsDividerReady

LL_RCC_HSI_SetDivider

Function name

__STATIC_INLINE void LL_RCC_HSI_SetDivider (uint32_t Divider)

Function description

Set HSI divider.

Parameters

- **Divider:** This parameter can be one of the following values:
  - LL_RCC_HSI_DIV1
  - LL_RCC_HSI_DIV2
  - LL_RCC_HSI_DIV4
  - LL_RCC_HSI_DIV8

Return values

- **None.:**

Reference Manual to LL API cross reference:

- CR HSIDIV LL_RCC_HSI_SetDivider

LL_RCC_HSI_GetDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI_GetDivider (void)

Function description

Get HSI divider.

Return values

- **can:** be one of the following values:
  - LL_RCC_HSI_DIV1
  - LL_RCC_HSI_DIV2
  - LL_RCC_HSI_DIV4
  - LL_RCC_HSI_DIV8

Reference Manual to LL API cross reference:

- CR HSIDIV LL_RCC_HSI_GetDivider

LL_RCC_HSI_EnableStopMode

Function name

__STATIC_INLINE void LL_RCC_HSI_EnableStopMode (void)

Function description

Enable HSI oscillator in Stop mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSIKERON LL_RCC_HSI_EnableStopMode

LL_RCC_HSI_DisableStopMode

Function name

__STATIC_INLINE void LL_RCC_HSI_DisableStopMode (void)

Function description

Disable HSI oscillator in Stop mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSION LL_RCC_HSI_DisableStopMode

LL_RCC_HSI_GetCalibration

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI_GetCalibration (void)

Function description

Get HSI Calibration value.

Return values

- **A:** value between 0 and 4095 (0xFFF)

Notes

- When HSITRIM is written, HSICAL is updated with the sum of HSITRIM and the factory trim value

Reference Manual to LL API cross reference:

- HSICFGR HSICAL LL_RCC_HSI_GetCalibration

LL_RCC_HSI_SetCalibTrimming

Function name

__STATIC_INLINE void LL_RCC_HSI_SetCalibTrimming (uint32_t Value)

Function description

Set HSI Calibration trimming.

Parameters

- **Value:** can be a value between 0 and 127 (63 for Cut1.x)

Return values

- **None:**

Notes

- user-programmable trimming value that is added to the HSICAL
- Default value is 64 (32 for Cut1.x), which, when added to the HSICAL value, should trim the HSI to 64 MHz +/- 1 %

Reference Manual to LL API cross reference:

- HSICFGR HSITRIM LL_RCC_HSI_SetCalibTrimming

LL_RCC_HSI_GetCalibTrimming

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI_GetCalibTrimming (void)

Function description

Get HSI Calibration trimming.

Return values

- **A:** value between 0 and 127 (63 for Cut1.x)

Reference Manual to LL API cross reference:

- HSICFGR HSITRIM LL_RCC_HSI_GetCalibTrimming

LL_RCC_CSI_Enable

Function name

__STATIC_INLINE void LL_RCC_CSI_Enable (void)

Function description

Enable CSI oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CSION LL_RCC_CSI_Enable

LL_RCC_CSI_Disable

Function name

__STATIC_INLINE void LL_RCC_CSI_Disable (void)

Function description

Disable CSI oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CSION LL_RCC_CSI_Disable

LL_RCC_CSI_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_CSI_IsReady (void)

Function description

Check if CSI clock is ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CSIRDY LL_RCC_CSI_IsReady

LL_RCC_CSI_EnableStopMode

Function name

__STATIC_INLINE void LL_RCC_CSI_EnableStopMode (void)

Function description

Enable CSI oscillator in Stop mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CSIKERON LL_RCC_CSI_EnableStopMode

LL_RCC_CSI_DisableStopMode

Function name

__STATIC_INLINE void LL_RCC_CSI_DisableStopMode (void)

Function description

Disable CSI oscillator in Stop mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CSIKERON LL_RCC_CSI_DisableStopMode

LL_RCC_CSI_GetCalibration

Function name

__STATIC_INLINE uint32_t LL_RCC_CSI_GetCalibration (void)

Function description

Get CSI Calibration value.

Return values

- **A:** value between 0 and 255 (0xFF)

Notes

- When CSITRIM is written, CSICAL is updated with the sum of CSITRIM and the factory trim value

Reference Manual to LL API cross reference:

- CSICFGR CSICAL LL_RCC_CSI_GetCalibration

LL_RCC_CSI_SetCalibTrimming

Function name

__STATIC_INLINE void LL_RCC_CSI_SetCalibTrimming (uint32_t Value)

Function description

Set CSI Calibration trimming.

Parameters

- **Value:** can be a value between 0 and 31

Return values

- **None:**

Notes

- user-programmable trimming value that is added to the CSICAL
- Default value is 16, which, when added to the CSICAL value, should trim the CSI to 4 MHz +/- 1 %

Reference Manual to LL API cross reference:

- CSICFGR CSITRIM LL_RCC_CSI_SetCalibTrimming

LL_RCC_CSI_GetCalibTrimming

Function name

__STATIC_INLINE uint32_t LL_RCC_CSI_GetCalibTrimming (void)

Function description

Get CSI Calibration trimming.

Return values

- **A:** value between 0 and 31

Reference Manual to LL API cross reference:

- CSICFGR CSITRIM LL_RCC_CSI_GetCalibTrimming

LL_RCC_HSI48_Enable

Function name

__STATIC_INLINE void LL_RCC_HSI48_Enable (void)

Function description

Enable HSI48 oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSI48ON LL_RCC_HSI48_Enable

LL_RCC_HSI48_Disable

Function name

__STATIC_INLINE void LL_RCC_HSI48_Disable (void)

Function description

Disable HSI48 oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSI48ON LL_RCC_HSI48_Disable

LL_RCC_HSI48_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI48_IsReady (void)

Function description

Check if HSI48 clock is ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR HSI48RDY LL_RCC_HSI48_IsReady

LL_RCC_HSI48_GetCalibration

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI48_GetCalibration (void)

Function description

Get HSI48 Calibration value.

Return values

- **A:** value between 0 and 1023 (0x3FF)

Notes

- When HSI48TRIM is written, HSI48CAL is updated with the sum of HSI48TRIM and the factory trim value

Reference Manual to LL API cross reference:

- CRRCR HSI48CAL LL_RCC_HSI48_GetCalibration

LL_RCC_D1CK_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_D1CK_IsReady (void)

Function description

Check if D1 clock is ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR D1CKRDY LL_RCC_D1CK_IsReady

LL_RCC_D2CK_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_D2CK_IsReady (void)

Function description

Check if D2 clock is ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR D2CKRDY LL_RCC_D2CK_IsReady

LL_RCC_WWDG1_EnableSystemReset

Function name

__STATIC_INLINE void LL_RCC_WWDG1_EnableSystemReset (void)

Function description

Enable system wide reset for Window Watch Dog 1.

Return values

- **None.:**

Reference Manual to LL API cross reference:

- GCR WW1RSC LL_RCC_WWDG1_EnableSystemReset

LL_RCC_WWDG1_IsSystemReset

Function name

__STATIC_INLINE uint32_t LL_RCC_WWDG1_IsSystemReset (void)

Function description

Check if Window Watch Dog 1 reset is system wide.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- GCR WW1RSC LL_RCC_WWDG1_IsSystemReset

LL_RCC_WWDG2_EnableSystemReset

Function name

__STATIC_INLINE void LL_RCC_WWDG2_EnableSystemReset (void)

Function description

Enable system wide reset for Window Watch Dog 2.

Return values

- **None.:**

Reference Manual to LL API cross reference:

- GCR WW1RSC LL_RCC_WWDG2_EnableSystemReset

LL_RCC_WWDG2_IsSystemReset

Function name

__STATIC_INLINE uint32_t LL_RCC_WWDG2_IsSystemReset (void)

Function description

Check if Window Watch Dog 2 reset is system wide.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- GCR WW2RSC LL_RCC_WWDG2_IsSystemReset

LL_RCC_ForceCM4Boot

Function name

__STATIC_INLINE void LL_RCC_ForceCM4Boot (void)

Function description

Force CM4 boot (if hold by option byte BCM4 = 0)

Return values

- **None.:**

Reference Manual to LL API cross reference:

- GCR BOOT_C2 LL_RCC_ForceCM4Boot

LL_RCC_IsCM4BootForced

Function name

__STATIC_INLINE uint32_t LL_RCC_IsCM4BootForced (void)

Function description

Check if CM4 boot is forced.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- GCR BOOT_C2 LL_RCC_IsCM4BootForced

LL_RCC_ForceCM7Boot

Function name

__STATIC_INLINE void LL_RCC_ForceCM7Boot (void)

Function description

Force CM7 boot (if hold by option byte BCM7 = 0)

Return values

- **None.:**

Reference Manual to LL API cross reference:

- GCR BOOT_C1 LL_RCC_ForceCM7Boot

LL_RCC_IsCM7BootForced

Function name

__STATIC_INLINE uint32_t LL_RCC_IsCM7BootForced (void)

Function description

Check if CM7 boot is forced.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- GCR BOOT_C1 LL_RCC_IsCM7BootForced

LL_RCC_LSE_EnableCSS

Function name

__STATIC_INLINE void LL_RCC_LSE_EnableCSS (void)

Function description

Enable the Clock Security System on LSE.

Return values

- **None:**

Notes

- Once LSE Clock Security System is enabled it cannot be changed anymore unless a clock failure is detected.

Reference Manual to LL API cross reference:

- BDCR LSECSSON LL_RCC_LSE_EnableCSS

LL_RCC_LSE_IsFailureDetected

Function name

__STATIC_INLINE uint32_t LL_RCC_LSE_IsFailureDetected (void)

Function description

Check if LSE failure is detected by Clock Security System.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDCR LSECSSD LL_RCC_LSE_IsFailureDetected

LL_RCC_LSE_Enable

Function name

__STATIC_INLINE void LL_RCC_LSE_Enable (void)

Function description

Enable Low Speed External (LSE) crystal.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSEON LL_RCC_LSE_Enable

LL_RCC_LSE_Disable

Function name

__STATIC_INLINE void LL_RCC_LSE_Disable (void)

Function description

Disable Low Speed External (LSE) crystal.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSEON LL_RCC_LSE_Disable

LL_RCC_LSE_EnableBypass

Function name

__STATIC_INLINE void LL_RCC_LSE_EnableBypass (void)

Function description

Enable external clock source (LSE bypass).

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSEBYP LL_RCC_LSE_EnableBypass

LL_RCC_LSE_DisableBypass

Function name

__STATIC_INLINE void LL_RCC_LSE_DisableBypass (void)

Function description

Disable external clock source (LSE bypass).

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSEBYP LL_RCC_LSE_DisableBypass

LL_RCC_LSE_SetDriveCapability

Function name

__STATIC_INLINE void LL_RCC_LSE_SetDriveCapability (uint32_t LSEDrive)

Function description

Set LSE oscillator drive capability.

Parameters

- **LSEDrive:** This parameter can be one of the following values:
  - LL_RCC_LSEDRIVE_LOW
  - LL_RCC_LSEDRIVE_MEDIUMLOW
  - LL_RCC_LSEDRIVE_MEDIUMHIGH
  - LL_RCC_LSEDRIVE_HIGH

Return values

- **None:**

Notes

- The oscillator is in Xtal mode when it is not in bypass mode.

Reference Manual to LL API cross reference:

- BDCR LSEDRV LL_RCC_LSE_SetDriveCapability

LL_RCC_LSE_GetDriveCapability

Function name

__STATIC_INLINE uint32_t LL_RCC_LSE_GetDriveCapability (void)

Function description

Get LSE oscillator drive capability.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_LSEDRIVE_LOW
  - LL_RCC_LSEDRIVE_MEDIUMLOW
  - LL_RCC_LSEDRIVE_MEDIUMHIGH
  - LL_RCC_LSEDRIVE_HIGH

Reference Manual to LL API cross reference:

- BDCR LSEDRV LL_RCC_LSE_GetDriveCapability

LL_RCC_LSE_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_LSE_IsReady (void)

Function description

Check if LSE oscillator Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDCR LSERDY LL_RCC_LSE_IsReady

LL_RCC_LSI_Enable

Function name

__STATIC_INLINE void LL_RCC_LSI_Enable (void)

Function description

Enable LSI Oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR LSION LL_RCC_LSI_Enable

LL_RCC_LSI_Disable

Function name

__STATIC_INLINE void LL_RCC_LSI_Disable (void)

Function description

Disable LSI Oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR LSION LL_RCC_LSI_Disable

LL_RCC_LSI_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_LSI_IsReady (void)

Function description

Check if LSI is Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR LSIRDY LL_RCC_LSI_IsReady

LL_RCC_SetSysClkSource

Function name

__STATIC_INLINE void LL_RCC_SetSysClkSource (uint32_t Source)

Function description

Configure the system clock source.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_SYS_CLKSOURCE_HSI
  - LL_RCC_SYS_CLKSOURCE_CSI
  - LL_RCC_SYS_CLKSOURCE_HSE
  - LL_RCC_SYS_CLKSOURCE_PLL1

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR SW LL_RCC_SetSysClkSource

LL_RCC_GetSysClkSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSysClkSource (void)

Function description

Get the system clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SYS_CLKSOURCE_STATUS_HSI
  - LL_RCC_SYS_CLKSOURCE_STATUS_CSI
  - LL_RCC_SYS_CLKSOURCE_STATUS_HSE
  - LL_RCC_SYS_CLKSOURCE_STATUS_PLL1

Reference Manual to LL API cross reference:

- CFGR SWS LL_RCC_GetSysClkSource

LL_RCC_SetSysWakeUpClkSource

Function name

__STATIC_INLINE void LL_RCC_SetSysWakeUpClkSource (uint32_t Source)

Function description

Configure the system wakeup clock source.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_SYSWAKEUP_CLKSOURCE_HSI
  - LL_RCC_SYSWAKEUP_CLKSOURCE_CSI

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR STOPWUCK LL_RCC_SetSysWakeUpClkSource

LL_RCC_GetSysWakeUpClkSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSysWakeUpClkSource (void)

Function description

Get the system wakeup clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SYSWAKEUP_CLKSOURCE_HSI
  - LL_RCC_SYSWAKEUP_CLKSOURCE_CSI

Reference Manual to LL API cross reference:

- CFGR STOPWUCK LL_RCC_GetSysWakeUpClkSource

LL_RCC_SetKerWakeUpClkSource

Function name

__STATIC_INLINE void LL_RCC_SetKerWakeUpClkSource (uint32_t Source)

Function description

Configure the kernel wakeup clock source.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_KERWAKEUP_CLKSOURCE_HSI
  - LL_RCC_KERWAKEUP_CLKSOURCE_CSI

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR STOPKERWUCK LL_RCC_SetKerWakeUpClkSource

LL_RCC_GetKerWakeUpClkSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetKerWakeUpClkSource (void)

Function description

Get the kernel wakeup clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_KERWAKEUP_CLKSOURCE_HSI
  - LL_RCC_KERWAKEUP_CLKSOURCE_CSI

Reference Manual to LL API cross reference:

- CFGR STOPKERWUCK LL_RCC_GetKerWakeUpClkSource

LL_RCC_SetSysPrescaler

Function name

__STATIC_INLINE void LL_RCC_SetSysPrescaler (uint32_t Prescaler)

Function description

Set System prescaler.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_SYSCLK_DIV_1
  - LL_RCC_SYSCLK_DIV_2
  - LL_RCC_SYSCLK_DIV_4
  - LL_RCC_SYSCLK_DIV_8
  - LL_RCC_SYSCLK_DIV_16
  - LL_RCC_SYSCLK_DIV_64
  - LL_RCC_SYSCLK_DIV_128
  - LL_RCC_SYSCLK_DIV_256
  - LL_RCC_SYSCLK_DIV_512

Return values

- **None:**

Reference Manual to LL API cross reference:

- D1CFGR/CDCFGR1 D1CPRE/CDCPRE LL_RCC_SetSysPrescaler

LL_RCC_SetAHBPrescaler

Function name

__STATIC_INLINE void LL_RCC_SetAHBPrescaler (uint32_t Prescaler)

Function description

Set AHB prescaler.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_AHB_DIV_1
  - LL_RCC_AHB_DIV_2
  - LL_RCC_AHB_DIV_4
  - LL_RCC_AHB_DIV_8
  - LL_RCC_AHB_DIV_16
  - LL_RCC_AHB_DIV_64
  - LL_RCC_AHB_DIV_128
  - LL_RCC_AHB_DIV_256
  - LL_RCC_AHB_DIV_512

Return values

- **None:**

Reference Manual to LL API cross reference:

- D1CFGR/CDCFGR1 HPRE LL_RCC_SetAHBPrescaler

LL_RCC_SetAPB1Prescaler

Function name

__STATIC_INLINE void LL_RCC_SetAPB1Prescaler (uint32_t Prescaler)

Function description

Set APB1 prescaler.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_APB1_DIV_1
  - LL_RCC_APB1_DIV_2
  - LL_RCC_APB1_DIV_4
  - LL_RCC_APB1_DIV_8
  - LL_RCC_APB1_DIV_16

Return values

- **None:**

Reference Manual to LL API cross reference:

- D2CFGR/CDCFGR2 D2PPRE1/CDPPRE1 LL_RCC_SetAPB1Prescaler

LL_RCC_SetAPB2Prescaler

Function name

__STATIC_INLINE void LL_RCC_SetAPB2Prescaler (uint32_t Prescaler)

Function description

Set APB2 prescaler.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_APB2_DIV_1
  - LL_RCC_APB2_DIV_2
  - LL_RCC_APB2_DIV_4
  - LL_RCC_APB2_DIV_8
  - LL_RCC_APB2_DIV_16

Return values

- **None:**

Reference Manual to LL API cross reference:

- D2CFGR/CDCFGR2 D2PPRE2/CDPPRE2 LL_RCC_SetAPB2Prescaler

LL_RCC_SetAPB3Prescaler

Function name

__STATIC_INLINE void LL_RCC_SetAPB3Prescaler (uint32_t Prescaler)

Function description

Set APB3 prescaler.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_APB3_DIV_1
  - LL_RCC_APB3_DIV_2
  - LL_RCC_APB3_DIV_4
  - LL_RCC_APB3_DIV_8
  - LL_RCC_APB3_DIV_16

Return values

- **None:**

Reference Manual to LL API cross reference:

- D1CFGR/CDCFGR1 D1PPRE/CDPPRE LL_RCC_SetAPB3Prescaler

LL_RCC_SetAPB4Prescaler

Function name

__STATIC_INLINE void LL_RCC_SetAPB4Prescaler (uint32_t Prescaler)

Function description

Set APB4 prescaler.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_APB4_DIV_1
  - LL_RCC_APB4_DIV_2
  - LL_RCC_APB4_DIV_4
  - LL_RCC_APB4_DIV_8
  - LL_RCC_APB4_DIV_16

Return values

- **None:**

Reference Manual to LL API cross reference:

- D3CFGR/SRDCFGR D3PPRE/SRDPPRE LL_RCC_SetAPB4Prescaler

LL_RCC_GetSysPrescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSysPrescaler (void)

Function description

Get System prescaler.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SYSCLK_DIV_1
  - LL_RCC_SYSCLK_DIV_2
  - LL_RCC_SYSCLK_DIV_4
  - LL_RCC_SYSCLK_DIV_8
  - LL_RCC_SYSCLK_DIV_16
  - LL_RCC_SYSCLK_DIV_64
  - LL_RCC_SYSCLK_DIV_128
  - LL_RCC_SYSCLK_DIV_256
  - LL_RCC_SYSCLK_DIV_512

Reference Manual to LL API cross reference:

- D1CFGR/CDCFGR1 D1CPRE/CDCPRE LL_RCC_GetSysPrescaler

LL_RCC_GetAHBPrescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetAHBPrescaler (void)

Function description

Get AHB prescaler.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_AHB_DIV_1
  - LL_RCC_AHB_DIV_2
  - LL_RCC_AHB_DIV_4
  - LL_RCC_AHB_DIV_8
  - LL_RCC_AHB_DIV_16
  - LL_RCC_AHB_DIV_64
  - LL_RCC_AHB_DIV_128
  - LL_RCC_AHB_DIV_256
  - LL_RCC_AHB_DIV_512

Reference Manual to LL API cross reference:

- D1CFGR/ CDCFGR1 HPRE LL_RCC_GetAHBPrescaler

LL_RCC_GetAPB1Prescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetAPB1Prescaler (void)

Function description

Get APB1 prescaler.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_APB1_DIV_1
  - LL_RCC_APB1_DIV_2
  - LL_RCC_APB1_DIV_4
  - LL_RCC_APB1_DIV_8
  - LL_RCC_APB1_DIV_16

Reference Manual to LL API cross reference:

- D2CFGR/CDCFGR2 D2PPRE1/CDPPRE1 LL_RCC_GetAPB1Prescaler

LL_RCC_GetAPB2Prescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetAPB2Prescaler (void)

Function description

Get APB2 prescaler.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_APB2_DIV_1
  - LL_RCC_APB2_DIV_2
  - LL_RCC_APB2_DIV_4
  - LL_RCC_APB2_DIV_8
  - LL_RCC_APB2_DIV_16

Reference Manual to LL API cross reference:

- D2CFGR/CDCFGR2 D2PPRE2/CDPPRE2 LL_RCC_GetAPB2Prescaler

LL_RCC_GetAPB3Prescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetAPB3Prescaler (void)

Function description

Get APB3 prescaler.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_APB3_DIV_1
  - LL_RCC_APB3_DIV_2
  - LL_RCC_APB3_DIV_4
  - LL_RCC_APB3_DIV_8
  - LL_RCC_APB3_DIV_16

Reference Manual to LL API cross reference:

- D1CFGR/CDCFGR1 D1PPRE/CDPPRE LL_RCC_GetAPB3Prescaler

LL_RCC_GetAPB4Prescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetAPB4Prescaler (void)

Function description

Get APB4 prescaler.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_APB4_DIV_1
  - LL_RCC_APB4_DIV_2
  - LL_RCC_APB4_DIV_4
  - LL_RCC_APB4_DIV_8
  - LL_RCC_APB4_DIV_16

Reference Manual to LL API cross reference:

- D3CFGR/SRDCFGR D3PPRE/SRDPPRE LL_RCC_GetAPB4Prescaler

LL_RCC_ConfigMCO

Function name

__STATIC_INLINE void LL_RCC_ConfigMCO (uint32_t MCOxSource, uint32_t MCOxPrescaler)

Function description

Configure MCOx.

Parameters

- **MCOxSource:** This parameter can be one of the following values:
  - LL_RCC_MCO1SOURCE_HSI
  - LL_RCC_MCO1SOURCE_LSE
  - LL_RCC_MCO1SOURCE_HSE
  - LL_RCC_MCO1SOURCE_PLL1QCLK
  - LL_RCC_MCO1SOURCE_HSI48
  - LL_RCC_MCO2SOURCE_SYSCLK
  - LL_RCC_MCO2SOURCE_PLL2PCLK
  - LL_RCC_MCO2SOURCE_HSE
  - LL_RCC_MCO2SOURCE_PLL1PCLK
  - LL_RCC_MCO2SOURCE_CSI
  - LL_RCC_MCO2SOURCE_LSI
- **MCOxPrescaler:** This parameter can be one of the following values:
  - LL_RCC_MCO1_DIV_1
  - LL_RCC_MCO1_DIV_2
  - LL_RCC_MCO1_DIV_3
  - LL_RCC_MCO1_DIV_4
  - LL_RCC_MCO1_DIV_5
  - LL_RCC_MCO1_DIV_6
  - LL_RCC_MCO1_DIV_7
  - LL_RCC_MCO1_DIV_8
  - LL_RCC_MCO1_DIV_9
  - LL_RCC_MCO1_DIV_10
  - LL_RCC_MCO1_DIV_11
  - LL_RCC_MCO1_DIV_12
  - LL_RCC_MCO1_DIV_13
  - LL_RCC_MCO1_DIV_14
  - LL_RCC_MCO1_DIV_15
  - LL_RCC_MCO2_DIV_1
  - LL_RCC_MCO2_DIV_2
  - LL_RCC_MCO2_DIV_3
  - LL_RCC_MCO2_DIV_4
  - LL_RCC_MCO2_DIV_5
  - LL_RCC_MCO2_DIV_6
  - LL_RCC_MCO2_DIV_7
  - LL_RCC_MCO2_DIV_8
  - LL_RCC_MCO2_DIV_9
  - LL_RCC_MCO2_DIV_10
  - LL_RCC_MCO2_DIV_11
  - LL_RCC_MCO2_DIV_12
  - LL_RCC_MCO2_DIV_13
  - LL_RCC_MCO2_DIV_14
  - LL_RCC_MCO2_DIV_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR MCO1 LL_RCC_ConfigMCO
- CFGR MCO1PRE LL_RCC_ConfigMCO
- CFGR MCO2 LL_RCC_ConfigMCO
- CFGR MCO2PRE LL_RCC_ConfigMCO

LL_RCC_SetClockSource

Function name

__STATIC_INLINE void LL_RCC_SetClockSource (uint32_t ClkSource)

Function description

Configure periph clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_USART16_CLKSOURCE_PCLK2
  - LL_RCC_USART16_CLKSOURCE_PLL2Q
  - LL_RCC_USART16_CLKSOURCE_PLL3Q
  - LL_RCC_USART16_CLKSOURCE_HSI
  - LL_RCC_USART16_CLKSOURCE_CSI
  - LL_RCC_USART16_CLKSOURCE_LSE
  - LL_RCC_USART234578_CLKSOURCE_PCLK1
  - LL_RCC_USART234578_CLKSOURCE_PLL2Q
  - LL_RCC_USART234578_CLKSOURCE_PLL3Q
  - LL_RCC_USART234578_CLKSOURCE_HSI
  - LL_RCC_USART234578_CLKSOURCE_CSI
  - LL_RCC_USART234578_CLKSOURCE_LSE
  - LL_RCC_I2C123_CLKSOURCE_PCLK1
  - LL_RCC_I2C123_CLKSOURCE_PLL3R
  - LL_RCC_I2C123_CLKSOURCE_HSI
  - LL_RCC_I2C123_CLKSOURCE_CSI
  - LL_RCC_I2C4_CLKSOURCE_PCLK4
  - LL_RCC_I2C4_CLKSOURCE_PLL3R
  - LL_RCC_I2C4_CLKSOURCE_HSI
  - LL_RCC_I2C4_CLKSOURCE_CSI
  - LL_RCC_LPTIM1_CLKSOURCE_PCLK1
  - LL_RCC_LPTIM1_CLKSOURCE_PLL2P
  - LL_RCC_LPTIM1_CLKSOURCE_PLL3R
  - LL_RCC_LPTIM1_CLKSOURCE_LSE
  - LL_RCC_LPTIM1_CLKSOURCE_LSI
  - LL_RCC_LPTIM1_CLKSOURCE_CLKP
  - LL_RCC_LPTIM2_CLKSOURCE_PCLK4
  - LL_RCC_LPTIM2_CLKSOURCE_PLL2P
  - LL_RCC_LPTIM2_CLKSOURCE_PLL3R
  - LL_RCC_LPTIM2_CLKSOURCE_LSE
  - LL_RCC_LPTIM2_CLKSOURCE_LSI
  - LL_RCC_LPTIM2_CLKSOURCE_CLKP
  - LL_RCC_LPTIM345_CLKSOURCE_PCLK4
  - LL_RCC_LPTIM345_CLKSOURCE_PLL2P
  - LL_RCC_LPTIM345_CLKSOURCE_PLL3R
  - LL_RCC_LPTIM345_CLKSOURCE_LSE
  - LL_RCC_LPTIM345_CLKSOURCE_LSI
  - LL_RCC_LPTIM345_CLKSOURCE_CLKP
  - LL_RCC_SAI1_CLKSOURCE_PLL1Q
  - LL_RCC_SAI1_CLKSOURCE_PLL2P
  - LL_RCC_SAI1_CLKSOURCE_PLL3P
  - LL_RCC_SAI1_CLKSOURCE_I2S_CKIN
  - LL_RCC_SAI1_CLKSOURCE_CLKP
  - LL_RCC_SAI23_CLKSOURCE_PLL1Q (*)
  - LL_RCC_SAI23_CLKSOURCE_PLL2P (*)
  - LL_RCC_SAI23_CLKSOURCE_PLL3P (*)
  - LL_RCC_SAI23_CLKSOURCE_I2S_CKIN (*)
  - LL_RCC_SAI23_CLKSOURCE_CLKP (*)
  - LL_RCC_SAI4A_CLKSOURCE_PLL1Q (*)
  - LL_RCC_SAI4A_CLKSOURCE_PLL2P (*)

Return values

- **None:**

Reference Manual to LL API cross reference:

- D2CCIP1R/CDCCIP1R * LL_RCC_SetClockSource
- D2CCIP2R/CDCCIP2R * LL_RCC_SetClockSource
- D3CCIPR/SRDCCIPR * LL_RCC_SetClockSource

LL_RCC_SetUSARTClockSource

Function name

__STATIC_INLINE void LL_RCC_SetUSARTClockSource (uint32_t ClkSource)

Function description

Configure USARTx clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_USART16_CLKSOURCE_PCLK2
  - LL_RCC_USART16_CLKSOURCE_PLL2Q
  - LL_RCC_USART16_CLKSOURCE_PLL3Q
  - LL_RCC_USART16_CLKSOURCE_HSI
  - LL_RCC_USART16_CLKSOURCE_CSI
  - LL_RCC_USART16_CLKSOURCE_LSE
  - LL_RCC_USART234578_CLKSOURCE_PCLK1
  - LL_RCC_USART234578_CLKSOURCE_PLL2Q
  - LL_RCC_USART234578_CLKSOURCE_PLL3Q
  - LL_RCC_USART234578_CLKSOURCE_HSI
  - LL_RCC_USART234578_CLKSOURCE_CSI
  - LL_RCC_USART234578_CLKSOURCE_LSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- D2CCIP2R / D2CCIP2R USART16SEL LL_RCC_SetUSARTClockSource
- D2CCIP2R / D2CCIP2R USART28SEL LL_RCC_SetUSARTClockSource

LL_RCC_SetLPUARTClockSource

Function name

__STATIC_INLINE void LL_RCC_SetLPUARTClockSource (uint32_t ClkSource)

Function description

Configure LPUARTx clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_LPUART1_CLKSOURCE_PCLK4
  - LL_RCC_LPUART1_CLKSOURCE_PLL2Q
  - LL_RCC_LPUART1_CLKSOURCE_PLL3Q
  - LL_RCC_LPUART1_CLKSOURCE_HSI
  - LL_RCC_LPUART1_CLKSOURCE_CSI
  - LL_RCC_LPUART1_CLKSOURCE_LSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- D3CCIPR / SRDCCIPR LPUART1SEL LL_RCC_SetLPUARTClockSource

LL_RCC_SetI2CClockSource

Function name

__STATIC_INLINE void LL_RCC_SetI2CClockSource (uint32_t ClkSource)

Function description

Configure I2Cx clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_I2C123_CLKSOURCE_PCLK1
  - LL_RCC_I2C123_CLKSOURCE_PLL3R
  - LL_RCC_I2C123_CLKSOURCE_HSI
  - LL_RCC_I2C123_CLKSOURCE_CSI
  - LL_RCC_I2C4_CLKSOURCE_PCLK4
  - LL_RCC_I2C4_CLKSOURCE_PLL3R
  - LL_RCC_I2C4_CLKSOURCE_HSI
  - LL_RCC_I2C4_CLKSOURCE_CSI

Return values

- **None:**

Reference Manual to LL API cross reference:

- D2CCIP2R / CDCCIP2R I2C123SEL LL_RCC_SetI2CClockSource
- D3CCIPR / SRDCCIPR I2C4SEL LL_RCC_SetI2CClockSource

LL_RCC_SetLPTIMClockSource

Function name

__STATIC_INLINE void LL_RCC_SetLPTIMClockSource (uint32_t ClkSource)

Function description

Configure LPTIMx clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_LPTIM1_CLKSOURCE_PCLK1
  - LL_RCC_LPTIM1_CLKSOURCE_PLL2P
  - LL_RCC_LPTIM1_CLKSOURCE_PLL3R
  - LL_RCC_LPTIM1_CLKSOURCE_LSE
  - LL_RCC_LPTIM1_CLKSOURCE_LSI
  - LL_RCC_LPTIM1_CLKSOURCE_CLKP
  - LL_RCC_LPTIM2_CLKSOURCE_PCLK4
  - LL_RCC_LPTIM2_CLKSOURCE_PLL2P
  - LL_RCC_LPTIM2_CLKSOURCE_PLL3R
  - LL_RCC_LPTIM2_CLKSOURCE_LSE
  - LL_RCC_LPTIM2_CLKSOURCE_LSI
  - LL_RCC_LPTIM2_CLKSOURCE_CLKP
  - LL_RCC_LPTIM345_CLKSOURCE_PCLK4
  - LL_RCC_LPTIM345_CLKSOURCE_PLL2P
  - LL_RCC_LPTIM345_CLKSOURCE_PLL3R
  - LL_RCC_LPTIM345_CLKSOURCE_LSE
  - LL_RCC_LPTIM345_CLKSOURCE_LSI
  - LL_RCC_LPTIM345_CLKSOURCE_CLKP

Return values

- **None:**

Reference Manual to LL API cross reference:

- D2CCIP2R / CDCCIP2R LPTIM1SEL LL_RCC_SetLPTIMClockSource D3CCIPR / SRDCCIPR LPTIM2SEL LL_RCC_SetLPTIMClockSource
- D3CCIPR / SRDCCIPR LPTIM345SEL LL_RCC_SetLPTIMClockSource

LL_RCC_SetSAIClockSource

Function name

__STATIC_INLINE void LL_RCC_SetSAIClockSource (uint32_t ClkSource)

Function description

Configure SAIx clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_SAI1_CLKSOURCE_PLL1Q
  - LL_RCC_SAI1_CLKSOURCE_PLL2P
  - LL_RCC_SAI1_CLKSOURCE_PLL3P
  - LL_RCC_SAI1_CLKSOURCE_I2S_CKIN
  - LL_RCC_SAI1_CLKSOURCE_CLKP
  - LL_RCC_SAI23_CLKSOURCE_PLL1Q (*)
  - LL_RCC_SAI23_CLKSOURCE_PLL2P (*)
  - LL_RCC_SAI23_CLKSOURCE_PLL3P (*)
  - LL_RCC_SAI23_CLKSOURCE_I2S_CKIN (*)
  - LL_RCC_SAI23_CLKSOURCE_CLKP (*)
  - LL_RCC_SAI4A_CLKSOURCE_PLL1Q (*)
  - LL_RCC_SAI4A_CLKSOURCE_PLL2P (*)
  - LL_RCC_SAI4A_CLKSOURCE_PLL3P (*)
  - LL_RCC_SAI4A_CLKSOURCE_I2S_CKIN (*)
  - LL_RCC_SAI4A_CLKSOURCE_SPDIF (*)
  - LL_RCC_SAI2A_CLKSOURCE_PLL1Q (*)
  - LL_RCC_SAI2A_CLKSOURCE_PLL2P (*)
  - LL_RCC_SAI2A_CLKSOURCE_PLL3P (*)
  - LL_RCC_SAI2A_CLKSOURCE_I2S_CKIN (*)
  - LL_RCC_SAI2A_CLKSOURCE_CLKP (*)
  - LL_RCC_SAI2A_CLKSOURCE_SPDIF (*)
  - LL_RCC_SAI4B_CLKSOURCE_PLL1Q (*)
  - LL_RCC_SAI4B_CLKSOURCE_PLL2P (*)
  - LL_RCC_SAI4B_CLKSOURCE_PLL3P (*)
  - LL_RCC_SAI4B_CLKSOURCE_I2S_CKIN (*)
  - LL_RCC_SAI4B_CLKSOURCE_CLKP (*)
  - LL_RCC_SAI4B_CLKSOURCE_SPDIF (*)
  - LL_RCC_SAI2B_CLKSOURCE_PLL1Q (*)
  - LL_RCC_SAI2B_CLKSOURCE_PLL2P (*)
  - LL_RCC_SAI2B_CLKSOURCE_PLL3P (*)
  - LL_RCC_SAI2B_CLKSOURCE_I2S_CKIN (*)
  - LL_RCC_SAI2B_CLKSOURCE_CLKP (*)
  - LL_RCC_SAI2B_CLKSOURCE_SPDIF (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- D2CCIP1R / CDCCIP1R SAI1SEL LL_RCC_SetSAIClockSource
- D2CCIP1R / CDCCIP1R SAI23SEL LL_RCC_SetSAIClockSource D3CCIPR / SRDCCIPR SAI4ASEL LL_RCC_SetSAI4xClockSource
- D3CCIPR / SRDCCIPR SAI4BSEL LL_RCC_SetSAI4xClockSource

LL_RCC_SetSDMMCClockSource

Function name

__STATIC_INLINE void LL_RCC_SetSDMMCClockSource (uint32_t ClkSource)

Function description

Configure SDMMCx clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_SDMMC_CLKSOURCE_PLL1Q
  - LL_RCC_SDMMC_CLKSOURCE_PLL2R

Return values

- **None:**

Reference Manual to LL API cross reference:

- D1CCIPR / CDCCIPR SDMMCSEL LL_RCC_SetSDMMCClockSource

LL_RCC_SetRNGClockSource

Function name

__STATIC_INLINE void LL_RCC_SetRNGClockSource (uint32_t ClkSource)

Function description

Configure RNGx clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_RNG_CLKSOURCE_HSI48
  - LL_RCC_RNG_CLKSOURCE_PLL1Q
  - LL_RCC_RNG_CLKSOURCE_LSE
  - LL_RCC_RNG_CLKSOURCE_LSI

Return values

- **None:**

Reference Manual to LL API cross reference:

- D2CCIP2R / CDCCIP2R RNGSEL LL_RCC_SetRNGClockSource

LL_RCC_SetUSBClockSource

Function name

__STATIC_INLINE void LL_RCC_SetUSBClockSource (uint32_t ClkSource)

Function description

Configure USBx clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_USB_CLKSOURCE_DISABLE
  - LL_RCC_USB_CLKSOURCE_PLL1Q
  - LL_RCC_USB_CLKSOURCE_PLL3Q
  - LL_RCC_USB_CLKSOURCE_HSI48

Return values

- **None:**

Reference Manual to LL API cross reference:

- D2CCIP2R / CDCCIP2R USBSEL LL_RCC_SetUSBClockSource

LL_RCC_SetCECClockSource

Function name

__STATIC_INLINE void LL_RCC_SetCECClockSource (uint32_t ClkSource)

Function description

Configure CECx clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_CEC_CLKSOURCE_LSE
  - LL_RCC_CEC_CLKSOURCE_LSI
  - LL_RCC_CEC_CLKSOURCE_CSI_DIV122

Return values

- **None:**

Reference Manual to LL API cross reference:

- D2CCIP2R / CDCCIP2R CECSEL LL_RCC_SetCECClockSource

LL_RCC_SetDFSDMClockSource

Function name

__STATIC_INLINE void LL_RCC_SetDFSDMClockSource (uint32_t ClkSource)

Function description

Configure DFSDMx Kernel clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_DFSDM1_CLKSOURCE_PCLK2
  - LL_RCC_DFSDM1_CLKSOURCE_SYSCLK

Return values

- **None:**

Reference Manual to LL API cross reference:

- D2CCIP1R / CDCCIP1R DFSDM1SEL LL_RCC_SetDFSDMClockSource

LL_RCC_SetFMCClockSource

Function name

__STATIC_INLINE void LL_RCC_SetFMCClockSource (uint32_t ClkSource)

Function description

Configure FMCx Kernel clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_FMC_CLKSOURCE_HCLK
  - LL_RCC_FMC_CLKSOURCE_PLL1Q
  - LL_RCC_FMC_CLKSOURCE_PLL2R
  - LL_RCC_FMC_CLKSOURCE_CLKP

Return values

- **None:**

Reference Manual to LL API cross reference:

- D1CCIPR / CDCCIPR FMCSEL LL_RCC_SetFMCClockSource

LL_RCC_SetQSPIClockSource

Function name

__STATIC_INLINE void LL_RCC_SetQSPIClockSource (uint32_t ClkSource)

Function description

Configure QSPIx Kernel clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_QSPI_CLKSOURCE_HCLK
  - LL_RCC_QSPI_CLKSOURCE_PLL1Q
  - LL_RCC_QSPI_CLKSOURCE_PLL2R
  - LL_RCC_QSPI_CLKSOURCE_CLKP

Return values

- **None:**

Reference Manual to LL API cross reference:

- D1CCIPR QSPISEL LL_RCC_SetQSPIClockSource

LL_RCC_SetCLKPClockSource

Function name

__STATIC_INLINE void LL_RCC_SetCLKPClockSource (uint32_t ClkSource)

Function description

Configure CLKP Kernel clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_CLKP_CLKSOURCE_HSI
  - LL_RCC_CLKP_CLKSOURCE_CSI
  - LL_RCC_CLKP_CLKSOURCE_HSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- D1CCIPR / CDCCIPR CKPERSEL LL_RCC_SetCLKPClockSource

LL_RCC_SetSPIClockSource

Function name

__STATIC_INLINE void LL_RCC_SetSPIClockSource (uint32_t ClkSource)

Function description

Configure SPIx Kernel clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_SPI123_CLKSOURCE_PLL1Q
  - LL_RCC_SPI123_CLKSOURCE_PLL2P
  - LL_RCC_SPI123_CLKSOURCE_PLL3P
  - LL_RCC_SPI123_CLKSOURCE_I2S_CKIN
  - LL_RCC_SPI123_CLKSOURCE_CLKP
  - LL_RCC_SPI45_CLKSOURCE_PCLK2
  - LL_RCC_SPI45_CLKSOURCE_PLL2Q
  - LL_RCC_SPI45_CLKSOURCE_PLL3Q
  - LL_RCC_SPI45_CLKSOURCE_HSI
  - LL_RCC_SPI45_CLKSOURCE_CSI
  - LL_RCC_SPI45_CLKSOURCE_HSE
  - LL_RCC_SPI6_CLKSOURCE_PCLK4
  - LL_RCC_SPI6_CLKSOURCE_PLL2Q
  - LL_RCC_SPI6_CLKSOURCE_PLL3Q
  - LL_RCC_SPI6_CLKSOURCE_HSI
  - LL_RCC_SPI6_CLKSOURCE_CSI
  - LL_RCC_SPI6_CLKSOURCE_HSE
  - LL_RCC_SPI6_CLKSOURCE_I2S_CKIN (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- D2CCIP1R / CDCCIP1R SPI123SEL LL_RCC_SetSPIClockSource
- D2CCIP1R / CDCCIP1R SPI45SEL LL_RCC_SetSPIClockSource
- D3CCIPR / SRDCCIPR SPI6SEL LL_RCC_SetSPIClockSource

LL_RCC_SetSPDIFClockSource

Function name

__STATIC_INLINE void LL_RCC_SetSPDIFClockSource (uint32_t ClkSource)

Function description

Configure SPDIFx Kernel clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_SPDIF_CLKSOURCE_PLL1Q
  - LL_RCC_SPDIF_CLKSOURCE_PLL2R
  - LL_RCC_SPDIF_CLKSOURCE_PLL3R
  - LL_RCC_SPDIF_CLKSOURCE_HSI

Return values

- **None:**

Reference Manual to LL API cross reference:

- D2CCIP1R / CDCCIP1R SPDIFSEL LL_RCC_SetSPDIFClockSource

LL_RCC_SetFDCANClockSource

Function name

__STATIC_INLINE void LL_RCC_SetFDCANClockSource (uint32_t ClkSource)

Function description

Configure FDCANx Kernel clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_FDCAN_CLKSOURCE_HSE
  - LL_RCC_FDCAN_CLKSOURCE_PLL1Q
  - LL_RCC_FDCAN_CLKSOURCE_PLL2Q

Return values

- **None:**

Reference Manual to LL API cross reference:

- D2CCIP1R / CDCCIP1R FDCANSEL LL_RCC_SetFDCANClockSource

LL_RCC_SetSWPClockSource

Function name

__STATIC_INLINE void LL_RCC_SetSWPClockSource (uint32_t ClkSource)

Function description

Configure SWPx Kernel clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_SWP_CLKSOURCE_PCLK1
  - LL_RCC_SWP_CLKSOURCE_HSI

Return values

- **None:**

Reference Manual to LL API cross reference:

- D2CCIP1R / CDCCIP1R SWPSEL LL_RCC_SetSWPClockSource

LL_RCC_SetADCClockSource

Function name

__STATIC_INLINE void LL_RCC_SetADCClockSource (uint32_t ClkSource)

Function description

Configure ADCx Kernel clock source.

Parameters

- **ClkSource:** This parameter can be one of the following values:
  - LL_RCC_ADC_CLKSOURCE_PLL2P
  - LL_RCC_ADC_CLKSOURCE_PLL3R
  - LL_RCC_ADC_CLKSOURCE_CLKP

Return values

- **None:**

Reference Manual to LL API cross reference:

- D3CCIPR / SRDCCIPR ADCSEL LL_RCC_SetADCClockSource

LL_RCC_GetClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetClockSource (uint32_t Periph)

Function description

Get periph clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_USART16_CLKSOURCE
  - LL_RCC_USART234578_CLKSOURCE
  - LL_RCC_I2C123_CLKSOURCE
  - LL_RCC_I2C4_CLKSOURCE
  - LL_RCC_LPTIM1_CLKSOURCE
  - LL_RCC_LPTIM2_CLKSOURCE
  - LL_RCC_LPTIM345_CLKSOURCE
  - LL_RCC_SAI1_CLKSOURCE
  - LL_RCC_SAI23_CLKSOURCE
  - LL_RCC_SAI4A_CLKSOURCE (*)
  - LL_RCC_SAI4B_CLKSOURCE (*)
  - LL_RCC_SAI2A_CLKSOURCE (*)
  - LL_RCC_SAI2B_CLKSOURCE (*)
  - LL_RCC_SPI123_CLKSOURCE (*)
  - LL_RCC_SPI45_CLKSOURCE (*)
  - LL_RCC_SPI6_CLKSOURCE (*)

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_USART16_CLKSOURCE_PCLK2
  - LL_RCC_USART16_CLKSOURCE_PLL2Q
  - LL_RCC_USART16_CLKSOURCE_PLL3Q
  - LL_RCC_USART16_CLKSOURCE_HSI
  - LL_RCC_USART16_CLKSOURCE_CSI
  - LL_RCC_USART16_CLKSOURCE_LSE
  - LL_RCC_USART234578_CLKSOURCE_PCLK1
  - LL_RCC_USART234578_CLKSOURCE_PLL2Q
  - LL_RCC_USART234578_CLKSOURCE_PLL3Q
  - LL_RCC_USART234578_CLKSOURCE_HSI
  - LL_RCC_USART234578_CLKSOURCE_CSI
  - LL_RCC_USART234578_CLKSOURCE_LSE
  - LL_RCC_I2C123_CLKSOURCE_PCLK1
  - LL_RCC_I2C123_CLKSOURCE_PLL3R
  - LL_RCC_I2C123_CLKSOURCE_HSI
  - LL_RCC_I2C123_CLKSOURCE_CSI
  - LL_RCC_I2C4_CLKSOURCE_PCLK4
  - LL_RCC_I2C4_CLKSOURCE_PLL3R
  - LL_RCC_I2C4_CLKSOURCE_HSI
  - LL_RCC_I2C4_CLKSOURCE_CSI
  - LL_RCC_LPTIM1_CLKSOURCE_PCLK1
  - LL_RCC_LPTIM1_CLKSOURCE_PLL2P
  - LL_RCC_LPTIM1_CLKSOURCE_PLL3R
  - LL_RCC_LPTIM1_CLKSOURCE_LSE
  - LL_RCC_LPTIM1_CLKSOURCE_LSI
  - LL_RCC_LPTIM1_CLKSOURCE_CLKP
  - LL_RCC_LPTIM2_CLKSOURCE_PCLK4
  - LL_RCC_LPTIM2_CLKSOURCE_PLL2P
  - LL_RCC_LPTIM2_CLKSOURCE_PLL3R
  - LL_RCC_LPTIM2_CLKSOURCE_LSE
  - LL_RCC_LPTIM2_CLKSOURCE_LSI
  - LL_RCC_LPTIM2_CLKSOURCE_CLKP
  - LL_RCC_LPTIM345_CLKSOURCE_PCLK4
  - LL_RCC_LPTIM345_CLKSOURCE_PLL2P
  - LL_RCC_LPTIM345_CLKSOURCE_PLL3R
  - LL_RCC_LPTIM345_CLKSOURCE_LSE
  - LL_RCC_LPTIM345_CLKSOURCE_LSI
  - LL_RCC_LPTIM345_CLKSOURCE_CLKP
  - LL_RCC_SAI1_CLKSOURCE_PLL1Q
  - LL_RCC_SAI1_CLKSOURCE_PLL2P
  - LL_RCC_SAI1_CLKSOURCE_PLL3P
  - LL_RCC_SAI1_CLKSOURCE_I2S_CKIN
  - LL_RCC_SAI1_CLKSOURCE_CLKP
  - LL_RCC_SAI23_CLKSOURCE_PLL1Q (*)
  - LL_RCC_SAI23_CLKSOURCE_PLL2P (*)
  - LL_RCC_SAI23_CLKSOURCE_PLL3P (*)
  - LL_RCC_SAI23_CLKSOURCE_I2S_CKIN (*)
  - LL_RCC_SAI23_CLKSOURCE_CLKP (*)
  - LL_RCC_SAI4A_CLKSOURCE_PLL1Q (*)
  - LL_RCC_SAI4A_CLKSOURCE_PLL2P (*)

- **None:**

Reference Manual to LL API cross reference:

- D1CCIPR / CDCCIPR * LL_RCC_GetClockSource
- D2CCIP1R / CDCCIP1R * LL_RCC_GetClockSource
- D2CCIP2R / CDCCIP2R * LL_RCC_GetClockSource
- D3CCIPR / SRDCCIPR * LL_RCC_GetClockSource

LL_RCC_GetUSARTClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetUSARTClockSource (uint32_t Periph)

Function description

Get USARTx clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_USART16_CLKSOURCE
  - LL_RCC_USART234578_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_USART16_CLKSOURCE_PCLK2
  - LL_RCC_USART16_CLKSOURCE_PLL2Q
  - LL_RCC_USART16_CLKSOURCE_PLL3Q
  - LL_RCC_USART16_CLKSOURCE_HSI
  - LL_RCC_USART16_CLKSOURCE_CSI
  - LL_RCC_USART16_CLKSOURCE_LSE
  - LL_RCC_USART234578_CLKSOURCE_PCLK1
  - LL_RCC_USART234578_CLKSOURCE_PLL2Q
  - LL_RCC_USART234578_CLKSOURCE_PLL3Q
  - LL_RCC_USART234578_CLKSOURCE_HSI
  - LL_RCC_USART234578_CLKSOURCE_CSI
  - LL_RCC_USART234578_CLKSOURCE_LSE

Reference Manual to LL API cross reference:

- D2CCIP2R / CDCCIP2R USART16SEL LL_RCC_GetUSARTClockSource
- D2CCIP2R / CDCCIP2R USART28SEL LL_RCC_GetUSARTClockSource

LL_RCC_GetLPUARTClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetLPUARTClockSource (uint32_t Periph)

Function description

Get LPUART clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_LPUART1_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_LPUART1_CLKSOURCE_PCLK4
  - LL_RCC_LPUART1_CLKSOURCE_PLL2Q
  - LL_RCC_LPUART1_CLKSOURCE_PLL3Q
  - LL_RCC_LPUART1_CLKSOURCE_HSI
  - LL_RCC_LPUART1_CLKSOURCE_CSI
  - LL_RCC_LPUART1_CLKSOURCE_LSE

Reference Manual to LL API cross reference:

- D3CCIPR / SRDCCIPR LPUART1SEL LL_RCC_GetLPUARTClockSource

LL_RCC_GetI2CClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetI2CClockSource (uint32_t Periph)

Function description

Get I2Cx clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_I2C123_CLKSOURCE
  - LL_RCC_I2C4_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_I2C123_CLKSOURCE_PCLK1
  - LL_RCC_I2C123_CLKSOURCE_PLL3R
  - LL_RCC_I2C123_CLKSOURCE_HSI
  - LL_RCC_I2C123_CLKSOURCE_CSI
  - LL_RCC_I2C4_CLKSOURCE_PCLK4
  - LL_RCC_I2C4_CLKSOURCE_PLL3R
  - LL_RCC_I2C4_CLKSOURCE_HSI
  - LL_RCC_I2C4_CLKSOURCE_CSI

Reference Manual to LL API cross reference:

- D2CCIP2R / CDCCIP2R I2C123SEL LL_RCC_GetI2CClockSource
- D3CCIPR / SRDCCIPR I2C4SEL LL_RCC_GetI2CClockSource

LL_RCC_GetLPTIMClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetLPTIMClockSource (uint32_t Periph)

Function description

Get LPTIM clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_LPTIM1_CLKSOURCE
  - LL_RCC_LPTIM2_CLKSOURCE
  - LL_RCC_LPTIM345_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_LPTIM1_CLKSOURCE_PCLK1
  - LL_RCC_LPTIM1_CLKSOURCE_PLL2P
  - LL_RCC_LPTIM1_CLKSOURCE_PLL3R
  - LL_RCC_LPTIM1_CLKSOURCE_LSE
  - LL_RCC_LPTIM1_CLKSOURCE_LSI
  - LL_RCC_LPTIM1_CLKSOURCE_CLKP
  - LL_RCC_LPTIM2_CLKSOURCE_PCLK4
  - LL_RCC_LPTIM2_CLKSOURCE_PLL2P
  - LL_RCC_LPTIM2_CLKSOURCE_PLL3R
  - LL_RCC_LPTIM2_CLKSOURCE_LSE
  - LL_RCC_LPTIM2_CLKSOURCE_LSI
  - LL_RCC_LPTIM2_CLKSOURCE_CLKP
  - LL_RCC_LPTIM345_CLKSOURCE_PCLK4
  - LL_RCC_LPTIM345_CLKSOURCE_PLL2P
  - LL_RCC_LPTIM345_CLKSOURCE_PLL3R
  - LL_RCC_LPTIM345_CLKSOURCE_LSE
  - LL_RCC_LPTIM345_CLKSOURCE_LSI
  - LL_RCC_LPTIM345_CLKSOURCE_CLKP
- **None:**

Reference Manual to LL API cross reference:

- D2CCIP2R / CDCCIP2R LPTIM1SEL LL_RCC_GetLPTIMClockSource
- D3CCIPR / SRDCCIPR LPTIM2SEL LL_RCC_GetLPTIMClockSource
- D3CCIPR / SRDCCIPR LPTIM345SEL LL_RCC_GetLPTIMClockSource

LL_RCC_GetSAIClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSAIClockSource (uint32_t Periph)

Function description

Get SAIx clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_SAI1_CLKSOURCE (*)
  - LL_RCC_SAI2A_CLKSOURCE (*)
  - LL_RCC_SAI2B_CLKSOURCE (*)
  - LL_RCC_SAI23_CLKSOURCE (*)
  - LL_RCC_SAI4A_CLKSOURCE (*)
  - LL_RCC_SAI4B_CLKSOURCE (*)

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SAI1_CLKSOURCE_PLL1Q
  - LL_RCC_SAI1_CLKSOURCE_PLL2P
  - LL_RCC_SAI1_CLKSOURCE_PLL3P
  - LL_RCC_SAI1_CLKSOURCE_I2S_CKIN
  - LL_RCC_SAI1_CLKSOURCE_CLKP
  - LL_RCC_SAI23_CLKSOURCE_PLL1Q (*)
  - LL_RCC_SAI23_CLKSOURCE_PLL2P (*)
  - LL_RCC_SAI23_CLKSOURCE_PLL3P (*)
  - LL_RCC_SAI23_CLKSOURCE_I2S_CKIN (*)
  - LL_RCC_SAI23_CLKSOURCE_CLKP (*)
  - LL_RCC_SAI2A_CLKSOURCE_PLL1Q (*)
  - LL_RCC_SAI2A_CLKSOURCE_PLL2P (*)
  - LL_RCC_SAI2A_CLKSOURCE_PLL3P (*)
  - LL_RCC_SAI2A_CLKSOURCE_I2S_CKIN (*)
  - LL_RCC_SAI2A_CLKSOURCE_CLKP (*)
  - LL_RCC_SAI2A_CLKSOURCE_SPDIF (*)
  - LL_RCC_SAI2B_CLKSOURCE_PLL1Q (*)
  - LL_RCC_SAI2B_CLKSOURCE_PLL2P (*)
  - LL_RCC_SAI2B_CLKSOURCE_PLL3P (*)
  - LL_RCC_SAI2B_CLKSOURCE_I2S_CKIN (*)
  - LL_RCC_SAI2B_CLKSOURCE_CLKP (*)
  - LL_RCC_SAI2B_CLKSOURCE_SPDIF (*)
  - LL_RCC_SAI4A_CLKSOURCE_PLL1Q (*)
  - LL_RCC_SAI4A_CLKSOURCE_PLL2P (*)
  - LL_RCC_SAI4A_CLKSOURCE_PLL3P (*)
  - LL_RCC_SAI4A_CLKSOURCE_I2S_CKIN (*)
  - LL_RCC_SAI4A_CLKSOURCE_CLKP (*)
  - LL_RCC_SAI4B_CLKSOURCE_PLL1Q (*)
  - LL_RCC_SAI4B_CLKSOURCE_PLL2P (*)
  - LL_RCC_SAI4B_CLKSOURCE_PLL3P (*)
  - LL_RCC_SAI4B_CLKSOURCE_I2S_CKIN (*)
  - LL_RCC_SAI4B_CLKSOURCE_CLKP (*)

  (*) value not defined in all devices.

Reference Manual to LL API cross reference:

- D2CCIP1R / CDCCIP1R SAI1SEL LL_RCC_GetSAIClockSource
- D2CCIP1R / CDCCIP1R SAI23SEL LL_RCC_GetSAIClockSource D3CCIPR / SRDCCIPR SAI4ASEL LL_RCC_GetSAIClockSource
- D3CCIPR / SRDCCIPR SAI4BSEL LL_RCC_GetSAIClockSource

LL_RCC_GetSDMMCClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSDMMCClockSource (uint32_t Periph)

Function description

Get SDMMC clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_SDMMC_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SDMMC_CLKSOURCE_PLL1Q
  - LL_RCC_SDMMC_CLKSOURCE_PLL2R

Reference Manual to LL API cross reference:

- D1CCIPR / CDCCIPR SDMMCSEL LL_RCC_GetSDMMCClockSource

LL_RCC_GetRNGClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetRNGClockSource (uint32_t Periph)

Function description

Get RNG clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_RNG_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_RNG_CLKSOURCE_HSI48
  - LL_RCC_RNG_CLKSOURCE_PLL1Q
  - LL_RCC_RNG_CLKSOURCE_LSE
  - LL_RCC_RNG_CLKSOURCE_LSI

Reference Manual to LL API cross reference:

- D2CCIP2R RNGSEL LL_RCC_GetRNGClockSource

LL_RCC_GetUSBClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetUSBClockSource (uint32_t Periph)

Function description

Get USB clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_USB_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_USB_CLKSOURCE_DISABLE
  - LL_RCC_USB_CLKSOURCE_PLL1Q
  - LL_RCC_USB_CLKSOURCE_PLL3Q
  - LL_RCC_USB_CLKSOURCE_HSI48

Reference Manual to LL API cross reference:

- D2CCIP2R / CDCCIP2R USBSEL LL_RCC_GetUSBClockSource

LL_RCC_GetCECClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetCECClockSource (uint32_t Periph)

Function description

Get CEC clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_CEC_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_CEC_CLKSOURCE_LSE
  - LL_RCC_CEC_CLKSOURCE_LSI
  - LL_RCC_CEC_CLKSOURCE_CSI_DIV122

Reference Manual to LL API cross reference:

- D2CCIP2R / CDCCIP2R CECSEL LL_RCC_GetCECClockSource

LL_RCC_GetDFSDMClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetDFSDMClockSource (uint32_t Periph)

Function description

Get DFSDM Kernel clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_DFSDM1_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_DFSDM1_CLKSOURCE_PCLK2
  - LL_RCC_DFSDM1_CLKSOURCE_SYSCLK

Reference Manual to LL API cross reference:

- D2CCIP1R / CDCCIP1R DFSDM1SEL LL_RCC_GetDFSDMClockSource

LL_RCC_GetFMCClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetFMCClockSource (uint32_t Periph)

Function description

Get FMC Kernel clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_FMC_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_FMC_CLKSOURCE_HCLK
  - LL_RCC_FMC_CLKSOURCE_PLL1Q
  - LL_RCC_FMC_CLKSOURCE_PLL2R
  - LL_RCC_FMC_CLKSOURCE_CLKP

Reference Manual to LL API cross reference:

- D1CCIPR / D1CCIPR FMCSEL LL_RCC_GetFMCClockSource

LL_RCC_GetQSPIClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetQSPIClockSource (uint32_t Periph)

Function description

Get QSPI Kernel clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_QSPI_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_QSPI_CLKSOURCE_HCLK
  - LL_RCC_QSPI_CLKSOURCE_PLL1Q
  - LL_RCC_QSPI_CLKSOURCE_PLL2R
  - LL_RCC_QSPI_CLKSOURCE_CLKP

Reference Manual to LL API cross reference:

- D1CCIPR / CDCCIPR QSPISEL LL_RCC_GetQSPIClockSource

LL_RCC_GetCLKPClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetCLKPClockSource (uint32_t Periph)

Function description

Get CLKP Kernel clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_CLKP_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_CLKP_CLKSOURCE_HSI
  - LL_RCC_CLKP_CLKSOURCE_CSI
  - LL_RCC_CLKP_CLKSOURCE_HSE

Reference Manual to LL API cross reference:

- D1CCIPR / CDCCIPR CKPERSEL LL_RCC_GetCLKPClockSource

LL_RCC_GetSPIClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSPIClockSource (uint32_t Periph)

Function description

Get SPIx Kernel clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_SPI123_CLKSOURCE
  - LL_RCC_SPI45_CLKSOURCE
  - LL_RCC_SPI6_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SPI123_CLKSOURCE_PLL1Q
  - LL_RCC_SPI123_CLKSOURCE_PLL2P
  - LL_RCC_SPI123_CLKSOURCE_PLL3P
  - LL_RCC_SPI123_CLKSOURCE_I2S_CKIN
  - LL_RCC_SPI123_CLKSOURCE_CLKP
  - LL_RCC_SPI45_CLKSOURCE_PCLK2
  - LL_RCC_SPI45_CLKSOURCE_PLL2Q
  - LL_RCC_SPI45_CLKSOURCE_PLL3Q
  - LL_RCC_SPI45_CLKSOURCE_HSI
  - LL_RCC_SPI45_CLKSOURCE_CSI
  - LL_RCC_SPI45_CLKSOURCE_HSE
  - LL_RCC_SPI6_CLKSOURCE_PCLK4
  - LL_RCC_SPI6_CLKSOURCE_PLL2Q
  - LL_RCC_SPI6_CLKSOURCE_PLL3Q
  - LL_RCC_SPI6_CLKSOURCE_HSI
  - LL_RCC_SPI6_CLKSOURCE_CSI
  - LL_RCC_SPI6_CLKSOURCE_HSE
  - LL_RCC_SPI6_CLKSOURCE_I2S_CKIN (*)

  (*) value not defined in all stm32h7xx lines.

Reference Manual to LL API cross reference:

- D2CCIP1R / CDCCIP1R SPI123SEL LL_RCC_GetSPIClockSource
- D2CCIP1R / CDCCIP1R SPI45SEL LL_RCC_GetSPIClockSource
- D3CCIPR / SRDCCIPR SPI6SEL LL_RCC_GetSPIClockSource

LL_RCC_GetSPDIFClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSPDIFClockSource (uint32_t Periph)

Function description

Get SPDIF Kernel clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_SPDIF_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SPDIF_CLKSOURCE_PLL1Q
  - LL_RCC_SPDIF_CLKSOURCE_PLL2R
  - LL_RCC_SPDIF_CLKSOURCE_PLL3R
  - LL_RCC_SPDIF_CLKSOURCE_HSI

Reference Manual to LL API cross reference:

- D2CCIP1R / CDCCIP1R SPDIFSEL LL_RCC_GetSPDIFClockSource

LL_RCC_GetFDCANClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetFDCANClockSource (uint32_t Periph)

Function description

Get FDCAN Kernel clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_FDCAN_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_FDCAN_CLKSOURCE_HSE
  - LL_RCC_FDCAN_CLKSOURCE_PLL1Q
  - LL_RCC_FDCAN_CLKSOURCE_PLL2Q

Reference Manual to LL API cross reference:

- D2CCIP1R / CDCCIP1R FDCANSEL LL_RCC_GetFDCANClockSource

LL_RCC_GetSWPClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSWPClockSource (uint32_t Periph)

Function description

Get SWP Kernel clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_SWP_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SWP_CLKSOURCE_PCLK1
  - LL_RCC_SWP_CLKSOURCE_HSI

Reference Manual to LL API cross reference:

- D2CCIP1R / CDCCIP1R SWPSEL LL_RCC_GetSWPClockSource

LL_RCC_GetADCClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetADCClockSource (uint32_t Periph)

Function description

Get ADC Kernel clock source.

Parameters

- **Periph:** This parameter can be one of the following values:
  - LL_RCC_ADC_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ADC_CLKSOURCE_PLL2P
  - LL_RCC_ADC_CLKSOURCE_PLL3R
  - LL_RCC_ADC_CLKSOURCE_CLKP

Reference Manual to LL API cross reference:

- D3CCIPR / SRDCCIPR ADCSEL LL_RCC_GetADCClockSource

LL_RCC_SetRTCClockSource

Function name

__STATIC_INLINE void LL_RCC_SetRTCClockSource (uint32_t Source)

Function description

Set RTC Clock Source.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_RTC_CLKSOURCE_NONE
  - LL_RCC_RTC_CLKSOURCE_LSE
  - LL_RCC_RTC_CLKSOURCE_LSI
  - LL_RCC_RTC_CLKSOURCE_HSE

Return values

- **None:**

Notes

- Once the RTC clock source has been selected, it cannot be changed anymore unless the Backup domain is reset, or unless a failure is detected on LSE (LSECSSD is set). The BDRST bit can be used to reset them.

Reference Manual to LL API cross reference:

- BDCR RTCSEL LL_RCC_SetRTCClockSource

LL_RCC_GetRTCClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetRTCClockSource (void)

Function description

Get RTC Clock Source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_RTC_CLKSOURCE_NONE
  - LL_RCC_RTC_CLKSOURCE_LSE
  - LL_RCC_RTC_CLKSOURCE_LSI
  - LL_RCC_RTC_CLKSOURCE_HSE

Reference Manual to LL API cross reference:

- BDCR RTCSEL LL_RCC_GetRTCClockSource

LL_RCC_EnableRTC

Function name

__STATIC_INLINE void LL_RCC_EnableRTC (void)

Function description

Enable RTC.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR RTCEN LL_RCC_EnableRTC

LL_RCC_DisableRTC

Function name

__STATIC_INLINE void LL_RCC_DisableRTC (void)

Function description

Disable RTC.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR RTCEN LL_RCC_DisableRTC

LL_RCC_IsEnabledRTC

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledRTC (void)

Function description

Check if RTC has been enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDCR RTCEN LL_RCC_IsEnabledRTC

LL_RCC_ForceBackupDomainReset

Function name

__STATIC_INLINE void LL_RCC_ForceBackupDomainReset (void)

Function description

Force the Backup domain reset.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR BDRST / VSWRST LL_RCC_ForceBackupDomainReset

LL_RCC_ReleaseBackupDomainReset

Function name

__STATIC_INLINE void LL_RCC_ReleaseBackupDomainReset (void)

Function description

Release the Backup domain reset.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR BDRST / VSWRST LL_RCC_ReleaseBackupDomainReset

LL_RCC_SetRTC_HSEPrescaler

Function name

__STATIC_INLINE void LL_RCC_SetRTC_HSEPrescaler (uint32_t Prescaler)

Function description

Set HSE Prescalers for RTC Clock.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_RTC_NOCLOCK
  - LL_RCC_RTC_HSE_DIV_2
  - LL_RCC_RTC_HSE_DIV_3
  - LL_RCC_RTC_HSE_DIV_4
  - LL_RCC_RTC_HSE_DIV_5
  - LL_RCC_RTC_HSE_DIV_6
  - LL_RCC_RTC_HSE_DIV_7
  - LL_RCC_RTC_HSE_DIV_8
  - LL_RCC_RTC_HSE_DIV_9
  - LL_RCC_RTC_HSE_DIV_10
  - LL_RCC_RTC_HSE_DIV_11
  - LL_RCC_RTC_HSE_DIV_12
  - LL_RCC_RTC_HSE_DIV_13
  - LL_RCC_RTC_HSE_DIV_14
  - LL_RCC_RTC_HSE_DIV_15
  - LL_RCC_RTC_HSE_DIV_16
  - LL_RCC_RTC_HSE_DIV_17
  - LL_RCC_RTC_HSE_DIV_18
  - LL_RCC_RTC_HSE_DIV_19
  - LL_RCC_RTC_HSE_DIV_20
  - LL_RCC_RTC_HSE_DIV_21
  - LL_RCC_RTC_HSE_DIV_22
  - LL_RCC_RTC_HSE_DIV_23
  - LL_RCC_RTC_HSE_DIV_24
  - LL_RCC_RTC_HSE_DIV_25
  - LL_RCC_RTC_HSE_DIV_26
  - LL_RCC_RTC_HSE_DIV_27
  - LL_RCC_RTC_HSE_DIV_28
  - LL_RCC_RTC_HSE_DIV_29
  - LL_RCC_RTC_HSE_DIV_30
  - LL_RCC_RTC_HSE_DIV_31
  - LL_RCC_RTC_HSE_DIV_32
  - LL_RCC_RTC_HSE_DIV_33
  - LL_RCC_RTC_HSE_DIV_34
  - LL_RCC_RTC_HSE_DIV_35
  - LL_RCC_RTC_HSE_DIV_36
  - LL_RCC_RTC_HSE_DIV_37
  - LL_RCC_RTC_HSE_DIV_38
  - LL_RCC_RTC_HSE_DIV_39
  - LL_RCC_RTC_HSE_DIV_40
  - LL_RCC_RTC_HSE_DIV_41
  - LL_RCC_RTC_HSE_DIV_42
  - LL_RCC_RTC_HSE_DIV_43
  - LL_RCC_RTC_HSE_DIV_44
  - LL_RCC_RTC_HSE_DIV_45
  - LL_RCC_RTC_HSE_DIV_46
  - LL_RCC_RTC_HSE_DIV_47
  - LL_RCC_RTC_HSE_DIV_48
  - LL_RCC_RTC_HSE_DIV_49
  - LL_RCC_RTC_HSE_DIV_50

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR RTCPRE LL_RCC_SetRTC_HSEPrescaler

LL_RCC_GetRTC_HSEPrescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetRTC_HSEPrescaler (void)

Function description

Get HSE Prescalers for RTC Clock.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_RTC_NOCLOCK
  - LL_RCC_RTC_HSE_DIV_2
  - LL_RCC_RTC_HSE_DIV_3
  - LL_RCC_RTC_HSE_DIV_4
  - LL_RCC_RTC_HSE_DIV_5
  - LL_RCC_RTC_HSE_DIV_6
  - LL_RCC_RTC_HSE_DIV_7
  - LL_RCC_RTC_HSE_DIV_8
  - LL_RCC_RTC_HSE_DIV_9
  - LL_RCC_RTC_HSE_DIV_10
  - LL_RCC_RTC_HSE_DIV_11
  - LL_RCC_RTC_HSE_DIV_12
  - LL_RCC_RTC_HSE_DIV_13
  - LL_RCC_RTC_HSE_DIV_14
  - LL_RCC_RTC_HSE_DIV_15
  - LL_RCC_RTC_HSE_DIV_16
  - LL_RCC_RTC_HSE_DIV_17
  - LL_RCC_RTC_HSE_DIV_18
  - LL_RCC_RTC_HSE_DIV_19
  - LL_RCC_RTC_HSE_DIV_20
  - LL_RCC_RTC_HSE_DIV_21
  - LL_RCC_RTC_HSE_DIV_22
  - LL_RCC_RTC_HSE_DIV_23
  - LL_RCC_RTC_HSE_DIV_24
  - LL_RCC_RTC_HSE_DIV_25
  - LL_RCC_RTC_HSE_DIV_26
  - LL_RCC_RTC_HSE_DIV_27
  - LL_RCC_RTC_HSE_DIV_28
  - LL_RCC_RTC_HSE_DIV_29
  - LL_RCC_RTC_HSE_DIV_30
  - LL_RCC_RTC_HSE_DIV_31
  - LL_RCC_RTC_HSE_DIV_32
  - LL_RCC_RTC_HSE_DIV_33
  - LL_RCC_RTC_HSE_DIV_34
  - LL_RCC_RTC_HSE_DIV_35
  - LL_RCC_RTC_HSE_DIV_36
  - LL_RCC_RTC_HSE_DIV_37
  - LL_RCC_RTC_HSE_DIV_38
  - LL_RCC_RTC_HSE_DIV_39
  - LL_RCC_RTC_HSE_DIV_40
  - LL_RCC_RTC_HSE_DIV_41
  - LL_RCC_RTC_HSE_DIV_42
  - LL_RCC_RTC_HSE_DIV_43
  - LL_RCC_RTC_HSE_DIV_44
  - LL_RCC_RTC_HSE_DIV_45
  - LL_RCC_RTC_HSE_DIV_46
  - LL_RCC_RTC_HSE_DIV_47
  - LL_RCC_RTC_HSE_DIV_48
  - LL_RCC_RTC_HSE_DIV_49
  - LL_RCC_RTC_HSE_DIV_50

Reference Manual to LL API cross reference:

- CFGR RTCPRE LL_RCC_GetRTC_HSEPrescaler

LL_RCC_SetTIMPrescaler

Function name

__STATIC_INLINE void LL_RCC_SetTIMPrescaler (uint32_t Prescaler)

Function description

Set Timers Clock Prescalers.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_TIM_PRESCALER_TWICE
  - LL_RCC_TIM_PRESCALER_FOUR_TIMES

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR TIMPRE LL_RCC_SetTIMPrescaler

LL_RCC_GetTIMPrescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetTIMPrescaler (void)

Function description

Get Timers Clock Prescalers.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_TIM_PRESCALER_TWICE
  - LL_RCC_TIM_PRESCALER_FOUR_TIMES

Reference Manual to LL API cross reference:

- CFGR TIMPRE LL_RCC_GetTIMPrescaler

LL_RCC_SetHRTIMClockSource

Function name

__STATIC_INLINE void LL_RCC_SetHRTIMClockSource (uint32_t Prescaler)

Function description

Set High Resolution Timers Clock Source.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_HRTIM_CLKSOURCE_TIM
  - LL_RCC_HRTIM_CLKSOURCE_CPU

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR HRTIMSEL LL_RCC_SetHRTIMClockSource

LL_RCC_GetHRTIMClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetHRTIMClockSource (void)

Function description

Get High Resolution Timers Clock Source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_HRTIM_CLKSOURCE_TIM
  - LL_RCC_HRTIM_CLKSOURCE_CPU

Reference Manual to LL API cross reference:

- CFGR HRTIMSEL LL_RCC_GetHRTIMClockSource

LL_RCC_PLL_SetSource

Function name

__STATIC_INLINE void LL_RCC_PLL_SetSource (uint32_t PLLSource)

Function description

Set the oscillator used as PLL clock source.

Parameters

- **PLLSource:** parameter can be one of the following values:
  - LL_RCC_PLLSOURCE_HSI
  - LL_RCC_PLLSOURCE_CSI
  - LL_RCC_PLLSOURCE_HSE
  - LL_RCC_PLLSOURCE_NONE

Return values

- **None:**

Notes

- PLLSRC can be written only when All PLLs are disabled.

Reference Manual to LL API cross reference:

- PLLCKSELR PLLSRC LL_RCC_PLL_SetSource

LL_RCC_PLL_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL_GetSource (void)

Function description

Get the oscillator used as PLL clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_PLLSOURCE_HSI
  - LL_RCC_PLLSOURCE_CSI
  - LL_RCC_PLLSOURCE_HSE
  - LL_RCC_PLLSOURCE_NONE

Reference Manual to LL API cross reference:

- PLLCKSELR PLLSRC LL_RCC_PLL_GetSource

LL_RCC_PLL1_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL1_Enable (void)

Function description

Enable PLL1.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PLL1ON LL_RCC_PLL1_Enable

LL_RCC_PLL1_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL1_Disable (void)

Function description

Disable PLL1.

Return values

- **None:**

Notes

- Cannot be disabled if the PLL1 clock is used as the system clock

Reference Manual to LL API cross reference:

- CR PLL1ON LL_RCC_PLL1_Disable

LL_RCC_PLL1_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_IsReady (void)

Function description

Check if PLL1 Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR PLL1RDY LL_RCC_PLL1_IsReady

LL_RCC_PLL1P_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL1P_Enable (void)

Function description

Enable PLL1P.

Return values

- **None:**

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLLCFGR DIVP1EN LL_RCC_PLL1P_Enable

LL_RCC_PLL1Q_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL1Q_Enable (void)

Function description

Enable PLL1Q.

Return values

- **None:**

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLLCFGR DIVQ1EN LL_RCC_PLL1Q_Enable

LL_RCC_PLL1R_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL1R_Enable (void)

Function description

Enable PLL1R.

Return values

- **None:**

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLLCFGR DIVR1EN LL_RCC_PLL1R_Enable

LL_RCC_PLL1FRACN_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL1FRACN_Enable (void)

Function description

Enable PLL1 FRACN.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLLCFGR PLL1FRACEN LL_RCC_PLL1FRACN_Enable

LL_RCC_PLL1P_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1P_IsEnabled (void)

Function description

Check if PLL1 P is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLLCFGR DIVP1EN LL_RCC_PLL1P_IsEnabled

LL_RCC_PLL1Q_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1Q_IsEnabled (void)

Function description

Check if PLL1 Q is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLLCFGR DIVQ1EN LL_RCC_PLL1Q_IsEnabled

LL_RCC_PLL1R_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1R_IsEnabled (void)

Function description

Check if PLL1 R is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLLCFGR DIVR1EN LL_RCC_PLL1R_IsEnabled

LL_RCC_PLL1FRACN_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1FRACN_IsEnabled (void)

Function description

Check if PLL1 FRACN is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLLCFGR PLL1FRACEN LL_RCC_PLL1FRACN_IsEnabled

LL_RCC_PLL1P_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL1P_Disable (void)

Function description

Disable PLL1P.

Return values

- **None:**

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLLCFGR DIVP1EN LL_RCC_PLL1P_Disable

LL_RCC_PLL1Q_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL1Q_Disable (void)

Function description

Disable PLL1Q.

Return values

- **None:**

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLLCFGR DIVQ1EN LL_RCC_PLL1Q_Disable

LL_RCC_PLL1R_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL1R_Disable (void)

Function description

Disable PLL1R.

Return values

- **None:**

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLLCFGR DIVR1EN LL_RCC_PLL1R_Disable

LL_RCC_PLL1FRACN_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL1FRACN_Disable (void)

Function description

Disable PLL1 FRACN.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLLCFGR PLL1FRACEN LL_RCC_PLL1FRACN_Enable

LL_RCC_PLL1_SetVCOOutputRange

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetVCOOutputRange (uint32_t VCORange)

Function description

Set PLL1 VCO OutputRange.

Parameters

- **VCORange:** This parameter can be one of the following values:
  - LL_RCC_PLLVCORANGE_WIDE
  - LL_RCC_PLLVCORANGE_MEDIUM

Return values

- **None:**

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLLCFGR PLL1VCOSEL LL_RCC_PLL1_SetVCOOuputRange

LL_RCC_PLL1_SetVCOInputRange

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetVCOInputRange (uint32_t InputRange)

Function description

Set PLL1 VCO Input Range.

Parameters

- **InputRange:** This parameter can be one of the following values:
  - LL_RCC_PLLINPUTRANGE_1_2
  - LL_RCC_PLLINPUTRANGE_2_4
  - LL_RCC_PLLINPUTRANGE_4_8
  - LL_RCC_PLLINPUTRANGE_8_16

Return values

- **None:**

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLLCFGR PLL1RGE LL_RCC_PLL1_SetVCOInputRange

LL_RCC_PLL1_GetN

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_GetN (void)

Function description

Get PLL1 N Coefficient.

Return values

- **A:** value between 4 and 512

Reference Manual to LL API cross reference:

- PLL1DIVR N1 LL_RCC_PLL1_GetN

LL_RCC_PLL1_GetM

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_GetM (void)

Function description

Get PLL1 M Coefficient.

Return values

- **A:** value between 0 and 63

Reference Manual to LL API cross reference:

- PLLCKSELR DIVM1 LL_RCC_PLL1_GetM

LL_RCC_PLL1_GetP

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_GetP (void)

Function description

Get PLL1 P Coefficient.

Return values

- **A:** value between 2 and 128

Reference Manual to LL API cross reference:

- PLL1DIVR P1 LL_RCC_PLL1_GetP

LL_RCC_PLL1_GetQ

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_GetQ (void)

Function description

Get PLL1 Q Coefficient.

Return values

- **A:** value between 1 and 128

Reference Manual to LL API cross reference:

- PLL1DIVR Q1 LL_RCC_PLL1_GetQ

LL_RCC_PLL1_GetR

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_GetR (void)

Function description

Get PLL1 R Coefficient.

Return values

- **A:** value between 1 and 128

Reference Manual to LL API cross reference:

- PLL1DIVR R1 LL_RCC_PLL1_GetR

LL_RCC_PLL1_GetFRACN

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_GetFRACN (void)

Function description

Get PLL1 FRACN Coefficient.

Return values

- **A:** value between 0 and 8191 (0x1FFF)

Reference Manual to LL API cross reference:

- PLL1FRACR FRACN1 LL_RCC_PLL1_GetFRACN

LL_RCC_PLL1_SetN

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetN (uint32_t N)

Function description

Set PLL1 N Coefficient.

Parameters

- **N:** parameter can be a value between 4 and 512

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLL1DIVR N1 LL_RCC_PLL1_SetN

LL_RCC_PLL1_SetM

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetM (uint32_t M)

Function description

Set PLL1 M Coefficient.

Parameters

- **M:** parameter can be a value between 0 and 63

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLLCKSELR DIVM1 LL_RCC_PLL1_SetM

LL_RCC_PLL1_SetP

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetP (uint32_t P)

Function description

Set PLL1 P Coefficient.

Parameters

- **P:** parameter can be a value between 2 (or 1*) and 128 (ODD division factor not supported)

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLL1DIVR P1 LL_RCC_PLL1_SetP

LL_RCC_PLL1_SetQ

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetQ (uint32_t Q)

Function description

Set PLL1 Q Coefficient.

Parameters

- **Q:** parameter can be a value between 1 and 128

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLL1DIVR Q1 LL_RCC_PLL1_SetQ

LL_RCC_PLL1_SetR

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetR (uint32_t R)

Function description

Set PLL1 R Coefficient.

Parameters

- **R:** parameter can be a value between 1 and 128

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLL1DIVR R1 LL_RCC_PLL1_SetR

LL_RCC_PLL1_SetFRACN

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetFRACN (uint32_t FRACN)

Function description

Set PLL1 FRACN Coefficient.

Parameters

- **FRACN:** parameter can be a value between 0 and 8191 (0x1FFF)

Reference Manual to LL API cross reference:

- PLL1FRACR FRACN1 LL_RCC_PLL1_SetFRACN

LL_RCC_PLL2_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL2_Enable (void)

Function description

Enable PLL2.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PLL2ON LL_RCC_PLL2_Enable

LL_RCC_PLL2_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL2_Disable (void)

Function description

Disable PLL2.

Return values

- **None:**

Notes

- Cannot be disabled if the PLL2 clock is used as the system clock

Reference Manual to LL API cross reference:

- CR PLL2ON LL_RCC_PLL2_Disable

LL_RCC_PLL2_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_IsReady (void)

Function description

Check if PLL2 Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR PLL2RDY LL_RCC_PLL2_IsReady

LL_RCC_PLL2P_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL2P_Enable (void)

Function description

Enable PLL2P.

Return values

- **None:**

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLLCFGR DIVP2EN LL_RCC_PLL2P_Enable

LL_RCC_PLL2Q_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL2Q_Enable (void)

Function description

Enable PLL2Q.

Return values

- **None:**

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLLCFGR DIVQ2EN LL_RCC_PLL2Q_Enable

LL_RCC_PLL2R_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL2R_Enable (void)

Function description

Enable PLL2R.

Return values

- **None:**

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLLCFGR DIVR2EN LL_RCC_PLL2R_Enable

LL_RCC_PLL2FRACN_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL2FRACN_Enable (void)

Function description

Enable PLL2 FRACN.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLLCFGR PLL2FRACEN LL_RCC_PLL2FRACN_Enable

LL_RCC_PLL2P_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2P_IsEnabled (void)

Function description

Check if PLL2 P is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLLCFGR DIVP2EN LL_RCC_PLL2P_IsEnabled

LL_RCC_PLL2Q_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2Q_IsEnabled (void)

Function description

Check if PLL2 Q is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLLCFGR DIVQ2EN LL_RCC_PLL2Q_IsEnabled

LL_RCC_PLL2R_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2R_IsEnabled (void)

Function description

Check if PLL2 R is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLLCFGR DIVR2EN LL_RCC_PLL2R_IsEnabled

LL_RCC_PLL2FRACN_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2FRACN_IsEnabled (void)

Function description

Check if PLL2 FRACN is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLLCFGR PLL2FRACEN LL_RCC_PLL2FRACN_IsEnabled

LL_RCC_PLL2P_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL2P_Disable (void)

Function description

Disable PLL2P.

Return values

- **None:**

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLLCFGR DIVP2EN LL_RCC_PLL2P_Disable

LL_RCC_PLL2Q_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL2Q_Disable (void)

Function description

Disable PLL2Q.

Return values

- **None:**

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLLCFGR DIVQ2EN LL_RCC_PLL2Q_Disable

LL_RCC_PLL2R_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL2R_Disable (void)

Function description

Disable PLL2R.

Return values

- **None:**

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLLCFGR DIVR2EN LL_RCC_PLL2R_Disable

LL_RCC_PLL2FRACN_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL2FRACN_Disable (void)

Function description

Disable PLL2 FRACN.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLLCFGR PLL2FRACEN LL_RCC_PLL2FRACN_Enable

LL_RCC_PLL2_SetVCOOutputRange

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetVCOOutputRange (uint32_t VCORange)

Function description

Set PLL2 VCO OutputRange.

Parameters

- **VCORange:** This parameter can be one of the following values:
  - LL_RCC_PLLVCORANGE_WIDE
  - LL_RCC_PLLVCORANGE_MEDIUM

Return values

- **None:**

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLLCFGR PLL2VCOSEL LL_RCC_PLL2_SetVCOOuputRange

LL_RCC_PLL2_SetVCOInputRange

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetVCOInputRange (uint32_t InputRange)

Function description

Set PLL2 VCO Input Range.

Parameters

- **InputRange:** This parameter can be one of the following values:
  - LL_RCC_PLLINPUTRANGE_1_2
  - LL_RCC_PLLINPUTRANGE_2_4
  - LL_RCC_PLLINPUTRANGE_4_8
  - LL_RCC_PLLINPUTRANGE_8_16

Return values

- **None:**

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLLCFGR PLL2RGE LL_RCC_PLL2_SetVCOInputRange

LL_RCC_PLL2_GetN

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_GetN (void)

Function description

Get PLL2 N Coefficient.

Return values

- **A:** value between 4 and 512

Reference Manual to LL API cross reference:

- PLL2DIVR N2 LL_RCC_PLL2_GetN

LL_RCC_PLL2_GetM

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_GetM (void)

Function description

Get PLL2 M Coefficient.

Return values

- **A:** value between 0 and 63

Reference Manual to LL API cross reference:

- PLLCKSELR DIVM2 LL_RCC_PLL2_GetM

LL_RCC_PLL2_GetP

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_GetP (void)

Function description

Get PLL2 P Coefficient.

Return values

- **A:** value between 1 and 128

Reference Manual to LL API cross reference:

- PLL2DIVR P2 LL_RCC_PLL2_GetP

LL_RCC_PLL2_GetQ

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_GetQ (void)

Function description

Get PLL2 Q Coefficient.

Return values

- **A:** value between 1 and 128

Reference Manual to LL API cross reference:

- PLL2DIVR Q2 LL_RCC_PLL2_GetQ

LL_RCC_PLL2_GetR

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_GetR (void)

Function description

Get PLL2 R Coefficient.

Return values

- **A:** value between 1 and 128

Reference Manual to LL API cross reference:

- PLL2DIVR R2 LL_RCC_PLL2_GetR

LL_RCC_PLL2_GetFRACN

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_GetFRACN (void)

Function description

Get PLL2 FRACN Coefficient.

Return values

- **A:** value between 0 and 8191 (0x1FFF)

Reference Manual to LL API cross reference:

- PLL2FRACR FRACN2 LL_RCC_PLL2_GetFRACN

LL_RCC_PLL2_SetN

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetN (uint32_t N)

Function description

Set PLL2 N Coefficient.

Parameters

- **N:** parameter can be a value between 4 and 512

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLL2DIVR N2 LL_RCC_PLL2_SetN

LL_RCC_PLL2_SetM

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetM (uint32_t M)

Function description

Set PLL2 M Coefficient.

Parameters

- **M:** parameter can be a value between 0 and 63

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLLCKSELR DIVM2 LL_RCC_PLL2_SetM

LL_RCC_PLL2_SetP

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetP (uint32_t P)

Function description

Set PLL2 P Coefficient.

Parameters

- **P:** parameter can be a value between 1 and 128

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLL2DIVR P2 LL_RCC_PLL2_SetP

LL_RCC_PLL2_SetQ

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetQ (uint32_t Q)

Function description

Set PLL2 Q Coefficient.

Parameters

- **Q:** parameter can be a value between 1 and 128

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLL2DIVR Q2 LL_RCC_PLL2_SetQ

LL_RCC_PLL2_SetR

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetR (uint32_t R)

Function description

Set PLL2 R Coefficient.

Parameters

- **R:** parameter can be a value between 1 and 128

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLL2DIVR R2 LL_RCC_PLL2_SetR

LL_RCC_PLL2_SetFRACN

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetFRACN (uint32_t FRACN)

Function description

Set PLL2 FRACN Coefficient.

Parameters

- **FRACN:** parameter can be a value between 0 and 8191 (0x1FFF)

Reference Manual to LL API cross reference:

- PLL2FRACR FRACN2 LL_RCC_PLL2_SetFRACN

LL_RCC_PLL3_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL3_Enable (void)

Function description

Enable PLL3.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PLL3ON LL_RCC_PLL3_Enable

LL_RCC_PLL3_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL3_Disable (void)

Function description

Disable PLL3.

Return values

- **None:**

Notes

- Cannot be disabled if the PLL3 clock is used as the system clock

Reference Manual to LL API cross reference:

- CR PLL3ON LL_RCC_PLL3_Disable

LL_RCC_PLL3_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_IsReady (void)

Function description

Check if PLL3 Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR PLL3RDY LL_RCC_PLL3_IsReady

LL_RCC_PLL3P_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL3P_Enable (void)

Function description

Enable PLL3P.

Return values

- **None:**

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLLCFGR DIVP3EN LL_RCC_PLL3P_Enable

LL_RCC_PLL3Q_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL3Q_Enable (void)

Function description

Enable PLL3Q.

Return values

- **None:**

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLLCFGR DIVQ3EN LL_RCC_PLL3Q_Enable

LL_RCC_PLL3R_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL3R_Enable (void)

Function description

Enable PLL3R.

Return values

- **None:**

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLLCFGR DIVR3EN LL_RCC_PLL3R_Enable

LL_RCC_PLL3FRACN_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL3FRACN_Enable (void)

Function description

Enable PLL3 FRACN.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLLCFGR PLL3FRACEN LL_RCC_PLL3FRACN_Enable

LL_RCC_PLL3P_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3P_IsEnabled (void)

Function description

Check if PLL3 P is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLLCFGR DIVP3EN LL_RCC_PLL3P_IsEnabled

LL_RCC_PLL3Q_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3Q_IsEnabled (void)

Function description

Check if PLL3 Q is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLLCFGR DIVQ3EN LL_RCC_PLL3Q_IsEnabled

LL_RCC_PLL3R_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3R_IsEnabled (void)

Function description

Check if PLL3 R is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLLCFGR DIVR3EN LL_RCC_PLL3R_IsEnabled

LL_RCC_PLL3FRACN_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3FRACN_IsEnabled (void)

Function description

Check if PLL3 FRACN is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLLCFGR PLL3FRACEN LL_RCC_PLL3FRACN_IsEnabled

LL_RCC_PLL3P_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL3P_Disable (void)

Function description

Disable PLL3P.

Return values

- **None:**

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLLCFGR DIVP2EN LL_RCC_PLL3P_Disable

LL_RCC_PLL3Q_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL3Q_Disable (void)

Function description

Disable PLL3Q.

Return values

- **None:**

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLLCFGR DIVQ3EN LL_RCC_PLL3Q_Disable

LL_RCC_PLL3R_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL3R_Disable (void)

Function description

Disable PLL3R.

Return values

- **None:**

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLLCFGR DIVR3EN LL_RCC_PLL3R_Disable

LL_RCC_PLL3FRACN_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL3FRACN_Disable (void)

Function description

Disable PLL3 FRACN.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLLCFGR PLL3FRACEN LL_RCC_PLL3FRACN_Enable

LL_RCC_PLL3_SetVCOOutputRange

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetVCOOutputRange (uint32_t VCORange)

Function description

Set PLL3 VCO OutputRange.

Parameters

- **VCORange:** This parameter can be one of the following values:
  - LL_RCC_PLLVCORANGE_WIDE
  - LL_RCC_PLLVCORANGE_MEDIUM

Return values

- **None:**

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLLCFGR PLL3VCOSEL LL_RCC_PLL3_SetVCOOuputRange

LL_RCC_PLL3_SetVCOInputRange

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetVCOInputRange (uint32_t InputRange)

Function description

Set PLL3 VCO Input Range.

Parameters

- **InputRange:** This parameter can be one of the following values:
  - LL_RCC_PLLINPUTRANGE_1_2
  - LL_RCC_PLLINPUTRANGE_2_4
  - LL_RCC_PLLINPUTRANGE_4_8
  - LL_RCC_PLLINPUTRANGE_8_16

Return values

- **None:**

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLLCFGR PLL3RGE LL_RCC_PLL3_SetVCOInputRange

LL_RCC_PLL3_GetN

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_GetN (void)

Function description

Get PLL3 N Coefficient.

Return values

- **A:** value between 4 and 512

Reference Manual to LL API cross reference:

- PLL3DIVR N3 LL_RCC_PLL3_GetN

LL_RCC_PLL3_GetM

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_GetM (void)

Function description

Get PLL3 M Coefficient.

Return values

- **A:** value between 0 and 63

Reference Manual to LL API cross reference:

- PLLCKSELR DIVM3 LL_RCC_PLL3_GetM

LL_RCC_PLL3_GetP

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_GetP (void)

Function description

Get PLL3 P Coefficient.

Return values

- **A:** value between 1 and 128

Reference Manual to LL API cross reference:

- PLL3DIVR P3 LL_RCC_PLL3_GetP

LL_RCC_PLL3_GetQ

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_GetQ (void)

Function description

Get PLL3 Q Coefficient.

Return values

- **A:** value between 1 and 128

Reference Manual to LL API cross reference:

- PLL3DIVR Q3 LL_RCC_PLL3_GetQ

LL_RCC_PLL3_GetR

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_GetR (void)

Function description

Get PLL3 R Coefficient.

Return values

- **A:** value between 1 and 128

Reference Manual to LL API cross reference:

- PLL3DIVR R3 LL_RCC_PLL3_GetR

LL_RCC_PLL3_GetFRACN

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_GetFRACN (void)

Function description

Get PLL3 FRACN Coefficient.

Return values

- **A:** value between 0 and 8191 (0x1FFF)

Reference Manual to LL API cross reference:

- PLL3FRACR FRACN3 LL_RCC_PLL3_GetFRACN

LL_RCC_PLL3_SetN

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetN (uint32_t N)

Function description

Set PLL3 N Coefficient.

Parameters

- **N:** parameter can be a value between 4 and 512

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLL3DIVR N3 LL_RCC_PLL3_SetN

LL_RCC_PLL3_SetM

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetM (uint32_t M)

Function description

Set PLL3 M Coefficient.

Parameters

- **M:** parameter can be a value between 0 and 63

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLLCKSELR DIVM3 LL_RCC_PLL3_SetM

LL_RCC_PLL3_SetP

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetP (uint32_t P)

Function description

Set PLL3 P Coefficient.

Parameters

- **P:** parameter can be a value between 1 and 128

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLL3DIVR P3 LL_RCC_PLL3_SetP

LL_RCC_PLL3_SetQ

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetQ (uint32_t Q)

Function description

Set PLL3 Q Coefficient.

Parameters

- **Q:** parameter can be a value between 1 and 128

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLL3DIVR Q3 LL_RCC_PLL3_SetQ

LL_RCC_PLL3_SetR

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetR (uint32_t R)

Function description

Set PLL3 R Coefficient.

Parameters

- **R:** parameter can be a value between 1 and 128

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLL3DIVR R3 LL_RCC_PLL3_SetR

LL_RCC_PLL3_SetFRACN

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetFRACN (uint32_t FRACN)

Function description

Set PLL3 FRACN Coefficient.

Parameters

- **FRACN:** parameter can be a value between 0 and 8191 (0x1FFF)

Reference Manual to LL API cross reference:

- PLL3FRACR FRACN3 LL_RCC_PLL3_SetFRACN

LL_RCC_ClearFlag_LSIRDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_LSIRDY (void)

Function description

Clear LSI ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR LSIRDYC LL_RCC_ClearFlag_LSIRDY

LL_RCC_ClearFlag_LSERDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_LSERDY (void)

Function description

Clear LSE ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR LSERDYC LL_RCC_ClearFlag_LSERDY

LL_RCC_ClearFlag_HSIRDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_HSIRDY (void)

Function description

Clear HSI ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR HSIRDYC LL_RCC_ClearFlag_HSIRDY

LL_RCC_ClearFlag_HSERDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_HSERDY (void)

Function description

Clear HSE ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR HSERDYC LL_RCC_ClearFlag_HSERDY

LL_RCC_ClearFlag_CSIRDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_CSIRDY (void)

Function description

Clear CSI ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR CSIRDYC LL_RCC_ClearFlag_CSIRDY

LL_RCC_ClearFlag_HSI48RDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_HSI48RDY (void)

Function description

Clear HSI48 ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR HSI48RDYC LL_RCC_ClearFlag_HSI48RDY

LL_RCC_ClearFlag_PLL1RDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_PLL1RDY (void)

Function description

Clear PLL1 ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR PLL1RDYC LL_RCC_ClearFlag_PLL1RDY

LL_RCC_ClearFlag_PLL2RDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_PLL2RDY (void)

Function description

Clear PLL2 ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR PLL2RDYC LL_RCC_ClearFlag_PLL2RDY

LL_RCC_ClearFlag_PLL3RDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_PLL3RDY (void)

Function description

Clear PLL3 ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR PLL3RDYC LL_RCC_ClearFlag_PLL3RDY

LL_RCC_ClearFlag_LSECSS

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_LSECSS (void)

Function description

Clear LSE Clock security system interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR LSECSSC LL_RCC_ClearFlag_LSECSS

LL_RCC_ClearFlag_HSECSS

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_HSECSS (void)

Function description

Clear HSE Clock security system interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR HSECSSC LL_RCC_ClearFlag_HSECSS

LL_RCC_IsActiveFlag_LSIRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LSIRDY (void)

Function description

Check if LSI ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR LSIRDYF LL_RCC_IsActiveFlag_LSIRDY

LL_RCC_IsActiveFlag_LSERDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LSERDY (void)

Function description

Check if LSE ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR LSERDYF LL_RCC_IsActiveFlag_LSERDY

LL_RCC_IsActiveFlag_HSIRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSIRDY (void)

Function description

Check if HSI ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR HSIRDYF LL_RCC_IsActiveFlag_HSIRDY

LL_RCC_IsActiveFlag_HSERDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSERDY (void)

Function description

Check if HSE ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR HSERDYF LL_RCC_IsActiveFlag_HSERDY

LL_RCC_IsActiveFlag_CSIRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_CSIRDY (void)

Function description

Check if CSI ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR CSIRDYF LL_RCC_IsActiveFlag_CSIRDY

LL_RCC_IsActiveFlag_HSI48RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSI48RDY (void)

Function description

Check if HSI48 ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR HSI48RDYF LL_RCC_IsActiveFlag_HSI48RDY

LL_RCC_IsActiveFlag_PLL1RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PLL1RDY (void)

Function description

Check if PLL1 ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR PLLRDYF LL_RCC_IsActiveFlag_PLL1RDY

LL_RCC_IsActiveFlag_PLL2RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PLL2RDY (void)

Function description

Check if PLL2 ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR PLL2RDYF LL_RCC_IsActiveFlag_PLL2RDY

LL_RCC_IsActiveFlag_PLL3RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PLL3RDY (void)

Function description

Check if PLL3 ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR PLL3RDYF LL_RCC_IsActiveFlag_PLL3RDY

LL_RCC_IsActiveFlag_LSECSS

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LSECSS (void)

Function description

Check if LSE Clock security system interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR LSECSSF LL_RCC_IsActiveFlag_LSECSS

LL_RCC_IsActiveFlag_HSECSS

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSECSS (void)

Function description

Check if HSE Clock security system interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR HSECSSF LL_RCC_IsActiveFlag_HSECSS

LL_RCC_IsActiveFlag_LPWRRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LPWRRST (void)

Function description

Check if RCC flag Low Power D1 reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR LPWRRSTF LL_RCC_IsActiveFlag_LPWRRST (*)
- RSR LPWR1RSTF LL_RCC_IsActiveFlag_LPWRRST (**)

LL_RCC_IsActiveFlag_LPWR2RST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LPWR2RST (void)

Function description

Check if RCC flag Low Power D2 reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR LPWR2RSTF LL_RCC_IsActiveFlag_LPWR2RST

LL_RCC_IsActiveFlag_WWDG1RST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_WWDG1RST (void)

Function description

Check if RCC flag Window Watchdog 1 reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR WWDG1RSTF LL_RCC_IsActiveFlag_WWDG1RST

LL_RCC_IsActiveFlag_WWDG2RST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_WWDG2RST (void)

Function description

Check if RCC flag Window Watchdog 2 reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR WWDG2RSTF LL_RCC_IsActiveFlag_WWDG2RST

LL_RCC_IsActiveFlag_IWDG1RST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_IWDG1RST (void)

Function description

Check if RCC flag Independent Watchdog 1 reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR IWDG1RSTF LL_RCC_IsActiveFlag_IWDG1RST

LL_RCC_IsActiveFlag_IWDG2RST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_IWDG2RST (void)

Function description

Check if RCC flag Independent Watchdog 2 reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR IWDG2RSTF LL_RCC_IsActiveFlag_IWDG2RST

LL_RCC_IsActiveFlag_SFTRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_SFTRST (void)

Function description

Check if RCC flag Software reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR SFTRSTF LL_RCC_IsActiveFlag_SFTRST (*)
- RSR SFT1RSTF LL_RCC_IsActiveFlag_SFTRST (**)

LL_RCC_IsActiveFlag_SFT2RST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_SFT2RST (void)

Function description

Check if RCC flag Software reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR SFT2RSTF LL_RCC_IsActiveFlag_SFT2RST

LL_RCC_IsActiveFlag_PORRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PORRST (void)

Function description

Check if RCC flag POR/PDR reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR PORRSTF LL_RCC_IsActiveFlag_PORRST

LL_RCC_IsActiveFlag_PINRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PINRST (void)

Function description

Check if RCC flag Pin reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR PINRSTF LL_RCC_IsActiveFlag_PINRST

LL_RCC_IsActiveFlag_BORRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_BORRST (void)

Function description

Check if RCC flag BOR reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR BORRSTF LL_RCC_IsActiveFlag_BORRST

LL_RCC_IsActiveFlag_D1RST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_D1RST (void)

Function description

Check if RCC flag D1 reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR D1RSTF LL_RCC_IsActiveFlag_D1RST

LL_RCC_IsActiveFlag_D2RST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_D2RST (void)

Function description

Check if RCC flag D2 reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR D2RSTF LL_RCC_IsActiveFlag_D2RST

LL_RCC_IsActiveFlag_CPURST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_CPURST (void)

Function description

Check if RCC flag CPU reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR CPURSTF LL_RCC_IsActiveFlag_CPURST (*)
- RSR C1RSTF LL_RCC_IsActiveFlag_CPURST (**)

LL_RCC_IsActiveFlag_CPU2RST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_CPU2RST (void)

Function description

Check if RCC flag CPU2 reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR C2RSTF LL_RCC_IsActiveFlag_CPU2RST

LL_RCC_ClearResetFlags

Function name

__STATIC_INLINE void LL_RCC_ClearResetFlags (void)

Function description

Set RMVF bit to clear all reset flags.

Return values

- **None:**

Reference Manual to LL API cross reference:

- RSR RMVF LL_RCC_ClearResetFlags

LL_C1_RCC_IsActiveFlag_LPWRRST

Function name

__STATIC_INLINE uint32_t LL_C1_RCC_IsActiveFlag_LPWRRST (void)

Function description

Check if RCC_C1 flag Low Power D1 reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR LPWR1RSTF LL_C1_RCC_IsActiveFlag_LPWRRST

LL_C1_RCC_IsActiveFlag_LPWR2RST

Function name

__STATIC_INLINE uint32_t LL_C1_RCC_IsActiveFlag_LPWR2RST (void)

Function description

Check if RCC_C1 flag Low Power D2 reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR LPWR2RSTF LL_C1_RCC_IsActiveFlag_LPWR2RST

LL_C1_RCC_IsActiveFlag_WWDG1RST

Function name

__STATIC_INLINE uint32_t LL_C1_RCC_IsActiveFlag_WWDG1RST (void)

Function description

Check if RCC_C1 flag Window Watchdog 1 reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR WWDG1RSTF LL_C1_RCC_IsActiveFlag_WWDG1RST

LL_C1_RCC_IsActiveFlag_WWDG2RST

Function name

__STATIC_INLINE uint32_t LL_C1_RCC_IsActiveFlag_WWDG2RST (void)

Function description

Check if RCC_C1 flag Window Watchdog 2 reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR WWDG2RSTF LL_C1_RCC_IsActiveFlag_WWDG2RST

LL_C1_RCC_IsActiveFlag_IWDG1RST

Function name

__STATIC_INLINE uint32_t LL_C1_RCC_IsActiveFlag_IWDG1RST (void)

Function description

Check if RCC_C1 flag Independent Watchdog 1 reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR IWDG1RSTF LL_C1_RCC_IsActiveFlag_IWDG1RST

LL_C1_RCC_IsActiveFlag_IWDG2RST

Function name

__STATIC_INLINE uint32_t LL_C1_RCC_IsActiveFlag_IWDG2RST (void)

Function description

Check if RCC_C1 flag Independent Watchdog 2 reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR IWDG2RSTF LL_C1_RCC_IsActiveFlag_IWDG2RST

LL_C1_RCC_IsActiveFlag_SFTRST

Function name

__STATIC_INLINE uint32_t LL_C1_RCC_IsActiveFlag_SFTRST (void)

Function description

Check if RCC_C1 flag Software reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR SFT1RSTF LL_C1_RCC_IsActiveFlag_SFTRST

LL_C1_RCC_IsActiveFlag_SFT2RST

Function name

__STATIC_INLINE uint32_t LL_C1_RCC_IsActiveFlag_SFT2RST (void)

Function description

Check if RCC_C1 flag Software reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR SFT2RSTF LL_C1_RCC_IsActiveFlag_SFT2RST

LL_C1_RCC_IsActiveFlag_PORRST

Function name

__STATIC_INLINE uint32_t LL_C1_RCC_IsActiveFlag_PORRST (void)

Function description

Check if RCC_C1 flag POR/PDR reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR PORRSTF LL_C1_RCC_IsActiveFlag_PORRST

LL_C1_RCC_IsActiveFlag_PINRST

Function name

__STATIC_INLINE uint32_t LL_C1_RCC_IsActiveFlag_PINRST (void)

Function description

Check if RCC_C1 flag Pin reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR PINRSTF LL_C1_RCC_IsActiveFlag_PINRST

LL_C1_RCC_IsActiveFlag_BORRST

Function name

__STATIC_INLINE uint32_t LL_C1_RCC_IsActiveFlag_BORRST (void)

Function description

Check if RCC_C1 flag BOR reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR BORRSTF LL_C1_RCC_IsActiveFlag_BORRST

LL_C1_RCC_IsActiveFlag_D1RST

Function name

__STATIC_INLINE uint32_t LL_C1_RCC_IsActiveFlag_D1RST (void)

Function description

Check if RCC_C1 flag D1 reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR D1RSTF LL_C1_RCC_IsActiveFlag_D1RST

LL_C1_RCC_IsActiveFlag_D2RST

Function name

__STATIC_INLINE uint32_t LL_C1_RCC_IsActiveFlag_D2RST (void)

Function description

Check if RCC_C1 flag D2 reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR D2RSTF LL_C1_RCC_IsActiveFlag_D2RST

LL_C1_RCC_IsActiveFlag_CPURST

Function name

__STATIC_INLINE uint32_t LL_C1_RCC_IsActiveFlag_CPURST (void)

Function description

Check if RCC_C1 flag CPU reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR C1RSTF LL_C1_RCC_IsActiveFlag_CPURST

LL_C1_RCC_IsActiveFlag_CPU2RST

Function name

__STATIC_INLINE uint32_t LL_C1_RCC_IsActiveFlag_CPU2RST (void)

Function description

Check if RCC_C1 flag CPU2 reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR C2RSTF LL_C1_RCC_IsActiveFlag_CPU2RST

LL_C1_RCC_ClearResetFlags

Function name

__STATIC_INLINE void LL_C1_RCC_ClearResetFlags (void)

Function description

Set RMVF bit to clear the reset flags.

Return values

- **None:**

Reference Manual to LL API cross reference:

- RSR RMVF LL_C1_RCC_ClearResetFlags

LL_C2_RCC_IsActiveFlag_LPWRRST

Function name

__STATIC_INLINE uint32_t LL_C2_RCC_IsActiveFlag_LPWRRST (void)

Function description

Check if RCC_C2 flag Low Power D1 reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR LPWR1RSTF LL_C2_RCC_IsActiveFlag_LPWRRST

LL_C2_RCC_IsActiveFlag_LPWR2RST

Function name

__STATIC_INLINE uint32_t LL_C2_RCC_IsActiveFlag_LPWR2RST (void)

Function description

Check if RCC_C2 flag Low Power D2 reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR LPWR2RSTF LL_C2_RCC_IsActiveFlag_LPWR2RST

LL_C2_RCC_IsActiveFlag_WWDG1RST

Function name

__STATIC_INLINE uint32_t LL_C2_RCC_IsActiveFlag_WWDG1RST (void)

Function description

Check if RCC_C2 flag Window Watchdog 1 reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR WWDG1RSTF LL_C2_RCC_IsActiveFlag_WWDG1RST

LL_C2_RCC_IsActiveFlag_WWDG2RST

Function name

__STATIC_INLINE uint32_t LL_C2_RCC_IsActiveFlag_WWDG2RST (void)

Function description

Check if RCC_C2 flag Window Watchdog 2 reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR WWDG2RSTF LL_C2_RCC_IsActiveFlag_WWDG2RST

LL_C2_RCC_IsActiveFlag_IWDG1RST

Function name

__STATIC_INLINE uint32_t LL_C2_RCC_IsActiveFlag_IWDG1RST (void)

Function description

Check if RCC_C2 flag Independent Watchdog 1 reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR IWDG1RSTF LL_C2_RCC_IsActiveFlag_IWDG1RST

LL_C2_RCC_IsActiveFlag_IWDG2RST

Function name

__STATIC_INLINE uint32_t LL_C2_RCC_IsActiveFlag_IWDG2RST (void)

Function description

Check if RCC_C2 flag Independent Watchdog 2 reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR IWDG2RSTF LL_C2_RCC_IsActiveFlag_IWDG2RST

LL_C2_RCC_IsActiveFlag_SFTRST

Function name

__STATIC_INLINE uint32_t LL_C2_RCC_IsActiveFlag_SFTRST (void)

Function description

Check if RCC_C2 flag Software reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR SFT1RSTF LL_C2_RCC_IsActiveFlag_SFTRST

LL_C2_RCC_IsActiveFlag_SFT2RST

Function name

__STATIC_INLINE uint32_t LL_C2_RCC_IsActiveFlag_SFT2RST (void)

Function description

Check if RCC_C2 flag Software reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR SFT2RSTF LL_C2_RCC_IsActiveFlag_SFT2RST

LL_C2_RCC_IsActiveFlag_PORRST

Function name

__STATIC_INLINE uint32_t LL_C2_RCC_IsActiveFlag_PORRST (void)

Function description

Check if RCC_C2 flag POR/PDR reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR PORRSTF LL_C2_RCC_IsActiveFlag_PORRST

LL_C2_RCC_IsActiveFlag_PINRST

Function name

__STATIC_INLINE uint32_t LL_C2_RCC_IsActiveFlag_PINRST (void)

Function description

Check if RCC_C2 flag Pin reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR PINRSTF LL_C2_RCC_IsActiveFlag_PINRST

LL_C2_RCC_IsActiveFlag_BORRST

Function name

__STATIC_INLINE uint32_t LL_C2_RCC_IsActiveFlag_BORRST (void)

Function description

Check if RCC_C2 flag BOR reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR BORRSTF LL_C2_RCC_IsActiveFlag_BORRST

LL_C2_RCC_IsActiveFlag_D1RST

Function name

__STATIC_INLINE uint32_t LL_C2_RCC_IsActiveFlag_D1RST (void)

Function description

Check if RCC_C2 flag D1 reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR D1RSTF LL_C2_RCC_IsActiveFlag_D1RST

LL_C2_RCC_IsActiveFlag_D2RST

Function name

__STATIC_INLINE uint32_t LL_C2_RCC_IsActiveFlag_D2RST (void)

Function description

Check if RCC_C2 flag D2 reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR D2RSTF LL_C2_RCC_IsActiveFlag_D2RST

LL_C2_RCC_IsActiveFlag_CPURST

Function name

__STATIC_INLINE uint32_t LL_C2_RCC_IsActiveFlag_CPURST (void)

Function description

Check if RCC_C2 flag CPU reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR C1RSTF LL_C2_RCC_IsActiveFlag_CPURST

LL_C2_RCC_IsActiveFlag_CPU2RST

Function name

__STATIC_INLINE uint32_t LL_C2_RCC_IsActiveFlag_CPU2RST (void)

Function description

Check if RCC_C2 flag CPU2 reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RSR C2RSTF LL_C2_RCC_IsActiveFlag_CPU2RST

LL_C2_RCC_ClearResetFlags

Function name

__STATIC_INLINE void LL_C2_RCC_ClearResetFlags (void)

Function description

Set RMVF bit to clear the reset flags.

Return values

- **None:**

Reference Manual to LL API cross reference:

- RSR RMVF LL_C2_RCC_ClearResetFlags

LL_RCC_EnableIT_LSIRDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_LSIRDY (void)

Function description

Enable LSI ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER LSIRDYIE LL_RCC_EnableIT_LSIRDY

LL_RCC_EnableIT_LSERDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_LSERDY (void)

Function description

Enable LSE ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER LSERDYIE LL_RCC_EnableIT_LSERDY

LL_RCC_EnableIT_HSIRDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_HSIRDY (void)

Function description

Enable HSI ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSIRDYIE LL_RCC_EnableIT_HSIRDY

LL_RCC_EnableIT_HSERDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_HSERDY (void)

Function description

Enable HSE ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSERDYIE LL_RCC_EnableIT_HSERDY

LL_RCC_EnableIT_CSIRDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_CSIRDY (void)

Function description

Enable CSI ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER CSIRDYIE LL_RCC_EnableIT_CSIRDY

LL_RCC_EnableIT_HSI48RDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_HSI48RDY (void)

Function description

Enable HSI48 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSI48RDYIE LL_RCC_EnableIT_HSI48RDY

LL_RCC_EnableIT_PLL1RDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_PLL1RDY (void)

Function description

Enable PLL1 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER PLL1RDYIE LL_RCC_EnableIT_PLL1RDY

LL_RCC_EnableIT_PLL2RDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_PLL2RDY (void)

Function description

Enable PLL2 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER PLL2RDYIE LL_RCC_EnableIT_PLL2RDY

LL_RCC_EnableIT_PLL3RDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_PLL3RDY (void)

Function description

Enable PLL3 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER PLL3RDYIE LL_RCC_EnableIT_PLL3RDY

LL_RCC_EnableIT_LSECSS

Function name

__STATIC_INLINE void LL_RCC_EnableIT_LSECSS (void)

Function description

Enable LSECSS interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER LSECSSIE LL_RCC_EnableIT_LSECSS

LL_RCC_DisableIT_LSIRDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_LSIRDY (void)

Function description

Disable LSI ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER LSIRDYIE LL_RCC_DisableIT_LSIRDY

LL_RCC_DisableIT_LSERDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_LSERDY (void)

Function description

Disable LSE ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER LSERDYIE LL_RCC_DisableIT_LSERDY

LL_RCC_DisableIT_HSIRDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_HSIRDY (void)

Function description

Disable HSI ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSIRDYIE LL_RCC_DisableIT_HSIRDY

LL_RCC_DisableIT_HSERDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_HSERDY (void)

Function description

Disable HSE ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSERDYIE LL_RCC_DisableIT_HSERDY

LL_RCC_DisableIT_CSIRDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_CSIRDY (void)

Function description

Disable CSI ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER CSIRDYIE LL_RCC_DisableIT_CSIRDY

LL_RCC_DisableIT_HSI48RDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_HSI48RDY (void)

Function description

Disable HSI48 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSI48RDYIE LL_RCC_DisableIT_HSI48RDY

LL_RCC_DisableIT_PLL1RDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_PLL1RDY (void)

Function description

Disable PLL1 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER PLL1RDYIE LL_RCC_DisableIT_PLL1RDY

LL_RCC_DisableIT_PLL2RDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_PLL2RDY (void)

Function description

Disable PLL2 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER PLL2RDYIE LL_RCC_DisableIT_PLL2RDY

LL_RCC_DisableIT_PLL3RDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_PLL3RDY (void)

Function description

Disable PLL3 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER PLL3RDYIE LL_RCC_DisableIT_PLL3RDY

LL_RCC_DisableIT_LSECSS

Function name

__STATIC_INLINE void LL_RCC_DisableIT_LSECSS (void)

Function description

Disable LSECSS interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER LSECSSIE LL_RCC_DisableIT_LSECSS

LL_RCC_IsEnableIT_LSIRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnableIT_LSIRDY (void)

Function description

Checks if LSI ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER LSIRDYIE LL_RCC_IsEnableIT_LSIRDY

LL_RCC_IsEnableIT_LSERDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnableIT_LSERDY (void)

Function description

Checks if LSE ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER LSERDYIE LL_RCC_IsEnableIT_LSERDY

LL_RCC_IsEnableIT_HSIRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnableIT_HSIRDY (void)

Function description

Checks if HSI ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER HSIRDYIE LL_RCC_IsEnableIT_HSIRDY

LL_RCC_IsEnableIT_HSERDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnableIT_HSERDY (void)

Function description

Checks if HSE ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER HSERDYIE LL_RCC_IsEnableIT_HSERDY

LL_RCC_IsEnableIT_CSIRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnableIT_CSIRDY (void)

Function description

Checks if CSI ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER CSIRDYIE LL_RCC_IsEnableIT_CSIRDY

LL_RCC_IsEnableIT_HSI48RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnableIT_HSI48RDY (void)

Function description

Checks if HSI48 ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER HSI48RDYIE LL_RCC_IsEnableIT_HSI48RDY

LL_RCC_IsEnableIT_PLL1RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnableIT_PLL1RDY (void)

Function description

Checks if PLL1 ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER PLL1RDYIE LL_RCC_IsEnableIT_PLL1RDY

LL_RCC_IsEnableIT_PLL2RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnableIT_PLL2RDY (void)

Function description

Checks if PLL2 ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER PLL2RDYIE LL_RCC_IsEnableIT_PLL2RDY

LL_RCC_IsEnableIT_PLL3RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnableIT_PLL3RDY (void)

Function description

Checks if PLL3 ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER PLL3RDYIE LL_RCC_IsEnableIT_PLL3RDY

LL_RCC_IsEnableIT_LSECSS

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnableIT_LSECSS (void)

Function description

Checks if LSECSS interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER LSECSSIE LL_RCC_IsEnableIT_LSECSS

LL_RCC_DeInit

Function name

void LL_RCC_DeInit (void)

Function description

Resets the RCC clock configuration to the default reset state.

Return values

- **None:**

Notes

- The default reset state of the clock configuration is given below: HSI ON and used as system clock sourceHSE, PLL1, PLL2 and PLL3 OFFAHB, APB Bus pre-scaler set to 1.CSS, MCO1 and MCO2 OFFAll interrupts disabled
- This function doesn't modify the configuration of the Peripheral clocksLSI, LSE and RTC clocks

LL_RCC_CalcPLLClockFreq

Function name

uint32_t LL_RCC_CalcPLLClockFreq (uint32_t PLLInputFreq, uint32_t M, uint32_t N, uint32_t FRACN, uint32_t PQR)

Function description

Helper function to calculate the PLL frequency output.

Parameters

- **PLLInputFreq:** PLL Input frequency (based on HSE/(HSI/HSIDIV)/CSI)
- **M:** Between 1 and 63
- **N:** Between 4 and 512
- **FRACN:** Between 0 and 0x1FFF
- **PQR:** VCO output divider (P, Q or R) Between 1 and 128, except for PLL1P Odd value not allowed

Return values

- **PLL1:** clock frequency (in Hz)

Notes

- ex: LL_RCC_CalcPLLClockFreq (HSE_VALUE, LL_RCC_PLL1_GetM (), LL_RCC_PLL1_GetN (), LL_RCC_PLL1_GetFRACN (), LL_RCC_PLL1_GetP ());

LL_RCC_GetPLL1ClockFreq

Function name

void LL_RCC_GetPLL1ClockFreq (LL_PLL_ClocksTypeDef * PLL_Clocks)

Function description

Return PLL1 clocks frequencies.

Return values

- **None:**

Notes

- LL_RCC_PERIPH_FREQUENCY_NO returned for non activated output or oscillator not ready

LL_RCC_GetPLL2ClockFreq

Function name

void LL_RCC_GetPLL2ClockFreq (LL_PLL_ClocksTypeDef * PLL_Clocks)

Function description

Return PLL2 clocks frequencies.

Return values

- **None:**

Notes

- LL_RCC_PERIPH_FREQUENCY_NO returned for non activated output or oscillator not ready

LL_RCC_GetPLL3ClockFreq

Function name

void LL_RCC_GetPLL3ClockFreq (LL_PLL_ClocksTypeDef * PLL_Clocks)

Function description

Return PLL3 clocks frequencies.

Return values

- **None:**

Notes

- LL_RCC_PERIPH_FREQUENCY_NO returned for non activated output or oscillator not ready

LL_RCC_GetSystemClocksFreq

Function name

void LL_RCC_GetSystemClocksFreq (LL_RCC_ClocksTypeDef * RCC_Clocks)

Function description

Return the frequencies of different on chip clocks; System, AHB, APB1, APB2, APB3 and APB4 buses clocks.

Parameters

- **RCC_Clocks:** pointer to a LL_RCC_ClocksTypeDef structure which will hold the clocks frequencies

Return values

- **None:**

Notes

- Each time SYSCLK, HCLK, PCLK1, PCLK2, PCLK3 and/or PCLK4 clock changes, this function must be called to update structure fields. Otherwise, any configuration based on this function will be incorrect.

LL_RCC_GetUSARTClockFreq

Function name

uint32_t LL_RCC_GetUSARTClockFreq (uint32_t USARTxSource)

Function description

Return USARTx clock frequency.

Parameters

- **USARTxSource:** This parameter can be one of the following values:
  - LL_RCC_USART16_CLKSOURCE
  - LL_RCC_USART234578_CLKSOURCE

Return values

- **USART:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetLPUARTClockFreq

Function name

uint32_t LL_RCC_GetLPUARTClockFreq (uint32_t LPUARTxSource)

Function description

Return LPUART clock frequency.

Parameters

- **LPUARTxSource:** This parameter can be one of the following values:
  - LL_RCC_LPUART1_CLKSOURCE

Return values

- **LPUART:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetI2CClockFreq

Function name

uint32_t LL_RCC_GetI2CClockFreq (uint32_t I2CxSource)

Function description

Return I2Cx clock frequency.

Parameters

- **I2CxSource:** This parameter can be one of the following values:
  - LL_RCC_I2C123_CLKSOURCE
  - LL_RCC_I2C4_CLKSOURCE

Return values

- **I2C:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetLPTIMClockFreq

Function name

uint32_t LL_RCC_GetLPTIMClockFreq (uint32_t LPTIMxSource)

Function description

Return LPTIMx clock frequency.

Parameters

- **LPTIMxSource:** This parameter can be one of the following values:
  - LL_RCC_LPTIM1_CLKSOURCE
  - LL_RCC_LPTIM2_CLKSOURCE
  - LL_RCC_LPTIM345_CLKSOURCE

Return values

- **LPTIM:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetSAIClockFreq

Function name

uint32_t LL_RCC_GetSAIClockFreq (uint32_t SAIxSource)

Function description

Return SAIx clock frequency.

Parameters

- **SAIxSource:** This parameter can be one of the following values:
  - LL_RCC_SAI1_CLKSOURCE
  - LL_RCC_SAI23_CLKSOURCE (*)
  - LL_RCC_SAI2A_CLKSOURCE (*)
  - LL_RCC_SAI2B_CLKSOURCE (*)
  - LL_RCC_SAI4A_CLKSOURCE (*)
  - LL_RCC_SAI4B_CLKSOURCE (*)

Return values

- **SAI:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetADCClockFreq

Function name

uint32_t LL_RCC_GetADCClockFreq (uint32_t ADCxSource)

Function description

Return ADC clock frequency.

Parameters

- **ADCxSource:** This parameter can be one of the following values:
  - LL_RCC_ADC_CLKSOURCE

Return values

- **ADC:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetSDMMCClockFreq

Function name

uint32_t LL_RCC_GetSDMMCClockFreq (uint32_t SDMMCxSource)

Function description

Return SDMMC clock frequency.

Parameters

- **SDMMCxSource:** This parameter can be one of the following values:
  - LL_RCC_SDMMC_CLKSOURCE

Return values

- **SDMMC:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetRNGClockFreq

Function name

uint32_t LL_RCC_GetRNGClockFreq (uint32_t RNGxSource)

Function description

Return RNG clock frequency.

Parameters

- **RNGxSource:** This parameter can be one of the following values:
  - LL_RCC_RNG_CLKSOURCE

Return values

- **RNG:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetCECClockFreq

Function name

uint32_t LL_RCC_GetCECClockFreq (uint32_t CECxSource)

Function description

Return CEC clock frequency.

Parameters

- **CECxSource:** This parameter can be one of the following values:
  - LL_RCC_RNG_CLKSOURCE

Return values

- **CEC:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetUSBClockFreq

Function name

uint32_t LL_RCC_GetUSBClockFreq (uint32_t USBxSource)

Function description

Return USB clock frequency.

Parameters

- **USBxSource:** This parameter can be one of the following values:
  - LL_RCC_USB_CLKSOURCE

Return values

- **USB:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready or Disabled

LL_RCC_GetDFSDMClockFreq

Function name

uint32_t LL_RCC_GetDFSDMClockFreq (uint32_t DFSDMxSource)

Function description

Return DFSDM clock frequency.

Parameters

- **DFSDMxSource:** This parameter can be one of the following values:
  - LL_RCC_DFSDM1_CLKSOURCE

Return values

- **DFSDM:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetSPDIFClockFreq

Function name

uint32_t LL_RCC_GetSPDIFClockFreq (uint32_t SPDIFxSource)

Function description

Return SPDIF clock frequency.

Parameters

- **SPDIFxSource:** This parameter can be one of the following values:
  - LL_RCC_SPDIF_CLKSOURCE

Return values

- **SPDIF:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetSPIClockFreq

Function name

uint32_t LL_RCC_GetSPIClockFreq (uint32_t SPIxSource)

Function description

Return SPIx clock frequency.

Parameters

- **SPIxSource:** This parameter can be one of the following values:
  - LL_RCC_SPI123_CLKSOURCE
  - LL_RCC_SPI45_CLKSOURCE
  - LL_RCC_SPI6_CLKSOURCE

Return values

- **SPI:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetSWPClockFreq

Function name

uint32_t LL_RCC_GetSWPClockFreq (uint32_t SWPxSource)

Function description

Return SWP clock frequency.

Parameters

- **SWPxSource:** This parameter can be one of the following values:
  - LL_RCC_SWP_CLKSOURCE

Return values

- **SWP:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetFDCANClockFreq

Function name

uint32_t LL_RCC_GetFDCANClockFreq (uint32_t FDCANxSource)

Function description

Return FDCAN clock frequency.

Parameters

- **FDCANxSource:** This parameter can be one of the following values:
  - LL_RCC_FDCAN_CLKSOURCE

Return values

- **FDCAN:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetFMCClockFreq

Function name

uint32_t LL_RCC_GetFMCClockFreq (uint32_t FMCxSource)

Function description

Return FMC clock frequency.

Parameters

- **FMCxSource:** This parameter can be one of the following values:
  - LL_RCC_FMC_CLKSOURCE

Return values

- **FMC:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetQSPIClockFreq

Function name

uint32_t LL_RCC_GetQSPIClockFreq (uint32_t QSPIxSource)

Function description

Return QSPI clock frequency.

Parameters

- **QSPIxSource:** This parameter can be one of the following values:
  - LL_RCC_QSPI_CLKSOURCE

Return values

- **QSPI:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetCLKPClockFreq

Function name

uint32_t LL_RCC_GetCLKPClockFreq (uint32_t CLKPxSource)

Function description

Return CLKP clock frequency.

Parameters

- **CLKPxSource:** This parameter can be one of the following values:
  - LL_RCC_CLKP_CLKSOURCE

Return values

- **CLKP:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

120.3 RCC Firmware driver defines

The following section lists the various define and macros of the module.

120.3.1 RCC

RCC

Peripheral ADC get clock source

LL_RCC_ADC_CLKSOURCE

Peripheral ADC clock source selection

LL_RCC_ADC_CLKSOURCE_PLL2P

LL_RCC_ADC_CLKSOURCE_PLL3R

LL_RCC_ADC_CLKSOURCE_CLKP

AHB prescaler

LL_RCC_AHB_DIV_1

LL_RCC_AHB_DIV_2

LL_RCC_AHB_DIV_4

LL_RCC_AHB_DIV_8

LL_RCC_AHB_DIV_16

LL_RCC_AHB_DIV_64

LL_RCC_AHB_DIV_128

LL_RCC_AHB_DIV_256

LL_RCC_AHB_DIV_512

APB low-speed prescaler (APB1)

LL_RCC_APB1_DIV_1

LL_RCC_APB1_DIV_2

LL_RCC_APB1_DIV_4

LL_RCC_APB1_DIV_8

LL_RCC_APB1_DIV_16

APB low-speed prescaler (APB2)

LL_RCC_APB2_DIV_1

LL_RCC_APB2_DIV_2

LL_RCC_APB2_DIV_4

LL_RCC_APB2_DIV_8

LL_RCC_APB2_DIV_16

APB low-speed prescaler (APB3)

LL_RCC_APB3_DIV_1

LL_RCC_APB3_DIV_2

LL_RCC_APB3_DIV_4

LL_RCC_APB3_DIV_8

LL_RCC_APB3_DIV_16

APB low-speed prescaler (APB4)

LL_RCC_APB4_DIV_1

LL_RCC_APB4_DIV_2

LL_RCC_APB4_DIV_4

LL_RCC_APB4_DIV_8

LL_RCC_APB4_DIV_16

Peripheral CEC get clock source

LL_RCC_CEC_CLKSOURCE

Peripheral CEC clock source selection

LL_RCC_CEC_CLKSOURCE_LSE

LL_RCC_CEC_CLKSOURCE_LSI

LL_RCC_CEC_CLKSOURCE_CSI_DIV122

Peripheral CLKP get clock source

LL_RCC_CLKP_CLKSOURCE

Peripheral CLKP clock source selection

LL_RCC_CLKP_CLKSOURCE_HSI

LL_RCC_CLKP_CLKSOURCE_CSI

LL_RCC_CLKP_CLKSOURCE_HSE

Peripheral DFSDM get clock source

LL_RCC_DFSDM1_CLKSOURCE

Peripheral DFSDM clock source selection

LL_RCC_DFSDM1_CLKSOURCE_PCLK2

LL_RCC_DFSDM1_CLKSOURCE_SYSCLK

Peripheral FDCAN get clock source

LL_RCC_FDCAN_CLKSOURCE

Peripheral FDCAN clock source selection

LL_RCC_FDCAN_CLKSOURCE_HSE

LL_RCC_FDCAN_CLKSOURCE_PLL1Q

LL_RCC_FDCAN_CLKSOURCE_PLL2Q

Peripheral FMC get clock source

LL_RCC_FMC_CLKSOURCE

Peripheral FMC clock source selection

LL_RCC_FMC_CLKSOURCE_HCLK

LL_RCC_FMC_CLKSOURCE_PLL1Q

LL_RCC_FMC_CLKSOURCE_PLL2R

LL_RCC_FMC_CLKSOURCE_CLKP

High Resolution Timers clock selection

LL_RCC_HRTIM_CLKSOURCE_TIM

LL_RCC_HRTIM_CLKSOURCE_CPU

HSI oscillator divider

LL_RCC_HSI_DIV1

LL_RCC_HSI_DIV2

LL_RCC_HSI_DIV4

LL_RCC_HSI_DIV8

Peripheral I2C get clock source

LL_RCC_I2C123_CLKSOURCE

LL_RCC_I2C1235_CLKSOURCE

LL_RCC_I2C4_CLKSOURCE

Peripheral I2C clock source selection

LL_RCC_I2C123_CLKSOURCE_PCLK1

LL_RCC_I2C123_CLKSOURCE_PLL3R

LL_RCC_I2C123_CLKSOURCE_HSI

LL_RCC_I2C123_CLKSOURCE_CSI

LL_RCC_I2C1235_CLKSOURCE_PCLK1

LL_RCC_I2C1235_CLKSOURCE_PLL3R

LL_RCC_I2C1235_CLKSOURCE_HSI

LL_RCC_I2C1235_CLKSOURCE_CSI

LL_RCC_I2C4_CLKSOURCE_PCLK4

LL_RCC_I2C4_CLKSOURCE_PLL3R

LL_RCC_I2C4_CLKSOURCE_HSI

LL_RCC_I2C4_CLKSOURCE_CSI

Kernel wakeup clock source

LL_RCC_KERWAKEUP_CLKSOURCE_HSI

LL_RCC_KERWAKEUP_CLKSOURCE_CSI

Peripheral LPTIM get clock source

LL_RCC_LPTIM1_CLKSOURCE

LL_RCC_LPTIM2_CLKSOURCE

LL_RCC_LPTIM345_CLKSOURCE

Peripheral LPTIM clock source selection

LL_RCC_LPTIM1_CLKSOURCE_PCLK1

LL_RCC_LPTIM1_CLKSOURCE_PLL2P

LL_RCC_LPTIM1_CLKSOURCE_PLL3R

LL_RCC_LPTIM1_CLKSOURCE_LSE

LL_RCC_LPTIM1_CLKSOURCE_LSI

LL_RCC_LPTIM1_CLKSOURCE_CLKP

LL_RCC_LPTIM2_CLKSOURCE_PCLK4

LL_RCC_LPTIM2_CLKSOURCE_PLL2P

LL_RCC_LPTIM2_CLKSOURCE_PLL3R

LL_RCC_LPTIM2_CLKSOURCE_LSE

LL_RCC_LPTIM2_CLKSOURCE_LSI

LL_RCC_LPTIM2_CLKSOURCE_CLKP

LL_RCC_LPTIM345_CLKSOURCE_PCLK4

LL_RCC_LPTIM345_CLKSOURCE_PLL2P

LL_RCC_LPTIM345_CLKSOURCE_PLL3R

LL_RCC_LPTIM345_CLKSOURCE_LSE

LL_RCC_LPTIM345_CLKSOURCE_LSI

LL_RCC_LPTIM345_CLKSOURCE_CLKP

Peripheral LPUART get clock source

LL_RCC_LPUART1_CLKSOURCE

Peripheral LPUART clock source selection

LL_RCC_LPUART1_CLKSOURCE_PCLK4

LL_RCC_LPUART1_CLKSOURCE_PLL2Q

LL_RCC_LPUART1_CLKSOURCE_PLL3Q

LL_RCC_LPUART1_CLKSOURCE_HSI

LL_RCC_LPUART1_CLKSOURCE_CSI

LL_RCC_LPUART1_CLKSOURCE_LSE

LSE oscillator drive capability

LL_RCC_LSEDRIVE_LOW

LL_RCC_LSEDRIVE_MEDIUMLOW

LL_RCC_LSEDRIVE_MEDIUMHIGH

LL_RCC_LSEDRIVE_HIGH

MCO source selection

LL_RCC_MCO1SOURCE_HSI

LL_RCC_MCO1SOURCE_LSE

LL_RCC_MCO1SOURCE_HSE

LL_RCC_MCO1SOURCE_PLL1QCLK

LL_RCC_MCO1SOURCE_HSI48

LL_RCC_MCO2SOURCE_SYSCLK

LL_RCC_MCO2SOURCE_PLL2PCLK

LL_RCC_MCO2SOURCE_HSE

LL_RCC_MCO2SOURCE_PLL1PCLK

LL_RCC_MCO2SOURCE_CSI

LL_RCC_MCO2SOURCE_LSI

MCO prescaler

LL_RCC_MCO1_DIV_1

LL_RCC_MCO1_DIV_2

LL_RCC_MCO1_DIV_3

LL_RCC_MCO1_DIV_4

LL_RCC_MCO1_DIV_5

LL_RCC_MCO1_DIV_6

LL_RCC_MCO1_DIV_7

LL_RCC_MCO1_DIV_8

LL_RCC_MCO1_DIV_9

LL_RCC_MCO1_DIV_10

LL_RCC_MCO1_DIV_11

LL_RCC_MCO1_DIV_12

LL_RCC_MCO1_DIV_13

LL_RCC_MCO1_DIV_14

LL_RCC_MCO1_DIV_15

LL_RCC_MCO2_DIV_1

LL_RCC_MCO2_DIV_2

LL_RCC_MCO2_DIV_3

LL_RCC_MCO2_DIV_4

LL_RCC_MCO2_DIV_5

LL_RCC_MCO2_DIV_6

LL_RCC_MCO2_DIV_7

LL_RCC_MCO2_DIV_8

LL_RCC_MCO2_DIV_9

LL_RCC_MCO2_DIV_10

LL_RCC_MCO2_DIV_11

LL_RCC_MCO2_DIV_12

LL_RCC_MCO2_DIV_13

LL_RCC_MCO2_DIV_14

LL_RCC_MCO2_DIV_15

Oscillator Values adaptation

HSI48_VALUE

Value of the HSI48 oscillator in Hz

Peripheral clock frequency

LL_RCC_PERIPH_FREQUENCY_NO

No clock enabled for the peripheral

LL_RCC_PERIPH_FREQUENCY_NA

Frequency cannot be provided as external clock

All PLLs input range

LL_RCC_PLLINPUTRANGE_1_2

LL_RCC_PLLINPUTRANGE_2_4

LL_RCC_PLLINPUTRANGE_4_8

LL_RCC_PLLINPUTRANGE_8_16

All PLLs entry clock source

LL_RCC_PLLSOURCE_HSI

LL_RCC_PLLSOURCE_CSI

LL_RCC_PLLSOURCE_HSE

LL_RCC_PLLSOURCE_NONE

All PLLs VCO range

LL_RCC_PLLVCORANGE_WIDE

LL_RCC_PLLVCORANGE_MEDIUM

Peripheral QSPI get clock source

LL_RCC_QSPI_CLKSOURCE

Peripheral QSPI clock source selection

LL_RCC_QSPI_CLKSOURCE_HCLK

LL_RCC_QSPI_CLKSOURCE_PLL1Q

LL_RCC_QSPI_CLKSOURCE_PLL2R

LL_RCC_QSPI_CLKSOURCE_CLKP

Peripheral RNG get clock source

LL_RCC_RNG_CLKSOURCE

Peripheral RNG clock source selection

LL_RCC_RNG_CLKSOURCE_HSI48

LL_RCC_RNG_CLKSOURCE_PLL1Q

LL_RCC_RNG_CLKSOURCE_LSE

LL_RCC_RNG_CLKSOURCE_LSI

RTC clock source selection

LL_RCC_RTC_CLKSOURCE_NONE

LL_RCC_RTC_CLKSOURCE_LSE

LL_RCC_RTC_CLKSOURCE_LSI

LL_RCC_RTC_CLKSOURCE_HSE

HSE prescaler for RTC clock

LL_RCC_RTC_NOCLOCK

LL_RCC_RTC_HSE_DIV_2

LL_RCC_RTC_HSE_DIV_3

LL_RCC_RTC_HSE_DIV_4

LL_RCC_RTC_HSE_DIV_5

LL_RCC_RTC_HSE_DIV_6

LL_RCC_RTC_HSE_DIV_7

LL_RCC_RTC_HSE_DIV_8

LL_RCC_RTC_HSE_DIV_9

LL_RCC_RTC_HSE_DIV_10

LL_RCC_RTC_HSE_DIV_11

LL_RCC_RTC_HSE_DIV_12

LL_RCC_RTC_HSE_DIV_13

LL_RCC_RTC_HSE_DIV_14

LL_RCC_RTC_HSE_DIV_15

LL_RCC_RTC_HSE_DIV_16

LL_RCC_RTC_HSE_DIV_17

LL_RCC_RTC_HSE_DIV_18

LL_RCC_RTC_HSE_DIV_19

LL_RCC_RTC_HSE_DIV_20

LL_RCC_RTC_HSE_DIV_21

LL_RCC_RTC_HSE_DIV_22

LL_RCC_RTC_HSE_DIV_23

LL_RCC_RTC_HSE_DIV_24

LL_RCC_RTC_HSE_DIV_25

LL_RCC_RTC_HSE_DIV_26

LL_RCC_RTC_HSE_DIV_27

LL_RCC_RTC_HSE_DIV_28

LL_RCC_RTC_HSE_DIV_29

LL_RCC_RTC_HSE_DIV_30

LL_RCC_RTC_HSE_DIV_31

LL_RCC_RTC_HSE_DIV_32

LL_RCC_RTC_HSE_DIV_33

LL_RCC_RTC_HSE_DIV_34

LL_RCC_RTC_HSE_DIV_35

LL_RCC_RTC_HSE_DIV_36

LL_RCC_RTC_HSE_DIV_37

LL_RCC_RTC_HSE_DIV_38

LL_RCC_RTC_HSE_DIV_39

LL_RCC_RTC_HSE_DIV_40

LL_RCC_RTC_HSE_DIV_41

LL_RCC_RTC_HSE_DIV_42

LL_RCC_RTC_HSE_DIV_43

LL_RCC_RTC_HSE_DIV_44

LL_RCC_RTC_HSE_DIV_45

LL_RCC_RTC_HSE_DIV_46

LL_RCC_RTC_HSE_DIV_47

LL_RCC_RTC_HSE_DIV_48

LL_RCC_RTC_HSE_DIV_49

LL_RCC_RTC_HSE_DIV_50

LL_RCC_RTC_HSE_DIV_51

LL_RCC_RTC_HSE_DIV_52

LL_RCC_RTC_HSE_DIV_53

LL_RCC_RTC_HSE_DIV_54

LL_RCC_RTC_HSE_DIV_55

LL_RCC_RTC_HSE_DIV_56

LL_RCC_RTC_HSE_DIV_57

LL_RCC_RTC_HSE_DIV_58

LL_RCC_RTC_HSE_DIV_59

LL_RCC_RTC_HSE_DIV_60

LL_RCC_RTC_HSE_DIV_61

LL_RCC_RTC_HSE_DIV_62

LL_RCC_RTC_HSE_DIV_63

Peripheral SAI get clock source

LL_RCC_SAI1_CLKSOURCE

LL_RCC_SAI23_CLKSOURCE

LL_RCC_SAI4A_CLKSOURCE

LL_RCC_SAI4B_CLKSOURCE

Peripheral SAI clock source selection

LL_RCC_SAI1_CLKSOURCE_PLL1Q

LL_RCC_SAI1_CLKSOURCE_PLL2P

LL_RCC_SAI1_CLKSOURCE_PLL3P

LL_RCC_SAI1_CLKSOURCE_I2S_CKIN

LL_RCC_SAI1_CLKSOURCE_CLKP

LL_RCC_SAI23_CLKSOURCE_PLL1Q

LL_RCC_SAI23_CLKSOURCE_PLL2P

LL_RCC_SAI23_CLKSOURCE_PLL3P

LL_RCC_SAI23_CLKSOURCE_I2S_CKIN

LL_RCC_SAI23_CLKSOURCE_CLKP

LL_RCC_SAI4A_CLKSOURCE_PLL1Q

LL_RCC_SAI4A_CLKSOURCE_PLL2P

LL_RCC_SAI4A_CLKSOURCE_PLL3P

LL_RCC_SAI4A_CLKSOURCE_I2S_CKIN

LL_RCC_SAI4A_CLKSOURCE_CLKP

LL_RCC_SAI4B_CLKSOURCE_PLL1Q

LL_RCC_SAI4B_CLKSOURCE_PLL2P

LL_RCC_SAI4B_CLKSOURCE_PLL3P

LL_RCC_SAI4B_CLKSOURCE_I2S_CKIN

LL_RCC_SAI4B_CLKSOURCE_CLKP

Peripheral SDMMC get clock source

LL_RCC_SDMMC_CLKSOURCE

Peripheral SDMMC clock source selection

LL_RCC_SDMMC_CLKSOURCE_PLL1Q

LL_RCC_SDMMC_CLKSOURCE_PLL2R

Peripheral SPDIF get clock source

LL_RCC_SPDIF_CLKSOURCE

Peripheral SPDIF clock source selection

LL_RCC_SPDIF_CLKSOURCE_PLL1Q

LL_RCC_SPDIF_CLKSOURCE_PLL2R

LL_RCC_SPDIF_CLKSOURCE_PLL3R

LL_RCC_SPDIF_CLKSOURCE_HSI

Peripheral SPI get clock source

LL_RCC_SPI123_CLKSOURCE

LL_RCC_SPI45_CLKSOURCE

LL_RCC_SPI6_CLKSOURCE

Peripheral SPI clock source selection

LL_RCC_SPI123_CLKSOURCE_PLL1Q

LL_RCC_SPI123_CLKSOURCE_PLL2P

LL_RCC_SPI123_CLKSOURCE_PLL3P

LL_RCC_SPI123_CLKSOURCE_I2S_CKIN

LL_RCC_SPI123_CLKSOURCE_CLKP

LL_RCC_SPI45_CLKSOURCE_PCLK2

LL_RCC_SPI45_CLKSOURCE_PLL2Q

LL_RCC_SPI45_CLKSOURCE_PLL3Q

LL_RCC_SPI45_CLKSOURCE_HSI

LL_RCC_SPI45_CLKSOURCE_CSI

LL_RCC_SPI45_CLKSOURCE_HSE

LL_RCC_SPI6_CLKSOURCE_PCLK4

LL_RCC_SPI6_CLKSOURCE_PLL2Q

LL_RCC_SPI6_CLKSOURCE_PLL3Q

LL_RCC_SPI6_CLKSOURCE_HSI

LL_RCC_SPI6_CLKSOURCE_CSI

LL_RCC_SPI6_CLKSOURCE_HSE

Peripheral SWP get clock source

LL_RCC_SWP_CLKSOURCE

Peripheral SWP clock source selection

LL_RCC_SWP_CLKSOURCE_PCLK1

LL_RCC_SWP_CLKSOURCE_HSI

System prescaler

LL_RCC_SYSCLK_DIV_1

LL_RCC_SYSCLK_DIV_2

LL_RCC_SYSCLK_DIV_4

LL_RCC_SYSCLK_DIV_8

LL_RCC_SYSCLK_DIV_16

LL_RCC_SYSCLK_DIV_64

LL_RCC_SYSCLK_DIV_128

LL_RCC_SYSCLK_DIV_256

LL_RCC_SYSCLK_DIV_512

System wakeup clock source

LL_RCC_SYSWAKEUP_CLKSOURCE_HSI

LL_RCC_SYSWAKEUP_CLKSOURCE_CSI

System clock switch

LL_RCC_SYS_CLKSOURCE_HSI

LL_RCC_SYS_CLKSOURCE_CSI

LL_RCC_SYS_CLKSOURCE_HSE

LL_RCC_SYS_CLKSOURCE_PLL1

System clock switch status

LL_RCC_SYS_CLKSOURCE_STATUS_HSI

HSI used as system clock

LL_RCC_SYS_CLKSOURCE_STATUS_CSI

CSI used as system clock

LL_RCC_SYS_CLKSOURCE_STATUS_HSE

HSE used as system clock

LL_RCC_SYS_CLKSOURCE_STATUS_PLL1

PLL1 used as system clock

Timers clocks prescalers selection

LL_RCC_TIM_PRESCALER_TWICE

LL_RCC_TIM_PRESCALER_FOUR_TIMES

Peripheral USART get clock source

LL_RCC_USART16_CLKSOURCE

LL_RCC_USART234578_CLKSOURCE

Peripheral USART clock source selection

LL_RCC_USART16_CLKSOURCE_PCLK2

LL_RCC_USART16_CLKSOURCE_PLL2Q

LL_RCC_USART16_CLKSOURCE_PLL3Q

LL_RCC_USART16_CLKSOURCE_HSI

LL_RCC_USART16_CLKSOURCE_CSI

LL_RCC_USART16_CLKSOURCE_LSE

LL_RCC_USART16910_CLKSOURCE_PCLK2

LL_RCC_USART16910_CLKSOURCE_PLL2Q

LL_RCC_USART16910_CLKSOURCE_PLL3Q

LL_RCC_USART16910_CLKSOURCE_HSI

LL_RCC_USART16910_CLKSOURCE_CSI

LL_RCC_USART16910_CLKSOURCE_LSE

LL_RCC_USART234578_CLKSOURCE_PCLK1

LL_RCC_USART234578_CLKSOURCE_PLL2Q

LL_RCC_USART234578_CLKSOURCE_PLL3Q

LL_RCC_USART234578_CLKSOURCE_HSI

LL_RCC_USART234578_CLKSOURCE_CSI

LL_RCC_USART234578_CLKSOURCE_LSE

Peripheral USB get clock source

LL_RCC_USB_CLKSOURCE

Peripheral USB clock source selection

LL_RCC_USB_CLKSOURCE_DISABLE

LL_RCC_USB_CLKSOURCE_PLL1Q

LL_RCC_USB_CLKSOURCE_PLL3Q

LL_RCC_USB_CLKSOURCE_HSI48

Calculate frequencies

LL_RCC_CALC_SYSCLK_FREQ

Description:

- Helper macro to calculate the SYSCLK frequency.

Parameters:

- __SYSINPUTCLKFREQ__: Frequency of the input of sys_ck (based on HSE/CSI/HSI/PLL1P)
- __SYSPRESCALER__: This parameter can be one of the following values:
  - LL_RCC_SYSCLK_DIV_1
  - LL_RCC_SYSCLK_DIV_2
  - LL_RCC_SYSCLK_DIV_4
  - LL_RCC_SYSCLK_DIV_8
  - LL_RCC_SYSCLK_DIV_16
  - LL_RCC_SYSCLK_DIV_64
  - LL_RCC_SYSCLK_DIV_128
  - LL_RCC_SYSCLK_DIV_256
  - LL_RCC_SYSCLK_DIV_512

Return value:

- SYSCLK: clock frequency (in Hz)

LL_RCC_CALC_HCLK_FREQ

Description:

- Helper macro to calculate the HCLK frequency.

Parameters:

- __SYSCLKFREQ__: SYSCLK frequency.
- __HPRESCALER__: This parameter can be one of the following values:
  - LL_RCC_AHB_DIV_1
  - LL_RCC_AHB_DIV_2
  - LL_RCC_AHB_DIV_4
  - LL_RCC_AHB_DIV_8
  - LL_RCC_AHB_DIV_16
  - LL_RCC_AHB_DIV_64
  - LL_RCC_AHB_DIV_128
  - LL_RCC_AHB_DIV_256
  - LL_RCC_AHB_DIV_512

Return value:

- HCLK: clock frequency (in Hz)

LL_RCC_CALC_PCLK1_FREQ

Description:

- Helper macro to calculate the PCLK1 frequency (ABP1)

Parameters:

- __HCLKFREQ__: HCLK frequency
- __APB1PRESCALER__: This parameter can be one of the following values:
  - LL_RCC_APB1_DIV_1
  - LL_RCC_APB1_DIV_2
  - LL_RCC_APB1_DIV_4
  - LL_RCC_APB1_DIV_8
  - LL_RCC_APB1_DIV_16

Return value:

- PCLK1: clock frequency (in Hz)

LL_RCC_CALC_PCLK2_FREQ

Description:

- Helper macro to calculate the PCLK2 frequency (ABP2)

Parameters:

- __HCLKFREQ__: HCLK frequency
- __APB2PRESCALER__: This parameter can be one of the following values:
  - LL_RCC_APB2_DIV_1
  - LL_RCC_APB2_DIV_2
  - LL_RCC_APB2_DIV_4
  - LL_RCC_APB2_DIV_8
  - LL_RCC_APB2_DIV_16

Return value:

- PCLK2: clock frequency (in Hz)

LL_RCC_CALC_PCLK3_FREQ

Description:

- Helper macro to calculate the PCLK3 frequency (APB3)

Parameters:

- __HCLKFREQ__: HCLK frequency
- __APB3PRESCALER__: This parameter can be one of the following values:
  - LL_RCC_APB3_DIV_1
  - LL_RCC_APB3_DIV_2
  - LL_RCC_APB3_DIV_4
  - LL_RCC_APB3_DIV_8
  - LL_RCC_APB3_DIV_16

Return value:

- PCLK1: clock frequency (in Hz)

LL_RCC_CALC_PCLK4_FREQ

Description:

- Helper macro to calculate the PCLK4 frequency (ABP4)

Parameters:

- __HCLKFREQ__: HCLK frequency
- __APB4PRESCALER__: This parameter can be one of the following values:
  - LL_RCC_APB4_DIV_1
  - LL_RCC_APB4_DIV_2
  - LL_RCC_APB4_DIV_4
  - LL_RCC_APB4_DIV_8
  - LL_RCC_APB4_DIV_16

Return value:

- PCLK1: clock frequency (in Hz)

Common Write and read registers Macros

LL_RCC_WriteReg

Description:

- Write a value in RCC register.

Parameters:

- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_RCC_ReadReg

Description:

- Read a value in RCC register.

Parameters:

- __REG__: Register to be read

Return value:

- Register: value

121 LL RNG Generic Driver

121.1 RNG Firmware driver registers structures

121.1.1 LL_RNG_InitTypeDef

LL_RNG_InitTypeDef is defined in the stm32h7xx_ll_rng.h

Data Fields

- ***uint32_t ClockErrorDetection***

Field Documentation

- ***uint32_t LL_RNG_InitTypeDef::ClockErrorDetection***
  Clock error detection. This parameter can be one value of ***RNG_LL_CED***. This parameter can be modified using unitary functions **LL_RNG_EnableClkErrorDetect()**.

121.2 RNG Firmware driver API description

The following section lists the various functions of the RNG library.

121.2.1 Detailed description of functions

LL_RNG_Enable

Function name

__STATIC_INLINE void LL_RNG_Enable (RNG_TypeDef * RNGx)

Function description

Enable Random Number Generation.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RNGEN LL_RNG_Enable

LL_RNG_Disable

Function name

__STATIC_INLINE void LL_RNG_Disable (RNG_TypeDef * RNGx)

Function description

Disable Random Number Generation.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RNGEN LL_RNG_Disable

LL_RNG_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RNG_IsEnabled (RNG_TypeDef * RNGx)

Function description

Check if Random Number Generator is enabled.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR RNGEN LL_RNG_IsEnabled

LL_RNG_EnableClkErrorDetect

Function name

__STATIC_INLINE void LL_RNG_EnableClkErrorDetect (RNG_TypeDef * RNGx)

Function description

Enable Clock Error Detection.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CED LL_RNG_EnableClkErrorDetect

LL_RNG_DisableClkErrorDetect

Function name

__STATIC_INLINE void LL_RNG_DisableClkErrorDetect (RNG_TypeDef * RNGx)

Function description

Disable RNG Clock Error Detection.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CED LL_RNG_DisableClkErrorDetect

LL_RNG_IsEnabledClkErrorDetect

Function name

__STATIC_INLINE uint32_t LL_RNG_IsEnabledClkErrorDetect (RNG_TypeDef * RNGx)

Function description

Check if RNG Clock Error Detection is enabled.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CED LL_RNG_IsEnabledClkErrorDetect

LL_RNG_IsActiveFlag_DRDY

Function name

__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_DRDY (RNG_TypeDef * RNGx)

Function description

Indicate if the RNG Data ready Flag is set or not.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR DRDY LL_RNG_IsActiveFlag_DRDY

LL_RNG_IsActiveFlag_CECS

Function name

__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_CECS (RNG_TypeDef * RNGx)

Function description

Indicate if the Clock Error Current Status Flag is set or not.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CECS LL_RNG_IsActiveFlag_CECS

LL_RNG_IsActiveFlag_SECS

Function name

__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_SECS (RNG_TypeDef * RNGx)

Function description

Indicate if the Seed Error Current Status Flag is set or not.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR SECS LL_RNG_IsActiveFlag_SECS

LL_RNG_IsActiveFlag_CEIS

Function name

__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_CEIS (RNG_TypeDef * RNGx)

Function description

Indicate if the Clock Error Interrupt Status Flag is set or not.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CEIS LL_RNG_IsActiveFlag_CEIS

LL_RNG_IsActiveFlag_SEIS

Function name

__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_SEIS (RNG_TypeDef * RNGx)

Function description

Indicate if the Seed Error Interrupt Status Flag is set or not.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR SEIS LL_RNG_IsActiveFlag_SEIS

LL_RNG_ClearFlag_CEIS

Function name

__STATIC_INLINE void LL_RNG_ClearFlag_CEIS (RNG_TypeDef * RNGx)

Function description

Clear Clock Error interrupt Status (CEIS) Flag.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CEIS LL_RNG_ClearFlag_CEIS

LL_RNG_ClearFlag_SEIS

Function name

__STATIC_INLINE void LL_RNG_ClearFlag_SEIS (RNG_TypeDef * RNGx)

Function description

Clear Seed Error interrupt Status (SEIS) Flag.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR SEIS LL_RNG_ClearFlag_SEIS

LL_RNG_EnableIT

Function name

__STATIC_INLINE void LL_RNG_EnableIT (RNG_TypeDef * RNGx)

Function description

Enable Random Number Generator Interrupt (applies for either Seed error, Clock Error or Data ready interrupts)

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR IE LL_RNG_EnableIT

LL_RNG_DisableIT

Function name

__STATIC_INLINE void LL_RNG_DisableIT (RNG_TypeDef * RNGx)

Function description

Disable Random Number Generator Interrupt (applies for either Seed error, Clock Error or Data ready interrupts)

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR IE LL_RNG_DisableIT

LL_RNG_IsEnabledIT

Function name

__STATIC_INLINE uint32_t LL_RNG_IsEnabledIT (RNG_TypeDef * RNGx)

Function description

Check if Random Number Generator Interrupt is enabled (applies for either Seed error, Clock Error or Data ready interrupts)

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR IE LL_RNG_IsEnabledIT

LL_RNG_ReadRandData32

Function name

__STATIC_INLINE uint32_t LL_RNG_ReadRandData32 (RNG_TypeDef * RNGx)

Function description

Return32-bit Random Number value.

Parameters

- **RNGx:** RNG Instance

Return values

- **Generated:** 32-bit random value

Reference Manual to LL API cross reference:

- DR RNDATA LL_RNG_ReadRandData32

LL_RNG_Init

Function name

ErrorStatus LL_RNG_Init (RNG_TypeDef * RNGx, LL_RNG_InitTypeDef * RNG_InitStruct)

Function description

Initialize RNG registers according to the specified parameters in RNG_InitStruct.

Parameters

- **RNGx:** RNG Instance
- **RNG_InitStruct:** pointer to a LL_RNG_InitTypeDef structure that contains the configuration information for the specified RNG peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RNG registers are initialized according to RNG_InitStruct content
  - ERROR: not applicable

LL_RNG_StructInit

Function name

void LL_RNG_StructInit (LL_RNG_InitTypeDef * RNG_InitStruct)

Function description

Set each LL_RNG_InitTypeDef field to default value.

Parameters

- **RNG_InitStruct:** pointer to a LL_RNG_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_RNG_DeInit

Function name

ErrorStatus LL_RNG_DeInit (RNG_TypeDef * RNGx)

Function description

De-initialize RNG registers (Registers restored to their default values).

Parameters

- **RNGx:** RNG Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RNG registers are de-initialized
  - ERROR: not applicable

121.3 RNG Firmware driver defines

The following section lists the various define and macros of the module.

121.3.1 RNG

RNG

Clock Error Detection

LL_RNG_CED_ENABLE

Clock error detection enabled

LL_RNG_CED_DISABLE

Clock error detection disabled

Get Flags Defines

LL_RNG_SR_DRDY

Register contains valid random data

LL_RNG_SR_CECS

Clock error current status

LL_RNG_SR_SECS

Seed error current status

LL_RNG_SR_CEIS

Clock error interrupt status

LL_RNG_SR_SEIS

Seed error interrupt status

IT Defines

LL_RNG_CR_IE

RNG Interrupt enable

Common Write and read registers Macros

LL_RNG_WriteReg

Description:

- Write a value in RNG register.

Parameters:

- __INSTANCE__: RNG Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_RNG_ReadReg

Description:

- Read a value in RNG register.

Parameters:

- __INSTANCE__: RNG Instance
- __REG__: Register to be read

Return value:

- Register: value

122 LL RTC Generic Driver

122.1 RTC Firmware driver registers structures

122.1.1 LL_RTC_InitTypeDef

LL_RTC_InitTypeDef is defined in the stm32h7xx_ll_rtc.h

Data Fields

- ***uint32_t HourFormat***
- ***uint32_t AsynchPrescaler***
- ***uint32_t SynchPrescaler***

Field Documentation

- ***uint32_t LL_RTC_InitTypeDef::HourFormat***

  Specifies the RTC Hours Format. This parameter can be a value of ***RTC_LL_EC_HOURFORMAT***This feature can be modified afterwards using unitary function **LL_RTC_SetHourFormat()**.
- ***uint32_t LL_RTC_InitTypeDef::AsynchPrescaler***

  Specifies the RTC Asynchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7FThis feature can be modified afterwards using unitary function **LL_RTC_SetAsynchPrescaler()**.
- ***uint32_t LL_RTC_InitTypeDef::SynchPrescaler***

  Specifies the RTC Synchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7FFFThis feature can be modified afterwards using unitary function **LL_RTC_SetSynchPrescaler()**.

122.1.2 LL_RTC_TimeTypeDef

LL_RTC_TimeTypeDef is defined in the stm32h7xx_ll_rtc.h

Data Fields

- ***uint32_t TimeFormat***
- ***uint8_t Hours***
- ***uint8_t Minutes***
- ***uint8_t Seconds***

Field Documentation

- ***uint32_t LL_RTC_TimeTypeDef::TimeFormat***

  Specifies the RTC AM/PM Time. This parameter can be a value of ***RTC_LL_EC_TIME_FORMAT***This feature can be modified afterwards using unitary function **LL_RTC_TIME_SetFormat()**.
- ***uint8_t LL_RTC_TimeTypeDef::Hours***

  Specifies the RTC Time Hours. This parameter must be a number between Min_Data = 0 and Max_Data = 12 if the **LL_RTC_TIME_FORMAT_PM** is selected. This parameter must be a number between Min_Data = 0 and Max_Data = 23 if the **LL_RTC_TIME_FORMAT_AM_OR_24** is selected.This feature can be modified afterwards using unitary function **LL_RTC_TIME_SetHour()**.
- ***uint8_t LL_RTC_TimeTypeDef::Minutes***

  Specifies the RTC Time Minutes. This parameter must be a number between Min_Data = 0 and Max_Data = 59This feature can be modified afterwards using unitary function **LL_RTC_TIME_SetMinute()**.
- ***uint8_t LL_RTC_TimeTypeDef::Seconds***

  Specifies the RTC Time Seconds. This parameter must be a number between Min_Data = 0 and Max_Data = 59This feature can be modified afterwards using unitary function **LL_RTC_TIME_SetSecond()**.

122.1.3 LL_RTC_DateTypeDef

LL_RTC_DateTypeDef is defined in the stm32h7xx_ll_rtc.h

Data Fields

- ***uint8_t WeekDay***
- ***uint8_t Month***
- ***uint8_t Day***
- ***uint8_t Year***

Field Documentation

- ***uint8_t LL_RTC_DateTypeDef::WeekDay***

  Specifies the RTC Date WeekDay. This parameter can be a value of ***RTC_LL_EC_WEEKDAY***This feature can be modified afterwards using unitary function **LL_RTC_DATE_SetWeekDay()**.
- ***uint8_t LL_RTC_DateTypeDef::Month***

  Specifies the RTC Date Month. This parameter can be a value of ***RTC_LL_EC_MONTH***This feature can be modified afterwards using unitary function **LL_RTC_DATE_SetMonth()**.
- ***uint8_t LL_RTC_DateTypeDef::Day***

  Specifies the RTC Date Day. This parameter must be a number between Min_Data = 1 and Max_Data = 31This feature can be modified afterwards using unitary function **LL_RTC_DATE_SetDay()**.
- ***uint8_t LL_RTC_DateTypeDef::Year***

  Specifies the RTC Date Year. This parameter must be a number between Min_Data = 0 and Max_Data = 99This feature can be modified afterwards using unitary function **LL_RTC_DATE_SetYear()**.

122.1.4 LL_RTC_AlarmTypeDef

LL_RTC_AlarmTypeDef is defined in the stm32h7xx_ll_rtc.h

Data Fields

- ***LL_RTC_TimeTypeDef AlarmTime***
- ***uint32_t AlarmMask***
- ***uint32_t AlarmDateWeekDaySel***
- ***uint8_t AlarmDateWeekDay***

Field Documentation

- ***LL_RTC_TimeTypeDef LL_RTC_AlarmTypeDef::AlarmTime***

  Specifies the RTC Alarm Time members.
- ***uint32_t LL_RTC_AlarmTypeDef::AlarmMask***

  Specifies the RTC Alarm Masks. This parameter can be a value of ***RTC_LL_EC_ALMA_MASK*** for ALARM A or ***RTC_LL_EC_ALMB_MASK*** for ALARM B.This feature can be modified afterwards using unitary function **LL_RTC_ALMA_SetMask()** for ALARM A or **LL_RTC_ALMB_SetMask()** for ALARM B
- ***uint32_t LL_RTC_AlarmTypeDef::AlarmDateWeekDaySel***

  Specifies the RTC Alarm is on day or WeekDay. This parameter can be a value of ***RTC_LL_EC_ALMA_WEEKDAY_SELECTION*** for ALARM A or ***RTC_LL_EC_ALMB_WEEKDAY_SELECTION*** for ALARM BThis feature can be modified afterwards using unitary function **LL_RTC_ALMA_EnableWeekday()** or **LL_RTC_ALMA_DisableWeekday()** for ALARM A or **LL_RTC_ALMB_EnableWeekday()** or **LL_RTC_ALMB_DisableWeekday()** for ALARM B
- ***uint8_t LL_RTC_AlarmTypeDef::AlarmDateWeekDay***

  Specifies the RTC Alarm Day/WeekDay. If AlarmDateWeekDaySel set to day, this parameter must be a number between Min_Data = 1 and Max_Data = 31.This feature can be modified afterwards using unitary function **LL_RTC_ALMA_SetDay()** for ALARM A or **LL_RTC_ALMB_SetDay()** for ALARM B.If AlarmDateWeekDaySel set to Weekday, this parameter can be a value of ***RTC_LL_EC_WEEKDAY***.This feature can be modified afterwards using unitary function **LL_RTC_ALMA_SetWeekDay()** for ALARM A or **LL_RTC_ALMB_SetWeekDay()** for ALARM B.

122.2 RTC Firmware driver API description

The following section lists the various functions of the RTC library.

122.2.1 Detailed description of functions

LL_RTC_SetHourFormat

Function name

__STATIC_INLINE void LL_RTC_SetHourFormat (RTC_TypeDef * RTCx, uint32_t HourFormat)

Function description

Set Hours format (24 hour/day or AM/PM hour format)

Parameters

- **RTCx:** RTC Instance
- **HourFormat:** This parameter can be one of the following values:
  - LL_RTC_HOURFORMAT_24HOUR
  - LL_RTC_HOURFORMAT_AMPM

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)

Reference Manual to LL API cross reference:

- RTC_CR FMT LL_RTC_SetHourFormat

LL_RTC_GetHourFormat

Function name

__STATIC_INLINE uint32_t LL_RTC_GetHourFormat (RTC_TypeDef * RTCx)

Function description

Get Hours format (24 hour/day or AM/PM hour format)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_HOURFORMAT_24HOUR
  - LL_RTC_HOURFORMAT_AMPM

Reference Manual to LL API cross reference:

- RTC_CR FMT LL_RTC_GetHourFormat

LL_RTC_SetAlarmOutEvent

Function name

__STATIC_INLINE void LL_RTC_SetAlarmOutEvent (RTC_TypeDef * RTCx, uint32_t AlarmOutput)

Function description

Select the flag to be routed to RTC_ALARM output.

Parameters

- **RTCx:** RTC Instance
- **AlarmOutput:** This parameter can be one of the following values:
  - LL_RTC_ALARMOUT_DISABLE
  - LL_RTC_ALARMOUT_ALMA
  - LL_RTC_ALARMOUT_ALMB
  - LL_RTC_ALARMOUT_WAKEUP

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR OSEL LL_RTC_SetAlarmOutEvent

LL_RTC_GetAlarmOutEvent

Function name

__STATIC_INLINE uint32_t LL_RTC_GetAlarmOutEvent (RTC_TypeDef * RTCx)

Function description

Get the flag to be routed to RTC_ALARM output.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_ALARMOUT_DISABLE
  - LL_RTC_ALARMOUT_ALMA
  - LL_RTC_ALARMOUT_ALMB
  - LL_RTC_ALARMOUT_WAKEUP

Reference Manual to LL API cross reference:

- RTC_CR OSEL LL_RTC_GetAlarmOutEvent

LL_RTC_SetAlarmOutputType

Function name

__STATIC_INLINE void LL_RTC_SetAlarmOutputType (RTC_TypeDef * RTCx, uint32_t Output)

Function description

Set RTC_ALARM output type (ALARM in push-pull or open-drain output)

Parameters

- **RTCx:** RTC Instance
- **Output:** This parameter can be one of the following values:
  - LL_RTC_ALARM_OUTPUTTYPE_OPENDRAIN
  - LL_RTC_ALARM_OUTPUTTYPE_PUSHPULL

Return values

- **None:**

Notes

- Used only when RTC_ALARM is mapped on PC13

Reference Manual to LL API cross reference:

- OR ALARMOUTTYPE LL_RTC_SetAlarmOutputType

LL_RTC_GetAlarmOutputType

Function name

__STATIC_INLINE uint32_t LL_RTC_GetAlarmOutputType (RTC_TypeDef * RTCx)

Function description

Get RTC_ALARM output type (ALARM in push-pull or open-drain output)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_ALARM_OUTPUTTYPE_OPENDRAIN
  - LL_RTC_ALARM_OUTPUTTYPE_PUSHPULL

Notes

- used only when RTC_ALARM is mapped on PC13

Reference Manual to LL API cross reference:

- OR ALARMOUTTYPE LL_RTC_GetAlarmOutputType

LL_RTC_EnableInitMode

Function name

__STATIC_INLINE void LL_RTC_EnableInitMode (RTC_TypeDef * RTCx)

Function description

Enable initialization mode.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Initialization mode is used to program time and date register (RTC_TR and RTC_DR) and prescaler register (RTC_PRER). Counters are stopped and start counting from the new value when INIT is reset.

Reference Manual to LL API cross reference:

- ISR INIT LL_RTC_EnableInitMode

LL_RTC_DisableInitMode

Function name

__STATIC_INLINE void LL_RTC_DisableInitMode (RTC_TypeDef * RTCx)

Function description

Disable initialization mode (Free running mode)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR INIT LL_RTC_DisableInitMode

LL_RTC_SetOutputPolarity

Function name

__STATIC_INLINE void LL_RTC_SetOutputPolarity (RTC_TypeDef * RTCx, uint32_t Polarity)

Function description

Set Output polarity (pin is low when ALRAF/ALRBF/WUTF is asserted)

Parameters

- **RTCx:** RTC Instance
- **Polarity:** This parameter can be one of the following values:
  - LL_RTC_OUTPUTPOLARITY_PIN_HIGH
  - LL_RTC_OUTPUTPOLARITY_PIN_LOW

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR POL LL_RTC_SetOutputPolarity

LL_RTC_GetOutputPolarity

Function name

__STATIC_INLINE uint32_t LL_RTC_GetOutputPolarity (RTC_TypeDef * RTCx)

Function description

Get Output polarity.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_OUTPUTPOLARITY_PIN_HIGH
  - LL_RTC_OUTPUTPOLARITY_PIN_LOW

Reference Manual to LL API cross reference:

- RTC_CR POL LL_RTC_GetOutputPolarity

LL_RTC_EnableShadowRegBypass

Function name

__STATIC_INLINE void LL_RTC_EnableShadowRegBypass (RTC_TypeDef * RTCx)

Function description

Enable Bypass the shadow registers.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR BYPSHAD LL_RTC_EnableShadowRegBypass

LL_RTC_DisableShadowRegBypass

Function name

__STATIC_INLINE void LL_RTC_DisableShadowRegBypass (RTC_TypeDef * RTCx)

Function description

Disable Bypass the shadow registers.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR BYPSHAD LL_RTC_DisableShadowRegBypass

LL_RTC_IsShadowRegBypassEnabled

Function name

__STATIC_INLINE uint32_t LL_RTC_IsShadowRegBypassEnabled (RTC_TypeDef * RTCx)

Function description

Check if Shadow registers bypass is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR BYPSHAD LL_RTC_IsShadowRegBypassEnabled

LL_RTC_EnableRefClock

Function name

__STATIC_INLINE void LL_RTC_EnableRefClock (RTC_TypeDef * RTCx)

Function description

Enable RTC_REFIN reference clock detection (50 or 60 Hz)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)

Reference Manual to LL API cross reference:

- RTC_CR REFCKON LL_RTC_EnableRefClock

LL_RTC_DisableRefClock

Function name

__STATIC_INLINE void LL_RTC_DisableRefClock (RTC_TypeDef * RTCx)

Function description

Disable RTC_REFIN reference clock detection (50 or 60 Hz)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)

Reference Manual to LL API cross reference:

- RTC_CR REFCKON LL_RTC_DisableRefClock

LL_RTC_SetAsynchPrescaler

Function name

__STATIC_INLINE void LL_RTC_SetAsynchPrescaler (RTC_TypeDef * RTCx, uint32_t AsynchPrescaler)

Function description

Set Asynchronous prescaler factor.

Parameters

- **RTCx:** RTC Instance
- **AsynchPrescaler:** Value between Min_Data = 0 and Max_Data = 0x7F

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_PRER PREDIV_A LL_RTC_SetAsynchPrescaler

LL_RTC_SetSynchPrescaler

Function name

__STATIC_INLINE void LL_RTC_SetSynchPrescaler (RTC_TypeDef * RTCx, uint32_t SynchPrescaler)

Function description

Set Synchronous prescaler factor.

Parameters

- **RTCx:** RTC Instance
- **SynchPrescaler:** Value between Min_Data = 0 and Max_Data = 0x7FFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_PRER PREDIV_S LL_RTC_SetSynchPrescaler

LL_RTC_GetAsynchPrescaler

Function name

__STATIC_INLINE uint32_t LL_RTC_GetAsynchPrescaler (RTC_TypeDef * RTCx)

Function description

Get Asynchronous prescaler factor.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data = 0 and Max_Data = 0x7F

Reference Manual to LL API cross reference:

- RTC_PRER PREDIV_A LL_RTC_GetAsynchPrescaler

LL_RTC_GetSynchPrescaler

Function name

__STATIC_INLINE uint32_t LL_RTC_GetSynchPrescaler (RTC_TypeDef * RTCx)

Function description

Get Synchronous prescaler factor.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data = 0 and Max_Data = 0x7FFF

Reference Manual to LL API cross reference:

- RTC_PRER PREDIV_S LL_RTC_GetSynchPrescaler

LL_RTC_EnableWriteProtection

Function name

__STATIC_INLINE void LL_RTC_EnableWriteProtection (RTC_TypeDef * RTCx)

Function description

Enable the write protection for RTC registers.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_WPR KEY LL_RTC_EnableWriteProtection

LL_RTC_DisableWriteProtection

Function name

__STATIC_INLINE void LL_RTC_DisableWriteProtection (RTC_TypeDef * RTCx)

Function description

Disable the write protection for RTC registers.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_WPR KEY LL_RTC_DisableWriteProtection

LL_RTC_EnableOutRemap

Function name

__STATIC_INLINE void LL_RTC_EnableOutRemap (RTC_TypeDef * RTCx)

Function description

Enable RTC_OUT remap.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- OR OUT_RMP LL_RTC_EnableOutRemap

LL_RTC_DisableOutRemap

Function name

__STATIC_INLINE void LL_RTC_DisableOutRemap (RTC_TypeDef * RTCx)

Function description

Disable RTC_OUT remap.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- OR OUT_RMP LL_RTC_DisableOutRemap

LL_RTC_TIME_SetFormat

Function name

__STATIC_INLINE void LL_RTC_TIME_SetFormat (RTC_TypeDef * RTCx, uint32_t TimeFormat)

Function description

Set time format (AM/24-hour or PM notation)

Parameters

- **RTCx:** RTC Instance
- **TimeFormat:** This parameter can be one of the following values:
  - LL_RTC_TIME_FORMAT_AM_OR_24
  - LL_RTC_TIME_FORMAT_PM

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)

Reference Manual to LL API cross reference:

- RTC_TR PM LL_RTC_TIME_SetFormat

LL_RTC_TIME_GetFormat

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_GetFormat (RTC_TypeDef * RTCx)

Function description

Get time format (AM or PM notation)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_TIME_FORMAT_AM_OR_24
  - LL_RTC_TIME_FORMAT_PM

Notes

- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).

Reference Manual to LL API cross reference:

- RTC_TR PM LL_RTC_TIME_GetFormat

LL_RTC_TIME_SetHour

Function name

__STATIC_INLINE void LL_RTC_TIME_SetHour (RTC_TypeDef * RTCx, uint32_t Hours)

Function description

Set Hours in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)
- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert hour from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_TR HT LL_RTC_TIME_SetHour RTC_TR HU LL_RTC_TIME_SetHour

LL_RTC_TIME_GetHour

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_GetHour (RTC_TypeDef * RTCx)

Function description

Get Hours in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Notes

- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert hour from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TR HT LL_RTC_TIME_GetHour RTC_TR HU LL_RTC_TIME_GetHour

LL_RTC_TIME_SetMinute

Function name

__STATIC_INLINE void LL_RTC_TIME_SetMinute (RTC_TypeDef * RTCx, uint32_t Minutes)

Function description

Set Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)
- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Minutes from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_TR MNT LL_RTC_TIME_SetMinute RTC_TR MNU LL_RTC_TIME_SetMinute

LL_RTC_TIME_GetMinute

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_GetMinute (RTC_TypeDef * RTCx)

Function description

Get Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert minute from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TR MNT LL_RTC_TIME_GetMinute RTC_TR MNU LL_RTC_TIME_GetMinute

LL_RTC_TIME_SetSecond

Function name

__STATIC_INLINE void LL_RTC_TIME_SetSecond (RTC_TypeDef * RTCx, uint32_t Seconds)

Function description

Set Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)
- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Seconds from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_TR ST LL_RTC_TIME_SetSecond RTC_TR SU LL_RTC_TIME_SetSecond

LL_RTC_TIME_GetSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_GetSecond (RTC_TypeDef * RTCx)

Function description

Get Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Seconds from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TR ST LL_RTC_TIME_GetSecond RTC_TR SU LL_RTC_TIME_GetSecond

LL_RTC_TIME_Config

Function name

__STATIC_INLINE void LL_RTC_TIME_Config (RTC_TypeDef * RTCx, uint32_t Format12_24, uint32_t Hours, uint32_t Minutes, uint32_t Seconds)

Function description

Set time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Format12_24:** This parameter can be one of the following values:
  - LL_RTC_TIME_FORMAT_AM_OR_24
  - LL_RTC_TIME_FORMAT_PM
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
- **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59
- **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)
- TimeFormat and Hours should follow the same format

Reference Manual to LL API cross reference:

- RTC_TR PM LL_RTC_TIME_Config RTC_TR HT LL_RTC_TIME_Config RTC_TR HU LL_RTC_TIME_Config RTC_TR MNT LL_RTC_TIME_Config RTC_TR MNU LL_RTC_TIME_Config RTC_TR ST LL_RTC_TIME_Config RTC_TR SU LL_RTC_TIME_Config

LL_RTC_TIME_Get

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_Get (RTC_TypeDef * RTCx)

Function description

Get time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of hours, minutes and seconds (Format: 0x00HHMMSS).

Notes

- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).
- helper macros __LL_RTC_GET_HOUR, __LL_RTC_GET_MINUTE and __LL_RTC_GET_SECOND are available to get independently each parameter.

Reference Manual to LL API cross reference:

- RTC_TR HT LL_RTC_TIME_Get RTC_TR HU LL_RTC_TIME_Get RTC_TR MNT LL_RTC_TIME_Get RTC_TR MNU LL_RTC_TIME_Get RTC_TR ST LL_RTC_TIME_Get RTC_TR SU LL_RTC_TIME_Get

LL_RTC_TIME_EnableDayLightStore

Function name

__STATIC_INLINE void LL_RTC_TIME_EnableDayLightStore (RTC_TypeDef * RTCx)

Function description

Memorize whether the daylight saving time change has been performed.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR BKP LL_RTC_TIME_EnableDayLightStore

LL_RTC_TIME_DisableDayLightStore

Function name

__STATIC_INLINE void LL_RTC_TIME_DisableDayLightStore (RTC_TypeDef * RTCx)

Function description

Disable memorization whether the daylight saving time change has been performed.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR BKP LL_RTC_TIME_DisableDayLightStore

LL_RTC_TIME_IsDayLightStoreEnabled

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_IsDayLightStoreEnabled (RTC_TypeDef * RTCx)

Function description

Check if RTC Day Light Saving stored operation has been enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR BKP LL_RTC_TIME_IsDayLightStoreEnabled

LL_RTC_TIME_DecHour

Function name

__STATIC_INLINE void LL_RTC_TIME_DecHour (RTC_TypeDef * RTCx)

Function description

Subtract 1 hour (winter time change)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR SUB1H LL_RTC_TIME_DecHour

LL_RTC_TIME_IncHour

Function name

__STATIC_INLINE void LL_RTC_TIME_IncHour (RTC_TypeDef * RTCx)

Function description

Add 1 hour (summer time change)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ADD1H LL_RTC_TIME_IncHour

LL_RTC_TIME_GetSubSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_GetSubSecond (RTC_TypeDef * RTCx)

Function description

Get Sub second value in the synchronous prescaler counter.

Parameters

- **RTCx:** RTC Instance

Return values

- **Sub:** second value (number between 0 and 65535)

Notes

- You can use both SubSeconds value and SecondFraction (PREDIV_S through LL_RTC_GetSynchPrescaler function) terms returned to convert Calendar SubSeconds value in second fraction ratio with time unit following generic formula: ==> Seconds fraction ratio * time_unit= [(SecondFraction-SubSeconds)/(SecondFraction+1)] * time_unit This conversion can be performed only if no shift operation is pending (ie. SHFP=0) when PREDIV_S >= SS.

Reference Manual to LL API cross reference:

- RTC_SSR SS LL_RTC_TIME_GetSubSecond

LL_RTC_TIME_Synchronize

Function name

__STATIC_INLINE void LL_RTC_TIME_Synchronize (RTC_TypeDef * RTCx, uint32_t ShiftSecond, uint32_t Fraction)

Function description

Synchronize to a remote clock with a high degree of precision.

Parameters

- **RTCx:** RTC Instance
- **ShiftSecond:** This parameter can be one of the following values:
  - LL_RTC_SHIFT_SECOND_DELAY
  - LL_RTC_SHIFT_SECOND_ADVANCE
- **Fraction:** Number of Seconds Fractions (any value from 0 to 0x7FFF)

Return values

- **None:**

Notes

- This operation effectively subtracts from (delays) or advance the clock of a fraction of a second.
- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- When REFCKON is set, firmware must not write to Shift control register.

Reference Manual to LL API cross reference:

- RTC_SHIFTR ADD1S LL_RTC_TIME_Synchronize RTC_SHIFTR SUBFS LL_RTC_TIME_Synchronize

LL_RTC_DATE_SetYear

Function name

__STATIC_INLINE void LL_RTC_DATE_SetYear (RTC_TypeDef * RTCx, uint32_t Year)

Function description

Set Year in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Year:** Value between Min_Data=0x00 and Max_Data=0x99

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Year from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_DR YT LL_RTC_DATE_SetYear RTC_DR YU LL_RTC_DATE_SetYear

LL_RTC_DATE_GetYear

Function name

__STATIC_INLINE uint32_t LL_RTC_DATE_GetYear (RTC_TypeDef * RTCx)

Function description

Get Year in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x99

Notes

- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Year from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_DR YT LL_RTC_DATE_GetYear RTC_DR YU LL_RTC_DATE_GetYear

LL_RTC_DATE_SetWeekDay

Function name

__STATIC_INLINE void LL_RTC_DATE_SetWeekDay (RTC_TypeDef * RTCx, uint32_t WeekDay)

Function description

Set Week day.

Parameters

- **RTCx:** RTC Instance
- **WeekDay:** This parameter can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_DR WDU LL_RTC_DATE_SetWeekDay

LL_RTC_DATE_GetWeekDay

Function name

__STATIC_INLINE uint32_t LL_RTC_DATE_GetWeekDay (RTC_TypeDef * RTCx)

Function description

Get Week day.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

Notes

- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit

Reference Manual to LL API cross reference:

- RTC_DR WDU LL_RTC_DATE_GetWeekDay

LL_RTC_DATE_SetMonth

Function name

__STATIC_INLINE void LL_RTC_DATE_SetMonth (RTC_TypeDef * RTCx, uint32_t Month)

Function description

Set Month in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Month:** This parameter can be one of the following values:
  - LL_RTC_MONTH_JANUARY
  - LL_RTC_MONTH_FEBRUARY
  - LL_RTC_MONTH_MARCH
  - LL_RTC_MONTH_APRIL
  - LL_RTC_MONTH_MAY
  - LL_RTC_MONTH_JUNE
  - LL_RTC_MONTH_JULY
  - LL_RTC_MONTH_AUGUST
  - LL_RTC_MONTH_SEPTEMBER
  - LL_RTC_MONTH_OCTOBER
  - LL_RTC_MONTH_NOVEMBER
  - LL_RTC_MONTH_DECEMBER

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Month from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_DR MT LL_RTC_DATE_SetMonth RTC_DR MU LL_RTC_DATE_SetMonth

LL_RTC_DATE_GetMonth

Function name

__STATIC_INLINE uint32_t LL_RTC_DATE_GetMonth (RTC_TypeDef * RTCx)

Function description

Get Month in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_MONTH_JANUARY
  - LL_RTC_MONTH_FEBRUARY
  - LL_RTC_MONTH_MARCH
  - LL_RTC_MONTH_APRIL
  - LL_RTC_MONTH_MAY
  - LL_RTC_MONTH_JUNE
  - LL_RTC_MONTH_JULY
  - LL_RTC_MONTH_AUGUST
  - LL_RTC_MONTH_SEPTEMBER
  - LL_RTC_MONTH_OCTOBER
  - LL_RTC_MONTH_NOVEMBER
  - LL_RTC_MONTH_DECEMBER

Notes

- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Month from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_DR MT LL_RTC_DATE_GetMonth RTC_DR MU LL_RTC_DATE_GetMonth

LL_RTC_DATE_SetDay

Function name

__STATIC_INLINE void LL_RTC_DATE_SetDay (RTC_TypeDef * RTCx, uint32_t Day)

Function description

Set Day in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Day:** Value between Min_Data=0x01 and Max_Data=0x31

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Day from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_DR DT LL_RTC_DATE_SetDay RTC_DR DU LL_RTC_DATE_SetDay

LL_RTC_DATE_GetDay

Function name

__STATIC_INLINE uint32_t LL_RTC_DATE_GetDay (RTC_TypeDef * RTCx)

Function description

Get Day in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x31

Notes

- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_DR DT LL_RTC_DATE_GetDay RTC_DR DU LL_RTC_DATE_GetDay

LL_RTC_DATE_Config

Function name

__STATIC_INLINE void LL_RTC_DATE_Config (RTC_TypeDef * RTCx, uint32_t WeekDay, uint32_t Day, uint32_t Month, uint32_t Year)

Function description

Set date (WeekDay, Day, Month and Year) in BCD format.

Parameters

- **RTCx:** RTC Instance
- **WeekDay:** This parameter can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY
- **Day:** Value between Min_Data=0x01 and Max_Data=0x31
- **Month:** This parameter can be one of the following values:
  - LL_RTC_MONTH_JANUARY
  - LL_RTC_MONTH_FEBRUARY
  - LL_RTC_MONTH_MARCH
  - LL_RTC_MONTH_APRIL
  - LL_RTC_MONTH_MAY
  - LL_RTC_MONTH_JUNE
  - LL_RTC_MONTH_JULY
  - LL_RTC_MONTH_AUGUST
  - LL_RTC_MONTH_SEPTEMBER
  - LL_RTC_MONTH_OCTOBER
  - LL_RTC_MONTH_NOVEMBER
  - LL_RTC_MONTH_DECEMBER
- **Year:** Value between Min_Data=0x00 and Max_Data=0x99

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_DR WDU LL_RTC_DATE_Config RTC_DR MT LL_RTC_DATE_Config RTC_DR MU LL_RTC_DATE_Config RTC_DR DT LL_RTC_DATE_Config RTC_DR DU LL_RTC_DATE_Config RTC_DR YT LL_RTC_DATE_Config RTC_DR YU LL_RTC_DATE_Config

LL_RTC_DATE_Get

Function name

__STATIC_INLINE uint32_t LL_RTC_DATE_Get (RTC_TypeDef * RTCx)

Function description

Get date (WeekDay, Day, Month and Year) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of WeekDay, Day, Month and Year (Format: 0xWWDDMMYY).

Notes

- if shadow mode is disabled (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- helper macros __LL_RTC_GET_WEEKDAY, __LL_RTC_GET_YEAR, __LL_RTC_GET_MONTH, and __LL_RTC_GET_DAY are available to get independently each parameter.

Reference Manual to LL API cross reference:

- RTC_DR WDU LL_RTC_DATE_Get RTC_DR MT LL_RTC_DATE_Get RTC_DR MU LL_RTC_DATE_Get RTC_DR DT LL_RTC_DATE_Get RTC_DR DU LL_RTC_DATE_Get RTC_DR YT LL_RTC_DATE_Get RTC_DR YU LL_RTC_DATE_Get

LL_RTC_ALMA_Enable

Function name

__STATIC_INLINE void LL_RTC_ALMA_Enable (RTC_TypeDef * RTCx)

Function description

Enable Alarm A.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRAE LL_RTC_ALMA_Enable

LL_RTC_ALMA_Disable

Function name

__STATIC_INLINE void LL_RTC_ALMA_Disable (RTC_TypeDef * RTCx)

Function description

Disable Alarm A.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRAE LL_RTC_ALMA_Disable

LL_RTC_ALMA_SetMask

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetMask (RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Specify the Alarm A masks.

Parameters

- **RTCx:** RTC Instance
- **Mask:** This parameter can be a combination of the following values:
  - LL_RTC_ALMA_MASK_NONE
  - LL_RTC_ALMA_MASK_DATEWEEKDAY
  - LL_RTC_ALMA_MASK_HOURS
  - LL_RTC_ALMA_MASK_MINUTES
  - LL_RTC_ALMA_MASK_SECONDS
  - LL_RTC_ALMA_MASK_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMAR MSK4 LL_RTC_ALMA_SetMask RTC_ALRMAR MSK3 LL_RTC_ALMA_SetMask RTC_ALRMAR MSK2 LL_RTC_ALMA_SetMask RTC_ALRMAR MSK1 LL_RTC_ALMA_SetMask

LL_RTC_ALMA_GetMask

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetMask (RTC_TypeDef * RTCx)

Function description

Get the Alarm A masks.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be can be a combination of the following values:
  - LL_RTC_ALMA_MASK_NONE
  - LL_RTC_ALMA_MASK_DATEWEEKDAY
  - LL_RTC_ALMA_MASK_HOURS
  - LL_RTC_ALMA_MASK_MINUTES
  - LL_RTC_ALMA_MASK_SECONDS
  - LL_RTC_ALMA_MASK_ALL

Reference Manual to LL API cross reference:

- RTC_ALRMAR MSK4 LL_RTC_ALMA_GetMask RTC_ALRMAR MSK3 LL_RTC_ALMA_GetMask RTC_ALRMAR MSK2 LL_RTC_ALMA_GetMask RTC_ALRMAR MSK1 LL_RTC_ALMA_GetMask

LL_RTC_ALMA_EnableWeekday

Function name

__STATIC_INLINE void LL_RTC_ALMA_EnableWeekday (RTC_TypeDef * RTCx)

Function description

Enable AlarmA Week day selection (DU[3:0] represents the week day.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMAR WDSEL LL_RTC_ALMA_EnableWeekday

LL_RTC_ALMA_DisableWeekday

Function name

__STATIC_INLINE void LL_RTC_ALMA_DisableWeekday (RTC_TypeDef * RTCx)

Function description

Disable AlarmA Week day selection (DU[3:0] represents the date)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMAR WDSEL LL_RTC_ALMA_DisableWeekday

LL_RTC_ALMA_SetDay

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetDay (RTC_TypeDef * RTCx, uint32_t Day)

Function description

Set ALARM A Day in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Day:** Value between Min_Data=0x01 and Max_Data=0x31

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Day from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMAR DT LL_RTC_ALMA_SetDay RTC_ALRMAR DU LL_RTC_ALMA_SetDay

LL_RTC_ALMA_GetDay

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetDay (RTC_TypeDef * RTCx)

Function description

Get ALARM A Day in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x31

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMAR DT LL_RTC_ALMA_GetDay RTC_ALRMAR DU LL_RTC_ALMA_GetDay

LL_RTC_ALMA_SetWeekDay

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetWeekDay (RTC_TypeDef * RTCx, uint32_t WeekDay)

Function description

Set ALARM A Weekday.

Parameters

- **RTCx:** RTC Instance
- **WeekDay:** This parameter can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMAR DU LL_RTC_ALMA_SetWeekDay

LL_RTC_ALMA_GetWeekDay

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetWeekDay (RTC_TypeDef * RTCx)

Function description

Get ALARM A Weekday.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

Reference Manual to LL API cross reference:

- RTC_ALRMAR DU LL_RTC_ALMA_GetWeekDay

LL_RTC_ALMA_SetTimeFormat

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetTimeFormat (RTC_TypeDef * RTCx, uint32_t TimeFormat)

Function description

Set Alarm A time format (AM/24-hour or PM notation)

Parameters

- **RTCx:** RTC Instance
- **TimeFormat:** This parameter can be one of the following values:
  - LL_RTC_ALMA_TIME_FORMAT_AM
  - LL_RTC_ALMA_TIME_FORMAT_PM

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMAR PM LL_RTC_ALMA_SetTimeFormat

LL_RTC_ALMA_GetTimeFormat

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetTimeFormat (RTC_TypeDef * RTCx)

Function description

Get Alarm A time format (AM or PM notation)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_ALMA_TIME_FORMAT_AM
  - LL_RTC_ALMA_TIME_FORMAT_PM

Reference Manual to LL API cross reference:

- RTC_ALRMAR PM LL_RTC_ALMA_GetTimeFormat

LL_RTC_ALMA_SetHour

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetHour (RTC_TypeDef * RTCx, uint32_t Hours)

Function description

Set ALARM A Hours in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Hours from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMAR HT LL_RTC_ALMA_SetHour RTC_ALRMAR HU LL_RTC_ALMA_SetHour

LL_RTC_ALMA_GetHour

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetHour (RTC_TypeDef * RTCx)

Function description

Get ALARM A Hours in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Hours from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMAR HT LL_RTC_ALMA_GetHour RTC_ALRMAR HU LL_RTC_ALMA_GetHour

LL_RTC_ALMA_SetMinute

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetMinute (RTC_TypeDef * RTCx, uint32_t Minutes)

Function description

Set ALARM A Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Minutes from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMAR MNT LL_RTC_ALMA_SetMinute RTC_ALRMAR MNU LL_RTC_ALMA_SetMinute

LL_RTC_ALMA_GetMinute

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetMinute (RTC_TypeDef * RTCx)

Function description

Get ALARM A Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Minutes from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMAR MNT LL_RTC_ALMA_GetMinute RTC_ALRMAR MNU LL_RTC_ALMA_GetMinute

LL_RTC_ALMA_SetSecond

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetSecond (RTC_TypeDef * RTCx, uint32_t Seconds)

Function description

Set ALARM A Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Seconds from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMAR ST LL_RTC_ALMA_SetSecond RTC_ALRMAR SU LL_RTC_ALMA_SetSecond

LL_RTC_ALMA_GetSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSecond (RTC_TypeDef * RTCx)

Function description

Get ALARM A Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Seconds from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMAR ST LL_RTC_ALMA_GetSecond RTC_ALRMAR SU LL_RTC_ALMA_GetSecond

LL_RTC_ALMA_ConfigTime

Function name

__STATIC_INLINE void LL_RTC_ALMA_ConfigTime (RTC_TypeDef * RTCx, uint32_t Format12_24, uint32_t Hours, uint32_t Minutes, uint32_t Seconds)

Function description

Set Alarm A Time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Format12_24:** This parameter can be one of the following values:
  - LL_RTC_ALMA_TIME_FORMAT_AM
  - LL_RTC_ALMA_TIME_FORMAT_PM
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
- **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59
- **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMAR PM LL_RTC_ALMA_ConfigTime RTC_ALRMAR HT LL_RTC_ALMA_ConfigTime RTC_ALRMAR HU LL_RTC_ALMA_ConfigTime RTC_ALRMAR MNT LL_RTC_ALMA_ConfigTime RTC_ALRMAR MNU LL_RTC_ALMA_ConfigTime RTC_ALRMAR ST LL_RTC_ALMA_ConfigTime RTC_ALRMAR SU LL_RTC_ALMA_ConfigTime

LL_RTC_ALMA_GetTime

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetTime (RTC_TypeDef * RTCx)

Function description

Get Alarm B Time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of hours, minutes and seconds.

Notes

- helper macros __LL_RTC_GET_HOUR, __LL_RTC_GET_MINUTE and __LL_RTC_GET_SECOND are available to get independently each parameter.

Reference Manual to LL API cross reference:

- RTC_ALRMAR HT LL_RTC_ALMA_GetTime RTC_ALRMAR HU LL_RTC_ALMA_GetTime RTC_ALRMAR MNT LL_RTC_ALMA_GetTime RTC_ALRMAR MNU LL_RTC_ALMA_GetTime RTC_ALRMAR ST LL_RTC_ALMA_GetTime RTC_ALRMAR SU LL_RTC_ALMA_GetTime

LL_RTC_ALMA_SetSubSecondMask

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetSubSecondMask (RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Set Alarm A Mask the most-significant bits starting at this bit.

Parameters

- **RTCx:** RTC Instance
- **Mask:** Value between Min_Data=0x00 and Max_Data=0xF

Return values

- **None:**

Notes

- This register can be written only when ALRAE is reset in RTC_CR register, or in initialization mode.

Reference Manual to LL API cross reference:

- RTC_ALRMASSR MASKSS LL_RTC_ALMA_SetSubSecondMask

LL_RTC_ALMA_GetSubSecondMask

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSubSecondMask (RTC_TypeDef * RTCx)

Function description

Get Alarm A Mask the most-significant bits starting at this bit.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xF

Reference Manual to LL API cross reference:

- RTC_ALRMASSR MASKSS LL_RTC_ALMA_GetSubSecondMask

LL_RTC_ALMA_SetSubSecond

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetSubSecond (RTC_TypeDef * RTCx, uint32_t Subsecond)

Function description

Set Alarm A Sub seconds value.

Parameters

- **RTCx:** RTC Instance
- **Subsecond:** Value between Min_Data=0x00 and Max_Data=0x7FFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RCT_ALRMASSR SS LL_RTC_ALMA_SetSubSecond

LL_RTC_ALMA_GetSubSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSubSecond (RTC_TypeDef * RTCx)

Function description

Get Alarm A Sub seconds value.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x7FFF

Reference Manual to LL API cross reference:

- RCT_ALRMASSR SS LL_RTC_ALMA_GetSubSecond

LL_RTC_ALMB_Enable

Function name

__STATIC_INLINE void LL_RTC_ALMB_Enable (RTC_TypeDef * RTCx)

Function description

Enable Alarm B.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRBE LL_RTC_ALMB_Enable

LL_RTC_ALMB_Disable

Function name

__STATIC_INLINE void LL_RTC_ALMB_Disable (RTC_TypeDef * RTCx)

Function description

Disable Alarm B.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRBE LL_RTC_ALMB_Disable

LL_RTC_ALMB_SetMask

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetMask (RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Specify the Alarm B masks.

Parameters

- **RTCx:** RTC Instance
- **Mask:** This parameter can be a combination of the following values:
  - LL_RTC_ALMB_MASK_NONE
  - LL_RTC_ALMB_MASK_DATEWEEKDAY
  - LL_RTC_ALMB_MASK_HOURS
  - LL_RTC_ALMB_MASK_MINUTES
  - LL_RTC_ALMB_MASK_SECONDS
  - LL_RTC_ALMB_MASK_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBR MSK4 LL_RTC_ALMB_SetMask RTC_ALRMBR MSK3 LL_RTC_ALMB_SetMask RTC_ALRMBR MSK2 LL_RTC_ALMB_SetMask RTC_ALRMBR MSK1 LL_RTC_ALMB_SetMask

LL_RTC_ALMB_GetMask

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetMask (RTC_TypeDef * RTCx)

Function description

Get the Alarm B masks.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be can be a combination of the following values:
  - LL_RTC_ALMB_MASK_NONE
  - LL_RTC_ALMB_MASK_DATEWEEKDAY
  - LL_RTC_ALMB_MASK_HOURS
  - LL_RTC_ALMB_MASK_MINUTES
  - LL_RTC_ALMB_MASK_SECONDS
  - LL_RTC_ALMB_MASK_ALL

Reference Manual to LL API cross reference:

- RTC_ALRMBR MSK4 LL_RTC_ALMB_GetMask RTC_ALRMBR MSK3 LL_RTC_ALMB_GetMask RTC_ALRMBR MSK2 LL_RTC_ALMB_GetMask RTC_ALRMBR MSK1 LL_RTC_ALMB_GetMask

LL_RTC_ALMB_EnableWeekday

Function name

__STATIC_INLINE void LL_RTC_ALMB_EnableWeekday (RTC_TypeDef * RTCx)

Function description

Enable AlarmB Week day selection (DU[3:0] represents the week day.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBR WDSEL LL_RTC_ALMB_EnableWeekday

LL_RTC_ALMB_DisableWeekday

Function name

__STATIC_INLINE void LL_RTC_ALMB_DisableWeekday (RTC_TypeDef * RTCx)

Function description

Disable AlarmB Week day selection (DU[3:0] represents the date)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBR WDSEL LL_RTC_ALMB_DisableWeekday

LL_RTC_ALMB_SetDay

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetDay (RTC_TypeDef * RTCx, uint32_t Day)

Function description

Set ALARM B Day in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Day:** Value between Min_Data=0x01 and Max_Data=0x31

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Day from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMBR DT LL_RTC_ALMB_SetDay RTC_ALRMBR DU LL_RTC_ALMB_SetDay

LL_RTC_ALMB_GetDay

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetDay (RTC_TypeDef * RTCx)

Function description

Get ALARM B Day in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x31

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMBR DT LL_RTC_ALMB_GetDay RTC_ALRMBR DU LL_RTC_ALMB_GetDay

LL_RTC_ALMB_SetWeekDay

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetWeekDay (RTC_TypeDef * RTCx, uint32_t WeekDay)

Function description

Set ALARM B Weekday.

Parameters

- **RTCx:** RTC Instance
- **WeekDay:** This parameter can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBR DU LL_RTC_ALMB_SetWeekDay

LL_RTC_ALMB_GetWeekDay

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetWeekDay (RTC_TypeDef * RTCx)

Function description

Get ALARM B Weekday.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

Reference Manual to LL API cross reference:

- RTC_ALRMBR DU LL_RTC_ALMB_GetWeekDay

LL_RTC_ALMB_SetTimeFormat

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetTimeFormat (RTC_TypeDef * RTCx, uint32_t TimeFormat)

Function description

Set ALARM B time format (AM/24-hour or PM notation)

Parameters

- **RTCx:** RTC Instance
- **TimeFormat:** This parameter can be one of the following values:
  - LL_RTC_ALMB_TIME_FORMAT_AM
  - LL_RTC_ALMB_TIME_FORMAT_PM

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBR PM LL_RTC_ALMB_SetTimeFormat

LL_RTC_ALMB_GetTimeFormat

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetTimeFormat (RTC_TypeDef * RTCx)

Function description

Get ALARM B time format (AM or PM notation)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_ALMB_TIME_FORMAT_AM
  - LL_RTC_ALMB_TIME_FORMAT_PM

Reference Manual to LL API cross reference:

- RTC_ALRMBR PM LL_RTC_ALMB_GetTimeFormat

LL_RTC_ALMB_SetHour

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetHour (RTC_TypeDef * RTCx, uint32_t Hours)

Function description

Set ALARM B Hours in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Hours from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMBR HT LL_RTC_ALMB_SetHour RTC_ALRMBR HU LL_RTC_ALMB_SetHour

LL_RTC_ALMB_GetHour

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetHour (RTC_TypeDef * RTCx)

Function description

Get ALARM B Hours in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Hours from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMBR HT LL_RTC_ALMB_GetHour RTC_ALRMBR HU LL_RTC_ALMB_GetHour

LL_RTC_ALMB_SetMinute

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetMinute (RTC_TypeDef * RTCx, uint32_t Minutes)

Function description

Set ALARM B Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Minutes:** between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Minutes from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMBR MNT LL_RTC_ALMB_SetMinute RTC_ALRMBR MNU LL_RTC_ALMB_SetMinute

LL_RTC_ALMB_GetMinute

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetMinute (RTC_TypeDef * RTCx)

Function description

Get ALARM B Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Minutes from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMBR MNT LL_RTC_ALMB_GetMinute RTC_ALRMBR MNU LL_RTC_ALMB_GetMinute

LL_RTC_ALMB_SetSecond

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetSecond (RTC_TypeDef * RTCx, uint32_t Seconds)

Function description

Set ALARM B Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Seconds from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMBR ST LL_RTC_ALMB_SetSecond RTC_ALRMBR SU LL_RTC_ALMB_SetSecond

LL_RTC_ALMB_GetSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetSecond (RTC_TypeDef * RTCx)

Function description

Get ALARM B Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Seconds from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMBR ST LL_RTC_ALMB_GetSecond RTC_ALRMBR SU LL_RTC_ALMB_GetSecond

LL_RTC_ALMB_ConfigTime

Function name

__STATIC_INLINE void LL_RTC_ALMB_ConfigTime (RTC_TypeDef * RTCx, uint32_t Format12_24, uint32_t Hours, uint32_t Minutes, uint32_t Seconds)

Function description

Set Alarm B Time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Format12_24:** This parameter can be one of the following values:
  - LL_RTC_ALMB_TIME_FORMAT_AM
  - LL_RTC_ALMB_TIME_FORMAT_PM
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
- **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59
- **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBR PM LL_RTC_ALMB_ConfigTime RTC_ALRMBR HT LL_RTC_ALMB_ConfigTime RTC_ALRMBR HU LL_RTC_ALMB_ConfigTime RTC_ALRMBR MNT LL_RTC_ALMB_ConfigTime RTC_ALRMBR MNU LL_RTC_ALMB_ConfigTime RTC_ALRMBR ST LL_RTC_ALMB_ConfigTime RTC_ALRMBR SU LL_RTC_ALMB_ConfigTime

LL_RTC_ALMB_GetTime

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetTime (RTC_TypeDef * RTCx)

Function description

Get Alarm B Time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of hours, minutes and seconds.

Notes

- helper macros __LL_RTC_GET_HOUR, __LL_RTC_GET_MINUTE and __LL_RTC_GET_SECOND are available to get independently each parameter.

Reference Manual to LL API cross reference:

- RTC_ALRMBR HT LL_RTC_ALMB_GetTime RTC_ALRMBR HU LL_RTC_ALMB_GetTime RTC_ALRMBR MNT LL_RTC_ALMB_GetTime RTC_ALRMBR MNU LL_RTC_ALMB_GetTime RTC_ALRMBR ST LL_RTC_ALMB_GetTime RTC_ALRMBR SU LL_RTC_ALMB_GetTime

LL_RTC_ALMB_SetSubSecondMask

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetSubSecondMask (RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Set Alarm B Mask the most-significant bits starting at this bit.

Parameters

- **RTCx:** RTC Instance
- **Mask:** Value between Min_Data=0x00 and Max_Data=0xF

Return values

- **None:**

Notes

- This register can be written only when ALRBE is reset in RTC_CR register, or in initialization mode.

Reference Manual to LL API cross reference:

- RTC_ALRMBSSR MASKSS LL_RTC_ALMB_SetSubSecondMask

LL_RTC_ALMB_GetSubSecondMask

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetSubSecondMask (RTC_TypeDef * RTCx)

Function description

Get Alarm B Mask the most-significant bits starting at this bit.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xF

Reference Manual to LL API cross reference:

- RTC_ALRMBSSR MASKSS LL_RTC_ALMB_GetSubSecondMask

LL_RTC_ALMB_SetSubSecond

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetSubSecond (RTC_TypeDef * RTCx, uint32_t Subsecond)

Function description

Set Alarm B Sub seconds value.

Parameters

- **RTCx:** RTC Instance
- **Subsecond:** Value between Min_Data=0x00 and Max_Data=0x7FFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBSSR SS LL_RTC_ALMB_SetSubSecond

LL_RTC_ALMB_GetSubSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetSubSecond (RTC_TypeDef * RTCx)

Function description

Get Alarm B Sub seconds value.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x7FFF

Reference Manual to LL API cross reference:

- RTC_ALRMBSSR SS LL_RTC_ALMB_GetSubSecond

LL_RTC_TS_EnableInternalEvent

Function name

__STATIC_INLINE void LL_RTC_TS_EnableInternalEvent (RTC_TypeDef * RTCx)

Function description

Enable internal event timestamp.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ITSE LL_RTC_TS_EnableInternalEvent

LL_RTC_TS_DisableInternalEvent

Function name

__STATIC_INLINE void LL_RTC_TS_DisableInternalEvent (RTC_TypeDef * RTCx)

Function description

Disable internal event timestamp.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ITSE LL_RTC_TS_DisableInternalEvent

LL_RTC_TS_Enable

Function name

__STATIC_INLINE void LL_RTC_TS_Enable (RTC_TypeDef * RTCx)

Function description

Enable Timestamp.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR TSE LL_RTC_TS_Enable

LL_RTC_TS_Disable

Function name

__STATIC_INLINE void LL_RTC_TS_Disable (RTC_TypeDef * RTCx)

Function description

Disable Timestamp.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR TSE LL_RTC_TS_Disable

LL_RTC_TS_SetActiveEdge

Function name

__STATIC_INLINE void LL_RTC_TS_SetActiveEdge (RTC_TypeDef * RTCx, uint32_t Edge)

Function description

Set Time-stamp event active edge.

Parameters

- **RTCx:** RTC Instance
- **Edge:** This parameter can be one of the following values:
  - LL_RTC_TIMESTAMP_EDGE_RISING
  - LL_RTC_TIMESTAMP_EDGE_FALLING

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- TSE must be reset when TSEDGE is changed to avoid unwanted TSF setting

Reference Manual to LL API cross reference:

- RTC_CR TSEDGE LL_RTC_TS_SetActiveEdge

LL_RTC_TS_GetActiveEdge

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetActiveEdge (RTC_TypeDef * RTCx)

Function description

Get Time-stamp event active edge.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_TIMESTAMP_EDGE_RISING
  - LL_RTC_TIMESTAMP_EDGE_FALLING

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR TSEDGE LL_RTC_TS_GetActiveEdge

LL_RTC_TS_GetTimeFormat

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetTimeFormat (RTC_TypeDef * RTCx)

Function description

Get Timestamp AM/PM notation (AM or 24-hour format)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_TS_TIME_FORMAT_AM
  - LL_RTC_TS_TIME_FORMAT_PM

Reference Manual to LL API cross reference:

- RTC_TSTR PM LL_RTC_TS_GetTimeFormat

LL_RTC_TS_GetHour

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetHour (RTC_TypeDef * RTCx)

Function description

Get Timestamp Hours in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Hours from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TSTR HT LL_RTC_TS_GetHour RTC_TSTR HU LL_RTC_TS_GetHour

LL_RTC_TS_GetMinute

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetMinute (RTC_TypeDef * RTCx)

Function description

Get Timestamp Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Minutes from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TSTR MNT LL_RTC_TS_GetMinute RTC_TSTR MNU LL_RTC_TS_GetMinute

LL_RTC_TS_GetSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetSecond (RTC_TypeDef * RTCx)

Function description

Get Timestamp Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Seconds from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TSTR ST LL_RTC_TS_GetSecond RTC_TSTR SU LL_RTC_TS_GetSecond

LL_RTC_TS_GetTime

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetTime (RTC_TypeDef * RTCx)

Function description

Get Timestamp time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of hours, minutes and seconds.

Notes

- helper macros __LL_RTC_GET_HOUR, __LL_RTC_GET_MINUTE and __LL_RTC_GET_SECOND are available to get independently each parameter.

Reference Manual to LL API cross reference:

- RTC_TSTR HT LL_RTC_TS_GetTime RTC_TSTR HU LL_RTC_TS_GetTime RTC_TSTR MNT LL_RTC_TS_GetTime RTC_TSTR MNU LL_RTC_TS_GetTime RTC_TSTR ST LL_RTC_TS_GetTime RTC_TSTR SU LL_RTC_TS_GetTime

LL_RTC_TS_GetWeekDay

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetWeekDay (RTC_TypeDef * RTCx)

Function description

Get Timestamp Week day.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

Reference Manual to LL API cross reference:

- RTC_TSDR WDU LL_RTC_TS_GetWeekDay

LL_RTC_TS_GetMonth

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetMonth (RTC_TypeDef * RTCx)

Function description

Get Timestamp Month in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_MONTH_JANUARY
  - LL_RTC_MONTH_FEBRUARY
  - LL_RTC_MONTH_MARCH
  - LL_RTC_MONTH_APRIL
  - LL_RTC_MONTH_MAY
  - LL_RTC_MONTH_JUNE
  - LL_RTC_MONTH_JULY
  - LL_RTC_MONTH_AUGUST
  - LL_RTC_MONTH_SEPTEMBER
  - LL_RTC_MONTH_OCTOBER
  - LL_RTC_MONTH_NOVEMBER
  - LL_RTC_MONTH_DECEMBER

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Month from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TSDR MT LL_RTC_TS_GetMonth RTC_TSDR MU LL_RTC_TS_GetMonth

LL_RTC_TS_GetDay

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetDay (RTC_TypeDef * RTCx)

Function description

Get Timestamp Day in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x31

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TSDR DT LL_RTC_TS_GetDay RTC_TSDR DU LL_RTC_TS_GetDay

LL_RTC_TS_GetDate

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetDate (RTC_TypeDef * RTCx)

Function description

Get Timestamp date (WeekDay, Day and Month) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of Weekday, Day and Month

Notes

- helper macros __LL_RTC_GET_WEEKDAY, __LL_RTC_GET_MONTH, and __LL_RTC_GET_DAY are available to get independently each parameter.

Reference Manual to LL API cross reference:

- RTC_TSDR WDU LL_RTC_TS_GetDate RTC_TSDR MT LL_RTC_TS_GetDate RTC_TSDR MU LL_RTC_TS_GetDate RTC_TSDR DT LL_RTC_TS_GetDate RTC_TSDR DU LL_RTC_TS_GetDate

LL_RTC_TS_GetSubSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetSubSecond (RTC_TypeDef * RTCx)

Function description

Get time-stamp sub second value.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- RTC_TSSSR SS LL_RTC_TS_GetSubSecond

LL_RTC_TS_EnableOnTamper

Function name

__STATIC_INLINE void LL_RTC_TS_EnableOnTamper (RTC_TypeDef * RTCx)

Function description

Activate timestamp on tamper detection event.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR TAMPTS LL_RTC_TS_EnableOnTamper

LL_RTC_TS_DisableOnTamper

Function name

__STATIC_INLINE void LL_RTC_TS_DisableOnTamper (RTC_TypeDef * RTCx)

Function description

Disable timestamp on tamper detection event.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR TAMPTS LL_RTC_TS_DisableOnTamper

LL_RTC_TAMPER_Enable

Function name

__STATIC_INLINE void LL_RTC_TAMPER_Enable (RTC_TypeDef * RTCx, uint32_t Tamper)

Function description

Enable RTC_TAMPx input detection.

Parameters

- **RTCx:** RTC Instance
- **Tamper:** This parameter can be a combination of the following values:
  - LL_RTC_TAMPER_1
  - LL_RTC_TAMPER_2
  - LL_RTC_TAMPER_3

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMPCR TAMP1E LL_RTC_TAMPER_Enable TAMPCR TAMP2E LL_RTC_TAMPER_Enable TAMPCR TAMP3E LL_RTC_TAMPER_Enable

LL_RTC_TAMPER_Disable

Function name

__STATIC_INLINE void LL_RTC_TAMPER_Disable (RTC_TypeDef * RTCx, uint32_t Tamper)

Function description

Clear RTC_TAMPx input detection.

Parameters

- **RTCx:** RTC Instance
- **Tamper:** This parameter can be a combination of the following values:
  - LL_RTC_TAMPER_1
  - LL_RTC_TAMPER_2
  - LL_RTC_TAMPER_3

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMPCR TAMP1E LL_RTC_TAMPER_Disable TAMPCR TAMP2E LL_RTC_TAMPER_Disable TAMPCR TAMP3E LL_RTC_TAMPER_Disable

LL_RTC_TAMPER_EnableMask

Function name

__STATIC_INLINE void LL_RTC_TAMPER_EnableMask (RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Enable Tamper mask flag.

Parameters

- **RTCx:** RTC Instance
- **Mask:** This parameter can be a combination of the following values:
  - LL_RTC_TAMPER_MASK_TAMPER1
  - LL_RTC_TAMPER_MASK_TAMPER2
  - LL_RTC_TAMPER_MASK_TAMPER3

Return values

- **None:**

Notes

- Associated Tamper IT must not enabled when tamper mask is set.

Reference Manual to LL API cross reference:

- TAMPCR TAMP1MF LL_RTC_TAMPER_EnableMask TAMPCR TAMP2MF LL_RTC_TAMPER_EnableMask TAMPCR TAMP3MF LL_RTC_TAMPER_EnableMask

LL_RTC_TAMPER_DisableMask

Function name

__STATIC_INLINE void LL_RTC_TAMPER_DisableMask (RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Disable Tamper mask flag.

Parameters

- **RTCx:** RTC Instance
- **Mask:** This parameter can be a combination of the following values:
  - LL_RTC_TAMPER_MASK_TAMPER1
  - LL_RTC_TAMPER_MASK_TAMPER2
  - LL_RTC_TAMPER_MASK_TAMPER3

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMPCR TAMP1MF LL_RTC_TAMPER_DisableMask TAMPCR TAMP2MF LL_RTC_TAMPER_DisableMask TAMPCR TAMP3MF LL_RTC_TAMPER_DisableMask

LL_RTC_TAMPER_EnableEraseBKP

Function name

__STATIC_INLINE void LL_RTC_TAMPER_EnableEraseBKP (RTC_TypeDef * RTCx, uint32_t Tamper)

Function description

Enable backup register erase after Tamper event detection.

Parameters

- **RTCx:** RTC Instance
- **Tamper:** This parameter can be a combination of the following values:
  - LL_RTC_TAMPER_NOERASE_TAMPER1
  - LL_RTC_TAMPER_NOERASE_TAMPER2
  - LL_RTC_TAMPER_NOERASE_TAMPER3

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMPCR TAMP1NOERASE LL_RTC_TAMPER_EnableEraseBKP TAMPCR TAMP2NOERASE LL_RTC_TAMPER_EnableEraseBKP TAMPCR TAMP3NOERASE LL_RTC_TAMPER_EnableEraseBKP

LL_RTC_TAMPER_DisableEraseBKP

Function name

__STATIC_INLINE void LL_RTC_TAMPER_DisableEraseBKP (RTC_TypeDef * RTCx, uint32_t Tamper)

Function description

Disable backup register erase after Tamper event detection.

Parameters

- **RTCx:** RTC Instance
- **Tamper:** This parameter can be a combination of the following values:
  - LL_RTC_TAMPER_NOERASE_TAMPER1
  - LL_RTC_TAMPER_NOERASE_TAMPER2
  - LL_RTC_TAMPER_NOERASE_TAMPER3

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMPCR TAMP1NOERASE LL_RTC_TAMPER_DisableEraseBKP TAMPCR TAMP2NOERASE LL_RTC_TAMPER_DisableEraseBKP TAMPCR TAMP3NOERASE LL_RTC_TAMPER_DisableEraseBKP

LL_RTC_TAMPER_DisablePullUp

Function name

__STATIC_INLINE void LL_RTC_TAMPER_DisablePullUp (RTC_TypeDef * RTCx)

Function description

Disable RTC_TAMPx pull-up disable (Disable precharge of RTC_TAMPx pins)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMPCR TAMPPUDIS LL_RTC_TAMPER_DisablePullUp

LL_RTC_TAMPER_EnablePullUp

Function name

__STATIC_INLINE void LL_RTC_TAMPER_EnablePullUp (RTC_TypeDef * RTCx)

Function description

Enable RTC_TAMPx pull-up disable (Precharge RTC_TAMPx pins before sampling)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMPCR TAMPPUDIS LL_RTC_TAMPER_EnablePullUp

LL_RTC_TAMPER_SetPrecharge

Function name

__STATIC_INLINE void LL_RTC_TAMPER_SetPrecharge (RTC_TypeDef * RTCx, uint32_t Duration)

Function description

Set RTC_TAMPx precharge duration.

Parameters

- **RTCx:** RTC Instance
- **Duration:** This parameter can be one of the following values:
  - LL_RTC_TAMPER_DURATION_1RTCCLK
  - LL_RTC_TAMPER_DURATION_2RTCCLK
  - LL_RTC_TAMPER_DURATION_4RTCCLK
  - LL_RTC_TAMPER_DURATION_8RTCCLK

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMPCR TAMPPRCH LL_RTC_TAMPER_SetPrecharge

LL_RTC_TAMPER_GetPrecharge

Function name

__STATIC_INLINE uint32_t LL_RTC_TAMPER_GetPrecharge (RTC_TypeDef * RTCx)

Function description

Get RTC_TAMPx precharge duration.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_TAMPER_DURATION_1RTCCLK
  - LL_RTC_TAMPER_DURATION_2RTCCLK
  - LL_RTC_TAMPER_DURATION_4RTCCLK
  - LL_RTC_TAMPER_DURATION_8RTCCLK

Reference Manual to LL API cross reference:

- TAMPCR TAMPPRCH LL_RTC_TAMPER_GetPrecharge

LL_RTC_TAMPER_SetFilterCount

Function name

__STATIC_INLINE void LL_RTC_TAMPER_SetFilterCount (RTC_TypeDef * RTCx, uint32_t FilterCount)

Function description

Set RTC_TAMPx filter count.

Parameters

- **RTCx:** RTC Instance
- **FilterCount:** This parameter can be one of the following values:
  - LL_RTC_TAMPER_FILTER_DISABLE
  - LL_RTC_TAMPER_FILTER_2SAMPLE
  - LL_RTC_TAMPER_FILTER_4SAMPLE
  - LL_RTC_TAMPER_FILTER_8SAMPLE

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMPCR TAMPFLT LL_RTC_TAMPER_SetFilterCount

LL_RTC_TAMPER_GetFilterCount

Function name

__STATIC_INLINE uint32_t LL_RTC_TAMPER_GetFilterCount (RTC_TypeDef * RTCx)

Function description

Get RTC_TAMPx filter count.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_TAMPER_FILTER_DISABLE
  - LL_RTC_TAMPER_FILTER_2SAMPLE
  - LL_RTC_TAMPER_FILTER_4SAMPLE
  - LL_RTC_TAMPER_FILTER_8SAMPLE

Reference Manual to LL API cross reference:

- TAMPCR TAMPFLT LL_RTC_TAMPER_GetFilterCount

LL_RTC_TAMPER_SetSamplingFreq

Function name

__STATIC_INLINE void LL_RTC_TAMPER_SetSamplingFreq (RTC_TypeDef * RTCx, uint32_t SamplingFreq)

Function description

Set Tamper sampling frequency.

Parameters

- **RTCx:** RTC Instance
- **SamplingFreq:** This parameter can be one of the following values:
  - LL_RTC_TAMPER_SAMPLFREQDIV_32768
  - LL_RTC_TAMPER_SAMPLFREQDIV_16384
  - LL_RTC_TAMPER_SAMPLFREQDIV_8192
  - LL_RTC_TAMPER_SAMPLFREQDIV_4096
  - LL_RTC_TAMPER_SAMPLFREQDIV_2048
  - LL_RTC_TAMPER_SAMPLFREQDIV_1024
  - LL_RTC_TAMPER_SAMPLFREQDIV_512
  - LL_RTC_TAMPER_SAMPLFREQDIV_256

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMPCR TAMPFREQ LL_RTC_TAMPER_SetSamplingFreq

LL_RTC_TAMPER_GetSamplingFreq

Function name

__STATIC_INLINE uint32_t LL_RTC_TAMPER_GetSamplingFreq (RTC_TypeDef * RTCx)

Function description

Get Tamper sampling frequency.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_TAMPER_SAMPLFREQDIV_32768
  - LL_RTC_TAMPER_SAMPLFREQDIV_16384
  - LL_RTC_TAMPER_SAMPLFREQDIV_8192
  - LL_RTC_TAMPER_SAMPLFREQDIV_4096
  - LL_RTC_TAMPER_SAMPLFREQDIV_2048
  - LL_RTC_TAMPER_SAMPLFREQDIV_1024
  - LL_RTC_TAMPER_SAMPLFREQDIV_512
  - LL_RTC_TAMPER_SAMPLFREQDIV_256

Reference Manual to LL API cross reference:

- TAMPCR TAMPFREQ LL_RTC_TAMPER_GetSamplingFreq

LL_RTC_TAMPER_EnableActiveLevel

Function name

__STATIC_INLINE void LL_RTC_TAMPER_EnableActiveLevel (RTC_TypeDef * RTCx, uint32_t Tamper)

Function description

Enable Active level for Tamper input.

Parameters

- **RTCx:** RTC Instance
- **Tamper:** This parameter can be a combination of the following values:
  - LL_RTC_TAMPER_ACTIVELEVEL_TAMP1
  - LL_RTC_TAMPER_ACTIVELEVEL_TAMP2
  - LL_RTC_TAMPER_ACTIVELEVEL_TAMP3

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMPCR TAMP1TRG LL_RTC_TAMPER_EnableActiveLevel TAMPCR TAMP2TRG LL_RTC_TAMPER_EnableActiveLevel TAMPCR TAMP3TRG LL_RTC_TAMPER_EnableActiveLevel

LL_RTC_TAMPER_DisableActiveLevel

Function name

__STATIC_INLINE void LL_RTC_TAMPER_DisableActiveLevel (RTC_TypeDef * RTCx, uint32_t Tamper)

Function description

Disable Active level for Tamper input.

Parameters

- **RTCx:** RTC Instance
- **Tamper:** This parameter can be a combination of the following values:
  - LL_RTC_TAMPER_ACTIVELEVEL_TAMP1
  - LL_RTC_TAMPER_ACTIVELEVEL_TAMP2
  - LL_RTC_TAMPER_ACTIVELEVEL_TAMP3

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMPCR TAMP1TRG LL_RTC_TAMPER_DisableActiveLevel TAMPCR TAMP2TRG LL_RTC_TAMPER_DisableActiveLevel TAMPCR TAMP3TRG LL_RTC_TAMPER_DisableActiveLevel

LL_RTC_WAKEUP_Enable

Function name

__STATIC_INLINE void LL_RTC_WAKEUP_Enable (RTC_TypeDef * RTCx)

Function description

Enable Wakeup timer.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR WUTE LL_RTC_WAKEUP_Enable

LL_RTC_WAKEUP_Disable

Function name

__STATIC_INLINE void LL_RTC_WAKEUP_Disable (RTC_TypeDef * RTCx)

Function description

Disable Wakeup timer.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR WUTE LL_RTC_WAKEUP_Disable

LL_RTC_WAKEUP_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RTC_WAKEUP_IsEnabled (RTC_TypeDef * RTCx)

Function description

Check if Wakeup timer is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR WUTE LL_RTC_WAKEUP_IsEnabled

LL_RTC_WAKEUP_SetClock

Function name

__STATIC_INLINE void LL_RTC_WAKEUP_SetClock (RTC_TypeDef * RTCx, uint32_t WakeupClock)

Function description

Select Wakeup clock.

Parameters

- **RTCx:** RTC Instance
- **WakeupClock:** This parameter can be one of the following values:
  - LL_RTC_WAKEUPCLOCK_DIV_16
  - LL_RTC_WAKEUPCLOCK_DIV_8
  - LL_RTC_WAKEUPCLOCK_DIV_4
  - LL_RTC_WAKEUPCLOCK_DIV_2
  - LL_RTC_WAKEUPCLOCK_CKSPRE
  - LL_RTC_WAKEUPCLOCK_CKSPRE_WUT

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RTC_CR WUTE bit = 0 and RTC_ISR WUTWF bit = 1

Reference Manual to LL API cross reference:

- RTC_CR WUCKSEL LL_RTC_WAKEUP_SetClock

LL_RTC_WAKEUP_GetClock

Function name

__STATIC_INLINE uint32_t LL_RTC_WAKEUP_GetClock (RTC_TypeDef * RTCx)

Function description

Get Wakeup clock.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_WAKEUPCLOCK_DIV_16
  - LL_RTC_WAKEUPCLOCK_DIV_8
  - LL_RTC_WAKEUPCLOCK_DIV_4
  - LL_RTC_WAKEUPCLOCK_DIV_2
  - LL_RTC_WAKEUPCLOCK_CKSPRE
  - LL_RTC_WAKEUPCLOCK_CKSPRE_WUT

Reference Manual to LL API cross reference:

- RTC_CR WUCKSEL LL_RTC_WAKEUP_GetClock

LL_RTC_WAKEUP_SetAutoReload

Function name

__STATIC_INLINE void LL_RTC_WAKEUP_SetAutoReload (RTC_TypeDef * RTCx, uint32_t Value)

Function description

Set Wakeup auto-reload value.

Parameters

- **RTCx:** RTC Instance
- **Value:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Return values

- **None:**

Notes

- Bit can be written only when WUTWF is set to 1 in RTC_ISR

Reference Manual to LL API cross reference:

- RTC_WUTR WUT LL_RTC_WAKEUP_SetAutoReload

LL_RTC_WAKEUP_GetAutoReload

Function name

__STATIC_INLINE uint32_t LL_RTC_WAKEUP_GetAutoReload (RTC_TypeDef * RTCx)

Function description

Get Wakeup auto-reload value.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- RTC_WUTR WUT LL_RTC_WAKEUP_GetAutoReload

LL_RTC_BAK_SetRegister

Function name

__STATIC_INLINE void LL_RTC_BAK_SetRegister (RTC_TypeDef * RTCx, uint32_t BackupRegister, uint32_t Data)

Function description

Writes a data in a specified RTC Backup data register.

Parameters

- **RTCx:** RTC Instance
- **BackupRegister:** This parameter can be one of the following values:
  - LL_RTC_BKP_DR0
  - LL_RTC_BKP_DR1
  - LL_RTC_BKP_DR2
  - LL_RTC_BKP_DR3
  - LL_RTC_BKP_DR4
  - LL_RTC_BKP_DR5
  - LL_RTC_BKP_DR6
  - LL_RTC_BKP_DR7
  - LL_RTC_BKP_DR8
  - LL_RTC_BKP_DR9
  - LL_RTC_BKP_DR10
  - LL_RTC_BKP_DR11
  - LL_RTC_BKP_DR12
  - LL_RTC_BKP_DR13
  - LL_RTC_BKP_DR14
  - LL_RTC_BKP_DR15
  - LL_RTC_BKP_DR16
  - LL_RTC_BKP_DR17
  - LL_RTC_BKP_DR18
  - LL_RTC_BKP_DR19
  - LL_RTC_BKP_DR20
  - LL_RTC_BKP_DR21
  - LL_RTC_BKP_DR22
  - LL_RTC_BKP_DR23
  - LL_RTC_BKP_DR24
  - LL_RTC_BKP_DR25
  - LL_RTC_BKP_DR26
  - LL_RTC_BKP_DR27
  - LL_RTC_BKP_DR28
  - LL_RTC_BKP_DR29
  - LL_RTC_BKP_DR30
  - LL_RTC_BKP_DR31
- **Data:** Value between Min_Data=0x00 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- BKPxR BKP LL_RTC_BAK_SetRegister

LL_RTC_BAK_GetRegister

Function name

__STATIC_INLINE uint32_t LL_RTC_BAK_GetRegister (RTC_TypeDef * RTCx, uint32_t BackupRegister)

Function description

Reads data from the specified RTC Backup data Register.

Parameters

- **RTCx:** RTC Instance
- **BackupRegister:** This parameter can be one of the following values:
  - LL_RTC_BKP_DR0
  - LL_RTC_BKP_DR1
  - LL_RTC_BKP_DR2
  - LL_RTC_BKP_DR3
  - LL_RTC_BKP_DR4
  - LL_RTC_BKP_DR5
  - LL_RTC_BKP_DR6
  - LL_RTC_BKP_DR7
  - LL_RTC_BKP_DR8
  - LL_RTC_BKP_DR9
  - LL_RTC_BKP_DR10
  - LL_RTC_BKP_DR11
  - LL_RTC_BKP_DR12
  - LL_RTC_BKP_DR13
  - LL_RTC_BKP_DR14
  - LL_RTC_BKP_DR15
  - LL_RTC_BKP_DR16
  - LL_RTC_BKP_DR17
  - LL_RTC_BKP_DR18
  - LL_RTC_BKP_DR19
  - LL_RTC_BKP_DR20
  - LL_RTC_BKP_DR21
  - LL_RTC_BKP_DR22
  - LL_RTC_BKP_DR23
  - LL_RTC_BKP_DR24
  - LL_RTC_BKP_DR25
  - LL_RTC_BKP_DR26
  - LL_RTC_BKP_DR27
  - LL_RTC_BKP_DR28
  - LL_RTC_BKP_DR29
  - LL_RTC_BKP_DR30
  - LL_RTC_BKP_DR31

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- BKPxR BKP LL_RTC_BAK_GetRegister

LL_RTC_CAL_SetOutputFreq

Function name

__STATIC_INLINE void LL_RTC_CAL_SetOutputFreq (RTC_TypeDef * RTCx, uint32_t Frequency)

Function description

Set Calibration output frequency (1 Hz or 512 Hz)

Parameters

- **RTCx:** RTC Instance
- **Frequency:** This parameter can be one of the following values:
  - LL_RTC_CALIB_OUTPUT_NONE
  - LL_RTC_CALIB_OUTPUT_1HZ
  - LL_RTC_CALIB_OUTPUT_512HZ

Return values

- **None:**

Notes

- Bits are write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR COE LL_RTC_CAL_SetOutputFreq RTC_CR COSEL LL_RTC_CAL_SetOutputFreq

LL_RTC_CAL_GetOutputFreq

Function name

__STATIC_INLINE uint32_t LL_RTC_CAL_GetOutputFreq (RTC_TypeDef * RTCx)

Function description

Get Calibration output frequency (1 Hz or 512 Hz)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_CALIB_OUTPUT_NONE
  - LL_RTC_CALIB_OUTPUT_1HZ
  - LL_RTC_CALIB_OUTPUT_512HZ

Reference Manual to LL API cross reference:

- RTC_CR COE LL_RTC_CAL_GetOutputFreq RTC_CR COSEL LL_RTC_CAL_GetOutputFreq

LL_RTC_CAL_SetPulse

Function name

__STATIC_INLINE void LL_RTC_CAL_SetPulse (RTC_TypeDef * RTCx, uint32_t Pulse)

Function description

Insert or not One RTCCLK pulse every 2exp11 pulses (frequency increased by 488.5 ppm)

Parameters

- **RTCx:** RTC Instance
- **Pulse:** This parameter can be one of the following values:
  - LL_RTC_CALIB_INSERTPULSE_NONE
  - LL_RTC_CALIB_INSERTPULSE_SET

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RECALPF is set to 0 in RTC_ISR

Reference Manual to LL API cross reference:

- RTC_CALR CALP LL_RTC_CAL_SetPulse

LL_RTC_CAL_IsPulseInserted

Function name

__STATIC_INLINE uint32_t LL_RTC_CAL_IsPulseInserted (RTC_TypeDef * RTCx)

Function description

Check if one RTCCLK has been inserted or not every 2exp11 pulses (frequency increased by 488.5 ppm)

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CALR CALP LL_RTC_CAL_IsPulseInserted

LL_RTC_CAL_SetPeriod

Function name

__STATIC_INLINE void LL_RTC_CAL_SetPeriod (RTC_TypeDef * RTCx, uint32_t Period)

Function description

Set the calibration cycle period.

Parameters

- **RTCx:** RTC Instance
- **Period:** This parameter can be one of the following values:
  - LL_RTC_CALIB_PERIOD_32SEC
  - LL_RTC_CALIB_PERIOD_16SEC
  - LL_RTC_CALIB_PERIOD_8SEC

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RECALPF is set to 0 in RTC_ISR

Reference Manual to LL API cross reference:

- RTC_CALR CALW8 LL_RTC_CAL_SetPeriod RTC_CALR CALW16 LL_RTC_CAL_SetPeriod

LL_RTC_CAL_GetPeriod

Function name

__STATIC_INLINE uint32_t LL_RTC_CAL_GetPeriod (RTC_TypeDef * RTCx)

Function description

Get the calibration cycle period.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_CALIB_PERIOD_32SEC
  - LL_RTC_CALIB_PERIOD_16SEC
  - LL_RTC_CALIB_PERIOD_8SEC

Reference Manual to LL API cross reference:

- RTC_CALR CALW8 LL_RTC_CAL_GetPeriod RTC_CALR CALW16 LL_RTC_CAL_GetPeriod

LL_RTC_CAL_SetMinus

Function name

__STATIC_INLINE void LL_RTC_CAL_SetMinus (RTC_TypeDef * RTCx, uint32_t CalibMinus)

Function description

Set Calibration minus.

Parameters

- **RTCx:** RTC Instance
- **CalibMinus:** Value between Min_Data=0x00 and Max_Data=0x1FF

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RECALPF is set to 0 in RTC_ISR

Reference Manual to LL API cross reference:

- RTC_CALR CALM LL_RTC_CAL_SetMinus

LL_RTC_CAL_GetMinus

Function name

__STATIC_INLINE uint32_t LL_RTC_CAL_GetMinus (RTC_TypeDef * RTCx)

Function description

Get Calibration minus.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data= 0x1FF

Reference Manual to LL API cross reference:

- RTC_CALR CALM LL_RTC_CAL_GetMinus

LL_RTC_IsActiveFlag_ITS

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITS (RTC_TypeDef * RTCx)

Function description

Get Internal Time-stamp flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_ISR ITSF LL_RTC_IsActiveFlag_ITS

LL_RTC_IsActiveFlag_RECALP

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_RECALP (RTC_TypeDef * RTCx)

Function description

Get Recalibration pending Flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RECALPF LL_RTC_IsActiveFlag_RECALP

LL_RTC_IsActiveFlag_TAMP3

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP3 (RTC_TypeDef * RTCx)

Function description

Get RTC_TAMP3 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TAMP3F LL_RTC_IsActiveFlag_TAMP3

LL_RTC_IsActiveFlag_TAMP2

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP2 (RTC_TypeDef * RTCx)

Function description

Get RTC_TAMP2 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TAMP2F LL_RTC_IsActiveFlag_TAMP2

LL_RTC_IsActiveFlag_TAMP1

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP1 (RTC_TypeDef * RTCx)

Function description

Get RTC_TAMP1 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TAMP1F LL_RTC_IsActiveFlag_TAMP1

LL_RTC_IsActiveFlag_TSOV

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TSOV (RTC_TypeDef * RTCx)

Function description

Get Time-stamp overflow flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TSOVF LL_RTC_IsActiveFlag_TSOV

LL_RTC_IsActiveFlag_TS

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TS (RTC_TypeDef * RTCx)

Function description

Get Time-stamp flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TSF LL_RTC_IsActiveFlag_TS

LL_RTC_IsActiveFlag_WUT

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_WUT (RTC_TypeDef * RTCx)

Function description

Get Wakeup timer flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR WUTF LL_RTC_IsActiveFlag_WUT

LL_RTC_IsActiveFlag_ALRB

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRB (RTC_TypeDef * RTCx)

Function description

Get Alarm B flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ALRBF LL_RTC_IsActiveFlag_ALRB

LL_RTC_IsActiveFlag_ALRA

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRA (RTC_TypeDef * RTCx)

Function description

Get Alarm A flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ALRAF LL_RTC_IsActiveFlag_ALRA

LL_RTC_ClearFlag_ITS

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITS (RTC_TypeDef * RTCx)

Function description

Clear Internal Time-stamp flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR ITSF LL_RTC_ClearFlag_ITS

LL_RTC_ClearFlag_TAMP3

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TAMP3 (RTC_TypeDef * RTCx)

Function description

Clear RTC_TAMP3 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR TAMP3F LL_RTC_ClearFlag_TAMP3

LL_RTC_ClearFlag_TAMP2

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TAMP2 (RTC_TypeDef * RTCx)

Function description

Clear RTC_TAMP2 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR TAMP2F LL_RTC_ClearFlag_TAMP2

LL_RTC_ClearFlag_TAMP1

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TAMP1 (RTC_TypeDef * RTCx)

Function description

Clear RTC_TAMP1 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR TAMP1F LL_RTC_ClearFlag_TAMP1

LL_RTC_ClearFlag_TSOV

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TSOV (RTC_TypeDef * RTCx)

Function description

Clear Time-stamp overflow flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR TSOVF LL_RTC_ClearFlag_TSOV

LL_RTC_ClearFlag_TS

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TS (RTC_TypeDef * RTCx)

Function description

Clear Time-stamp flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR TSF LL_RTC_ClearFlag_TS

LL_RTC_ClearFlag_WUT

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_WUT (RTC_TypeDef * RTCx)

Function description

Clear Wakeup timer flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR WUTF LL_RTC_ClearFlag_WUT

LL_RTC_ClearFlag_ALRB

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ALRB (RTC_TypeDef * RTCx)

Function description

Clear Alarm B flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR ALRBF LL_RTC_ClearFlag_ALRB

LL_RTC_ClearFlag_ALRA

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ALRA (RTC_TypeDef * RTCx)

Function description

Clear Alarm A flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR ALRAF LL_RTC_ClearFlag_ALRA

LL_RTC_IsActiveFlag_INIT

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_INIT (RTC_TypeDef * RTCx)

Function description

Get Initialization flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR INITF LL_RTC_IsActiveFlag_INIT

LL_RTC_IsActiveFlag_RS

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_RS (RTC_TypeDef * RTCx)

Function description

Get Registers synchronization flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RSF LL_RTC_IsActiveFlag_RS

LL_RTC_ClearFlag_RS

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_RS (RTC_TypeDef * RTCx)

Function description

Clear Registers synchronization flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR RSF LL_RTC_ClearFlag_RS

LL_RTC_IsActiveFlag_INITS

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_INITS (RTC_TypeDef * RTCx)

Function description

Get Initialization status flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR INITS LL_RTC_IsActiveFlag_INITS

LL_RTC_IsActiveFlag_SHP

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_SHP (RTC_TypeDef * RTCx)

Function description

Get Shift operation pending flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR SHPF LL_RTC_IsActiveFlag_SHP

LL_RTC_IsActiveFlag_WUTW

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_WUTW (RTC_TypeDef * RTCx)

Function description

Get Wakeup timer write flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR WUTWF LL_RTC_IsActiveFlag_WUTW

LL_RTC_IsActiveFlag_ALRBW

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRBW (RTC_TypeDef * RTCx)

Function description

Get Alarm B write flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ALRBWF LL_RTC_IsActiveFlag_ALRBW

LL_RTC_IsActiveFlag_ALRAW

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRAW (RTC_TypeDef * RTCx)

Function description

Get Alarm A write flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ALRAWF LL_RTC_IsActiveFlag_ALRAW

LL_RTC_EnableIT_TS

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TS (RTC_TypeDef * RTCx)

Function description

Enable Time-stamp interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR TSIE LL_RTC_EnableIT_TS

LL_RTC_DisableIT_TS

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TS (RTC_TypeDef * RTCx)

Function description

Disable Time-stamp interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR TSIE LL_RTC_DisableIT_TS

LL_RTC_EnableIT_WUT

Function name

__STATIC_INLINE void LL_RTC_EnableIT_WUT (RTC_TypeDef * RTCx)

Function description

Enable Wakeup timer interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR WUTIE LL_RTC_EnableIT_WUT

LL_RTC_DisableIT_WUT

Function name

__STATIC_INLINE void LL_RTC_DisableIT_WUT (RTC_TypeDef * RTCx)

Function description

Disable Wakeup timer interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR WUTIE LL_RTC_DisableIT_WUT

LL_RTC_EnableIT_ALRB

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ALRB (RTC_TypeDef * RTCx)

Function description

Enable Alarm B interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRBIE LL_RTC_EnableIT_ALRB

LL_RTC_DisableIT_ALRB

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ALRB (RTC_TypeDef * RTCx)

Function description

Disable Alarm B interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRBIE LL_RTC_DisableIT_ALRB

LL_RTC_EnableIT_ALRA

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ALRA (RTC_TypeDef * RTCx)

Function description

Enable Alarm A interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRAIE LL_RTC_EnableIT_ALRA

LL_RTC_DisableIT_ALRA

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ALRA (RTC_TypeDef * RTCx)

Function description

Disable Alarm A interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRAIE LL_RTC_DisableIT_ALRA

LL_RTC_EnableIT_TAMP3

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TAMP3 (RTC_TypeDef * RTCx)

Function description

Enable Tamper 3 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMPCR TAMP3IE LL_RTC_EnableIT_TAMP3

LL_RTC_DisableIT_TAMP3

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TAMP3 (RTC_TypeDef * RTCx)

Function description

Disable Tamper 3 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMPCR TAMP3IE LL_RTC_DisableIT_TAMP3

LL_RTC_EnableIT_TAMP2

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TAMP2 (RTC_TypeDef * RTCx)

Function description

Enable Tamper 2 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMPCR TAMP2IE LL_RTC_EnableIT_TAMP2

LL_RTC_DisableIT_TAMP2

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TAMP2 (RTC_TypeDef * RTCx)

Function description

Disable Tamper 2 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMPCR TAMP2IE LL_RTC_DisableIT_TAMP2

LL_RTC_EnableIT_TAMP1

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TAMP1 (RTC_TypeDef * RTCx)

Function description

Enable Tamper 1 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMPCR TAMP1IE LL_RTC_EnableIT_TAMP1

LL_RTC_DisableIT_TAMP1

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TAMP1 (RTC_TypeDef * RTCx)

Function description

Disable Tamper 1 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMPCR TAMP1IE LL_RTC_DisableIT_TAMP1

LL_RTC_EnableIT_TAMP

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TAMP (RTC_TypeDef * RTCx)

Function description

Enable all Tamper Interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMPCR TAMPIE LL_RTC_EnableIT_TAMP

LL_RTC_DisableIT_TAMP

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TAMP (RTC_TypeDef * RTCx)

Function description

Disable all Tamper Interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMPCR TAMPIE LL_RTC_DisableIT_TAMP

LL_RTC_IsEnabledIT_TS

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TS (RTC_TypeDef * RTCx)

Function description

Check if Time-stamp interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR TSIE LL_RTC_IsEnabledIT_TS

LL_RTC_IsEnabledIT_WUT

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_WUT (RTC_TypeDef * RTCx)

Function description

Check if Wakeup timer interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR WUTIE LL_RTC_IsEnabledIT_WUT

LL_RTC_IsEnabledIT_ALRB

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ALRB (RTC_TypeDef * RTCx)

Function description

Check if Alarm B interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR ALRBIE LL_RTC_IsEnabledIT_ALRB

LL_RTC_IsEnabledIT_ALRA

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ALRA (RTC_TypeDef * RTCx)

Function description

Check if Alarm A interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR ALRAIE LL_RTC_IsEnabledIT_ALRA

LL_RTC_IsEnabledIT_TAMP3

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP3 (RTC_TypeDef * RTCx)

Function description

Check if Tamper 3 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMPCR TAMP3IE LL_RTC_IsEnabledIT_TAMP3

LL_RTC_IsEnabledIT_TAMP2

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP2 (RTC_TypeDef * RTCx)

Function description

Check if Tamper 2 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMPCR TAMP2IE LL_RTC_IsEnabledIT_TAMP2

LL_RTC_IsEnabledIT_TAMP1

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP1 (RTC_TypeDef * RTCx)

Function description

Check if Tamper 1 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMPCR TAMP1IE LL_RTC_IsEnabledIT_TAMP1

LL_RTC_IsEnabledIT_TAMP

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP (RTC_TypeDef * RTCx)

Function description

Check if all the TAMPER interrupts are enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMPCR TAMPIE LL_RTC_IsEnabledIT_TAMP

LL_RTC_DeInit

Function name

ErrorStatus LL_RTC_DeInit (RTC_TypeDef * RTCx)

Function description

De-Initializes the RTC registers to their default reset values.

Parameters

- **RTCx:** RTC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC registers are de-initialized
  - ERROR: RTC registers are not de-initialized

Notes

- This function does not reset the RTC Clock source and RTC Backup Data registers.

LL_RTC_Init

Function name

ErrorStatus LL_RTC_Init (RTC_TypeDef * RTCx, LL_RTC_InitTypeDef * RTC_InitStruct)

Function description

Initializes the RTC registers according to the specified parameters in RTC_InitStruct.

Parameters

- **RTCx:** RTC Instance
- **RTC_InitStruct:** pointer to a LL_RTC_InitTypeDef structure that contains the configuration information for the RTC peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC registers are initialized
  - ERROR: RTC registers are not initialized

Notes

- The RTC Prescaler register is write protected and can be written in initialization mode only.

LL_RTC_StructInit

Function name

void LL_RTC_StructInit (LL_RTC_InitTypeDef * RTC_InitStruct)

Function description

Set each LL_RTC_InitTypeDef field to default value.

Parameters

- **RTC_InitStruct:** pointer to a LL_RTC_InitTypeDef structure which will be initialized.

Return values

- **None:**

LL_RTC_TIME_Init

Function name

ErrorStatus LL_RTC_TIME_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_TimeTypeDef * RTC_TimeStruct)

Function description

Set the RTC current time.

Parameters

- **RTCx:** RTC Instance
- **RTC_Format:** This parameter can be one of the following values:
  - LL_RTC_FORMAT_BIN
  - LL_RTC_FORMAT_BCD
- **RTC_TimeStruct:** pointer to a RTC_TimeTypeDef structure that contains the time configuration information for the RTC.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC Time register is configured
  - ERROR: RTC Time register is not configured

LL_RTC_TIME_StructInit

Function name

void LL_RTC_TIME_StructInit (LL_RTC_TimeTypeDef * RTC_TimeStruct)

Function description

Set each LL_RTC_TimeTypeDef field to default value (Time = 00h:00min:00sec).

Parameters

- **RTC_TimeStruct:** pointer to a LL_RTC_TimeTypeDef structure which will be initialized.

Return values

- **None:**

LL_RTC_DATE_Init

Function name

ErrorStatus LL_RTC_DATE_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_DateTypeDef * RTC_DateStruct)

Function description

Set the RTC current date.

Parameters

- **RTCx:** RTC Instance
- **RTC_Format:** This parameter can be one of the following values:
  - LL_RTC_FORMAT_BIN
  - LL_RTC_FORMAT_BCD
- **RTC_DateStruct:** pointer to a RTC_DateTypeDef structure that contains the date configuration information for the RTC.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC Day register is configured
  - ERROR: RTC Day register is not configured

LL_RTC_DATE_StructInit

Function name

void LL_RTC_DATE_StructInit (LL_RTC_DateTypeDef * RTC_DateStruct)

Function description

Set each LL_RTC_DateTypeDef field to default value (date = Monday, January 01 xx00)

Parameters

- **RTC_DateStruct:** pointer to a LL_RTC_DateTypeDef structure which will be initialized.

Return values

- **None:**

LL_RTC_ALMA_Init

Function name

ErrorStatus LL_RTC_ALMA_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_AlarmTypeDef * RTC_AlarmStruct)

Function description

Set the RTC Alarm A.

Parameters

- **RTCx:** RTC Instance
- **RTC_Format:** This parameter can be one of the following values:
  - LL_RTC_FORMAT_BIN
  - LL_RTC_FORMAT_BCD
- **RTC_AlarmStruct:** pointer to a LL_RTC_AlarmTypeDef structure that contains the alarm configuration parameters.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ALARMA registers are configured
  - ERROR: ALARMA registers are not configured

Notes

- The Alarm register can only be written when the corresponding Alarm is disabled (Use LL_RTC_ALMA_Disable function).

LL_RTC_ALMB_Init

Function name

ErrorStatus LL_RTC_ALMB_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_AlarmTypeDef * RTC_AlarmStruct)

Function description

Set the RTC Alarm B.

Parameters

- **RTCx:** RTC Instance
- **RTC_Format:** This parameter can be one of the following values:
  - LL_RTC_FORMAT_BIN
  - LL_RTC_FORMAT_BCD
- **RTC_AlarmStruct:** pointer to a LL_RTC_AlarmTypeDef structure that contains the alarm configuration parameters.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ALARMB registers are configured
  - ERROR: ALARMB registers are not configured

Notes

- The Alarm register can only be written when the corresponding Alarm is disabled (LL_RTC_ALMB_Disable function).

LL_RTC_ALMA_StructInit

Function name

void LL_RTC_ALMA_StructInit (LL_RTC_AlarmTypeDef * RTC_AlarmStruct)

Function description

Set each LL_RTC_AlarmTypeDef of ALARMA field to default value (Time = 00h:00mn:00sec / Day = 1st day of the month/Mask = all fields are masked).

Parameters

- **RTC_AlarmStruct:** pointer to a LL_RTC_AlarmTypeDef structure which will be initialized.

Return values

- **None:**

LL_RTC_ALMB_StructInit

Function name

void LL_RTC_ALMB_StructInit (LL_RTC_AlarmTypeDef * RTC_AlarmStruct)

Function description

Set each LL_RTC_AlarmTypeDef of ALARMA field to default value (Time = 00h:00mn:00sec / Day = 1st day of the month/Mask = all fields are masked).

Parameters

- **RTC_AlarmStruct:** pointer to a LL_RTC_AlarmTypeDef structure which will be initialized.

Return values

- **None:**

LL_RTC_EnterInitMode

Function name

ErrorStatus LL_RTC_EnterInitMode (RTC_TypeDef * RTCx)

Function description

Enters the RTC Initialization mode.

Parameters

- **RTCx:** RTC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC is in Init mode
  - ERROR: RTC is not in Init mode

Notes

- The RTC Initialization mode is write protected, use the LL_RTC_DisableWriteProtection before calling this function.

LL_RTC_ExitInitMode

Function name

ErrorStatus LL_RTC_ExitInitMode (RTC_TypeDef * RTCx)

Function description

Exit the RTC Initialization mode.

Parameters

- **RTCx:** RTC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC exited from in Init mode
  - ERROR: Not applicable

Notes

- When the initialization sequence is complete, the calendar restarts counting after 4 RTCCLK cycles.
- The RTC Initialization mode is write protected, use the LL_RTC_DisableWriteProtection before calling this function.

LL_RTC_WaitForSynchro

Function name

ErrorStatus LL_RTC_WaitForSynchro (RTC_TypeDef * RTCx)

Function description

Waits until the RTC Time and Day registers (RTC_TR and RTC_DR) are synchronized with RTC APB clock.

Parameters

- **RTCx:** RTC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC registers are synchronised
  - ERROR: RTC registers are not synchronised

Notes

- The RTC Resynchronization mode is write protected, use the LL_RTC_DisableWriteProtection before calling this function.
- To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers.

122.3 RTC Firmware driver defines

The following section lists the various define and macros of the module.

122.3.1 RTC

RTC

ALARM OUTPUT

LL_RTC_ALARMOUT_DISABLE

Output disabled

LL_RTC_ALARMOUT_ALMA

Alarm A output enabled

LL_RTC_ALARMOUT_ALMB

Alarm B output enabled

LL_RTC_ALARMOUT_WAKEUP

Wakeup output enabled

ALARM OUTPUT TYPE

LL_RTC_ALARM_OUTPUTTYPE_OPENDRAIN

RTC_ALARM, when mapped on PC13, is open-drain output

LL_RTC_ALARM_OUTPUTTYPE_PUSHPULL

RTC_ALARM, when mapped on PC13, is push-pull output

ALARMA MASK

LL_RTC_ALMA_MASK_NONE

No masks applied on Alarm A

LL_RTC_ALMA_MASK_DATEWEEKDAY

Date/day do not care in Alarm A comparison

LL_RTC_ALMA_MASK_HOURS

Hours do not care in Alarm A comparison

LL_RTC_ALMA_MASK_MINUTES

Minutes do not care in Alarm A comparison

LL_RTC_ALMA_MASK_SECONDS

Seconds do not care in Alarm A comparison

LL_RTC_ALMA_MASK_ALL

Masks all

ALARMA TIME FORMAT

LL_RTC_ALMA_TIME_FORMAT_AM

AM or 24-hour format

LL_RTC_ALMA_TIME_FORMAT_PM

PM

RTC Alarm A Date WeekDay

LL_RTC_ALMA_DATEWEEKDAYSEL_DATE

Alarm A Date is selected

LL_RTC_ALMA_DATEWEEKDAYSEL_WEEKDAY

Alarm A WeekDay is selected

ALARMB MASK

LL_RTC_ALMB_MASK_NONE

No masks applied on Alarm B

LL_RTC_ALMB_MASK_DATEWEEKDAY

Date/day do not care in Alarm B comparison

LL_RTC_ALMB_MASK_HOURS

Hours do not care in Alarm B comparison

LL_RTC_ALMB_MASK_MINUTES

Minutes do not care in Alarm B comparison

LL_RTC_ALMB_MASK_SECONDS

Seconds do not care in Alarm B comparison

LL_RTC_ALMB_MASK_ALL

Masks all

ALARMB TIME FORMAT

LL_RTC_ALMB_TIME_FORMAT_AM

AM or 24-hour format

LL_RTC_ALMB_TIME_FORMAT_PM

PM

RTC Alarm B Date WeekDay

LL_RTC_ALMB_DATEWEEKDAYSEL_DATE

Alarm B Date is selected

LL_RTC_ALMB_DATEWEEKDAYSEL_WEEKDAY

Alarm B WeekDay is selected

BACKUP

LL_RTC_BKP_DR0

LL_RTC_BKP_DR1

LL_RTC_BKP_DR2

LL_RTC_BKP_DR3

LL_RTC_BKP_DR4

LL_RTC_BKP_DR5

LL_RTC_BKP_DR6

LL_RTC_BKP_DR7

LL_RTC_BKP_DR8

LL_RTC_BKP_DR9

LL_RTC_BKP_DR10

LL_RTC_BKP_DR11

LL_RTC_BKP_DR12

LL_RTC_BKP_DR13

LL_RTC_BKP_DR14

LL_RTC_BKP_DR15

LL_RTC_BKP_DR16

LL_RTC_BKP_DR17

LL_RTC_BKP_DR18

LL_RTC_BKP_DR19

LL_RTC_BKP_DR20

LL_RTC_BKP_DR21

LL_RTC_BKP_DR22

LL_RTC_BKP_DR23

LL_RTC_BKP_DR24

LL_RTC_BKP_DR25

LL_RTC_BKP_DR26

LL_RTC_BKP_DR27

LL_RTC_BKP_DR28

LL_RTC_BKP_DR29

LL_RTC_BKP_DR30

LL_RTC_BKP_DR31

Calibration pulse insertion

LL_RTC_CALIB_INSERTPULSE_NONE

No RTCCLK pulses are added

LL_RTC_CALIB_INSERTPULSE_SET

One RTCCLK pulse is effectively inserted every 2exp11 pulses (frequency increased by 488.5 ppm)

Calibration output

LL_RTC_CALIB_OUTPUT_NONE

Calibration output disabled

LL_RTC_CALIB_OUTPUT_1HZ

Calibration output is 1 Hz

LL_RTC_CALIB_OUTPUT_512HZ

Calibration output is 512 Hz

Calibration period

LL_RTC_CALIB_PERIOD_32SEC

Use a 32-second calibration cycle period

LL_RTC_CALIB_PERIOD_16SEC

Use a 16-second calibration cycle period

LL_RTC_CALIB_PERIOD_8SEC

Use a 8-second calibration cycle period

FORMAT

LL_RTC_FORMAT_BIN

Binary data format

LL_RTC_FORMAT_BCD

BCD data format

Get Flags Defines

LL_RTC_ISR_ITSF

LL_RTC_ISR_RECALPF

LL_RTC_ISR_TAMP3F

LL_RTC_ISR_TAMP2F

LL_RTC_ISR_TAMP1F

LL_RTC_ISR_TSOVF

LL_RTC_ISR_TSF

LL_RTC_ISR_WUTF

LL_RTC_ISR_ALRBF

LL_RTC_ISR_ALRAF

LL_RTC_ISR_INITF

LL_RTC_ISR_RSF

LL_RTC_ISR_INITS

LL_RTC_ISR_SHPF

LL_RTC_ISR_WUTWF

LL_RTC_ISR_ALRBWF

LL_RTC_ISR_ALRAWF

HOUR FORMAT

LL_RTC_HOURFORMAT_24HOUR

24 hour/day format

LL_RTC_HOURFORMAT_AMPM

AM/PM hour format

IT Defines

LL_RTC_CR_TSIE

LL_RTC_CR_WUTIE

LL_RTC_CR_ALRBIE

LL_RTC_CR_ALRAIE

LL_RTC_TAMPCR_TAMP3IE

LL_RTC_TAMPCR_TAMP2IE

LL_RTC_TAMPCR_TAMP1IE

LL_RTC_TAMPCR_TAMPIE

MONTH

LL_RTC_MONTH_JANUARY

January

LL_RTC_MONTH_FEBRUARY

February

LL_RTC_MONTH_MARCH

March

LL_RTC_MONTH_APRIL

April

LL_RTC_MONTH_MAY

May

LL_RTC_MONTH_JUNE

June

LL_RTC_MONTH_JULY

July

LL_RTC_MONTH_AUGUST

August

LL_RTC_MONTH_SEPTEMBER

September

LL_RTC_MONTH_OCTOBER

October

LL_RTC_MONTH_NOVEMBER

November

LL_RTC_MONTH_DECEMBER

December

OUTPUT POLARITY PIN

LL_RTC_OUTPUTPOLARITY_PIN_HIGH

Pin is high when ALRAF/ALRBF/WUTF is asserted (depending on OSEL)

LL_RTC_OUTPUTPOLARITY_PIN_LOW

Pin is low when ALRAF/ALRBF/WUTF is asserted (depending on OSEL)

SHIFT SECOND

LL_RTC_SHIFT_SECOND_DELAY

LL_RTC_SHIFT_SECOND_ADVANCE

TAMPER

LL_RTC_TAMPER_1

RTC_TAMP1 input detection

LL_RTC_TAMPER_2

RTC_TAMP2 input detection

LL_RTC_TAMPER_3

RTC_TAMP3 input detection

TAMPER ACTIVE LEVEL

LL_RTC_TAMPER_ACTIVELEVEL_TAMP1

RTC_TAMP1 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

LL_RTC_TAMPER_ACTIVELEVEL_TAMP2

RTC_TAMP2 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

LL_RTC_TAMPER_ACTIVELEVEL_TAMP3

RTC_TAMP3 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

TAMPER DURATION

LL_RTC_TAMPER_DURATION_1RTCCLK

Tamper pins are pre-charged before sampling during 1 RTCCLK cycle

LL_RTC_TAMPER_DURATION_2RTCCLK

Tamper pins are pre-charged before sampling during 2 RTCCLK cycles

LL_RTC_TAMPER_DURATION_4RTCCLK

Tamper pins are pre-charged before sampling during 4 RTCCLK cycles

LL_RTC_TAMPER_DURATION_8RTCCLK

Tamper pins are pre-charged before sampling during 8 RTCCLK cycles

TAMPER FILTER

LL_RTC_TAMPER_FILTER_DISABLE

Tamper filter is disabled

LL_RTC_TAMPER_FILTER_2SAMPLE

Tamper is activated after 2 consecutive samples at the active level

LL_RTC_TAMPER_FILTER_4SAMPLE

Tamper is activated after 4 consecutive samples at the active level

LL_RTC_TAMPER_FILTER_8SAMPLE

Tamper is activated after 8 consecutive samples at the active level.

TAMPER MASK

LL_RTC_TAMPER_MASK_TAMPER1

Tamper 1 event generates a trigger event. TAMP1F is masked and internally cleared by hardware.The backup registers are not erased

LL_RTC_TAMPER_MASK_TAMPER2

Tamper 2 event generates a trigger event. TAMP2F is masked and internally cleared by hardware. The backup registers are not erased.

LL_RTC_TAMPER_MASK_TAMPER3

Tamper 3 event generates a trigger event. TAMP3F is masked and internally cleared by hardware. The backup registers are not erased

TAMPER NO ERASE

LL_RTC_TAMPER_NOERASE_TAMPER1

Tamper 1 event does not erase the backup registers.

LL_RTC_TAMPER_NOERASE_TAMPER2

Tamper 2 event does not erase the backup registers.

LL_RTC_TAMPER_NOERASE_TAMPER3

Tamper 3 event does not erase the backup registers.

TAMPER SAMPLING FREQUENCY DIVIDER

LL_RTC_TAMPER_SAMPLFREQDIV_32768

Each of the tamper inputs are sampled with a frequency = RTCCLK / 32768

LL_RTC_TAMPER_SAMPLFREQDIV_16384

Each of the tamper inputs are sampled with a frequency = RTCCLK / 16384

LL_RTC_TAMPER_SAMPLFREQDIV_8192

Each of the tamper inputs are sampled with a frequency = RTCCLK / 8192

LL_RTC_TAMPER_SAMPLFREQDIV_4096

Each of the tamper inputs are sampled with a frequency = RTCCLK / 4096

LL_RTC_TAMPER_SAMPLFREQDIV_2048

Each of the tamper inputs are sampled with a frequency = RTCCLK / 2048

LL_RTC_TAMPER_SAMPLFREQDIV_1024

Each of the tamper inputs are sampled with a frequency = RTCCLK / 1024

LL_RTC_TAMPER_SAMPLFREQDIV_512

Each of the tamper inputs are sampled with a frequency = RTCCLK / 512

LL_RTC_TAMPER_SAMPLFREQDIV_256

Each of the tamper inputs are sampled with a frequency = RTCCLK / 256

TIMESTAMP EDGE

LL_RTC_TIMESTAMP_EDGE_RISING

RTC_TS input rising edge generates a time-stamp event

LL_RTC_TIMESTAMP_EDGE_FALLING

RTC_TS input falling edge generates a time-stamp even

TIME FORMAT

LL_RTC_TIME_FORMAT_AM_OR_24

AM or 24-hour format

LL_RTC_TIME_FORMAT_PM

PM

TIMESTAMP TIME FORMAT

LL_RTC_TS_TIME_FORMAT_AM

AM or 24-hour format

LL_RTC_TS_TIME_FORMAT_PM

PM

WAKEUP CLOCK DIV

LL_RTC_WAKEUPCLOCK_DIV_16

RTC/16 clock is selected

LL_RTC_WAKEUPCLOCK_DIV_8

RTC/8 clock is selected

LL_RTC_WAKEUPCLOCK_DIV_4

RTC/4 clock is selected

LL_RTC_WAKEUPCLOCK_DIV_2

RTC/2 clock is selected

LL_RTC_WAKEUPCLOCK_CKSPRE

ck_spre (usually 1 Hz) clock is selected

LL_RTC_WAKEUPCLOCK_CKSPRE_WUT

ck_spre (usually 1 Hz) clock is selected and 2exp16 is added to the WUT counter value

WEEK DAY

LL_RTC_WEEKDAY_MONDAY

Monday

LL_RTC_WEEKDAY_TUESDAY

Tuesday

LL_RTC_WEEKDAY_WEDNESDAY

Wednesday

LL_RTC_WEEKDAY_THURSDAY

Thrusday

LL_RTC_WEEKDAY_FRIDAY

Friday

LL_RTC_WEEKDAY_SATURDAY

Saturday

LL_RTC_WEEKDAY_SUNDAY

Sunday

Convert helper Macros

__LL_RTC_CONVERT_BIN2BCD

Description:

- Helper macro to convert a value from 2 digit decimal format to BCD format.

Parameters:

- __VALUE__: Byte to be converted

Return value:

- Converted: byte

__LL_RTC_CONVERT_BCD2BIN

Description:

- Helper macro to convert a value from BCD format to 2 digit decimal format.

Parameters:

- __VALUE__: BCD value to be converted

Return value:

- Converted: byte

Date helper Macros

__LL_RTC_GET_WEEKDAY

Description:

- Helper macro to retrieve weekday.

Parameters:

- __RTC_DATE__: Date returned by

Return value:

- Returned: value can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

__LL_RTC_GET_YEAR

Description:

- Helper macro to retrieve Year in BCD format.

Parameters:

- __RTC_DATE__: Value returned by

Return value:

- Year: in BCD format (0x00 . . . 0x99)

__LL_RTC_GET_MONTH

Description:

- Helper macro to retrieve Month in BCD format.

Parameters:

- __RTC_DATE__: Value returned by

Return value:

- Returned: value can be one of the following values:
  - LL_RTC_MONTH_JANUARY
  - LL_RTC_MONTH_FEBRUARY
  - LL_RTC_MONTH_MARCH
  - LL_RTC_MONTH_APRIL
  - LL_RTC_MONTH_MAY
  - LL_RTC_MONTH_JUNE
  - LL_RTC_MONTH_JULY
  - LL_RTC_MONTH_AUGUST
  - LL_RTC_MONTH_SEPTEMBER
  - LL_RTC_MONTH_OCTOBER
  - LL_RTC_MONTH_NOVEMBER
  - LL_RTC_MONTH_DECEMBER

__LL_RTC_GET_DAY

Description:

- Helper macro to retrieve Day in BCD format.

Parameters:

- __RTC_DATE__: Value returned by

Return value:

- Day: in BCD format (0x01 . . . 0x31)

Time helper Macros

__LL_RTC_GET_HOUR

Description:

- Helper macro to retrieve hour in BCD format.

Parameters:

- __RTC_TIME__: RTC time returned by

Return value:

- Hours: in BCD format (0x01. . .0x12 or between Min_Data=0x00 and Max_Data=0x23)

__LL_RTC_GET_MINUTE

Description:

- Helper macro to retrieve minute in BCD format.

Parameters:

- __RTC_TIME__: RTC time returned by

Return value:

- Minutes: in BCD format (0x00. . .0x59)

__LL_RTC_GET_SECOND

Description:

- Helper macro to retrieve second in BCD format.

Parameters:

- __RTC_TIME__: RTC time returned by

Return value:

- Seconds: in format (0x00. . .0x59)

Common Write and read registers Macros

LL_RTC_WriteReg

Description:

- Write a value in RTC register.

Parameters:

- __INSTANCE__: RTC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_RTC_ReadReg

Description:

- Read a value in RTC register.

Parameters:

- __INSTANCE__: RTC Instance
- __REG__: Register to be read

Return value:

- Register: value

123 LL SPI Generic Driver

123.1 SPI Firmware driver registers structures

123.1.1 LL_SPI_InitTypeDef

LL_SPI_InitTypeDef is defined in the stm32h7xx_ll_spi.h

Data Fields

- ***uint32_t TransferDirection***
- ***uint32_t Mode***
- ***uint32_t DataWidth***
- ***uint32_t ClockPolarity***
- ***uint32_t ClockPhase***
- ***uint32_t NSS***
- ***uint32_t BaudRate***
- ***uint32_t BitOrder***
- ***uint32_t CRCCalculation***
- ***uint32_t CRCPoly***

Field Documentation

- ***uint32_t LL_SPI_InitTypeDef::TransferDirection***

  Specifies the SPI unidirectional or bidirectional data mode. This parameter can be a value of ***SPI_LL_EC_TRANSFER_MODE***.This feature can be modified afterwards using unitary function **LL_SPI_SetTransferDirection()**.
- ***uint32_t LL_SPI_InitTypeDef::Mode***

  Specifies the SPI mode (Master/Slave). This parameter can be a value of ***SPI_LL_EC_MODE***.This feature can be modified afterwards using unitary function **LL_SPI_SetMode()**.
- ***uint32_t LL_SPI_InitTypeDef::DataWidth***

  Specifies the SPI data width. This parameter can be a value of ***SPI_LL_EC_DATAWIDTH***.This feature can be modified afterwards using unitary function **LL_SPI_SetDataWidth()**.
- ***uint32_t LL_SPI_InitTypeDef::ClockPolarity***

  Specifies the serial clock steady state. This parameter can be a value of ***SPI_LL_EC_POLARITY***.This feature can be modified afterwards using unitary function **LL_SPI_SetClockPolarity()**.
- ***uint32_t LL_SPI_InitTypeDef::ClockPhase***

  Specifies the clock active edge for the bit capture. This parameter can be a value of ***SPI_LL_EC_PHASE***.This feature can be modified afterwards using unitary function **LL_SPI_SetClockPhase()**.
- ***uint32_t LL_SPI_InitTypeDef::NSS***

  Specifies whether the NSS signal is managed by hardware (NSS pin) or by software using the SSI bit.This parameter can be a value of ***SPI_LL_EC_NSS_MODE***.This feature can be modified afterwards using unitary function **LL_SPI_SetNSSMode()**.
- ***uint32_t LL_SPI_InitTypeDef::BaudRate***

  Specifies the BaudRate prescaler value which will be used to configure the transmit and receive SCK clock. This parameter can be a value of ***SPI_LL_EC_BAUDRATEPRESCALER***.

  Note:

  – The communication clock is derived from the master clock. The slave clock does not need to be set.

  This feature can be modified afterwards using unitary function **LL_SPI_SetBaudRatePrescaler()**.

- ***uint32_t LL_SPI_InitTypeDef::BitOrder***

  Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of ***SPI_LL_EC_BIT_ORDER***.This feature can be modified afterwards using unitary function **LL_SPI_SetTransferBitOrder()**.
- ***uint32_t LL_SPI_InitTypeDef::CRCCalculation***

  Specifies if the CRC calculation is enabled or not. This parameter can be a value of ***SPI_LL_EC_CRC_CALCULATION***.This feature can be modified afterwards using unitary functions **LL_SPI_EnableCRC()** and **LL_SPI_DisableCRC()**.
- ***uint32_t LL_SPI_InitTypeDef::CRCPoly***

  Specifies the polynomial used for the CRC calculation. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFFFFFFFF.This feature can be modified afterwards using unitary function **LL_SPI_SetCRCPolynomial()**.

123.1.2 LL_I2S_InitTypeDef

LL_I2S_InitTypeDef is defined in the stm32h7xx_ll_spi.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t Standard***
- ***uint32_t DataFormat***
- ***uint32_t MCLKOutput***
- ***uint32_t AudioFreq***
- ***uint32_t ClockPolarity***

Field Documentation

- ***uint32_t LL_I2S_InitTypeDef::Mode***

  Specifies the I2S operating mode. This parameter can be a value of ***I2S_LL_EC_MODE***This feature can be modified afterwards using unitary function **LL_I2S_SetTransferMode()**.
- ***uint32_t LL_I2S_InitTypeDef::Standard***

  Specifies the standard used for the I2S communication. This parameter can be a value of ***I2S_LL_EC_STANDARD***This feature can be modified afterwards using unitary function **LL_I2S_SetStandard()**.
- ***uint32_t LL_I2S_InitTypeDef::DataFormat***

  Specifies the data format for the I2S communication. This parameter can be a value of ***I2S_LL_EC_DATA_FORMAT***This feature can be modified afterwards using unitary function **LL_I2S_SetDataFormat()**.
- ***uint32_t LL_I2S_InitTypeDef::MCLKOutput***

  Specifies whether the I2S MCLK output is enabled or not. This parameter can be a value of ***I2S_LL_EC_MCLK_OUTPUT***This feature can be modified afterwards using unitary functions **LL_I2S_EnableMasterClock()** or **LL_I2S_DisableMasterClock**.
- ***uint32_t LL_I2S_InitTypeDef::AudioFreq***

  Specifies the frequency selected for the I2S communication. This parameter can be a value of ***I2S_LL_EC_AUDIO_FREQ***Audio Frequency can be modified afterwards using Reference manual formulas to calculate Prescaler Linear, Parity and unitary functions **LL_I2S_SetPrescalerLinear()** and **LL_I2S_SetPrescalerParity()** to set it.
- ***uint32_t LL_I2S_InitTypeDef::ClockPolarity***

  Specifies the idle state of the I2S clock. This parameter can be a value of ***I2S_LL_EC_POLARITY***This feature can be modified afterwards using unitary function **LL_I2S_SetClockPolarity()**.

123.2 SPI Firmware driver API description

The following section lists the various functions of the SPI library.

123.2.1 Detailed description of functions

LL_SPI_Enable

Function name

__STATIC_INLINE void LL_SPI_Enable (SPI_TypeDef * SPIx)

Function description

Enable SPI peripheral.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 SPE LL_SPI_Enable

LL_SPI_Disable

Function name

__STATIC_INLINE void LL_SPI_Disable (SPI_TypeDef * SPIx)

Function description

Disable SPI peripheral.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- When disabling the SPI, follow the procedure described in the Reference Manual.

Reference Manual to LL API cross reference:

- CR1 SPE LL_SPI_Disable

LL_SPI_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabled (SPI_TypeDef * SPIx)

Function description

Check if SPI peripheral is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CR1 SPE LL_SPI_IsEnabled

LL_SPI_EnableIOSwap

Function name

__STATIC_INLINE void LL_SPI_EnableIOSwap (SPI_TypeDef * SPIx)

Function description

Swap the MOSI and MISO pin.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG2 IOSWP LL_SPI_EnableIOSwap

LL_SPI_DisableIOSwap

Function name

__STATIC_INLINE void LL_SPI_DisableIOSwap (SPI_TypeDef * SPIx)

Function description

Restore default function for MOSI and MISO pin.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG2 IOSWP LL_SPI_DisableIOSwap

LL_SPI_IsEnabledIOSwap

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIOSwap (SPI_TypeDef * SPIx)

Function description

Check if MOSI and MISO pin are swapped.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG2 IOSWP LL_SPI_IsEnabledIOSwap

LL_SPI_EnableGPIOControl

Function name

__STATIC_INLINE void LL_SPI_EnableGPIOControl (SPI_TypeDef * SPIx)

Function description

Enable GPIO control.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG2 AFCNTR LL_SPI_EnableGPIOControl

LL_SPI_DisableGPIOControl

Function name

__STATIC_INLINE void LL_SPI_DisableGPIOControl (SPI_TypeDef * SPIx)

Function description

Disable GPIO control.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG2 AFCNTR LL_SPI_DisableGPIOControl

LL_SPI_IsEnabledGPIOControl

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledGPIOControl (SPI_TypeDef * SPIx)

Function description

Check if GPIO control is active.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG2 AFCNTR LL_SPI_IsEnabledGPIOControl

LL_SPI_SetMode

Function name

__STATIC_INLINE void LL_SPI_SetMode (SPI_TypeDef * SPIx, uint32_t Mode)

Function description

Set SPI Mode to Master or Slave.

Parameters

- **SPIx:** SPI Instance
- **Mode:** This parameter can be one of the following values:
  - LL_SPI_MODE_MASTER
  - LL_SPI_MODE_SLAVE

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG2 MASTER LL_SPI_SetMode

LL_SPI_GetMode

Function name

__STATIC_INLINE uint32_t LL_SPI_GetMode (SPI_TypeDef * SPIx)

Function description

Get SPI Mode (Master or Slave)

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_MODE_MASTER
  - LL_SPI_MODE_SLAVE

Reference Manual to LL API cross reference:

- CFG2 MASTER LL_SPI_GetMode

LL_SPI_SetMasterSSIdleness

Function name

__STATIC_INLINE void LL_SPI_SetMasterSSIdleness (SPI_TypeDef * SPIx, uint32_t MasterSSIdleness)

Function description

Configure the Idleness applied by master between active edge of SS and first send data.

Parameters

- **SPIx:** SPI Instance
- **MasterSSIdleness:** This parameter can be one of the following values:
  - LL_SPI_SS_IDLENESS_00CYCLE
  - LL_SPI_SS_IDLENESS_01CYCLE
  - LL_SPI_SS_IDLENESS_02CYCLE
  - LL_SPI_SS_IDLENESS_03CYCLE
  - LL_SPI_SS_IDLENESS_04CYCLE
  - LL_SPI_SS_IDLENESS_05CYCLE
  - LL_SPI_SS_IDLENESS_06CYCLE
  - LL_SPI_SS_IDLENESS_07CYCLE
  - LL_SPI_SS_IDLENESS_08CYCLE
  - LL_SPI_SS_IDLENESS_09CYCLE
  - LL_SPI_SS_IDLENESS_10CYCLE
  - LL_SPI_SS_IDLENESS_11CYCLE
  - LL_SPI_SS_IDLENESS_12CYCLE
  - LL_SPI_SS_IDLENESS_13CYCLE
  - LL_SPI_SS_IDLENESS_14CYCLE
  - LL_SPI_SS_IDLENESS_15CYCLE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 MSSI LL_SPI_SetMasterSSIdleness

LL_SPI_GetMasterSSIdleness

Function name

__STATIC_INLINE uint32_t LL_SPI_GetMasterSSIdleness (SPI_TypeDef * SPIx)

Function description

Get the configured Idleness applied by master.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_SS_IDLENESS_00CYCLE
  - LL_SPI_SS_IDLENESS_01CYCLE
  - LL_SPI_SS_IDLENESS_02CYCLE
  - LL_SPI_SS_IDLENESS_03CYCLE
  - LL_SPI_SS_IDLENESS_04CYCLE
  - LL_SPI_SS_IDLENESS_05CYCLE
  - LL_SPI_SS_IDLENESS_06CYCLE
  - LL_SPI_SS_IDLENESS_07CYCLE
  - LL_SPI_SS_IDLENESS_08CYCLE
  - LL_SPI_SS_IDLENESS_09CYCLE
  - LL_SPI_SS_IDLENESS_10CYCLE
  - LL_SPI_SS_IDLENESS_11CYCLE
  - LL_SPI_SS_IDLENESS_12CYCLE
  - LL_SPI_SS_IDLENESS_13CYCLE
  - LL_SPI_SS_IDLENESS_14CYCLE
  - LL_SPI_SS_IDLENESS_15CYCLE

Reference Manual to LL API cross reference:

- CFG2 MSSI LL_SPI_GetMasterSSIdleness

LL_SPI_SetInterDataIdleness

Function name

__STATIC_INLINE void LL_SPI_SetInterDataIdleness (SPI_TypeDef * SPIx, uint32_t MasterInterDataIdleness)

Function description

Configure the idleness applied by master between data frame.

Parameters

- **SPIx:** SPI Instance
- **MasterInterDataIdleness:** This parameter can be one of the following values:
  - LL_SPI_ID_IDLENESS_00CYCLE
  - LL_SPI_ID_IDLENESS_01CYCLE
  - LL_SPI_ID_IDLENESS_02CYCLE
  - LL_SPI_ID_IDLENESS_03CYCLE
  - LL_SPI_ID_IDLENESS_04CYCLE
  - LL_SPI_ID_IDLENESS_05CYCLE
  - LL_SPI_ID_IDLENESS_06CYCLE
  - LL_SPI_ID_IDLENESS_07CYCLE
  - LL_SPI_ID_IDLENESS_08CYCLE
  - LL_SPI_ID_IDLENESS_09CYCLE
  - LL_SPI_ID_IDLENESS_10CYCLE
  - LL_SPI_ID_IDLENESS_11CYCLE
  - LL_SPI_ID_IDLENESS_12CYCLE
  - LL_SPI_ID_IDLENESS_13CYCLE
  - LL_SPI_ID_IDLENESS_14CYCLE
  - LL_SPI_ID_IDLENESS_15CYCLE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 MIDI LL_SPI_SetInterDataIdleness

LL_SPI_GetInterDataIdleness

Function name

__STATIC_INLINE uint32_t LL_SPI_GetInterDataIdleness (SPI_TypeDef * SPIx)

Function description

Get the configured inter data idleness.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_ID_IDLENESS_00CYCLE
  - LL_SPI_ID_IDLENESS_01CYCLE
  - LL_SPI_ID_IDLENESS_02CYCLE
  - LL_SPI_ID_IDLENESS_03CYCLE
  - LL_SPI_ID_IDLENESS_04CYCLE
  - LL_SPI_ID_IDLENESS_05CYCLE
  - LL_SPI_ID_IDLENESS_06CYCLE
  - LL_SPI_ID_IDLENESS_07CYCLE
  - LL_SPI_ID_IDLENESS_08CYCLE
  - LL_SPI_ID_IDLENESS_09CYCLE
  - LL_SPI_ID_IDLENESS_10CYCLE
  - LL_SPI_ID_IDLENESS_11CYCLE
  - LL_SPI_ID_IDLENESS_12CYCLE
  - LL_SPI_ID_IDLENESS_13CYCLE
  - LL_SPI_ID_IDLENESS_14CYCLE
  - LL_SPI_ID_IDLENESS_15CYCLE

Reference Manual to LL API cross reference:

- CFG2 MIDI LL_SPI_SetInterDataIdleness

LL_SPI_SetTransferSize

Function name

__STATIC_INLINE void LL_SPI_SetTransferSize (SPI_TypeDef * SPIx, uint32_t Count)

Function description

Set transfer size.

Parameters

- **SPIx:** SPI Instance
- **Count:** 0..0xFFFF

Return values

- **None:**

Notes

- Count is the number of frame to be transferred

Reference Manual to LL API cross reference:

- CR2 TSIZE LL_SPI_SetTransferSize

LL_SPI_GetTransferSize

Function name

__STATIC_INLINE uint32_t LL_SPI_GetTransferSize (SPI_TypeDef * SPIx)

Function description

Get transfer size.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFF:**

Notes

- Count is the number of frame to be transferred

Reference Manual to LL API cross reference:

- CR2 TSIZE LL_SPI_GetTransferSize

LL_SPI_SetReloadSize

Function name

__STATIC_INLINE void LL_SPI_SetReloadSize (SPI_TypeDef * SPIx, uint32_t Count)

Function description

Set reload transfer size.

Parameters

- **SPIx:** SPI Instance
- **Count:** 0..0xFFFF

Return values

- **None:**

Notes

- Count is the number of frame to be transferred

Reference Manual to LL API cross reference:

- CR2 TSER LL_SPI_SetReloadSize

LL_SPI_GetReloadSize

Function name

__STATIC_INLINE uint32_t LL_SPI_GetReloadSize (SPI_TypeDef * SPIx)

Function description

Get reload transfer size.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFF:**

Notes

- Count is the number of frame to be transferred

Reference Manual to LL API cross reference:

- CR2 TSER LL_SPI_GetReloadSize

LL_SPI_EnableIOLock

Function name

__STATIC_INLINE void LL_SPI_EnableIOLock (SPI_TypeDef * SPIx)

Function description

Lock the AF configuration of associated IOs.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- Once this bit is set, the AF configuration remains locked until a hardware reset occurs. the reset of the IOLock bit is done by hardware. for that, LL_SPI_DisableIOLock can not exist.

Reference Manual to LL API cross reference:

- CR1 IOLOCK LL_SPI_EnableIOLock

LL_SPI_IsEnabledIOLock

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIOLock (SPI_TypeDef * SPIx)

Function description

Check if the AF configuration is locked.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CR1 IOLOCK LL_SPI_IsEnabledIOLock

LL_SPI_SetTxCRCInitPattern

Function name

__STATIC_INLINE void LL_SPI_SetTxCRCInitPattern (SPI_TypeDef * SPIx, uint32_t TXCRCInitAll)

Function description

Set Tx CRC Initialization Pattern.

Parameters

- **SPIx:** SPI Instance
- **TXCRCInitAll:** This parameter can be one of the following values:
  - LL_SPI_TXCRCINIT_ALL_ZERO_PATTERN
  - LL_SPI_TXCRCINIT_ALL_ONES_PATTERN

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TCRCINI LL_SPI_SetTxCRCInitPattern

LL_SPI_GetTxCRCInitPattern

Function name

__STATIC_INLINE uint32_t LL_SPI_GetTxCRCInitPattern (SPI_TypeDef * SPIx)

Function description

Get Tx CRC Initialization Pattern.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_TXCRCINIT_ALL_ZERO_PATTERN
  - LL_SPI_TXCRCINIT_ALL_ONES_PATTERN

Reference Manual to LL API cross reference:

- CR1 TCRCINI LL_SPI_GetTxCRCInitPattern

LL_SPI_SetRxCRCInitPattern

Function name

__STATIC_INLINE void LL_SPI_SetRxCRCInitPattern (SPI_TypeDef * SPIx, uint32_t RXCRCInitAll)

Function description

Set Rx CRC Initialization Pattern.

Parameters

- **SPIx:** SPI Instance
- **RXCRCInitAll:** This parameter can be one of the following values:
  - LL_SPI_RXCRCINIT_ALL_ZERO_PATTERN
  - LL_SPI_RXCRCINIT_ALL_ONES_PATTERN

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RCRCINI LL_SPI_SetRxCRCInitPattern

LL_SPI_GetRxCRCInitPattern

Function name

__STATIC_INLINE uint32_t LL_SPI_GetRxCRCInitPattern (SPI_TypeDef * SPIx)

Function description

Get Rx CRC Initialization Pattern.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_RXCRCINIT_ALL_ZERO_PATTERN
  - LL_SPI_RXCRCINIT_ALL_ONES_PATTERN

Reference Manual to LL API cross reference:

- CR1 RCRCINI LL_SPI_GetRxCRCInitPattern

LL_SPI_SetInternalSSLevel

Function name

__STATIC_INLINE void LL_SPI_SetInternalSSLevel (SPI_TypeDef * SPIx, uint32_t SSLevel)

Function description

Set internal SS input level ignoring what comes from PIN.

Parameters

- **SPIx:** SPI Instance
- **SSLevel:** This parameter can be one of the following values:
  - LL_SPI_SS_LEVEL_HIGH
  - LL_SPI_SS_LEVEL_LOW

Return values

- **None:**

Notes

- This configuration has effect only with config LL_SPI_NSS_SOFT

Reference Manual to LL API cross reference:

- CR1 SSI LL_SPI_SetInternalSSLevel

LL_SPI_GetInternalSSLevel

Function name

__STATIC_INLINE uint32_t LL_SPI_GetInternalSSLevel (SPI_TypeDef * SPIx)

Function description

Get internal SS input level.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_SS_LEVEL_HIGH
  - LL_SPI_SS_LEVEL_LOW

Reference Manual to LL API cross reference:

- CR1 SSI LL_SPI_GetInternalSSLevel

LL_SPI_EnableFullSizeCRC

Function name

__STATIC_INLINE void LL_SPI_EnableFullSizeCRC (SPI_TypeDef * SPIx)

Function description

Enable CRC computation on 33/17 bits.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CRC33_17 LL_SPI_EnableFullSizeCRC

LL_SPI_DisableFullSizeCRC

Function name

__STATIC_INLINE void LL_SPI_DisableFullSizeCRC (SPI_TypeDef * SPIx)

Function description

Disable CRC computation on 33/17 bits.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CRC33_17 LL_SPI_DisableFullSizeCRC

LL_SPI_IsEnabledFullSizeCRC

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledFullSizeCRC (SPI_TypeDef * SPIx)

Function description

Check if Enable CRC computation on 33/17 bits is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CR1 CRC33_17 LL_SPI_IsEnabledFullSizeCRC

LL_SPI_SuspendMasterTransfer

Function name

__STATIC_INLINE void LL_SPI_SuspendMasterTransfer (SPI_TypeDef * SPIx)

Function description

Suspend an ongoing transfer for Master configuration.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CSUSP LL_SPI_SuspendMasterTransfer

LL_SPI_StartMasterTransfer

Function name

__STATIC_INLINE void LL_SPI_StartMasterTransfer (SPI_TypeDef * SPIx)

Function description

Start effective transfer on wire for Master configuration.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CSTART LL_SPI_StartMasterTransfer

LL_SPI_IsActiveMasterTransfer

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveMasterTransfer (SPI_TypeDef * SPIx)

Function description

Check if there is an unfinished master transfer.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CR1 CSTART LL_SPI_IsActiveMasterTransfer

LL_SPI_EnableMasterRxAutoSuspend

Function name

__STATIC_INLINE void LL_SPI_EnableMasterRxAutoSuspend (SPI_TypeDef * SPIx)

Function description

Enable Master Rx auto suspend in case of overrun.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 MASRX LL_SPI_EnableMasterRxAutoSuspend

LL_SPI_DisableMasterRxAutoSuspend

Function name

__STATIC_INLINE void LL_SPI_DisableMasterRxAutoSuspend (SPI_TypeDef * SPIx)

Function description

Disable Master Rx auto suspend in case of overrun.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 MASRX LL_SPI_DisableMasterRxAutoSuspend

LL_SPI_IsEnabledMasterRxAutoSuspend

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledMasterRxAutoSuspend (SPI_TypeDef * SPIx)

Function description

Check if Master Rx auto suspend is activated.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CR1 MASRX LL_SPI_IsEnabledMasterRxAutoSuspend

LL_SPI_SetUDRConfiguration

Function name

__STATIC_INLINE void LL_SPI_SetUDRConfiguration (SPI_TypeDef * SPIx, uint32_t UDRConfig)

Function description

Set Underrun behavior.

Parameters

- **SPIx:** SPI Instance
- **UDRConfig:** This parameter can be one of the following values:
  - LL_SPI_UDR_CONFIG_REGISTER_PATTERN
  - LL_SPI_UDR_CONFIG_LAST_RECEIVED
  - LL_SPI_UDR_CONFIG_LAST_TRANSMITTED

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG1 UDRCFG LL_SPI_SetUDRConfiguration

LL_SPI_GetUDRConfiguration

Function name

__STATIC_INLINE uint32_t LL_SPI_GetUDRConfiguration (SPI_TypeDef * SPIx)

Function description

Get Underrun behavior.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_UDR_CONFIG_REGISTER_PATTERN
  - LL_SPI_UDR_CONFIG_LAST_RECEIVED
  - LL_SPI_UDR_CONFIG_LAST_TRANSMITTED

Reference Manual to LL API cross reference:

- CFG1 UDRCFG LL_SPI_GetUDRConfiguration

LL_SPI_SetUDRDetection

Function name

__STATIC_INLINE void LL_SPI_SetUDRDetection (SPI_TypeDef * SPIx, uint32_t UDRDetection)

Function description

Set Underrun Detection method.

Parameters

- **SPIx:** SPI Instance
- **UDRDetection:** This parameter can be one of the following values:
  - LL_SPI_UDR_DETECT_BEGIN_DATA_FRAME
  - LL_SPI_UDR_DETECT_END_DATA_FRAME
  - LL_SPI_UDR_DETECT_BEGIN_ACTIVE_NSS

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG1 UDRDET LL_SPI_SetUDRDetection

LL_SPI_GetUDRDetection

Function name

__STATIC_INLINE uint32_t LL_SPI_GetUDRDetection (SPI_TypeDef * SPIx)

Function description

Get Underrun Detection method.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_UDR_DETECT_BEGIN_DATA_FRAME
  - LL_SPI_UDR_DETECT_END_DATA_FRAME
  - LL_SPI_UDR_DETECT_BEGIN_ACTIVE_NSS

Reference Manual to LL API cross reference:

- CFG1 UDRDET LL_SPI_GetUDRDetection

LL_SPI_SetStandard

Function name

__STATIC_INLINE void LL_SPI_SetStandard (SPI_TypeDef * SPIx, uint32_t Standard)

Function description

Set Serial protocol used.

Parameters

- **SPIx:** SPI Instance
- **Standard:** This parameter can be one of the following values:
  - LL_SPI_PROTOCOL_MOTOROLA
  - LL_SPI_PROTOCOL_TI

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG2 SP LL_SPI_SetStandard

LL_SPI_GetStandard

Function name

__STATIC_INLINE uint32_t LL_SPI_GetStandard (SPI_TypeDef * SPIx)

Function description

Get Serial protocol used.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_PROTOCOL_MOTOROLA
  - LL_SPI_PROTOCOL_TI

Reference Manual to LL API cross reference:

- CFG2 SP LL_SPI_GetStandard

LL_SPI_SetClockPhase

Function name

__STATIC_INLINE void LL_SPI_SetClockPhase (SPI_TypeDef * SPIx, uint32_t ClockPhase)

Function description

Set Clock phase.

Parameters

- **SPIx:** SPI Instance
- **ClockPhase:** This parameter can be one of the following values:
  - LL_SPI_PHASE_1EDGE
  - LL_SPI_PHASE_2EDGE

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CFG2 CPHA LL_SPI_SetClockPhase

LL_SPI_GetClockPhase

Function name

__STATIC_INLINE uint32_t LL_SPI_GetClockPhase (SPI_TypeDef * SPIx)

Function description

Get Clock phase.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_PHASE_1EDGE
  - LL_SPI_PHASE_2EDGE

Reference Manual to LL API cross reference:

- CFG2 CPHA LL_SPI_GetClockPhase

LL_SPI_SetClockPolarity

Function name

__STATIC_INLINE void LL_SPI_SetClockPolarity (SPI_TypeDef * SPIx, uint32_t ClockPolarity)

Function description

Set Clock polarity.

Parameters

- **SPIx:** SPI Instance
- **ClockPolarity:** This parameter can be one of the following values:
  - LL_SPI_POLARITY_LOW
  - LL_SPI_POLARITY_HIGH

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CFG2 CPOL LL_SPI_SetClockPolarity

LL_SPI_GetClockPolarity

Function name

__STATIC_INLINE uint32_t LL_SPI_GetClockPolarity (SPI_TypeDef * SPIx)

Function description

Get Clock polarity.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_POLARITY_LOW
  - LL_SPI_POLARITY_HIGH

Reference Manual to LL API cross reference:

- CFG2 CPOL LL_SPI_GetClockPolarity

LL_SPI_SetNSSPolarity

Function name

__STATIC_INLINE void LL_SPI_SetNSSPolarity (SPI_TypeDef * SPIx, uint32_t NSSPolarity)

Function description

Set NSS polarity.

Parameters

- **SPIx:** SPI Instance
- **NSSPolarity:** This parameter can be one of the following values:
  - LL_SPI_NSS_POLARITY_LOW
  - LL_SPI_NSS_POLARITY_HIGH

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CFG2 SSIOP LL_SPI_SetNSSPolarity

LL_SPI_GetNSSPolarity

Function name

__STATIC_INLINE uint32_t LL_SPI_GetNSSPolarity (SPI_TypeDef * SPIx)

Function description

Get NSS polarity.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_NSS_POLARITY_LOW
  - LL_SPI_NSS_POLARITY_HIGH

Reference Manual to LL API cross reference:

- CFG2 SSIOP LL_SPI_GetNSSPolarity

LL_SPI_SetBaudRatePrescaler

Function name

__STATIC_INLINE void LL_SPI_SetBaudRatePrescaler (SPI_TypeDef * SPIx, uint32_t Baudrate)

Function description

Set Baudrate Prescaler.

Parameters

- **SPIx:** SPI Instance
- **Baudrate:** This parameter can be one of the following values:
  - LL_SPI_BAUDRATEPRESCALER_DIV2
  - LL_SPI_BAUDRATEPRESCALER_DIV4
  - LL_SPI_BAUDRATEPRESCALER_DIV8
  - LL_SPI_BAUDRATEPRESCALER_DIV16
  - LL_SPI_BAUDRATEPRESCALER_DIV32
  - LL_SPI_BAUDRATEPRESCALER_DIV64
  - LL_SPI_BAUDRATEPRESCALER_DIV128
  - LL_SPI_BAUDRATEPRESCALER_DIV256

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. SPI BaudRate = fPCLK/Pescaler.

Reference Manual to LL API cross reference:

- CFG1 MBR LL_SPI_SetBaudRatePrescaler

LL_SPI_GetBaudRatePrescaler

Function name

__STATIC_INLINE uint32_t LL_SPI_GetBaudRatePrescaler (SPI_TypeDef * SPIx)

Function description

Get Baudrate Prescaler.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_BAUDRATEPRESCALER_DIV2
  - LL_SPI_BAUDRATEPRESCALER_DIV4
  - LL_SPI_BAUDRATEPRESCALER_DIV8
  - LL_SPI_BAUDRATEPRESCALER_DIV16
  - LL_SPI_BAUDRATEPRESCALER_DIV32
  - LL_SPI_BAUDRATEPRESCALER_DIV64
  - LL_SPI_BAUDRATEPRESCALER_DIV128
  - LL_SPI_BAUDRATEPRESCALER_DIV256

Reference Manual to LL API cross reference:

- CFG1 MBR LL_SPI_GetBaudRatePrescaler

LL_SPI_SetTransferBitOrder

Function name

__STATIC_INLINE void LL_SPI_SetTransferBitOrder (SPI_TypeDef * SPIx, uint32_t BitOrder)

Function description

Set Transfer Bit Order.

Parameters

- **SPIx:** SPI Instance
- **BitOrder:** This parameter can be one of the following values:
  - LL_SPI_LSB_FIRST
  - LL_SPI_MSB_FIRST

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CFG2 LSBFRST LL_SPI_SetTransferBitOrder

LL_SPI_GetTransferBitOrder

Function name

__STATIC_INLINE uint32_t LL_SPI_GetTransferBitOrder (SPI_TypeDef * SPIx)

Function description

Get Transfer Bit Order.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_LSB_FIRST
  - LL_SPI_MSB_FIRST

Reference Manual to LL API cross reference:

- CFG2 LSBFRST LL_SPI_GetTransferBitOrder

LL_SPI_SetTransferDirection

Function name

__STATIC_INLINE void LL_SPI_SetTransferDirection (SPI_TypeDef * SPIx, uint32_t TransferDirection)

Function description

Set Transfer Mode.

Parameters

- **SPIx:** SPI Instance
- **TransferDirection:** This parameter can be one of the following values:
  - LL_SPI_FULL_DUPLEX
  - LL_SPI_SIMPLEX_TX
  - LL_SPI_SIMPLEX_RX
  - LL_SPI_HALF_DUPLEX_RX
  - LL_SPI_HALF_DUPLEX_TX

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled except for half duplex direction using LL_SPI_SetHalfDuplexDirection.

Reference Manual to LL API cross reference:

- CR1 HDDIR LL_SPI_SetTransferDirection
- CFG2 COMM LL_SPI_SetTransferDirection

LL_SPI_GetTransferDirection

Function name

__STATIC_INLINE uint32_t LL_SPI_GetTransferDirection (SPI_TypeDef * SPIx)

Function description

Get Transfer Mode.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_FULL_DUPLEX
  - LL_SPI_SIMPLEX_TX
  - LL_SPI_SIMPLEX_RX
  - LL_SPI_HALF_DUPLEX_RX
  - LL_SPI_HALF_DUPLEX_TX

Reference Manual to LL API cross reference:

- CR1 HDDIR LL_SPI_GetTransferDirection
- CFG2 COMM LL_SPI_GetTransferDirection

LL_SPI_SetHalfDuplexDirection

Function name

__STATIC_INLINE void LL_SPI_SetHalfDuplexDirection (SPI_TypeDef * SPIx, uint32_t HalfDuplexDirection)

Function description

Set direction for Half-Duplex Mode.

Parameters

- **SPIx:** SPI Instance
- **HalfDuplexDirection:** This parameter can be one of the following values:
  - LL_SPI_HALF_DUPLEX_RX
  - LL_SPI_HALF_DUPLEX_TX

Return values

- **None:**

Notes

- In master mode the MOSI pin is used and in slave mode the MISO pin is used for Half-Duplex.

Reference Manual to LL API cross reference:

- CR1 HDDIR LL_SPI_SetHalfDuplexDirection

LL_SPI_GetHalfDuplexDirection

Function name

__STATIC_INLINE uint32_t LL_SPI_GetHalfDuplexDirection (SPI_TypeDef * SPIx)

Function description

Get direction for Half-Duplex Mode.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_HALF_DUPLEX_RX
  - LL_SPI_HALF_DUPLEX_TX

Notes

- In master mode the MOSI pin is used and in slave mode the MISO pin is used for Half-Duplex.

Reference Manual to LL API cross reference:

- CR1 HDDIR LL_SPI_GetHalfDuplexDirection

LL_SPI_SetDataWidth

Function name

__STATIC_INLINE void LL_SPI_SetDataWidth (SPI_TypeDef * SPIx, uint32_t DataWidth)

Function description

Set Frame Data Size.

Parameters

- **SPIx:** SPI Instance
- **DataWidth:** This parameter can be one of the following values:
  - LL_SPI_DATAWIDTH_4BIT
  - LL_SPI_DATAWIDTH_5BIT
  - LL_SPI_DATAWIDTH_6BIT
  - LL_SPI_DATAWIDTH_7BIT
  - LL_SPI_DATAWIDTH_8BIT
  - LL_SPI_DATAWIDTH_9BIT
  - LL_SPI_DATAWIDTH_10BIT
  - LL_SPI_DATAWIDTH_11BIT
  - LL_SPI_DATAWIDTH_12BIT
  - LL_SPI_DATAWIDTH_13BIT
  - LL_SPI_DATAWIDTH_14BIT
  - LL_SPI_DATAWIDTH_15BIT
  - LL_SPI_DATAWIDTH_16BIT
  - LL_SPI_DATAWIDTH_17BIT
  - LL_SPI_DATAWIDTH_18BIT
  - LL_SPI_DATAWIDTH_19BIT
  - LL_SPI_DATAWIDTH_20BIT
  - LL_SPI_DATAWIDTH_21BIT
  - LL_SPI_DATAWIDTH_22BIT
  - LL_SPI_DATAWIDTH_23BIT
  - LL_SPI_DATAWIDTH_24BIT
  - LL_SPI_DATAWIDTH_25BIT
  - LL_SPI_DATAWIDTH_26BIT
  - LL_SPI_DATAWIDTH_27BIT
  - LL_SPI_DATAWIDTH_28BIT
  - LL_SPI_DATAWIDTH_29BIT
  - LL_SPI_DATAWIDTH_30BIT
  - LL_SPI_DATAWIDTH_31BIT
  - LL_SPI_DATAWIDTH_32BIT

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG1 DSIZE LL_SPI_SetDataWidth

LL_SPI_GetDataWidth

Function name

__STATIC_INLINE uint32_t LL_SPI_GetDataWidth (SPI_TypeDef * SPIx)

Function description

Get Frame Data Size.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_DATAWIDTH_4BIT
  - LL_SPI_DATAWIDTH_5BIT
  - LL_SPI_DATAWIDTH_6BIT
  - LL_SPI_DATAWIDTH_7BIT
  - LL_SPI_DATAWIDTH_8BIT
  - LL_SPI_DATAWIDTH_9BIT
  - LL_SPI_DATAWIDTH_10BIT
  - LL_SPI_DATAWIDTH_11BIT
  - LL_SPI_DATAWIDTH_12BIT
  - LL_SPI_DATAWIDTH_13BIT
  - LL_SPI_DATAWIDTH_14BIT
  - LL_SPI_DATAWIDTH_15BIT
  - LL_SPI_DATAWIDTH_16BIT
  - LL_SPI_DATAWIDTH_17BIT
  - LL_SPI_DATAWIDTH_18BIT
  - LL_SPI_DATAWIDTH_19BIT
  - LL_SPI_DATAWIDTH_20BIT
  - LL_SPI_DATAWIDTH_21BIT
  - LL_SPI_DATAWIDTH_22BIT
  - LL_SPI_DATAWIDTH_23BIT
  - LL_SPI_DATAWIDTH_24BIT
  - LL_SPI_DATAWIDTH_25BIT
  - LL_SPI_DATAWIDTH_26BIT
  - LL_SPI_DATAWIDTH_27BIT
  - LL_SPI_DATAWIDTH_28BIT
  - LL_SPI_DATAWIDTH_29BIT
  - LL_SPI_DATAWIDTH_30BIT
  - LL_SPI_DATAWIDTH_31BIT
  - LL_SPI_DATAWIDTH_32BIT

Reference Manual to LL API cross reference:

- CFG1 DSIZE LL_SPI_GetDataWidth

LL_SPI_SetFIFOThreshold

Function name

__STATIC_INLINE void LL_SPI_SetFIFOThreshold (SPI_TypeDef * SPIx, uint32_t Threshold)

Function description

Set threshold of FIFO that triggers a transfer event.

Parameters

- **SPIx:** SPI Instance
- **Threshold:** This parameter can be one of the following values:
  - LL_SPI_FIFO_TH_01DATA
  - LL_SPI_FIFO_TH_02DATA
  - LL_SPI_FIFO_TH_03DATA
  - LL_SPI_FIFO_TH_04DATA
  - LL_SPI_FIFO_TH_05DATA
  - LL_SPI_FIFO_TH_06DATA
  - LL_SPI_FIFO_TH_07DATA
  - LL_SPI_FIFO_TH_08DATA
  - LL_SPI_FIFO_TH_09DATA
  - LL_SPI_FIFO_TH_10DATA
  - LL_SPI_FIFO_TH_11DATA
  - LL_SPI_FIFO_TH_12DATA
  - LL_SPI_FIFO_TH_13DATA
  - LL_SPI_FIFO_TH_14DATA
  - LL_SPI_FIFO_TH_15DATA
  - LL_SPI_FIFO_TH_16DATA

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG1 FTHLV LL_SPI_SetFIFOThreshold

LL_SPI_GetFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_SPI_GetFIFOThreshold (SPI_TypeDef * SPIx)

Function description

Get threshold of FIFO that triggers a transfer event.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_FIFO_TH_01DATA
  - LL_SPI_FIFO_TH_02DATA
  - LL_SPI_FIFO_TH_03DATA
  - LL_SPI_FIFO_TH_04DATA
  - LL_SPI_FIFO_TH_05DATA
  - LL_SPI_FIFO_TH_06DATA
  - LL_SPI_FIFO_TH_07DATA
  - LL_SPI_FIFO_TH_08DATA
  - LL_SPI_FIFO_TH_09DATA
  - LL_SPI_FIFO_TH_10DATA
  - LL_SPI_FIFO_TH_11DATA
  - LL_SPI_FIFO_TH_12DATA
  - LL_SPI_FIFO_TH_13DATA
  - LL_SPI_FIFO_TH_14DATA
  - LL_SPI_FIFO_TH_15DATA
  - LL_SPI_FIFO_TH_16DATA

Reference Manual to LL API cross reference:

- CFG1 FTHLV LL_SPI_GetFIFOThreshold

LL_SPI_EnableCRC

Function name

__STATIC_INLINE void LL_SPI_EnableCRC (SPI_TypeDef * SPIx)

Function description

Enable CRC.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG1 CRCEN LL_SPI_EnableCRC

LL_SPI_DisableCRC

Function name

__STATIC_INLINE void LL_SPI_DisableCRC (SPI_TypeDef * SPIx)

Function description

Disable CRC.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 CRCEN LL_SPI_DisableCRC

LL_SPI_IsEnabledCRC

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledCRC (SPI_TypeDef * SPIx)

Function description

Check if CRC is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFG1 CRCEN LL_SPI_IsEnabledCRC

LL_SPI_SetCRCWidth

Function name

__STATIC_INLINE void LL_SPI_SetCRCWidth (SPI_TypeDef * SPIx, uint32_t CRCLength)

Function description

Set CRC Length.

Parameters

- **SPIx:** SPI Instance
- **CRCLength:** This parameter can be one of the following values:
  - LL_SPI_CRC_4BIT
  - LL_SPI_CRC_5BIT
  - LL_SPI_CRC_6BIT
  - LL_SPI_CRC_7BIT
  - LL_SPI_CRC_8BIT
  - LL_SPI_CRC_9BIT
  - LL_SPI_CRC_10BIT
  - LL_SPI_CRC_11BIT
  - LL_SPI_CRC_12BIT
  - LL_SPI_CRC_13BIT
  - LL_SPI_CRC_14BIT
  - LL_SPI_CRC_15BIT
  - LL_SPI_CRC_16BIT
  - LL_SPI_CRC_17BIT
  - LL_SPI_CRC_18BIT
  - LL_SPI_CRC_19BIT
  - LL_SPI_CRC_20BIT
  - LL_SPI_CRC_21BIT
  - LL_SPI_CRC_22BIT
  - LL_SPI_CRC_23BIT
  - LL_SPI_CRC_24BIT
  - LL_SPI_CRC_25BIT
  - LL_SPI_CRC_26BIT
  - LL_SPI_CRC_27BIT
  - LL_SPI_CRC_28BIT
  - LL_SPI_CRC_29BIT
  - LL_SPI_CRC_30BIT
  - LL_SPI_CRC_31BIT
  - LL_SPI_CRC_32BIT

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG1 CRCSIZE LL_SPI_SetCRCWidth

LL_SPI_GetCRCWidth

Function name

__STATIC_INLINE uint32_t LL_SPI_GetCRCWidth (SPI_TypeDef * SPIx)

Function description

Get CRC Length.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_CRC_4BIT
  - LL_SPI_CRC_5BIT
  - LL_SPI_CRC_6BIT
  - LL_SPI_CRC_7BIT
  - LL_SPI_CRC_8BIT
  - LL_SPI_CRC_9BIT
  - LL_SPI_CRC_10BIT
  - LL_SPI_CRC_11BIT
  - LL_SPI_CRC_12BIT
  - LL_SPI_CRC_13BIT
  - LL_SPI_CRC_14BIT
  - LL_SPI_CRC_15BIT
  - LL_SPI_CRC_16BIT
  - LL_SPI_CRC_17BIT
  - LL_SPI_CRC_18BIT
  - LL_SPI_CRC_19BIT
  - LL_SPI_CRC_20BIT
  - LL_SPI_CRC_21BIT
  - LL_SPI_CRC_22BIT
  - LL_SPI_CRC_23BIT
  - LL_SPI_CRC_24BIT
  - LL_SPI_CRC_25BIT
  - LL_SPI_CRC_26BIT
  - LL_SPI_CRC_27BIT
  - LL_SPI_CRC_28BIT
  - LL_SPI_CRC_29BIT
  - LL_SPI_CRC_30BIT
  - LL_SPI_CRC_31BIT
  - LL_SPI_CRC_32BIT

Reference Manual to LL API cross reference:

- CFG1 CRCSIZE LL_SPI_GetCRCWidth

LL_SPI_SetNSSMode

Function name

__STATIC_INLINE void LL_SPI_SetNSSMode (SPI_TypeDef * SPIx, uint32_t NSS)

Function description

Set NSS Mode.

Parameters

- **SPIx:** SPI Instance
- **NSS:** This parameter can be one of the following values:
  - LL_SPI_NSS_SOFT
  - LL_SPI_NSS_HARD_INPUT
  - LL_SPI_NSS_HARD_OUTPUT

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CFG2 SSM LL_SPI_SetNSSMode
- CFG2 SSOE LL_SPI_SetNSSMode

LL_SPI_GetNSSMode

Function name

__STATIC_INLINE uint32_t LL_SPI_GetNSSMode (SPI_TypeDef * SPIx)

Function description

Set NSS Mode.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_NSS_SOFT
  - LL_SPI_NSS_HARD_INPUT
  - LL_SPI_NSS_HARD_OUTPUT

Reference Manual to LL API cross reference:

- CFG2 SSM LL_SPI_GetNSSMode
- CFG2 SSOE LL_SPI_GetNSSMode

LL_SPI_EnableNSSPulseMgt

Function name

__STATIC_INLINE void LL_SPI_EnableNSSPulseMgt (SPI_TypeDef * SPIx)

Function description

Enable NSS pulse mgt.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CFG2 SSOM LL_SPI_EnableNSSPulseMgt

LL_SPI_DisableNSSPulseMgt

Function name

__STATIC_INLINE void LL_SPI_DisableNSSPulseMgt (SPI_TypeDef * SPIx)

Function description

Disable NSS pulse mgt.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CFG2 SSOM LL_SPI_DisableNSSPulseMgt

LL_SPI_IsEnabledNSSPulse

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledNSSPulse (SPI_TypeDef * SPIx)

Function description

Check if NSS pulse is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG2 SSOM LL_SPI_IsEnabledNSSPulse

LL_SPI_IsActiveFlag_RXP

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_RXP (SPI_TypeDef * SPIx)

Function description

Check if there is enough data in FIFO to read a full packet.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- SR RXP LL_SPI_IsActiveFlag_RXP

LL_SPI_IsActiveFlag_TXP

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_TXP (SPI_TypeDef * SPIx)

Function description

Check if there is enough space in FIFO to hold a full packet.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- SR TXP LL_SPI_IsActiveFlag_TXP

LL_SPI_IsActiveFlag_DXP

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_DXP (SPI_TypeDef * SPIx)

Function description

Check if there enough space in FIFO to hold a full packet, AND enough data to read a full packet.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- SR DXP LL_SPI_IsActiveFlag_DXP

LL_SPI_IsActiveFlag_EOT

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_EOT (SPI_TypeDef * SPIx)

Function description

Check that end of transfer event occurred.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR EOT LL_SPI_IsActiveFlag_EOT

LL_SPI_IsActiveFlag_TXTF

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_TXTF (SPI_TypeDef * SPIx)

Function description

Check that all required data has been filled in the fifo according to transfer size.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXTF LL_SPI_IsActiveFlag_TXTF

LL_SPI_IsActiveFlag_UDR

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_UDR (SPI_TypeDef * SPIx)

Function description

Get Underrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR UDR LL_SPI_IsActiveFlag_UDR

LL_SPI_IsActiveFlag_CRCERR

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_CRCERR (SPI_TypeDef * SPIx)

Function description

Get CRC error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CRCE LL_SPI_IsActiveFlag_CRCERR

LL_SPI_IsActiveFlag_MODF

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_MODF (SPI_TypeDef * SPIx)

Function description

Get Mode fault error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR MODF LL_SPI_IsActiveFlag_MODF

LL_SPI_IsActiveFlag_OVR

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_OVR (SPI_TypeDef * SPIx)

Function description

Get Overrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR OVR LL_SPI_IsActiveFlag_OVR

LL_SPI_IsActiveFlag_FRE

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_FRE (SPI_TypeDef * SPIx)

Function description

Get TI Frame format error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TIFRE LL_SPI_IsActiveFlag_FRE

LL_SPI_IsActiveFlag_TSER

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_TSER (SPI_TypeDef * SPIx)

Function description

Check if the additional number of data has been reloaded.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TSERF LL_SPI_IsActiveFlag_TSER

LL_SPI_IsActiveFlag_SUSP

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_SUSP (SPI_TypeDef * SPIx)

Function description

Check if a suspend operation is done.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- SR SUSP LL_SPI_IsActiveFlag_SUSP

LL_SPI_IsActiveFlag_TXC

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_TXC (SPI_TypeDef * SPIx)

Function description

Check if last TxFIFO or CRC frame transmission is completed.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXC LL_SPI_IsActiveFlag_TXC

LL_SPI_IsActiveFlag_RXWNE

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_RXWNE (SPI_TypeDef * SPIx)

Function description

Check if at least one 32-bit data is available in RxFIFO.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- SR RXWNE LL_SPI_IsActiveFlag_RXWNE

LL_SPI_GetRemainingDataFrames

Function name

__STATIC_INLINE uint32_t LL_SPI_GetRemainingDataFrames (SPI_TypeDef * SPIx)

Function description

Get number of data framed remaining in current TSIZE.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFF:**

Reference Manual to LL API cross reference:

- SR CTSIZE LL_SPI_GetRemainingDataFrames

LL_SPI_GetRxFIFOPackingLevel

Function name

__STATIC_INLINE uint32_t LL_SPI_GetRxFIFOPackingLevel (SPI_TypeDef * SPIx)

Function description

Get RxFIFO packing Level.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_RX_FIFO_0PACKET
  - LL_SPI_RX_FIFO_1PACKET
  - LL_SPI_RX_FIFO_2PACKET
  - LL_SPI_RX_FIFO_3PACKET

Reference Manual to LL API cross reference:

- SR RXPLVL LL_SPI_GetRxFIFOPackingLevel

LL_SPI_ClearFlag_EOT

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_EOT (SPI_TypeDef * SPIx)

Function description

Clear End Of Transfer flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR EOTC LL_SPI_ClearFlag_EOT

LL_SPI_ClearFlag_TXTF

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_TXTF (SPI_TypeDef * SPIx)

Function description

Clear TXTF flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR TXTFC LL_SPI_ClearFlag_TXTF

LL_SPI_ClearFlag_UDR

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_UDR (SPI_TypeDef * SPIx)

Function description

Clear Underrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR UDRC LL_SPI_ClearFlag_UDR

LL_SPI_ClearFlag_OVR

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_OVR (SPI_TypeDef * SPIx)

Function description

Clear Overrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR OVRC LL_SPI_ClearFlag_OVR

LL_SPI_ClearFlag_CRCERR

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_CRCERR (SPI_TypeDef * SPIx)

Function description

Clear CRC error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CRCEC LL_SPI_ClearFlag_CRCERR

LL_SPI_ClearFlag_MODF

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_MODF (SPI_TypeDef * SPIx)

Function description

Clear Mode fault error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR MODFC LL_SPI_ClearFlag_MODF

LL_SPI_ClearFlag_FRE

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_FRE (SPI_TypeDef * SPIx)

Function description

Clear Frame format error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR TIFREC LL_SPI_ClearFlag_FRE

LL_SPI_ClearFlag_TSER

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_TSER (SPI_TypeDef * SPIx)

Function description

Clear TSER flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR TSERFC LL_SPI_ClearFlag_TSER

LL_SPI_ClearFlag_SUSP

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_SUSP (SPI_TypeDef * SPIx)

Function description

Clear SUSP flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR SUSPC LL_SPI_ClearFlag_SUSP

LL_SPI_EnableIT_RXP

Function name

__STATIC_INLINE void LL_SPI_EnableIT_RXP (SPI_TypeDef * SPIx)

Function description

Enable Rx Packet available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RXPIE LL_SPI_EnableIT_RXP

LL_SPI_EnableIT_TXP

Function name

__STATIC_INLINE void LL_SPI_EnableIT_TXP (SPI_TypeDef * SPIx)

Function description

Enable Tx Packet space available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TXPIE LL_SPI_EnableIT_TXP

LL_SPI_EnableIT_DXP

Function name

__STATIC_INLINE void LL_SPI_EnableIT_DXP (SPI_TypeDef * SPIx)

Function description

Enable Duplex Packet available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER DXPIE LL_SPI_EnableIT_DXP

LL_SPI_EnableIT_EOT

Function name

__STATIC_INLINE void LL_SPI_EnableIT_EOT (SPI_TypeDef * SPIx)

Function description

Enable End Of Transfer IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOTIE LL_SPI_EnableIT_EOT

LL_SPI_EnableIT_TXTF

Function name

__STATIC_INLINE void LL_SPI_EnableIT_TXTF (SPI_TypeDef * SPIx)

Function description

Enable TXTF IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TXTFIE LL_SPI_EnableIT_TXTF

LL_SPI_EnableIT_UDR

Function name

__STATIC_INLINE void LL_SPI_EnableIT_UDR (SPI_TypeDef * SPIx)

Function description

Enable Underrun IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER UDRIE LL_SPI_EnableIT_UDR

LL_SPI_EnableIT_OVR

Function name

__STATIC_INLINE void LL_SPI_EnableIT_OVR (SPI_TypeDef * SPIx)

Function description

Enable Overrun IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER OVRIE LL_SPI_EnableIT_OVR

LL_SPI_EnableIT_CRCERR

Function name

__STATIC_INLINE void LL_SPI_EnableIT_CRCERR (SPI_TypeDef * SPIx)

Function description

Enable CRC Error IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CRCEIE LL_SPI_EnableIT_CRCERR

LL_SPI_EnableIT_FRE

Function name

__STATIC_INLINE void LL_SPI_EnableIT_FRE (SPI_TypeDef * SPIx)

Function description

Enable TI Frame Format Error IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TIFREIE LL_SPI_EnableIT_FRE

LL_SPI_EnableIT_MODF

Function name

__STATIC_INLINE void LL_SPI_EnableIT_MODF (SPI_TypeDef * SPIx)

Function description

Enable MODF IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER MODFIE LL_SPI_EnableIT_MODF

LL_SPI_EnableIT_TSER

Function name

__STATIC_INLINE void LL_SPI_EnableIT_TSER (SPI_TypeDef * SPIx)

Function description

Enable TSER reload IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TSERFIE LL_SPI_EnableIT_TSER

LL_SPI_DisableIT_RXP

Function name

__STATIC_INLINE void LL_SPI_DisableIT_RXP (SPI_TypeDef * SPIx)

Function description

Disable Rx Packet available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RXPIE LL_SPI_DisableIT_RXP

LL_SPI_DisableIT_TXP

Function name

__STATIC_INLINE void LL_SPI_DisableIT_TXP (SPI_TypeDef * SPIx)

Function description

Disable Tx Packet space available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TXPIE LL_SPI_DisableIT_TXP

LL_SPI_DisableIT_DXP

Function name

__STATIC_INLINE void LL_SPI_DisableIT_DXP (SPI_TypeDef * SPIx)

Function description

Disable Duplex Packet available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER DXPIE LL_SPI_DisableIT_DXP

LL_SPI_DisableIT_EOT

Function name

__STATIC_INLINE void LL_SPI_DisableIT_EOT (SPI_TypeDef * SPIx)

Function description

Disable End Of Transfer IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOTIE LL_SPI_DisableIT_EOT

LL_SPI_DisableIT_TXTF

Function name

__STATIC_INLINE void LL_SPI_DisableIT_TXTF (SPI_TypeDef * SPIx)

Function description

Disable TXTF IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TXTFIE LL_SPI_DisableIT_TXTF

LL_SPI_DisableIT_UDR

Function name

__STATIC_INLINE void LL_SPI_DisableIT_UDR (SPI_TypeDef * SPIx)

Function description

Disable Underrun IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER UDRIE LL_SPI_DisableIT_UDR

LL_SPI_DisableIT_OVR

Function name

__STATIC_INLINE void LL_SPI_DisableIT_OVR (SPI_TypeDef * SPIx)

Function description

Disable Overrun IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER OVRIE LL_SPI_DisableIT_OVR

LL_SPI_DisableIT_CRCERR

Function name

__STATIC_INLINE void LL_SPI_DisableIT_CRCERR (SPI_TypeDef * SPIx)

Function description

Disable CRC Error IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CRCEIE LL_SPI_DisableIT_CRCERR

LL_SPI_DisableIT_FRE

Function name

__STATIC_INLINE void LL_SPI_DisableIT_FRE (SPI_TypeDef * SPIx)

Function description

Disable TI Frame Format Error IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TIFREIE LL_SPI_DisableIT_FRE

LL_SPI_DisableIT_MODF

Function name

__STATIC_INLINE void LL_SPI_DisableIT_MODF (SPI_TypeDef * SPIx)

Function description

Disable MODF IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER MODFIE LL_SPI_DisableIT_MODF

LL_SPI_DisableIT_TSER

Function name

__STATIC_INLINE void LL_SPI_DisableIT_TSER (SPI_TypeDef * SPIx)

Function description

Disable TSER reload IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TSERFIE LL_SPI_DisableIT_TSER

LL_SPI_IsEnabledIT_RXP

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_RXP (SPI_TypeDef * SPIx)

Function description

Check if Rx Packet available IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER RXPIE LL_SPI_IsEnabledIT_RXP

LL_SPI_IsEnabledIT_TXP

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_TXP (SPI_TypeDef * SPIx)

Function description

Check if Tx Packet space available IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER TXPIE LL_SPI_IsEnabledIT_TXP

LL_SPI_IsEnabledIT_DXP

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_DXP (SPI_TypeDef * SPIx)

Function description

Check if Duplex Packet available IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER DXPIE LL_SPI_IsEnabledIT_DXP

LL_SPI_IsEnabledIT_EOT

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_EOT (SPI_TypeDef * SPIx)

Function description

Check if End Of Transfer IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER EOTIE LL_SPI_IsEnabledIT_EOT

LL_SPI_IsEnabledIT_TXTF

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_TXTF (SPI_TypeDef * SPIx)

Function description

Check if TXTF IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER TXTFIE LL_SPI_IsEnabledIT_TXTF

LL_SPI_IsEnabledIT_UDR

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_UDR (SPI_TypeDef * SPIx)

Function description

Check if Underrun IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER UDRIE LL_SPI_IsEnabledIT_UDR

LL_SPI_IsEnabledIT_OVR

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_OVR (SPI_TypeDef * SPIx)

Function description

Check if Overrun IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER OVRIE LL_SPI_IsEnabledIT_OVR

LL_SPI_IsEnabledIT_CRCERR

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_CRCERR (SPI_TypeDef * SPIx)

Function description

Check if CRC Error IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER CRCEIE LL_SPI_IsEnabledIT_CRCERR

LL_SPI_IsEnabledIT_FRE

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_FRE (SPI_TypeDef * SPIx)

Function description

Check if TI Frame Format Error IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER TIFREIE LL_SPI_IsEnabledIT_FRE

LL_SPI_IsEnabledIT_MODF

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_MODF (SPI_TypeDef * SPIx)

Function description

Check if MODF IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER MODFIE LL_SPI_IsEnabledIT_MODF

LL_SPI_IsEnabledIT_TSER

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_TSER (SPI_TypeDef * SPIx)

Function description

Check if TSER reload IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER TSERFIE LL_SPI_IsEnabledIT_TSER

LL_SPI_EnableDMAReq_RX

Function name

__STATIC_INLINE void LL_SPI_EnableDMAReq_RX (SPI_TypeDef * SPIx)

Function description

Enable DMA Rx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 RXDMAEN LL_SPI_EnableDMAReq_RX

LL_SPI_DisableDMAReq_RX

Function name

__STATIC_INLINE void LL_SPI_DisableDMAReq_RX (SPI_TypeDef * SPIx)

Function description

Disable DMA Rx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 RXDMAEN LL_SPI_DisableDMAReq_RX

LL_SPI_IsEnabledDMAReq_RX

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledDMAReq_RX (SPI_TypeDef * SPIx)

Function description

Check if DMA Rx is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG1 RXDMAEN LL_SPI_IsEnabledDMAReq_RX

LL_SPI_EnableDMAReq_TX

Function name

__STATIC_INLINE void LL_SPI_EnableDMAReq_TX (SPI_TypeDef * SPIx)

Function description

Enable DMA Tx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 TXDMAEN LL_SPI_EnableDMAReq_TX

LL_SPI_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_SPI_DisableDMAReq_TX (SPI_TypeDef * SPIx)

Function description

Disable DMA Tx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 TXDMAEN LL_SPI_DisableDMAReq_TX

LL_SPI_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledDMAReq_TX (SPI_TypeDef * SPIx)

Function description

Check if DMA Tx is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG1 TXDMAEN LL_SPI_IsEnabledDMAReq_TX

LL_SPI_DMA_GetTxRegAddr

Function name

__STATIC_INLINE uint32_t LL_SPI_DMA_GetTxRegAddr (SPI_TypeDef * SPIx)

Function description

Get the data register address used for DMA transfer.

Parameters

- **SPIx:** SPI Instance

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- TXDR TXDR LL_SPI_DMA_GetTxRegAddr

LL_SPI_DMA_GetRxRegAddr

Function name

__STATIC_INLINE uint32_t LL_SPI_DMA_GetRxRegAddr (SPI_TypeDef * SPIx)

Function description

Get the data register address used for DMA transfer.

Parameters

- **SPIx:** SPI Instance

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- RXDR RXDR LL_SPI_DMA_GetRxRegAddr

LL_SPI_ReceiveData8

Function name

__STATIC_INLINE uint8_t LL_SPI_ReceiveData8 (SPI_TypeDef * SPIx)

Function description

Read Data Register.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFF:**

Reference Manual to LL API cross reference:

- RXDR . LL_SPI_ReceiveData8

LL_SPI_ReceiveData16

Function name

__STATIC_INLINE uint16_t LL_SPI_ReceiveData16 (SPI_TypeDef * SPIx)

Function description

Read Data Register.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFF:**

Reference Manual to LL API cross reference:

- RXDR . LL_SPI_ReceiveData16

LL_SPI_ReceiveData32

Function name

__STATIC_INLINE uint32_t LL_SPI_ReceiveData32 (SPI_TypeDef * SPIx)

Function description

Read Data Register.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFFFFFF:**

Reference Manual to LL API cross reference:

- RXDR . LL_SPI_ReceiveData32

LL_SPI_TransmitData8

Function name

__STATIC_INLINE void LL_SPI_TransmitData8 (SPI_TypeDef * SPIx, uint8_t TxData)

Function description

Write Data Register.

Parameters

- **SPIx:** SPI Instance
- **TxData:** 0..0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TXDR . LL_SPI_TransmitData8

LL_SPI_TransmitData16

Function name

__STATIC_INLINE void LL_SPI_TransmitData16 (SPI_TypeDef * SPIx, uint16_t TxData)

Function description

Write Data Register.

Parameters

- **SPIx:** SPI Instance
- **TxData:** 0..0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TXDR . LL_SPI_TransmitData16

LL_SPI_TransmitData32

Function name

__STATIC_INLINE void LL_SPI_TransmitData32 (SPI_TypeDef * SPIx, uint32_t TxData)

Function description

Write Data Register.

Parameters

- **SPIx:** SPI Instance
- **TxData:** 0..0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TXDR . LL_SPI_TransmitData32

LL_SPI_SetCRCPolynomial

Function name

__STATIC_INLINE void LL_SPI_SetCRCPolynomial (SPI_TypeDef * SPIx, uint32_t CRCPoly)

Function description

Set polynomial for CRC calcul.

Parameters

- **SPIx:** SPI Instance
- **CRCPoly:** 0..0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- CRCPOLY CRCPOLY LL_SPI_SetCRCPolynomial

LL_SPI_GetCRCPolynomial

Function name

__STATIC_INLINE uint32_t LL_SPI_GetCRCPolynomial (SPI_TypeDef * SPIx)

Function description

Get polynomial for CRC calcul.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFFFFFF:**

Reference Manual to LL API cross reference:

- CRCPOLY CRCPOLY LL_SPI_GetCRCPolynomial

LL_SPI_SetUDRPattern

Function name

__STATIC_INLINE void LL_SPI_SetUDRPattern (SPI_TypeDef * SPIx, uint32_t Pattern)

Function description

Set the underrun pattern.

Parameters

- **SPIx:** SPI Instance
- **Pattern:** 0..0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- UDRDR UDRDR LL_SPI_SetUDRPattern

LL_SPI_GetUDRPattern

Function name

__STATIC_INLINE uint32_t LL_SPI_GetUDRPattern (SPI_TypeDef * SPIx)

Function description

Get the underrun pattern.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFFFFFF:**

Reference Manual to LL API cross reference:

- UDRDR UDRDR LL_SPI_GetUDRPattern

LL_SPI_GetRxCRC

Function name

__STATIC_INLINE uint32_t LL_SPI_GetRxCRC (SPI_TypeDef * SPIx)

Function description

Get Rx CRC.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFFFFFF:**

Reference Manual to LL API cross reference:

- RXCRCR RXCRC LL_SPI_GetRxCRC

LL_SPI_GetTxCRC

Function name

__STATIC_INLINE uint32_t LL_SPI_GetTxCRC (SPI_TypeDef * SPIx)

Function description

Get Tx CRC.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFFFFFF:**

Reference Manual to LL API cross reference:

- TXCRCR TXCRC LL_SPI_GetTxCRC

LL_SPI_DeInit

Function name

ErrorStatus LL_SPI_DeInit (SPI_TypeDef * SPIx)

Function description

De-initialize the SPI registers to their default reset values.

Parameters

- **SPIx:** SPI Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: SPI registers are de-initialized
  - ERROR: SPI registers are not de-initialized

LL_SPI_Init

Function name

ErrorStatus LL_SPI_Init (SPI_TypeDef * SPIx, LL_SPI_InitTypeDef * SPI_InitStruct)

Function description

Initialize the SPI registers according to the specified parameters in SPI_InitStruct.

Parameters

- **SPIx:** SPI Instance
- **SPI_InitStruct:** pointer to a LL_SPI_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value. (Return always SUCCESS)

Notes

- As some bits in SPI configuration registers can only be written when the SPI is disabled (SPI_CR1_SPE bit =0), SPI IP should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.

LL_SPI_StructInit

Function name

void LL_SPI_StructInit (LL_SPI_InitTypeDef * SPI_InitStruct)

Function description

Set each LL_SPI_InitTypeDef field to default value.

Parameters

- **SPI_InitStruct:** pointer to a LL_SPI_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_I2S_SetDataFormat

Function name

__STATIC_INLINE void LL_I2S_SetDataFormat (SPI_TypeDef * SPIx, uint32_t DataLength)

Function description

Set I2S Data frame format.

Parameters

- **SPIx:** SPI Handle
- **DataLength:** This parameter can be one of the following values:
  - LL_I2S_DATAFORMAT_16B
  - LL_I2S_DATAFORMAT_16B_EXTENDED
  - LL_I2S_DATAFORMAT_24B
  - LL_I2S_DATAFORMAT_24B_LEFT_ALIGNED
  - LL_I2S_DATAFORMAT_32B

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR DATLEN LL_I2S_SetDataFormat
- I2SCFGR CHLEN LL_I2S_SetDataFormat
- I2SCFGR DATFMT LL_I2S_SetDataFormat

LL_I2S_GetDataFormat

Function name

__STATIC_INLINE uint32_t LL_I2S_GetDataFormat (SPI_TypeDef * SPIx)

Function description

Get I2S Data frame format.

Parameters

- **SPIx:** SPI Handle

Return values

- **Return:** value can be one of the following values:
  - LL_I2S_DATAFORMAT_16B
  - LL_I2S_DATAFORMAT_16B_EXTENDED
  - LL_I2S_DATAFORMAT_24B
  - LL_I2S_DATAFORMAT_24B_LEFT_ALIGNED
  - LL_I2S_DATAFORMAT_32B

Reference Manual to LL API cross reference:

- I2SCFGR DATLEN LL_I2S_GetDataFormat
- I2SCFGR CHLEN LL_I2S_GetDataFormat
- I2SCFGR DATFMT LL_I2S_GetDataFormat

LL_I2S_SetChannelLengthType

Function name

__STATIC_INLINE void LL_I2S_SetChannelLengthType (SPI_TypeDef * SPIx, uint32_t ChannelLengthType)

Function description

Set I2S Channel Length Type.

Parameters

- **SPIx:** SPI Handle
- **ChannelLengthType:** This parameter can be one of the following values:
  - LL_I2S_SLAVE_VARIABLE_CH_LENGTH
  - LL_I2S_SLAVE_FIXED_CH_LENGTH

Return values

- **None:**

Notes

- This feature is useful with SLAVE only

Reference Manual to LL API cross reference:

- I2SCFGR FIXCH LL_I2S_SetChannelLengthType

LL_I2S_GetChannelLengthType

Function name

__STATIC_INLINE uint32_t LL_I2S_GetChannelLengthType (SPI_TypeDef * SPIx)

Function description

Get I2S Channel Length Type.

Parameters

- **SPIx:** SPI Handle

Return values

- **Return:** value can be one of the following values:
  - LL_I2S_SLAVE_VARIABLE_CH_LENGTH
  - LL_I2S_SLAVE_FIXED_CH_LENGTH

Notes

- This feature is useful with SLAVE only

Reference Manual to LL API cross reference:

- I2SCFGR FIXCH LL_I2S_GetChannelLengthType

LL_I2S_EnableWordSelectInversion

Function name

__STATIC_INLINE void LL_I2S_EnableWordSelectInversion (SPI_TypeDef * SPIx)

Function description

Invert the default polarity of WS signal.

Parameters

- **SPIx:** SPI Handle

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR WSINV LL_I2S_EnableWordSelectInversion

LL_I2S_DisableWordSelectInversion

Function name

__STATIC_INLINE void LL_I2S_DisableWordSelectInversion (SPI_TypeDef * SPIx)

Function description

Use the default polarity of WS signal.

Parameters

- **SPIx:** SPI Handle

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR WSINV LL_I2S_DisableWordSelectInversion

LL_I2S_IsEnabledWordSelectInversion

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledWordSelectInversion (SPI_TypeDef * SPIx)

Function description

Check if polarity of WS signal is inverted.

Parameters

- **SPIx:** SPI Handle

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- I2SCFGR WSINV LL_I2S_IsEnabledWordSelectInversion

LL_I2S_SetClockPolarity

Function name

__STATIC_INLINE void LL_I2S_SetClockPolarity (SPI_TypeDef * SPIx, uint32_t ClockPolarity)

Function description

Set 2S Clock Polarity.

Parameters

- **SPIx:** SPI Handle
- **ClockPolarity:** This parameter can be one of the following values:
  - LL_I2S_POLARITY_LOW
  - LL_I2S_POLARITY_HIGH

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR CKPOL LL_I2S_SetClockPolarity

LL_I2S_GetClockPolarity

Function name

__STATIC_INLINE uint32_t LL_I2S_GetClockPolarity (SPI_TypeDef * SPIx)

Function description

Get 2S Clock Polarity.

Parameters

- **SPIx:** SPI Handle

Return values

- **Return:** value can be one of the following values:
  - LL_I2S_POLARITY_LOW
  - LL_I2S_POLARITY_HIGH

Reference Manual to LL API cross reference:

- I2SCFGR CKPOL LL_I2S_GetClockPolarity

LL_I2S_SetStandard

Function name

__STATIC_INLINE void LL_I2S_SetStandard (SPI_TypeDef * SPIx, uint32_t Standard)

Function description

Set I2S standard.

Parameters

- **SPIx:** SPI Handle
- **Standard:** This parameter can be one of the following values:
  - LL_I2S_STANDARD_PHILIPS
  - LL_I2S_STANDARD_MSB
  - LL_I2S_STANDARD_LSB
  - LL_I2S_STANDARD_PCM_SHORT
  - LL_I2S_STANDARD_PCM_LONG

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR I2SSTD LL_I2S_SetStandard
- I2SCFGR PCMSYNC LL_I2S_SetStandard

LL_I2S_GetStandard

Function name

__STATIC_INLINE uint32_t LL_I2S_GetStandard (SPI_TypeDef * SPIx)

Function description

Get I2S standard.

Parameters

- **SPIx:** SPI Handle

Return values

- **Return:** value can be one of the following values:
  - LL_I2S_STANDARD_PHILIPS
  - LL_I2S_STANDARD_MSB
  - LL_I2S_STANDARD_LSB
  - LL_I2S_STANDARD_PCM_SHORT
  - LL_I2S_STANDARD_PCM_LONG

Reference Manual to LL API cross reference:

- I2SCFGR I2SSTD LL_I2S_GetStandard
- I2SCFGR PCMSYNC LL_I2S_GetStandard

LL_I2S_SetTransferMode

Function name

__STATIC_INLINE void LL_I2S_SetTransferMode (SPI_TypeDef * SPIx, uint32_t Standard)

Function description

Set I2S config.

Parameters

- **SPIx:** SPI Handle
- **Standard:** This parameter can be one of the following values:
  - LL_I2S_MODE_SLAVE_TX
  - LL_I2S_MODE_SLAVE_RX
  - LL_I2S_MODE_SLAVE_FULL_DUPLEX
  - LL_I2S_MODE_MASTER_TX
  - LL_I2S_MODE_MASTER_RX
  - LL_I2S_MODE_MASTER_FULL_DUPLEX

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR I2SCFG LL_I2S_SetTransferMode

LL_I2S_GetTransferMode

Function name

__STATIC_INLINE uint32_t LL_I2S_GetTransferMode (SPI_TypeDef * SPIx)

Function description

Get I2S config.

Parameters

- **SPIx:** SPI Handle

Return values

- **Return:** value can be one of the following values:
  - LL_I2S_MODE_SLAVE_TX
  - LL_I2S_MODE_SLAVE_RX
  - LL_I2S_MODE_SLAVE_FULL_DUPLEX
  - LL_I2S_MODE_MASTER_TX
  - LL_I2S_MODE_MASTER_RX
  - LL_I2S_MODE_MASTER_FULL_DUPLEX

Reference Manual to LL API cross reference:

- I2SCFGR I2SCFG LL_I2S_GetTransferMode

LL_I2S_Enable

Function name

__STATIC_INLINE void LL_I2S_Enable (SPI_TypeDef * SPIx)

Function description

Select I2S mode and Enable I2S peripheral.

Parameters

- **SPIx:** SPI Handle

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR I2SMOD LL_I2S_Enable
- CR1 SPE LL_I2S_Enable

LL_I2S_Disable

Function name

__STATIC_INLINE void LL_I2S_Disable (SPI_TypeDef * SPIx)

Function description

Disable I2S peripheral and disable I2S mode.

Parameters

- **SPIx:** SPI Handle

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 SPE LL_I2S_Disable
- I2SCFGR I2SMOD LL_I2S_Disable

LL_I2S_EnableIOSwap

Function name

__STATIC_INLINE void LL_I2S_EnableIOSwap (SPI_TypeDef * SPIx)

Function description

Swap the SDO and SDI pin.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when I2S is enabled.

Reference Manual to LL API cross reference:

- CFG2 IOSWP LL_I2S_EnableIOSwap

LL_I2S_DisableIOSwap

Function name

__STATIC_INLINE void LL_I2S_DisableIOSwap (SPI_TypeDef * SPIx)

Function description

Restore default function for SDO and SDI pin.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when I2S is enabled.

Reference Manual to LL API cross reference:

- CFG2 IOSWP LL_I2S_DisableIOSwap

LL_I2S_IsEnabledIOSwap

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledIOSwap (SPI_TypeDef * SPIx)

Function description

Check if SDO and SDI pin are swapped.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG2 IOSWP LL_I2S_IsEnabledIOSwap

LL_I2S_EnableGPIOControl

Function name

__STATIC_INLINE void LL_I2S_EnableGPIOControl (SPI_TypeDef * SPIx)

Function description

Enable GPIO control.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when I2S is enabled.

Reference Manual to LL API cross reference:

- CFG2 AFCNTR LL_I2S_EnableGPIOControl

LL_I2S_DisableGPIOControl

Function name

__STATIC_INLINE void LL_I2S_DisableGPIOControl (SPI_TypeDef * SPIx)

Function description

Disable GPIO control.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when I2S is enabled.

Reference Manual to LL API cross reference:

- CFG2 AFCNTR LL_I2S_DisableGPIOControl

LL_I2S_IsEnabledGPIOControl

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledGPIOControl (SPI_TypeDef * SPIx)

Function description

Check if GPIO control is active.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG2 AFCNTR LL_I2S_IsEnabledGPIOControl

LL_I2S_EnableIOLock

Function name

__STATIC_INLINE void LL_I2S_EnableIOLock (SPI_TypeDef * SPIx)

Function description

Lock the AF configuration of associated IOs.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- Once this bit is set, the SPI_CFG2 register content can not be modified until a hardware reset occurs. The reset of the IOLock bit is done by hardware. for that, LL_SPI_DisableIOLock can not exist.

Reference Manual to LL API cross reference:

- CR1 IOLOCK LL_SPI_EnableIOLock

LL_I2S_IsEnabledIOLock

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledIOLock (SPI_TypeDef * SPIx)

Function description

Check if the the SPI_CFG2 register is locked.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CR1 IOLOCK LL_I2S_IsEnabledIOLock

LL_I2S_SetTransferBitOrder

Function name

__STATIC_INLINE void LL_I2S_SetTransferBitOrder (SPI_TypeDef * SPIx, uint32_t BitOrder)

Function description

Set Transfer Bit Order.

Parameters

- **SPIx:** SPI Instance
- **BitOrder:** This parameter can be one of the following values:
  - LL_I2S_LSB_FIRST
  - LL_I2S_MSB_FIRST

Return values

- **None:**

Notes

- This configuration can not be changed when I2S is enabled.

Reference Manual to LL API cross reference:

- CFG2 LSBFRST LL_I2S_SetTransferBitOrder

LL_I2S_GetTransferBitOrder

Function name

__STATIC_INLINE uint32_t LL_I2S_GetTransferBitOrder (SPI_TypeDef * SPIx)

Function description

Get Transfer Bit Order.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_I2S_LSB_FIRST
  - LL_I2S_MSB_FIRST

Reference Manual to LL API cross reference:

- CFG2 LSBFRST LL_I2S_GetTransferBitOrder

LL_I2S_StartTransfer

Function name

__STATIC_INLINE void LL_I2S_StartTransfer (SPI_TypeDef * SPIx)

Function description

Start effective transfer on wire.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CSTART LL_I2S_StartTransfer

LL_I2S_IsActiveTransfer

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveTransfer (SPI_TypeDef * SPIx)

Function description

Check if there is an unfinished transfer.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CR1 CSTART LL_I2S_IsTransferActive

LL_I2S_SetFIFOThreshold

Function name

__STATIC_INLINE void LL_I2S_SetFIFOThreshold (SPI_TypeDef * SPIx, uint32_t Threshold)

Function description

Set threshold of FIFO that triggers a transfer event.

Parameters

- **SPIx:** SPI Instance
- **Threshold:** This parameter can be one of the following values:
  - LL_I2S_FIFO_TH_01DATA
  - LL_I2S_FIFO_TH_02DATA
  - LL_I2S_FIFO_TH_03DATA
  - LL_I2S_FIFO_TH_04DATA
  - LL_I2S_FIFO_TH_05DATA
  - LL_I2S_FIFO_TH_06DATA
  - LL_I2S_FIFO_TH_07DATA
  - LL_I2S_FIFO_TH_08DATA

Return values

- **None:**

Notes

- This configuration can not be changed when I2S is enabled.

Reference Manual to LL API cross reference:

- CFG1 FTHLV LL_I2S_SetFIFOThreshold

LL_I2S_GetFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_I2S_GetFIFOThreshold (SPI_TypeDef * SPIx)

Function description

Get threshold of FIFO that triggers a transfer event.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_I2S_FIFO_TH_01DATA
  - LL_I2S_FIFO_TH_02DATA
  - LL_I2S_FIFO_TH_03DATA
  - LL_I2S_FIFO_TH_04DATA
  - LL_I2S_FIFO_TH_05DATA
  - LL_I2S_FIFO_TH_06DATA
  - LL_I2S_FIFO_TH_07DATA
  - LL_I2S_FIFO_TH_08DATA

Reference Manual to LL API cross reference:

- CFG1 FTHLV LL_I2S_GetFIFOThreshold

LL_I2S_SetPrescalerLinear

Function name

__STATIC_INLINE void LL_I2S_SetPrescalerLinear (SPI_TypeDef * SPIx, uint32_t PrescalerLinear)

Function description

Set I2S linear prescaler.

Parameters

- **SPIx:** SPI Instance
- **PrescalerLinear:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Notes

- PrescalerLinear '1' is not authorized with parity LL_I2S_PRESCALER_PARITY_ODD

Reference Manual to LL API cross reference:

- I2SCFGR I2SDIV LL_I2S_SetPrescalerLinear

LL_I2S_GetPrescalerLinear

Function name

__STATIC_INLINE uint32_t LL_I2S_GetPrescalerLinear (SPI_TypeDef * SPIx)

Function description

Get I2S linear prescaler.

Parameters

- **SPIx:** SPI Instance

Return values

- **PrescalerLinear:** Value between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- I2SCFGR I2SDIV LL_I2S_GetPrescalerLinear

LL_I2S_SetPrescalerParity

Function name

__STATIC_INLINE void LL_I2S_SetPrescalerParity (SPI_TypeDef * SPIx, uint32_t PrescalerParity)

Function description

Set I2S parity prescaler.

Parameters

- **SPIx:** SPI Instance
- **PrescalerParity:** This parameter can be one of the following values:
  - LL_I2S_PRESCALER_PARITY_EVEN
  - LL_I2S_PRESCALER_PARITY_ODD

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR ODD LL_I2S_SetPrescalerParity

LL_I2S_GetPrescalerParity

Function name

__STATIC_INLINE uint32_t LL_I2S_GetPrescalerParity (SPI_TypeDef * SPIx)

Function description

Get I2S parity prescaler.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_I2S_PRESCALER_PARITY_EVEN
  - LL_I2S_PRESCALER_PARITY_ODD

Reference Manual to LL API cross reference:

- I2SCFGR ODD LL_I2S_GetPrescalerParity

LL_I2S_EnableMasterClock

Function name

__STATIC_INLINE void LL_I2S_EnableMasterClock (SPI_TypeDef * SPIx)

Function description

Enable the Master Clock Output (Pin MCK)

Parameters

- **SPIx:** SPI Handle

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR MCKOE LL_I2S_EnableMasterClock

LL_I2S_DisableMasterClock

Function name

__STATIC_INLINE void LL_I2S_DisableMasterClock (SPI_TypeDef * SPIx)

Function description

Disable the Master Clock Output (Pin MCK)

Parameters

- **SPIx:** SPI Handle

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR MCKOE LL_I2S_DisableMasterClock

LL_I2S_IsEnabledMasterClock

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledMasterClock (SPI_TypeDef * SPIx)

Function description

Check if the master clock output (Pin MCK) is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- I2SCFGR MCKOE LL_I2S_IsEnabledMasterClock

LL_I2S_IsActiveFlag_RXP

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_RXP (SPI_TypeDef * SPIx)

Function description

Check if there enough data in FIFO to read a full packet.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- SR RXP LL_I2S_IsActiveFlag_RXP

LL_I2S_IsActiveFlag_TXP

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_TXP (SPI_TypeDef * SPIx)

Function description

Check if there enough space in FIFO to hold a full packet.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- SR TXP LL_I2S_IsActiveFlag_TXP

LL_I2S_IsActiveFlag_UDR

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_UDR (SPI_TypeDef * SPIx)

Function description

Get Underrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- SR UDR LL_I2S_IsActiveFlag_UDR

LL_I2S_IsActiveFlag_OVR

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_OVR (SPI_TypeDef * SPIx)

Function description

Get Overrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR OVR LL_I2S_IsActiveFlag_OVR

LL_I2S_IsActiveFlag_FRE

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_FRE (SPI_TypeDef * SPIx)

Function description

Get TI Frame format error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TIFRE LL_I2S_IsActiveFlag_FRE

LL_I2S_ClearFlag_UDR

Function name

__STATIC_INLINE void LL_I2S_ClearFlag_UDR (SPI_TypeDef * SPIx)

Function description

Clear Underrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR UDRC LL_I2S_ClearFlag_UDR

LL_I2S_ClearFlag_OVR

Function name

__STATIC_INLINE void LL_I2S_ClearFlag_OVR (SPI_TypeDef * SPIx)

Function description

Clear Overrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR OVRC LL_I2S_ClearFlag_OVR

LL_I2S_ClearFlag_FRE

Function name

__STATIC_INLINE void LL_I2S_ClearFlag_FRE (SPI_TypeDef * SPIx)

Function description

Clear Frame format error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR TIFREC LL_I2S_ClearFlag_FRE

LL_I2S_EnableIT_RXP

Function name

__STATIC_INLINE void LL_I2S_EnableIT_RXP (SPI_TypeDef * SPIx)

Function description

Enable Rx Packet available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RXPIE LL_I2S_EnableIT_RXP

LL_I2S_EnableIT_TXP

Function name

__STATIC_INLINE void LL_I2S_EnableIT_TXP (SPI_TypeDef * SPIx)

Function description

Enable Tx Packet space available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TXPIE LL_I2S_EnableIT_TXP

LL_I2S_EnableIT_UDR

Function name

__STATIC_INLINE void LL_I2S_EnableIT_UDR (SPI_TypeDef * SPIx)

Function description

Enable Underrun IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER UDRIE LL_I2S_EnableIT_UDR

LL_I2S_EnableIT_OVR

Function name

__STATIC_INLINE void LL_I2S_EnableIT_OVR (SPI_TypeDef * SPIx)

Function description

Enable Overrun IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER OVRIE LL_I2S_EnableIT_OVR

LL_I2S_EnableIT_FRE

Function name

__STATIC_INLINE void LL_I2S_EnableIT_FRE (SPI_TypeDef * SPIx)

Function description

Enable TI Frame Format Error IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TIFREIE LL_I2S_EnableIT_FRE

LL_I2S_DisableIT_RXP

Function name

__STATIC_INLINE void LL_I2S_DisableIT_RXP (SPI_TypeDef * SPIx)

Function description

Disable Rx Packet available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RXPIE LL_I2S_DisableIT_RXP

LL_I2S_DisableIT_TXP

Function name

__STATIC_INLINE void LL_I2S_DisableIT_TXP (SPI_TypeDef * SPIx)

Function description

Disable Tx Packet space available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TXPIE LL_I2S_DisableIT_TXP

LL_I2S_DisableIT_UDR

Function name

__STATIC_INLINE void LL_I2S_DisableIT_UDR (SPI_TypeDef * SPIx)

Function description

Disable Underrun IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER UDRIE LL_I2S_DisableIT_UDR

LL_I2S_DisableIT_OVR

Function name

__STATIC_INLINE void LL_I2S_DisableIT_OVR (SPI_TypeDef * SPIx)

Function description

Disable Overrun IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER OVRIE LL_I2S_DisableIT_OVR

LL_I2S_DisableIT_FRE

Function name

__STATIC_INLINE void LL_I2S_DisableIT_FRE (SPI_TypeDef * SPIx)

Function description

Disable TI Frame Format Error IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TIFREIE LL_I2S_DisableIT_FRE

LL_I2S_IsEnabledIT_RXP

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_RXP (SPI_TypeDef * SPIx)

Function description

Check if Rx Packet available IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER RXPIE LL_I2S_IsEnabledIT_RXP

LL_I2S_IsEnabledIT_TXP

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_TXP (SPI_TypeDef * SPIx)

Function description

Check if Tx Packet space available IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER TXPIE LL_I2S_IsEnabledIT_TXP

LL_I2S_IsEnabledIT_UDR

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_UDR (SPI_TypeDef * SPIx)

Function description

Check if Underrun IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER UDRIE LL_I2S_IsEnabledIT_UDR

LL_I2S_IsEnabledIT_OVR

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_OVR (SPI_TypeDef * SPIx)

Function description

Check if Overrun IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER OVRIE LL_I2S_IsEnabledIT_OVR

LL_I2S_IsEnabledIT_FRE

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_FRE (SPI_TypeDef * SPIx)

Function description

Check if TI Frame Format Error IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER TIFREIE LL_I2S_IsEnabledIT_FRE

LL_I2S_EnableDMAReq_RX

Function name

__STATIC_INLINE void LL_I2S_EnableDMAReq_RX (SPI_TypeDef * SPIx)

Function description

Enable DMA Rx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 RXDMAEN LL_I2S_EnableDMAReq_RX

LL_I2S_DisableDMAReq_RX

Function name

__STATIC_INLINE void LL_I2S_DisableDMAReq_RX (SPI_TypeDef * SPIx)

Function description

Disable DMA Rx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 RXDMAEN LL_I2S_DisableDMAReq_RX

LL_I2S_IsEnabledDMAReq_RX

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledDMAReq_RX (SPI_TypeDef * SPIx)

Function description

Check if DMA Rx is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG1 RXDMAEN LL_I2S_IsEnabledDMAReq_RX

LL_I2S_EnableDMAReq_TX

Function name

__STATIC_INLINE void LL_I2S_EnableDMAReq_TX (SPI_TypeDef * SPIx)

Function description

Enable DMA Tx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 TXDMAEN LL_I2S_EnableDMAReq_TX

LL_I2S_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_I2S_DisableDMAReq_TX (SPI_TypeDef * SPIx)

Function description

Disable DMA Tx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 TXDMAEN LL_I2S_DisableDMAReq_TX

LL_I2S_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledDMAReq_TX (SPI_TypeDef * SPIx)

Function description

Check if DMA Tx is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG1 TXDMAEN LL_I2S_IsEnabledDMAReq_TX

LL_I2S_ReceiveData16

Function name

__STATIC_INLINE uint16_t LL_I2S_ReceiveData16 (SPI_TypeDef * SPIx)

Function description

Read Data Register.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFF:**

Reference Manual to LL API cross reference:

- RXDR . LL_I2S_ReceiveData16

LL_I2S_ReceiveData32

Function name

__STATIC_INLINE uint32_t LL_I2S_ReceiveData32 (SPI_TypeDef * SPIx)

Function description

Read Data Register.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFFFFFF:**

Reference Manual to LL API cross reference:

- RXDR . LL_I2S_ReceiveData32

LL_I2S_TransmitData16

Function name

__STATIC_INLINE void LL_I2S_TransmitData16 (SPI_TypeDef * SPIx, uint16_t TxData)

Function description

Write Data Register.

Parameters

- **SPIx:** SPI Instance
- **TxData:** 0..0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TXDR . LL_I2S_TransmitData16

LL_I2S_TransmitData32

Function name

__STATIC_INLINE void LL_I2S_TransmitData32 (SPI_TypeDef * SPIx, uint32_t TxData)

Function description

Write Data Register.

Parameters

- **SPIx:** SPI Instance
- **TxData:** 0..0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TXDR . LL_I2S_TransmitData32

LL_I2S_DeInit

Function name

ErrorStatus LL_I2S_DeInit (SPI_TypeDef * SPIx)

Function description

De-initialize the SPI/I2S registers to their default reset values.

Parameters

- **SPIx:** SPI Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: SPI registers are de-initialized
  - ERROR: SPI registers are not de-initialized

LL_I2S_Init

Function name

ErrorStatus LL_I2S_Init (SPI_TypeDef * SPIx, LL_I2S_InitTypeDef * I2S_InitStruct)

Function description

Initializes the SPI/I2S registers according to the specified parameters in I2S_InitStruct.

Parameters

- **SPIx:** SPI Instance
- **I2S_InitStruct:** pointer to a LL_I2S_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: SPI registers are Initialized
  - ERROR: SPI registers are not Initialized

Notes

- As some bits in I2S configuration registers can only be written when the SPI is disabled (SPI_CR1_SPE bit =0), SPI IP should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.
- I2S (SPI) source clock must be ready before calling this function. Otherwise will results in wrong programming.

LL_I2S_StructInit

Function name

void LL_I2S_StructInit (LL_I2S_InitTypeDef * I2S_InitStruct)

Function description

Set each LL_I2S_InitTypeDef field to default value.

Parameters

- **I2S_InitStruct:** pointer to a LL_I2S_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_I2S_ConfigPrescaler

Function name

void LL_I2S_ConfigPrescaler (SPI_TypeDef * SPIx, uint32_t PrescalerLinear, uint32_t PrescalerParity)

Function description

Set linear and parity prescaler.

Parameters

- **SPIx:** SPI Instance
- **PrescalerLinear:** Value between Min_Data=0x00 and Max_Data=0xFF
- **PrescalerParity:** This parameter can be one of the following values:
  - LL_I2S_PRESCALER_PARITY_EVEN
  - LL_I2S_PRESCALER_PARITY_ODD

Return values

- **None:**

Notes

- To calculate value of PrescalerLinear(I2SDIV[7:0] bits) and PrescalerParity(ODD bit) Check Audio frequency table and formulas inside Reference Manual (SPI/I2S).
- PrescalerLinear '1' is not authorized with parity LL_I2S_PRESCALER_PARITY_ODD

123.3 SPI Firmware driver defines

The following section lists the various define and macros of the module.

123.3.1 SPI

SPI

Baud Rate Prescaler

LL_SPI_BAUDRATEPRESCALER_DIV2

LL_SPI_BAUDRATEPRESCALER_DIV4

LL_SPI_BAUDRATEPRESCALER_DIV8

LL_SPI_BAUDRATEPRESCALER_DIV16

LL_SPI_BAUDRATEPRESCALER_DIV32

LL_SPI_BAUDRATEPRESCALER_DIV64

LL_SPI_BAUDRATEPRESCALER_DIV128

LL_SPI_BAUDRATEPRESCALER_DIV256

Bit Order

LL_SPI_LSB_FIRST

LL_SPI_MSB_FIRST

CRC

LL_SPI_CRC_4BIT

LL_SPI_CRC_5BIT

LL_SPI_CRC_6BIT

LL_SPI_CRC_7BIT

LL_SPI_CRC_8BIT

LL_SPI_CRC_9BIT

LL_SPI_CRC_10BIT

LL_SPI_CRC_11BIT

LL_SPI_CRC_12BIT

LL_SPI_CRC_13BIT

LL_SPI_CRC_14BIT

LL_SPI_CRC_15BIT

LL_SPI_CRC_16BIT

LL_SPI_CRC_17BIT

LL_SPI_CRC_18BIT

LL_SPI_CRC_19BIT

LL_SPI_CRC_20BIT

LL_SPI_CRC_21BIT

LL_SPI_CRC_22BIT

LL_SPI_CRC_23BIT

LL_SPI_CRC_24BIT

LL_SPI_CRC_25BIT

LL_SPI_CRC_26BIT

LL_SPI_CRC_27BIT

LL_SPI_CRC_28BIT

LL_SPI_CRC_29BIT

LL_SPI_CRC_30BIT

LL_SPI_CRC_31BIT

LL_SPI_CRC_32BIT

CRC Calculation

LL_SPI_CRCCALCULATION_DISABLE

CRC calculation disabled

LL_SPI_CRCCALCULATION_ENABLE

CRC calculation enabled

Data Width

LL_SPI_DATAWIDTH_4BIT

LL_SPI_DATAWIDTH_5BIT

LL_SPI_DATAWIDTH_6BIT

LL_SPI_DATAWIDTH_7BIT

LL_SPI_DATAWIDTH_8BIT

LL_SPI_DATAWIDTH_9BIT

LL_SPI_DATAWIDTH_10BIT

LL_SPI_DATAWIDTH_11BIT

LL_SPI_DATAWIDTH_12BIT

LL_SPI_DATAWIDTH_13BIT

LL_SPI_DATAWIDTH_14BIT

LL_SPI_DATAWIDTH_15BIT

LL_SPI_DATAWIDTH_16BIT

LL_SPI_DATAWIDTH_17BIT

LL_SPI_DATAWIDTH_18BIT

LL_SPI_DATAWIDTH_19BIT

LL_SPI_DATAWIDTH_20BIT

LL_SPI_DATAWIDTH_21BIT

LL_SPI_DATAWIDTH_22BIT

LL_SPI_DATAWIDTH_23BIT

LL_SPI_DATAWIDTH_24BIT

LL_SPI_DATAWIDTH_25BIT

LL_SPI_DATAWIDTH_26BIT

LL_SPI_DATAWIDTH_27BIT

LL_SPI_DATAWIDTH_28BIT

LL_SPI_DATAWIDTH_29BIT

LL_SPI_DATAWIDTH_30BIT

LL_SPI_DATAWIDTH_31BIT

LL_SPI_DATAWIDTH_32BIT

FIFO Threshold

LL_SPI_FIFO_TH_01DATA

LL_SPI_FIFO_TH_02DATA

LL_SPI_FIFO_TH_03DATA

LL_SPI_FIFO_TH_04DATA

LL_SPI_FIFO_TH_05DATA

LL_SPI_FIFO_TH_06DATA

LL_SPI_FIFO_TH_07DATA

LL_SPI_FIFO_TH_08DATA

LL_SPI_FIFO_TH_09DATA

LL_SPI_FIFO_TH_10DATA

LL_SPI_FIFO_TH_11DATA

LL_SPI_FIFO_TH_12DATA

LL_SPI_FIFO_TH_13DATA

LL_SPI_FIFO_TH_14DATA

LL_SPI_FIFO_TH_15DATA

LL_SPI_FIFO_TH_16DATA

Get Flags Defines

LL_SPI_SR_RXP

LL_SPI_SR_TXP

LL_SPI_SR_DXP

LL_SPI_SR_EOT

LL_SPI_SR_TXTF

LL_SPI_SR_UDR

LL_SPI_SR_CRCE

LL_SPI_SR_MODF

LL_SPI_SR_OVR

LL_SPI_SR_TIFRE

LL_SPI_SR_TSERF

LL_SPI_SR_SUSP

LL_SPI_SR_TXC

LL_SPI_SR_RXWNE

Master Inter-Data Idleness

LL_SPI_ID_IDLENESS_00CYCLE

LL_SPI_ID_IDLENESS_01CYCLE

LL_SPI_ID_IDLENESS_02CYCLE

LL_SPI_ID_IDLENESS_03CYCLE

LL_SPI_ID_IDLENESS_04CYCLE

LL_SPI_ID_IDLENESS_05CYCLE

LL_SPI_ID_IDLENESS_06CYCLE

LL_SPI_ID_IDLENESS_07CYCLE

LL_SPI_ID_IDLENESS_08CYCLE

LL_SPI_ID_IDLENESS_09CYCLE

LL_SPI_ID_IDLENESS_10CYCLE

LL_SPI_ID_IDLENESS_11CYCLE

LL_SPI_ID_IDLENESS_12CYCLE

LL_SPI_ID_IDLENESS_13CYCLE

LL_SPI_ID_IDLENESS_14CYCLE

LL_SPI_ID_IDLENESS_15CYCLE

IT Defines

LL_SPI_IER_RXPIE

LL_SPI_IER_TXPIE

LL_SPI_IER_DXPIE

LL_SPI_IER_EOTIE

LL_SPI_IER_TXTFIE

LL_SPI_IER_UDRIE

LL_SPI_IER_OVRIE

LL_SPI_IER_CRCEIE

LL_SPI_IER_TIFREIE

LL_SPI_IER_MODFIE

LL_SPI_IER_TSERFIE

Mode

LL_SPI_MODE_MASTER

LL_SPI_MODE_SLAVE

NSS Mode

LL_SPI_NSS_SOFT

LL_SPI_NSS_HARD_INPUT

LL_SPI_NSS_HARD_OUTPUT

NSS Polarity

LL_SPI_NSS_POLARITY_LOW

LL_SPI_NSS_POLARITY_HIGH

Phase

LL_SPI_PHASE_1EDGE

LL_SPI_PHASE_2EDGE

Polarity

LL_SPI_POLARITY_LOW

LL_SPI_POLARITY_HIGH

Protocol

LL_SPI_PROTOCOL_MOTOROLA

LL_SPI_PROTOCOL_TI

RXCRC Init All

LL_SPI_RXCRCINIT_ALL_ZERO_PATTERN

LL_SPI_RXCRCINIT_ALL_ONES_PATTERN

RxFIFO Packing LeVel

LL_SPI_RX_FIFO_0PACKET

LL_SPI_RX_FIFO_1PACKET

LL_SPI_RX_FIFO_2PACKET

LL_SPI_RX_FIFO_3PACKET

SS Idleness

LL_SPI_SS_IDLENESS_00CYCLE

LL_SPI_SS_IDLENESS_01CYCLE

LL_SPI_SS_IDLENESS_02CYCLE

LL_SPI_SS_IDLENESS_03CYCLE

LL_SPI_SS_IDLENESS_04CYCLE

LL_SPI_SS_IDLENESS_05CYCLE

LL_SPI_SS_IDLENESS_06CYCLE

LL_SPI_SS_IDLENESS_07CYCLE

LL_SPI_SS_IDLENESS_08CYCLE

LL_SPI_SS_IDLENESS_09CYCLE

LL_SPI_SS_IDLENESS_10CYCLE

LL_SPI_SS_IDLENESS_11CYCLE

LL_SPI_SS_IDLENESS_12CYCLE

LL_SPI_SS_IDLENESS_13CYCLE

LL_SPI_SS_IDLENESS_14CYCLE

LL_SPI_SS_IDLENESS_15CYCLE

SS Level

LL_SPI_SS_LEVEL_HIGH

LL_SPI_SS_LEVEL_LOW

Transfer Mode

LL_SPI_FULL_DUPLEX

LL_SPI_SIMPLEX_TX

LL_SPI_SIMPLEX_RX

LL_SPI_HALF_DUPLEX_RX

LL_SPI_HALF_DUPLEX_TX

TXCRC Init All

LL_SPI_TXCRCINIT_ALL_ZERO_PATTERN

LL_SPI_TXCRCINIT_ALL_ONES_PATTERN

UDR Config Register

LL_SPI_UDR_CONFIG_REGISTER_PATTERN

LL_SPI_UDR_CONFIG_LAST_RECEIVED

LL_SPI_UDR_CONFIG_LAST_TRANSMITTED

UDR Detect Begin Data

LL_SPI_UDR_DETECT_BEGIN_DATA_FRAME

LL_SPI_UDR_DETECT_END_DATA_FRAME

LL_SPI_UDR_DETECT_BEGIN_ACTIVE_NSS

Common Write and read registers Macros

LL_SPI_WriteReg

Description:

- Write a value in SPI register.

Parameters:

- __INSTANCE__: SPI Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_SPI_ReadReg

Description:

- Read a value in SPI register.

Parameters:

- __INSTANCE__: SPI Instance
- __REG__: Register to be read

Return value:

- Register: value

124 LL SWPMI Generic Driver

124.1 SWPMI Firmware driver registers structures

124.1.1 LL_SWPMI_InitTypeDef

LL_SWPMI_InitTypeDef is defined in the stm32h7xx_ll_swpmi.h

Data Fields

- ***uint32_t VoltageClass***
- ***uint32_t BitRatePrescaler***
- ***uint32_t TxBufferingMode***
- ***uint32_t RxBufferingMode***

Field Documentation

- ***uint32_t LL_SWPMI_InitTypeDef::VoltageClass***

  Specifies the SWP Voltage Class. This parameter can be a value of ***SWPMI_LL_EC_VOLTAGE_CLASS***This feature can be modified afterwards using unitary function **LL_SWPMI_SetVoltageClass**.
- ***uint32_t LL_SWPMI_InitTypeDef::BitRatePrescaler***

  Specifies the SWPMI bitrate prescaler. This parameter must be a number between Min_Data=0 and Max_Data=255U.The value can be calculated thanks to helper macro **__LL_SWPMI_CALC_BITRATE_PRESCALER**This feature can be modified afterwards using unitary function **LL_SWPMI_SetBitRatePrescaler**.
- ***uint32_t LL_SWPMI_InitTypeDef::TxBufferingMode***

  Specifies the transmission buffering mode. This parameter can be a value of ***SWPMI_LL_EC_SW_BUFFER_TX***This feature can be modified afterwards using unitary function **LL_SWPMI_SetTransmissionMode**.
- ***uint32_t LL_SWPMI_InitTypeDef::RxBufferingMode***

  Specifies the reception buffering mode. This parameter can be a value of ***SWPMI_LL_EC_SW_BUFFER_RX***This feature can be modified afterwards using unitary function **LL_SWPMI_SetReceptionMode**.

124.2 SWPMI Firmware driver API description

The following section lists the various functions of the SWPMI library.

124.2.1 Detailed description of functions

LL_SWPMI_SetReceptionMode

Function name

__STATIC_INLINE void LL_SWPMI_SetReceptionMode (SWPMI_TypeDef * SWPMIx, uint32_t RxBufferingMode)

Function description

Set Reception buffering mode.

Parameters

- **SWPMIx:** SWPMI Instance
- **RxBufferingMode:** This parameter can be one of the following values:
  - LL_SWPMI_SW_BUFFER_RX_SINGLE
  - LL_SWPMI_SW_BUFFER_RX_MULTI

Return values

- **None:**

Notes

- If Multi software buffer mode is chosen, RXDMA bits must also be set.

Reference Manual to LL API cross reference:

- CR RXMODE LL_SWPMI_SetReceptionMode

LL_SWPMI_GetReceptionMode

Function name

__STATIC_INLINE uint32_t LL_SWPMI_GetReceptionMode (SWPMI_TypeDef * SWPMIx)

Function description

Get Reception buffering mode.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SWPMI_SW_BUFFER_RX_SINGLE
  - LL_SWPMI_SW_BUFFER_RX_MULTI

Reference Manual to LL API cross reference:

- CR RXMODE LL_SWPMI_GetReceptionMode

LL_SWPMI_SetTransmissionMode

Function name

__STATIC_INLINE void LL_SWPMI_SetTransmissionMode (SWPMI_TypeDef * SWPMIx, uint32_t TxBufferingMode)

Function description

Set Transmission buffering mode.

Parameters

- **SWPMIx:** SWPMI Instance
- **TxBufferingMode:** This parameter can be one of the following values:
  - LL_SWPMI_SW_BUFFER_TX_SINGLE
  - LL_SWPMI_SW_BUFFER_TX_MULTI

Return values

- **None:**

Notes

- If Multi software buffer mode is chosen, TXDMA bits must also be set.

Reference Manual to LL API cross reference:

- CR TXMODE LL_SWPMI_SetTransmissionMode

LL_SWPMI_GetTransmissionMode

Function name

__STATIC_INLINE uint32_t LL_SWPMI_GetTransmissionMode (SWPMI_TypeDef * SWPMIx)

Function description

Get Transmission buffering mode.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SWPMI_SW_BUFFER_TX_SINGLE
  - LL_SWPMI_SW_BUFFER_TX_MULTI

Reference Manual to LL API cross reference:

- CR TXMODE LL_SWPMI_GetTransmissionMode

LL_SWPMI_EnableLoopback

Function name

__STATIC_INLINE void LL_SWPMI_EnableLoopback (SWPMI_TypeDef * SWPMIx)

Function description

Enable loopback mode.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR LPBK LL_SWPMI_EnableLoopback

LL_SWPMI_DisableLoopback

Function name

__STATIC_INLINE void LL_SWPMI_DisableLoopback (SWPMI_TypeDef * SWPMIx)

Function description

Disable loopback mode.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR LPBK LL_SWPMI_DisableLoopback

LL_SWPMI_EnableTransceiver

Function name

__STATIC_INLINE void LL_SWPMI_EnableTransceiver (SWPMI_TypeDef * SWPMIx)

Function description

Enable SWPMI transceiver.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Notes

- SWPMI_IO pin is controlled by SWPMI

Reference Manual to LL API cross reference:

- CR SWPEN LL_SWPMI_EnableTransceiver

LL_SWPMI_DisableTransceiver

Function name

__STATIC_INLINE void LL_SWPMI_DisableTransceiver (SWPMI_TypeDef * SWPMIx)

Function description

Disable SWPMI transceiver.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Notes

- SWPMI_IO pin is controlled by GPIO controller

Reference Manual to LL API cross reference:

- CR SWPEN LL_SWPMI_DisableTransceiver

LL_SWPMI_IsEnabledTransceiver

Function name

__STATIC_INLINE uint32_t LL_SWPMI_IsEnabledTransceiver (SWPMI_TypeDef * SWPMIx)

Function description

Check if SWPMI transceiver is enabled.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR SWPEN LL_SWPMI_IsEnabledTransceiver

LL_SWPMI_Activate

Function name

__STATIC_INLINE void LL_SWPMI_Activate (SWPMI_TypeDef * SWPMIx)

Function description

Activate Single wire protocol bus (SUSPENDED or ACTIVATED state)

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Notes

- SWP bus stays in the ACTIVATED state as long as there is a communication with the slave, either in transmission or in reception. The SWP bus switches back to the SUSPENDED state as soon as there is no more transmission or reception activity, after 7 idle bits.

Reference Manual to LL API cross reference:

- CR SWPACT LL_SWPMI_Activate

LL_SWPMI_IsActivated

Function name

__STATIC_INLINE uint32_t LL_SWPMI_IsActivated (SWPMI_TypeDef * SWPMIx)

Function description

Check if Single wire protocol bus is in ACTIVATED state.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR SWPACT LL_SWPMI_Activate

LL_SWPMI_Deactivate

Function name

__STATIC_INLINE void LL_SWPMI_Deactivate (SWPMI_TypeDef * SWPMIx)

Function description

Deactivate immediately Single wire protocol bus (immediate transition to DEACTIVATED state)

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SWPACT LL_SWPMI_Deactivate

LL_SWPMI_RequestDeactivation

Function name

__STATIC_INLINE void LL_SWPMI_RequestDeactivation (SWPMI_TypeDef * SWPMIx)

Function description

Request a deactivation of Single wire protocol bus (request to go in DEACTIVATED state if no resume from slave)

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DEACT LL_SWPMI_RequestDeactivation

LL_SWPMI_SetBitRatePrescaler

Function name

__STATIC_INLINE void LL_SWPMI_SetBitRatePrescaler (SWPMI_TypeDef * SWPMIx, uint32_t BitRatePrescaler)

Function description

Set Bitrate prescaler SWPMI_freq = SWPMI_clk / (((BitRate) + 1) * 4)

Parameters

- **SWPMIx:** SWPMI Instance
- **BitRatePrescaler:** A number between Min_Data=0 and Max_Data=255U

Return values

- **None:**

Reference Manual to LL API cross reference:

- BRR BR LL_SWPMI_SetBitRatePrescaler

LL_SWPMI_GetBitRatePrescaler

Function name

__STATIC_INLINE uint32_t LL_SWPMI_GetBitRatePrescaler (SWPMI_TypeDef * SWPMIx)

Function description

Get Bitrate prescaler.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **A:** number between Min_Data=0 and Max_Data=255U

Reference Manual to LL API cross reference:

- BRR BR LL_SWPMI_GetBitRatePrescaler

LL_SWPMI_SetVoltageClass

Function name

__STATIC_INLINE void LL_SWPMI_SetVoltageClass (SWPMI_TypeDef * SWPMIx, uint32_t VoltageClass)

Function description

Set SWP Voltage Class.

Parameters

- **SWPMIx:** SWPMI Instance
- **VoltageClass:** This parameter can be one of the following values:
  - LL_SWPMI_VOLTAGE_CLASS_C
  - LL_SWPMI_VOLTAGE_CLASS_B

Return values

- **None:**

Reference Manual to LL API cross reference:

- OR CLASS LL_SWPMI_SetVoltageClass

LL_SWPMI_GetVoltageClass

Function name

__STATIC_INLINE uint32_t LL_SWPMI_GetVoltageClass (SWPMI_TypeDef * SWPMIx)

Function description

Get SWP Voltage Class.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SWPMI_VOLTAGE_CLASS_C
  - LL_SWPMI_VOLTAGE_CLASS_B

Reference Manual to LL API cross reference:

- OR CLASS LL_SWPMI_GetVoltageClass

LL_SWPMI_IsActiveFlag_RXBF

Function name

__STATIC_INLINE uint32_t LL_SWPMI_IsActiveFlag_RXBF (SWPMI_TypeDef * SWPMIx)

Function description

Check if the last word of the frame under reception has arrived in SWPMI_RDR.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RXBFF LL_SWPMI_IsActiveFlag_RXBF

LL_SWPMI_IsActiveFlag_TXBE

Function name

__STATIC_INLINE uint32_t LL_SWPMI_IsActiveFlag_TXBE (SWPMI_TypeDef * SWPMIx)

Function description

Check if Frame transmission buffer has been emptied.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TXBEF LL_SWPMI_IsActiveFlag_TXBE

LL_SWPMI_IsActiveFlag_RXBER

Function name

__STATIC_INLINE uint32_t LL_SWPMI_IsActiveFlag_RXBER (SWPMI_TypeDef * SWPMIx)

Function description

Check if CRC error in reception has been detected.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RXBERF LL_SWPMI_IsActiveFlag_RXBER

LL_SWPMI_IsActiveFlag_RXOVR

Function name

__STATIC_INLINE uint32_t LL_SWPMI_IsActiveFlag_RXOVR (SWPMI_TypeDef * SWPMIx)

Function description

Check if Overrun in reception has been detected.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RXOVRF LL_SWPMI_IsActiveFlag_RXOVR

LL_SWPMI_IsActiveFlag_TXUNR

Function name

__STATIC_INLINE uint32_t LL_SWPMI_IsActiveFlag_TXUNR (SWPMI_TypeDef * SWPMIx)

Function description

Check if underrun error in transmission has been detected.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TXUNRF LL_SWPMI_IsActiveFlag_TXUNR

LL_SWPMI_IsActiveFlag_RXNE

Function name

__STATIC_INLINE uint32_t LL_SWPMI_IsActiveFlag_RXNE (SWPMI_TypeDef * SWPMIx)

Function description

Check if Receive data register not empty (it means that Received data is ready to be read in the SWPMI_RDR register)

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RXNE LL_SWPMI_IsActiveFlag_RXNE

LL_SWPMI_IsActiveFlag_TXE

Function name

__STATIC_INLINE uint32_t LL_SWPMI_IsActiveFlag_TXE (SWPMI_TypeDef * SWPMIx)

Function description

Check if Transmit data register is empty (it means that Data written in transmit data register SWPMI_TDR has been transmitted and SWPMI_TDR can be written to again)

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TXE LL_SWPMI_IsActiveFlag_TXE

LL_SWPMI_IsActiveFlag_TC

Function name

__STATIC_INLINE uint32_t LL_SWPMI_IsActiveFlag_TC (SWPMI_TypeDef * SWPMIx)

Function description

Check if Both transmission and reception are completed and SWP is switched to the SUSPENDED state.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCF LL_SWPMI_IsActiveFlag_TC

LL_SWPMI_IsActiveFlag_SR

Function name

__STATIC_INLINE uint32_t LL_SWPMI_IsActiveFlag_SR (SWPMI_TypeDef * SWPMIx)

Function description

Check if a Resume by slave state has been detected during the SWP bus SUSPENDED state.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR SRF LL_SWPMI_IsActiveFlag_SR

LL_SWPMI_IsActiveFlag_SUSP

Function name

__STATIC_INLINE uint32_t LL_SWPMI_IsActiveFlag_SUSP (SWPMI_TypeDef * SWPMIx)

Function description

Check if SWP bus is in SUSPENDED or DEACTIVATED state.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR SUSP LL_SWPMI_IsActiveFlag_SUSP

LL_SWPMI_IsActiveFlag_DEACT

Function name

__STATIC_INLINE uint32_t LL_SWPMI_IsActiveFlag_DEACT (SWPMI_TypeDef * SWPMIx)

Function description

Check if SWP bus is in DEACTIVATED state.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR DEACTF LL_SWPMI_IsActiveFlag_DEACT

LL_SWPMI_IsActiveFlag_RDYF

Function name

__STATIC_INLINE uint32_t LL_SWPMI_IsActiveFlag_RDYF (SWPMI_TypeDef * SWPMIx)

Function description

Check if SWPMI transceiver is ready.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RDYF LL_SWPMI_IsActiveFlag_RDYF

LL_SWPMI_ClearFlag_RXBF

Function name

__STATIC_INLINE void LL_SWPMI_ClearFlag_RXBF (SWPMI_TypeDef * SWPMIx)

Function description

Clear receive buffer full flag.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CRXBFF LL_SWPMI_ClearFlag_RXBF

LL_SWPMI_ClearFlag_TXBE

Function name

__STATIC_INLINE void LL_SWPMI_ClearFlag_TXBE (SWPMI_TypeDef * SWPMIx)

Function description

Clear transmit buffer empty flag.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CTXBEF LL_SWPMI_ClearFlag_TXBE

LL_SWPMI_ClearFlag_RXBER

Function name

__STATIC_INLINE void LL_SWPMI_ClearFlag_RXBER (SWPMI_TypeDef * SWPMIx)

Function description

Clear receive CRC error flag.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CRXBERF LL_SWPMI_ClearFlag_RXBER

LL_SWPMI_ClearFlag_RXOVR

Function name

__STATIC_INLINE void LL_SWPMI_ClearFlag_RXOVR (SWPMI_TypeDef * SWPMIx)

Function description

Clear receive overrun error flag.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CRXOVRF LL_SWPMI_ClearFlag_RXOVR

LL_SWPMI_ClearFlag_TXUNR

Function name

__STATIC_INLINE void LL_SWPMI_ClearFlag_TXUNR (SWPMI_TypeDef * SWPMIx)

Function description

Clear transmit underrun error flag.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CTXUNRF LL_SWPMI_ClearFlag_TXUNR

LL_SWPMI_ClearFlag_TC

Function name

__STATIC_INLINE void LL_SWPMI_ClearFlag_TC (SWPMI_TypeDef * SWPMIx)

Function description

Clear transfer complete flag.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CTCF LL_SWPMI_ClearFlag_TC

LL_SWPMI_ClearFlag_SR

Function name

__STATIC_INLINE void LL_SWPMI_ClearFlag_SR (SWPMI_TypeDef * SWPMIx)

Function description

Clear slave resume flag.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CSRF LL_SWPMI_ClearFlag_SR

LL_SWPMI_ClearFlag_RDY

Function name

__STATIC_INLINE void LL_SWPMI_ClearFlag_RDY (SWPMI_TypeDef * SWPMIx)

Function description

Clear SWPMI transceiver ready flag.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR CRDYF LL_SWPMI_ClearFlag_RDY

LL_SWPMI_EnableIT_RDY

Function name

__STATIC_INLINE void LL_SWPMI_EnableIT_RDY (SWPMI_TypeDef * SWPMIx)

Function description

Enable SWPMI transceiver ready interrupt.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RDYIE LL_SWPMI_EnableIT_RDY

LL_SWPMI_EnableIT_SR

Function name

__STATIC_INLINE void LL_SWPMI_EnableIT_SR (SWPMI_TypeDef * SWPMIx)

Function description

Enable Slave resume interrupt.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER SRIE LL_SWPMI_EnableIT_SR

LL_SWPMI_EnableIT_TC

Function name

__STATIC_INLINE void LL_SWPMI_EnableIT_TC (SWPMI_TypeDef * SWPMIx)

Function description

Enable Transmit complete interrupt.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TCIE LL_SWPMI_EnableIT_TC

LL_SWPMI_EnableIT_TX

Function name

__STATIC_INLINE void LL_SWPMI_EnableIT_TX (SWPMI_TypeDef * SWPMIx)

Function description

Enable Transmit interrupt.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TIE LL_SWPMI_EnableIT_TX

LL_SWPMI_EnableIT_RX

Function name

__STATIC_INLINE void LL_SWPMI_EnableIT_RX (SWPMI_TypeDef * SWPMIx)

Function description

Enable Receive interrupt.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RIE LL_SWPMI_EnableIT_RX

LL_SWPMI_EnableIT_TXUNR

Function name

__STATIC_INLINE void LL_SWPMI_EnableIT_TXUNR (SWPMI_TypeDef * SWPMIx)

Function description

Enable Transmit underrun error interrupt.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TXUNRIE LL_SWPMI_EnableIT_TXUNR

LL_SWPMI_EnableIT_RXOVR

Function name

__STATIC_INLINE void LL_SWPMI_EnableIT_RXOVR (SWPMI_TypeDef * SWPMIx)

Function description

Enable Receive overrun error interrupt.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RXOVRIE LL_SWPMI_EnableIT_RXOVR

LL_SWPMI_EnableIT_RXBER

Function name

__STATIC_INLINE void LL_SWPMI_EnableIT_RXBER (SWPMI_TypeDef * SWPMIx)

Function description

Enable Receive CRC error interrupt.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RXBERIE LL_SWPMI_EnableIT_RXBER

LL_SWPMI_EnableIT_TXBE

Function name

__STATIC_INLINE void LL_SWPMI_EnableIT_TXBE (SWPMI_TypeDef * SWPMIx)

Function description

Enable Transmit buffer empty interrupt.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TXBEIE LL_SWPMI_EnableIT_TXBE

LL_SWPMI_EnableIT_RXBF

Function name

__STATIC_INLINE void LL_SWPMI_EnableIT_RXBF (SWPMI_TypeDef * SWPMIx)

Function description

Enable Receive buffer full interrupt.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RXBFIE LL_SWPMI_EnableIT_RXBF

LL_SWPMI_DisableIT_RDY

Function name

__STATIC_INLINE void LL_SWPMI_DisableIT_RDY (SWPMI_TypeDef * SWPMIx)

Function description

Disable SWPMI transceiver ready interrupt.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RDYIE LL_SWPMI_DisableIT_RDY

LL_SWPMI_DisableIT_SR

Function name

__STATIC_INLINE void LL_SWPMI_DisableIT_SR (SWPMI_TypeDef * SWPMIx)

Function description

Disable Slave resume interrupt.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER SRIE LL_SWPMI_DisableIT_SR

LL_SWPMI_DisableIT_TC

Function name

__STATIC_INLINE void LL_SWPMI_DisableIT_TC (SWPMI_TypeDef * SWPMIx)

Function description

Disable Transmit complete interrupt.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TCIE LL_SWPMI_DisableIT_TC

LL_SWPMI_DisableIT_TX

Function name

__STATIC_INLINE void LL_SWPMI_DisableIT_TX (SWPMI_TypeDef * SWPMIx)

Function description

Disable Transmit interrupt.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TIE LL_SWPMI_DisableIT_TX

LL_SWPMI_DisableIT_RX

Function name

__STATIC_INLINE void LL_SWPMI_DisableIT_RX (SWPMI_TypeDef * SWPMIx)

Function description

Disable Receive interrupt.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RIE LL_SWPMI_DisableIT_RX

LL_SWPMI_DisableIT_TXUNR

Function name

__STATIC_INLINE void LL_SWPMI_DisableIT_TXUNR (SWPMI_TypeDef * SWPMIx)

Function description

Disable Transmit underrun error interrupt.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TXUNRIE LL_SWPMI_DisableIT_TXUNR

LL_SWPMI_DisableIT_RXOVR

Function name

__STATIC_INLINE void LL_SWPMI_DisableIT_RXOVR (SWPMI_TypeDef * SWPMIx)

Function description

Disable Receive overrun error interrupt.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RXOVRIE LL_SWPMI_DisableIT_RXOVR

LL_SWPMI_DisableIT_RXBER

Function name

__STATIC_INLINE void LL_SWPMI_DisableIT_RXBER (SWPMI_TypeDef * SWPMIx)

Function description

Disable Receive CRC error interrupt.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RXBERIE LL_SWPMI_DisableIT_RXBER

LL_SWPMI_DisableIT_TXBE

Function name

__STATIC_INLINE void LL_SWPMI_DisableIT_TXBE (SWPMI_TypeDef * SWPMIx)

Function description

Disable Transmit buffer empty interrupt.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TXBEIE LL_SWPMI_DisableIT_TXBE

LL_SWPMI_DisableIT_RXBF

Function name

__STATIC_INLINE void LL_SWPMI_DisableIT_RXBF (SWPMI_TypeDef * SWPMIx)

Function description

Disable Receive buffer full interrupt.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RXBFIE LL_SWPMI_DisableIT_RXBF

LL_SWPMI_IsEnabledIT_RDY

Function name

__STATIC_INLINE uint32_t LL_SWPMI_IsEnabledIT_RDY (SWPMI_TypeDef * SWPMIx)

Function description

Check if SWPMI transceiver ready interrupt is enabled.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER RDYIE LL_SWPMI_IsEnabledIT_RDY

LL_SWPMI_IsEnabledIT_SR

Function name

__STATIC_INLINE uint32_t LL_SWPMI_IsEnabledIT_SR (SWPMI_TypeDef * SWPMIx)

Function description

Check if Slave resume interrupt is enabled.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER SRIE LL_SWPMI_IsEnabledIT_SR

LL_SWPMI_IsEnabledIT_TC

Function name

__STATIC_INLINE uint32_t LL_SWPMI_IsEnabledIT_TC (SWPMI_TypeDef * SWPMIx)

Function description

Check if Transmit complete interrupt is enabled.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER TCIE LL_SWPMI_IsEnabledIT_TC

LL_SWPMI_IsEnabledIT_TX

Function name

__STATIC_INLINE uint32_t LL_SWPMI_IsEnabledIT_TX (SWPMI_TypeDef * SWPMIx)

Function description

Check if Transmit interrupt is enabled.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER TIE LL_SWPMI_IsEnabledIT_TX

LL_SWPMI_IsEnabledIT_RX

Function name

__STATIC_INLINE uint32_t LL_SWPMI_IsEnabledIT_RX (SWPMI_TypeDef * SWPMIx)

Function description

Check if Receive interrupt is enabled.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER RIE LL_SWPMI_IsEnabledIT_RX

LL_SWPMI_IsEnabledIT_TXUNR

Function name

__STATIC_INLINE uint32_t LL_SWPMI_IsEnabledIT_TXUNR (SWPMI_TypeDef * SWPMIx)

Function description

Check if Transmit underrun error interrupt is enabled.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER TXUNRIE LL_SWPMI_IsEnabledIT_TXUNR

LL_SWPMI_IsEnabledIT_RXOVR

Function name

__STATIC_INLINE uint32_t LL_SWPMI_IsEnabledIT_RXOVR (SWPMI_TypeDef * SWPMIx)

Function description

Check if Receive overrun error interrupt is enabled.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER RXOVRIE LL_SWPMI_IsEnabledIT_RXOVR

LL_SWPMI_IsEnabledIT_RXBER

Function name

__STATIC_INLINE uint32_t LL_SWPMI_IsEnabledIT_RXBER (SWPMI_TypeDef * SWPMIx)

Function description

Check if Receive CRC error interrupt is enabled.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER RXBERIE LL_SWPMI_IsEnabledIT_RXBER

LL_SWPMI_IsEnabledIT_TXBE

Function name

__STATIC_INLINE uint32_t LL_SWPMI_IsEnabledIT_TXBE (SWPMI_TypeDef * SWPMIx)

Function description

Check if Transmit buffer empty interrupt is enabled.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER TXBEIE LL_SWPMI_IsEnabledIT_TXBE

LL_SWPMI_IsEnabledIT_RXBF

Function name

__STATIC_INLINE uint32_t LL_SWPMI_IsEnabledIT_RXBF (SWPMI_TypeDef * SWPMIx)

Function description

Check if Receive buffer full interrupt is enabled.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER RXBFIE LL_SWPMI_IsEnabledIT_RXBF

LL_SWPMI_EnableDMAReq_RX

Function name

__STATIC_INLINE void LL_SWPMI_EnableDMAReq_RX (SWPMI_TypeDef * SWPMIx)

Function description

Enable DMA mode for reception.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RXDMA LL_SWPMI_EnableDMAReq_RX

LL_SWPMI_DisableDMAReq_RX

Function name

__STATIC_INLINE void LL_SWPMI_DisableDMAReq_RX (SWPMI_TypeDef * SWPMIx)

Function description

Disable DMA mode for reception.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RXDMA LL_SWPMI_DisableDMAReq_RX

LL_SWPMI_IsEnabledDMAReq_RX

Function name

__STATIC_INLINE uint32_t LL_SWPMI_IsEnabledDMAReq_RX (SWPMI_TypeDef * SWPMIx)

Function description

Check if DMA mode for reception is enabled.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR RXDMA LL_SWPMI_IsEnabledDMAReq_RX

LL_SWPMI_EnableDMAReq_TX

Function name

__STATIC_INLINE void LL_SWPMI_EnableDMAReq_TX (SWPMI_TypeDef * SWPMIx)

Function description

Enable DMA mode for transmission.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TXDMA LL_SWPMI_EnableDMAReq_TX

LL_SWPMI_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_SWPMI_DisableDMAReq_TX (SWPMI_TypeDef * SWPMIx)

Function description

Disable DMA mode for transmission.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TXDMA LL_SWPMI_DisableDMAReq_TX

LL_SWPMI_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_SWPMI_IsEnabledDMAReq_TX (SWPMI_TypeDef * SWPMIx)

Function description

Check if DMA mode for transmission is enabled.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR TXDMA LL_SWPMI_IsEnabledDMAReq_TX

LL_SWPMI_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_SWPMI_DMA_GetRegAddr (SWPMI_TypeDef * SWPMIx, uint32_t Direction)

Function description

Get the data register address used for DMA transfer.

Parameters

- **SWPMIx:** SWPMI Instance
- **Direction:** This parameter can be one of the following values:
  - LL_SWPMI_DMA_REG_DATA_TRANSMIT
  - LL_SWPMI_DMA_REG_DATA_RECEIVE

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- TDR TD LL_SWPMI_DMA_GetRegAddr
- RDR RD LL_SWPMI_DMA_GetRegAddr

LL_SWPMI_GetReceiveFrameLength

Function name

__STATIC_INLINE uint32_t LL_SWPMI_GetReceiveFrameLength (SWPMI_TypeDef * SWPMIx)

Function description

Retrieve number of data bytes present in payload of received frame.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x1F

Reference Manual to LL API cross reference:

- RFL RFL LL_SWPMI_GetReceiveFrameLength

LL_SWPMI_TransmitData32

Function name

__STATIC_INLINE void LL_SWPMI_TransmitData32 (SWPMI_TypeDef * SWPMIx, uint32_t TxData)

Function description

Transmit Data Register.

Parameters

- **SWPMIx:** SWPMI Instance
- **TxData:** Value between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TDR TD LL_SWPMI_TransmitData32

LL_SWPMI_ReceiveData32

Function name

__STATIC_INLINE uint32_t LL_SWPMI_ReceiveData32 (SWPMI_TypeDef * SWPMIx)

Function description

Receive Data Register.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **Value:** between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- RDR RD LL_SWPMI_ReceiveData32

LL_SWPMI_EnableTXBypass

Function name

__STATIC_INLINE void LL_SWPMI_EnableTXBypass (SWPMI_TypeDef * SWPMIx)

Function description

Enable SWP Transceiver Bypass.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Notes

- The external interface for SWPMI is SWPMI_IO (SWPMI_RX, SWPMI_TX and SWPMI_SUSPEND signals are not available on GPIOs)

Reference Manual to LL API cross reference:

- OR TBYP LL_SWPMI_EnableTXBypass

LL_SWPMI_DisableTXBypass

Function name

__STATIC_INLINE void LL_SWPMI_DisableTXBypass (SWPMI_TypeDef * SWPMIx)

Function description

Disable SWP Transceiver Bypass.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **None:**

Notes

- SWPMI_RX, SWPMI_TX and SWPMI_SUSPEND signals are available as alternate function on GPIOs. This configuration is selected to connect an external transceiver
- In SWPMI_IO bypass mode, SWPEN bit in SWPMI_CR register must be kept cleared

Reference Manual to LL API cross reference:

- OR TBYP LL_SWPMI_DisableTXBypass

LL_SWPMI_DeInit

Function name

ErrorStatus LL_SWPMI_DeInit (SWPMI_TypeDef * SWPMIx)

Function description

De-initialize the SWPMI peripheral registers to their default reset values.

Parameters

- **SWPMIx:** SWPMI Instance

Return values

- **An:** ErrorStatus enumeration value
  - SUCCESS: SWPMI registers are de-initialized
  - ERROR: Not applicable

LL_SWPMI_Init

Function name

ErrorStatus LL_SWPMI_Init (SWPMI_TypeDef * SWPMIx, LL_SWPMI_InitTypeDef * SWPMI_InitStruct)

Function description

Initialize the SWPMI peripheral according to the specified parameters in the SWPMI_InitStruct.

Parameters

- **SWPMIx:** SWPMI Instance
- **SWPMI_InitStruct:** pointer to a LL_SWPMI_InitTypeDef structure that contains the configuration information for the SWPMI peripheral.

Return values

- **An:** ErrorStatus enumeration value
  - SUCCESS: SWPMI registers are initialized
  - ERROR: SWPMI registers are not initialized

Notes

- As some bits in SWPMI configuration registers can only be written when the SWPMI is deactivated (SWPMI_CR_SWPACT bit = 0), the SWPMI peripheral should be in deactivated state prior calling this function. Otherwise, ERROR result will be returned.

LL_SWPMI_StructInit

Function name

void LL_SWPMI_StructInit (LL_SWPMI_InitTypeDef * SWPMI_InitStruct)

Function description

Set each LL_SWPMI_InitTypeDef field to default value.

Parameters

- **SWPMI_InitStruct:** pointer to a LL_SWPMI_InitTypeDef structure that contains the configuration information for the SWPMI peripheral.

Return values

- **None:**

124.3 SWPMI Firmware driver defines

The following section lists the various define and macros of the module.

124.3.1 SWPMI

SWPMI

Clear Flags Defines

LL_SWPMI_ICR_CRXBFF

Clear receive buffer full flag

LL_SWPMI_ICR_CTXBEF

Clear transmit buffer empty flag

LL_SWPMI_ICR_CRXBERF

Clear receive CRC error flag

LL_SWPMI_ICR_CRXOVRF

Clear receive overrun error flag

LL_SWPMI_ICR_CTXUNRF

Clear transmit underrun error flag

LL_SWPMI_ICR_CTCF

Clear transfer complete flag

LL_SWPMI_ICR_CSRF

Clear slave resume flag

DMA register data

LL_SWPMI_DMA_REG_DATA_TRANSMIT

Get address of data register used for transmission

LL_SWPMI_DMA_REG_DATA_RECEIVE

Get address of data register used for reception

Get Flags Defines

LL_SWPMI_ISR_RXBFF

Receive buffer full flag

LL_SWPMI_ISR_TXBEF

Transmit buffer empty flag

LL_SWPMI_ISR_RXBERF

Receive CRC error flag

LL_SWPMI_ISR_RXOVRF

Receive overrun error flag

LL_SWPMI_ISR_TXUNRF

Transmit underrun error flag

LL_SWPMI_ISR_RXNE

Receive data register not empty

LL_SWPMI_ISR_TXE

Transmit data register empty

LL_SWPMI_ISR_TCF

Transfer complete flag

LL_SWPMI_ISR_SRF

Slave resume flag

LL_SWPMI_ISR_SUSP

SUSPEND flag

LL_SWPMI_ISR_DEACTF

DEACTIVATED flag

IT Defines

LL_SWPMI_IER_SRIE

Slave resume interrupt enable

LL_SWPMI_IER_TCIE

Transmit complete interrupt enable

LL_SWPMI_IER_TIE

Transmit interrupt enable

LL_SWPMI_IER_RIE

Receive interrupt enable

LL_SWPMI_IER_TXUNRIE

Transmit underrun error interrupt enable

LL_SWPMI_IER_RXOVRIE

Receive overrun error interrupt enable

LL_SWPMI_IER_RXBERIE

Receive CRC error interrupt enable

LL_SWPMI_IER_TXBEIE

Transmit buffer empty interrupt enable

LL_SWPMI_IER_RXBFIE

Receive buffer full interrupt enable

SW BUFFER RX

LL_SWPMI_SW_BUFFER_RX_SINGLE

Single software buffer mode for reception

LL_SWPMI_SW_BUFFER_RX_MULTI

Multi software buffermode for reception

SW BUFFER TX

LL_SWPMI_SW_BUFFER_TX_SINGLE

Single software buffer mode for transmission

LL_SWPMI_SW_BUFFER_TX_MULTI

Multi software buffermode for transmission

VOLTAGE CLASS

LL_SWPMI_VOLTAGE_CLASS_C

SWPMI_IO uses directly VDD voltage to operate in class C

LL_SWPMI_VOLTAGE_CLASS_B

SWPMI_IO uses an internal voltage regulator to operate in class B

Bit rate calculation helper Macros

__LL_SWPMI_CALC_BITRATE_PRESCALER

Description:

- Helper macro to calculate bit rate value to set in BRR register (

Parameters:

- __FSWP__: Within the following range: from 100 Kbit/s up to 2Mbit/s (in bit/s)
- __FSWPCLK__: PCLK or HSI frequency (in Hz)

Return value:

- Bitrate: prescaler (BRR register)

Notes:

- ex: __LL_SWPMI_CALC_BITRATE_PRESCALER(2000000, 80000000);

Common Write and read registers Macros

LL_SWPMI_WriteReg

Description:

- Write a value in SWPMI register.

Parameters:

- __INSTANCE__: SWPMI Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_SWPMI_ReadReg

Description:

- Read a value in SWPMI register.

Parameters:

- __INSTANCE__: SWPMI Instance
- __REG__: Register to be read

Return value:

- Register: value

125 LL SYSTEM Generic Driver

125.1 SYSTEM Firmware driver API description

The following section lists the various functions of the SYSTEM library.

125.1.1 Detailed description of functions

LL_SYSCFG_SetPHYInterface

Function name

__STATIC_INLINE void LL_SYSCFG_SetPHYInterface (uint32_t Interface)

Function description

Select Ethernet PHY interface.

Parameters

- **Interface:** This parameter can be one of the following values:
  - LL_SYSCFG_ETH_MII
  - LL_SYSCFG_ETH_RMII

Return values

- **None:**

Reference Manual to LL API cross reference:

- PMCR EPIS_SEL LL_SYSCFG_SetPHYInterface

LL_SYSCFG_GetPHYInterface

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetPHYInterface (void)

Function description

Get Ethernet PHY interface.

Return values

- **Returned:** value can be one of the following values:
  - LL_SYSCFG_ETH_MII
  - LL_SYSCFG_ETH_RMII

Reference Manual to LL API cross reference:

- PMCR EPIS_SEL LL_SYSCFG_GetPHYInterface

LL_SYSCFG_OpenAnalogSwitch

Function name

__STATIC_INLINE void LL_SYSCFG_OpenAnalogSwitch (uint32_t AnalogSwitch)

Function description

Open an Analog Switch.

Parameters

- **AnalogSwitch:** This parameter can be one of the following values:
  - LL_SYSCFG_ANALOG_SWITCH_PA0 : PA0 analog switch
  - LL_SYSCFG_ANALOG_SWITCH_PA1: PA1 analog switch
  - LL_SYSCFG_ANALOG_SWITCH_PC2 : PC2 analog switch
  - LL_SYSCFG_ANALOG_SWITCH_PC3: PC3 analog switch

Return values

- **None:**

Reference Manual to LL API cross reference:

- PMCR PA0SO LL_SYSCFG_OpenAnalogSwitch
- PMCR PA1SO LL_SYSCFG_OpenAnalogSwitch
- PMCR PC2SO LL_SYSCFG_OpenAnalogSwitch
- PMCR PC3SO LL_SYSCFG_OpenAnalogSwitch

LL_SYSCFG_CloseAnalogSwitch

Function name

__STATIC_INLINE void LL_SYSCFG_CloseAnalogSwitch (uint32_t AnalogSwitch)

Function description

Close an Analog Switch.

Parameters

- **AnalogSwitch:** This parameter can be one of the following values:
  - LL_SYSCFG_ANALOG_SWITCH_PA0 : PA0 analog switch
  - LL_SYSCFG_ANALOG_SWITCH_PA1: PA1 analog switch
  - LL_SYSCFG_ANALOG_SWITCH_PC2 : PC2 analog switch
  - LL_SYSCFG_ANALOG_SWITCH_PC3: PC3 analog switch

Return values

- **None:**

Reference Manual to LL API cross reference:

- PMCR PA0SO LL_SYSCFG_CloseAnalogSwitch
- PMCR PA1SO LL_SYSCFG_CloseAnalogSwitch
- PMCR PC2SO LL_SYSCFG_CloseAnalogSwitch
- PMCR PC3SO LL_SYSCFG_CloseAnalogSwitch

LL_SYSCFG_EnableAnalogBooster

Function name

__STATIC_INLINE void LL_SYSCFG_EnableAnalogBooster (void)

Function description

Enable the Analog booster to reduce the total harmonic distortion of the analog switch when the supply voltage is lower than 2.7 V.

Return values

- **None:**

Notes

- Activating the booster allows to guaranty the analog switch AC performance when the supply voltage is below 2.7 V: in this case, the analog switch performance is the same on the full voltage range

Reference Manual to LL API cross reference:

- PMCR BOOSTEN LL_SYSCFG_EnableAnalogBooster

LL_SYSCFG_DisableAnalogBooster

Function name

__STATIC_INLINE void LL_SYSCFG_DisableAnalogBooster (void)

Function description

Disable the Analog booster.

Return values

- **None:**

Notes

- Activating the booster allows to guaranty the analog switch AC performance when the supply voltage is below 2.7 V: in this case, the analog switch performance is the same on the full voltage range

Reference Manual to LL API cross reference:

- PMCR BOOSTEN LL_SYSCFG_DisableAnalogBooster

LL_SYSCFG_EnableFastModePlus

Function name

__STATIC_INLINE void LL_SYSCFG_EnableFastModePlus (uint32_t ConfigFastModePlus)

Function description

Enable the I2C fast mode plus driving capability.

Parameters

- **ConfigFastModePlus:** This parameter can be a combination of the following values:
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB6
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB7
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB8 (*)
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB9 (*)
  - LL_SYSCFG_I2C_FASTMODEPLUS_I2C1
  - LL_SYSCFG_I2C_FASTMODEPLUS_I2C2 (*)
  - LL_SYSCFG_I2C_FASTMODEPLUS_I2C3
  - LL_SYSCFG_I2C_FASTMODEPLUS_I2C4 (*)
  - LL_SYSCFG_I2C_FASTMODEPLUS_I2C5 (*)

  (*) value not defined in all devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_PMCR I2C_PBx_FMP LL_SYSCFG_EnableFastModePlus
- SYSCFG_PMCR I2Cx_FMP LL_SYSCFG_EnableFastModePlus

LL_SYSCFG_DisableFastModePlus

Function name

__STATIC_INLINE void LL_SYSCFG_DisableFastModePlus (uint32_t ConfigFastModePlus)

Function description

Disable the I2C fast mode plus driving capability.

Parameters

- **ConfigFastModePlus:** This parameter can be a combination of the following values:
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB6
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB7
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB8 (*)
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB9 (*)
  - LL_SYSCFG_I2C_FASTMODEPLUS_I2C1
  - LL_SYSCFG_I2C_FASTMODEPLUS_I2C2 (*)
  - LL_SYSCFG_I2C_FASTMODEPLUS_I2C3
  - LL_SYSCFG_I2C_FASTMODEPLUS_I2C4
  - LL_SYSCFG_I2C_FASTMODEPLUS_I2C5 (*)

  (*) value not defined in all devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_PMCR I2C_PBx_FMP LL_SYSCFG_DisableFastModePlus
- SYSCFG_PMCR I2Cx_FMP LL_SYSCFG_DisableFastModePlus

LL_SYSCFG_SetEXTISource

Function name

__STATIC_INLINE void LL_SYSCFG_SetEXTISource (uint32_t Port, uint32_t Line)

Function description

Configure source input for the EXTI external interrupt.

Parameters

- **Port:** This parameter can be one of the following values:
  - LL_SYSCFG_EXTI_PORTA
  - LL_SYSCFG_EXTI_PORTB
  - LL_SYSCFG_EXTI_PORTC
  - LL_SYSCFG_EXTI_PORTD
  - LL_SYSCFG_EXTI_PORTE
  - LL_SYSCFG_EXTI_PORTF
  - LL_SYSCFG_EXTI_PORTG
  - LL_SYSCFG_EXTI_PORTH
  - LL_SYSCFG_EXTI_PORTI (*)
  - LL_SYSCFG_EXTI_PORTJ
  - LL_SYSCFG_EXTI_PORTK

  (*) value not defined in all devices
- **Line:** This parameter can be one of the following values:
  - LL_SYSCFG_EXTI_LINE0
  - LL_SYSCFG_EXTI_LINE1
  - LL_SYSCFG_EXTI_LINE2
  - LL_SYSCFG_EXTI_LINE3
  - LL_SYSCFG_EXTI_LINE4
  - LL_SYSCFG_EXTI_LINE5
  - LL_SYSCFG_EXTI_LINE6
  - LL_SYSCFG_EXTI_LINE7
  - LL_SYSCFG_EXTI_LINE8
  - LL_SYSCFG_EXTI_LINE9
  - LL_SYSCFG_EXTI_LINE10
  - LL_SYSCFG_EXTI_LINE11
  - LL_SYSCFG_EXTI_LINE12
  - LL_SYSCFG_EXTI_LINE13
  - LL_SYSCFG_EXTI_LINE14
  - LL_SYSCFG_EXTI_LINE15

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_EXTICR1 EXTIx LL_SYSCFG_SetEXTISource
- SYSCFG_EXTICR2 EXTIx LL_SYSCFG_SetEXTISource
- SYSCFG_EXTICR3 EXTIx LL_SYSCFG_SetEXTISource
- SYSCFG_EXTICR4 EXTIx LL_SYSCFG_SetEXTISource

LL_SYSCFG_GetEXTISource

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetEXTISource (uint32_t Line)

Function description

Get the configured defined for specific EXTI Line.

Parameters

- **Line:** This parameter can be one of the following values:
  - LL_SYSCFG_EXTI_LINE0
  - LL_SYSCFG_EXTI_LINE1
  - LL_SYSCFG_EXTI_LINE2
  - LL_SYSCFG_EXTI_LINE3
  - LL_SYSCFG_EXTI_LINE4
  - LL_SYSCFG_EXTI_LINE5
  - LL_SYSCFG_EXTI_LINE6
  - LL_SYSCFG_EXTI_LINE7
  - LL_SYSCFG_EXTI_LINE8
  - LL_SYSCFG_EXTI_LINE9
  - LL_SYSCFG_EXTI_LINE10
  - LL_SYSCFG_EXTI_LINE11
  - LL_SYSCFG_EXTI_LINE12
  - LL_SYSCFG_EXTI_LINE13
  - LL_SYSCFG_EXTI_LINE14
  - LL_SYSCFG_EXTI_LINE15

Return values

- **Returned:** value can be one of the following values:
  - LL_SYSCFG_EXTI_PORTA
  - LL_SYSCFG_EXTI_PORTB
  - LL_SYSCFG_EXTI_PORTC
  - LL_SYSCFG_EXTI_PORTD
  - LL_SYSCFG_EXTI_PORTE
  - LL_SYSCFG_EXTI_PORTF
  - LL_SYSCFG_EXTI_PORTG
  - LL_SYSCFG_EXTI_PORTH
  - LL_SYSCFG_EXTI_PORTI (*)
  - LL_SYSCFG_EXTI_PORTJ
  - LL_SYSCFG_EXTI_PORTK (*) value not defined in all devices

Reference Manual to LL API cross reference:

- SYSCFG_EXTICR1 EXTIx LL_SYSCFG_GetEXTISource
- SYSCFG_EXTICR2 EXTIx LL_SYSCFG_GetEXTISource
- SYSCFG_EXTICR3 EXTIx LL_SYSCFG_GetEXTISource
- SYSCFG_EXTICR4 EXTIx LL_SYSCFG_GetEXTISource

LL_SYSCFG_SetTIMBreakInputs

Function name

__STATIC_INLINE void LL_SYSCFG_SetTIMBreakInputs (uint32_t Break)

Function description

Set connections to TIM1/8/15/16/17 and HRTIM Break inputs.

Parameters

- **Break:** This parameter can be a combination of the following values:
  - LL_SYSCFG_TIMBREAK_AXISRAM_DBL_ECC
  - LL_SYSCFG_TIMBREAK_ITCM_DBL_ECC
  - LL_SYSCFG_TIMBREAK_DTCM_DBL_ECC
  - LL_SYSCFG_TIMBREAK_SRAM1_DBL_ECC
  - LL_SYSCFG_TIMBREAK_SRAM2_DBL_ECC
  - LL_SYSCFG_TIMBREAK_SRAM3_DBL_ECC (*)
  - LL_SYSCFG_TIMBREAK_SRAM4_DBL_ECC
  - LL_SYSCFG_TIMBREAK_BKRAM_DBL_ECC
  - LL_SYSCFG_TIMBREAK_CM7_LOCKUP
  - LL_SYSCFG_TIMBREAK_FLASH_DBL_ECC
  - LL_SYSCFG_TIMBREAK_PVD
  - LL_SYSCFG_TIMBREAK_CM4_LOCKUP (available for dual core lines only)

Return values

- **None:** (*) value not defined in all devices

Notes

- this feature is available on STM32H7 rev.B and above

Reference Manual to LL API cross reference:

- SYSCFG_CFGR AXISRAML LL_SYSCFG_SetTIMBreakInputs
- SYSCFG_CFGR ITCML LL_SYSCFG_SetTIMBreakInputs
- SYSCFG_CFGR DTCML LL_SYSCFG_SetTIMBreakInputs
- SYSCFG_CFGR SRAM1L LL_SYSCFG_SetTIMBreakInputs
- SYSCFG_CFGR SRAM2L LL_SYSCFG_SetTIMBreakInputs
- SYSCFG_CFGR SRAM3L LL_SYSCFG_SetTIMBreakInputs
- SYSCFG_CFGR SRAM4L LL_SYSCFG_SetTIMBreakInputs
- SYSCFG_CFGR BKRAML LL_SYSCFG_SetTIMBreakInputs
- SYSCFG_CFGR CM7L LL_SYSCFG_SetTIMBreakInputs
- SYSCFG_CFGR FLASHL LL_SYSCFG_SetTIMBreakInputs
- SYSCFG_CFGR PVDL LL_SYSCFG_SetTIMBreakInputs
- SYSCFG_CFGR_CM4L LL_SYSCFG_SetTIMBreakInputs

LL_SYSCFG_GetTIMBreakInputs

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetTIMBreakInputs (void)

Function description

Get connections to TIM1/8/15/16/17 and HRTIM Break inputs.

Return values

- **Returned:** value can be can be a combination of the following values:
  - LL_SYSCFG_TIMBREAK_AXISRAM_DBL_ECC
  - LL_SYSCFG_TIMBREAK_ITCM_DBL_ECC
  - LL_SYSCFG_TIMBREAK_DTCM_DBL_ECC
  - LL_SYSCFG_TIMBREAK_SRAM1_DBL_ECC
  - LL_SYSCFG_TIMBREAK_SRAM2_DBL_ECC
  - LL_SYSCFG_TIMBREAK_SRAM3_DBL_ECC (*)
  - LL_SYSCFG_TIMBREAK_SRAM4_DBL_ECC
  - LL_SYSCFG_TIMBREAK_BKRAM_DBL_ECC
  - LL_SYSCFG_TIMBREAK_CM7_LOCKUP
  - LL_SYSCFG_TIMBREAK_FLASH_DBL_ECC
  - LL_SYSCFG_TIMBREAK_PVD
  - LL_SYSCFG_TIMBREAK_CM4_LOCKUP (available for dual core lines only) (*) value not defined in all devices

Notes

- this feature is available on STM32H7 rev.B and above

Reference Manual to LL API cross reference:

- SYSCFG_CFGR AXISRAML LL_SYSCFG_GetTIMBreakInputs
- SYSCFG_CFGR ITCML LL_SYSCFG_GetTIMBreakInputs
- SYSCFG_CFGR DTCML LL_SYSCFG_GetTIMBreakInputs
- SYSCFG_CFGR SRAM1L LL_SYSCFG_GetTIMBreakInputs
- SYSCFG_CFGR SRAM2L LL_SYSCFG_GetTIMBreakInputs
- SYSCFG_CFGR SRAM3L LL_SYSCFG_GetTIMBreakInputs
- SYSCFG_CFGR SRAM4L LL_SYSCFG_GetTIMBreakInputs
- SYSCFG_CFGR BKRAML LL_SYSCFG_GetTIMBreakInputs
- SYSCFG_CFGR CM7L LL_SYSCFG_GetTIMBreakInputs
- SYSCFG_CFGR FLASHL LL_SYSCFG_GetTIMBreakInputs
- SYSCFG_CFGR PVDL LL_SYSCFG_GetTIMBreakInputs
- SYSCFG_CFGR_CM4L LL_SYSCFG_GetTIMBreakInputs

LL_SYSCFG_EnableCompensationCell

Function name

__STATIC_INLINE void LL_SYSCFG_EnableCompensationCell (void)

Function description

Enable the Compensation Cell.

Return values

- **None:**

Notes

- The I/O compensation cell can be used only when the device supply voltage ranges from 1.62 to 2.0 V and from 2.7 to 3.6 V.

Reference Manual to LL API cross reference:

- CCCSR EN LL_SYSCFG_EnableCompensationCell

LL_SYSCFG_DisableCompensationCell

Function name

__STATIC_INLINE void LL_SYSCFG_DisableCompensationCell (void)

Function description

Disable the Compensation Cell.

Return values

- **None:**

Notes

- The I/O compensation cell can be used only when the device supply voltage ranges from 1.62 to 2.0 V and from 2.7 to 3.6 V.

Reference Manual to LL API cross reference:

- CCCSR EN LL_SYSCFG_DisableCompensationCell

LL_SYSCFG_IsEnabledCompensationCell

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsEnabledCompensationCell (void)

Function description

Check if the Compensation Cell is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCCSR EN LL_SYSCFG_IsEnabledCompensationCell

LL_SYSCFG_IsActiveFlag_CMPCR

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_CMPCR (void)

Function description

Get Compensation Cell ready Flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCCSR READY LL_SYSCFG_IsActiveFlag_CMPCR

LL_SYSCFG_EnableIOSpeedOptimization

Function name

__STATIC_INLINE void LL_SYSCFG_EnableIOSpeedOptimization (void)

Function description

Enable the I/O speed optimization when the product voltage is low.

Return values

- **None:**

Notes

- This bit is active only if IO_HSLV user option bit is set. It must be used only if the product supply voltage is below 2.7 V. Setting this bit when VDD is higher than 2.7 V might be destructive.

Reference Manual to LL API cross reference:

- CCCSR HSLV LL_SYSCFG_EnableIOSpeedOptimize

LL_SYSCFG_DisableIOSpeedOptimization

Function name

__STATIC_INLINE void LL_SYSCFG_DisableIOSpeedOptimization (void)

Function description

To Disable optimize the I/O speed when the product voltage is low.

Return values

- **None:**

Notes

- This bit is active only if IO_HSLV user option bit is set. It must be used only if the product supply voltage is below 2.7 V. Setting this bit when VDD is higher than 2.7 V might be destructive.

Reference Manual to LL API cross reference:

- CCCSR HSLV LL_SYSCFG_DisableIOSpeedOptimize

LL_SYSCFG_IsEnabledIOSpeedOptimization

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsEnabledIOSpeedOptimization (void)

Function description

Check if the I/O speed optimization is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCCSR HSLV LL_SYSCFG_IsEnabledIOSpeedOptimization

LL_SYSCFG_SetCellCompensationCode

Function name

__STATIC_INLINE void LL_SYSCFG_SetCellCompensationCode (uint32_t CompCode)

Function description

Set the code selection for the I/O Compensation cell.

Parameters

- **CompCode:** Selects the code to be applied for the I/O compensation cell This parameter can be one of the following values:
  - LL_SYSCFG_CELL_CODE : Select Code from the cell (available in the SYSCFG_CCVR)
  - LL_SYSCFG_REGISTER_CODE: Select Code from the SYSCFG compensation cell code register (SYSCFG_CCCR)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCCSR CS LL_SYSCFG_SetCellCompensationCode

LL_SYSCFG_GetCellCompensationCode

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetCellCompensationCode (void)

Function description

Get the code selected for the I/O Compensation cell.

Return values

- **Returned:** value can be one of the following values:
  - LL_SYSCFG_CELL_CODE : Selected Code is from the cell (available in the SYSCFG_CCVR)
  - LL_SYSCFG_REGISTER_CODE: Selected Code is from the SYSCFG compensation cell code register (SYSCFG_CCCR)

Reference Manual to LL API cross reference:

- CCCSR CS LL_SYSCFG_GetCellCompensationCode

LL_SYSCFG_GetPMOSCompensationValue

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetPMOSCompensationValue (void)

Function description

Get I/O compensation cell value for PMOS transistors.

Return values

- **Returned:** value is the I/O compensation cell value for PMOS transistors

Reference Manual to LL API cross reference:

- CCVR PCV LL_SYSCFG_GetPMOSCompensationValue

LL_SYSCFG_GetNMOSCompensationValue

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetNMOSCompensationValue (void)

Function description

Get I/O compensation cell value for NMOS transistors.

Return values

- **Returned:** value is the I/O compensation cell value for NMOS transistors

Reference Manual to LL API cross reference:

- CCVR NCV LL_SYSCFG_GetNMOSCompensationValue

LL_SYSCFG_SetPMOSCompensationCode

Function name

__STATIC_INLINE void LL_SYSCFG_SetPMOSCompensationCode (uint32_t PMOSCode)

Function description

Set I/O compensation cell code for PMOS transistors.

Parameters

- **PMOSCode:** PMOS compensation code This code is applied to the I/O compensation cell when the CS bit of the SYSCFG_CMPCR is set

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCCR PCC LL_SYSCFG_SetPMOSCompensationCode

LL_SYSCFG_GetPMOSCompensationCode

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetPMOSCompensationCode (void)

Function description

Get I/O compensation cell code for PMOS transistors.

Return values

- **Returned:** value is the I/O compensation cell code for PMOS transistors

Reference Manual to LL API cross reference:

- CCCR PCC LL_SYSCFG_GetPMOSCompensationCode

LL_SYSCFG_SetNMOSCompensationCode

Function name

__STATIC_INLINE void LL_SYSCFG_SetNMOSCompensationCode (uint32_t NMOSCode)

Function description

Set I/O compensation cell code for NMOS transistors.

Parameters

- **NMOSCode:** NMOS compensation code This code is applied to the I/O compensation cell when the CS bit of the SYSCFG_CMPCR is set

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCCR NCC LL_SYSCFG_SetNMOSCompensationCode

LL_SYSCFG_GetNMOSCompensationCode

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetNMOSCompensationCode (void)

Function description

Get I/O compensation cell code for NMOS transistors.

Return values

- **Returned:** value is the I/O compensation cell code for NMOS transistors

Reference Manual to LL API cross reference:

- CCCR NCC LL_SYSCFG_GetNMOSCompensationCode

LL_SYSCFG_GetPackage

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetPackage (void)

Function description

Get the device package.

Return values

- **Returned:** value can be one of the following values:
  - LL_SYSCFG_LQFP100_PACKAGE (*)
  - LL_SYSCFG_TQFP144_PACKAGE (*)
  - LL_SYSCFG_TQFP176_UFBGA176_PACKAGE (*)
  - LL_SYSCFG_LQFP208_TFBGA240_PACKAGE (*)
  - LL_SYSCFG_VFQFPN68_INDUS_PACKAGE (*)
  - LL_SYSCFG_TFBGA100_LQFP100_PACKAGE (*)
  - LL_SYSCFG_LQFP100_INDUS_PACKAGE (**)
  - LL_SYSCFG_TFBGA100_INDUS_PACKAGE (**)
  - LL_SYSCFG_WLCSP115_INDUS_PACKAGE (**)
  - LL_SYSCFG_LQFP144_PACKAGE (**)
  - LL_SYSCFG_UFBGA144_PACKAGE (**)
  - LL_SYSCFG_LQFP144_INDUS_PACKAGE (**)
  - LL_SYSCFG_UFBGA169_INDUS_PACKAGE (**)
  - LL_SYSCFG_UFBGA176PLUS25_INDUS_PACKAGE (**)
  - LL_SYSCFG_LQFP176_INDUS_PACKAGE (**)

  (*) : For stm32h74xxx and stm32h75xxx family lines. (**): For stm32h72xxx and stm32h73xxx family lines.

Reference Manual to LL API cross reference:

- PKGR PKG LL_SYSCFG_GetPackage

LL_SYSCFG_GetFLashProtectionLevel

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetFLashProtectionLevel (void)

Function description

Get the Flash memory protection level.

Return values

- **Returned:** value can be one of the following values: 0xAA : RDP level 0 0xCC : RDP level 2 Any other value : RDP level 1

Reference Manual to LL API cross reference:

- UR0 RDP LL_SYSCFG_GetFLashProtectionLevel

LL_SYSCFG_IsFLashBankAddressesSwaped

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsFLashBankAddressesSwaped (void)

Function description

Indicate if the Flash memory bank addresses are inverted or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UR0 BKS LL_SYSCFG_IsFLashBankAddressesSwaped

LL_SYSCFG_GetBrownoutResetLevel

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetBrownoutResetLevel (void)

Function description

Get the BOR Threshold Reset Level.

Return values

- **Returned:** value can be one of the following values:
  - LL_SYSCFG_BOR_HIGH_RESET_LEVEL
  - LL_SYSCFG_BOR_MEDIUM_RESET_LEVEL
  - LL_SYSCFG_BOR_LOW_RESET_LEVEL
  - LL_SYSCFG_BOR_OFF_RESET_LEVEL

Reference Manual to LL API cross reference:

- UR2 BORH LL_SYSCFG_GetBrownoutResetLevel

LL_SYSCFG_SetCM7BootAddress0

Function name

__STATIC_INLINE void LL_SYSCFG_SetCM7BootAddress0 (uint16_t BootAddress)

Function description

BootCM7 address 0 configuration.

Parameters

- **BootAddress:** :Specifies the CM7 Boot Address to be loaded in Address0

Return values

- **None:**

Reference Manual to LL API cross reference:

- UR2 BOOT_ADD0 LL_SYSCFG_SetCM7BootAddress0

LL_SYSCFG_GetCM7BootAddress0

Function name

__STATIC_INLINE uint16_t LL_SYSCFG_GetCM7BootAddress0 (void)

Function description

Get BootCM7 address 0.

Return values

- **Returned:** the CM7 Boot Address0

Reference Manual to LL API cross reference:

- UR2 BOOT_ADD0 LL_SYSCFG_GetCM7BootAddress0

LL_SYSCFG_SetCM7BootAddress1

Function name

__STATIC_INLINE void LL_SYSCFG_SetCM7BootAddress1 (uint16_t BootAddress)

Function description

BootCM7 address 1 configuration.

Parameters

- **BootAddress:** :Specifies the CM7 Boot Address to be loaded in Address1

Return values

- **None:**

Reference Manual to LL API cross reference:

- UR3 BOOT_ADD1 LL_SYSCFG_SetCM7BootAddress1

LL_SYSCFG_GetCM7BootAddress1

Function name

__STATIC_INLINE uint16_t LL_SYSCFG_GetCM7BootAddress1 (void)

Function description

Get BootCM7 address 1.

Return values

- **Returned:** the CM7 Boot Address0

Reference Manual to LL API cross reference:

- UR3 BOOT_ADD1 LL_SYSCFG_GetCM7BootAddress1

LL_SYSCFG_SetCM4BootAddress0

Function name

__STATIC_INLINE void LL_SYSCFG_SetCM4BootAddress0 (uint16_t BootAddress)

Function description

BootCM4 address 0 configuration.

Parameters

- **BootAddress:** :Specifies the CM4 Boot Address to be loaded in Address0

Return values

- **None:**

Reference Manual to LL API cross reference:

- UR3 BCM4_ADD0 LL_SYSCFG_SetCM4BootAddress0

LL_SYSCFG_GetCM4BootAddress0

Function name

__STATIC_INLINE uint16_t LL_SYSCFG_GetCM4BootAddress0 (void)

Function description

Get BootCM4 address 0.

Return values

- **Returned:** the CM4 Boot Address0

Reference Manual to LL API cross reference:

- UR3 BCM4_ADD0 LL_SYSCFG_GetCM4BootAddress0

LL_SYSCFG_SetCM4BootAddress1

Function name

__STATIC_INLINE void LL_SYSCFG_SetCM4BootAddress1 (uint16_t BootAddress)

Function description

BootCM4 address 1 configuration.

Parameters

- **BootAddress:** :Specifies the CM4 Boot Address to be loaded in Address1

Return values

- **None:**

Reference Manual to LL API cross reference:

- UR4 BCM4_ADD1 LL_SYSCFG_SetCM4BootAddress1

LL_SYSCFG_GetCM4BootAddress1

Function name

__STATIC_INLINE uint16_t LL_SYSCFG_GetCM4BootAddress1 (void)

Function description

Get BootCM4 address 1.

Return values

- **Returned:** the CM4 Boot Address0

Reference Manual to LL API cross reference:

- UR4 BCM4_ADD1 LL_SYSCFG_GetCM4BootAddress1

LL_SYSCFG_IsFlashB1ProtectedAreaErasable

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsFlashB1ProtectedAreaErasable (void)

Function description

Indicates if the flash protected area (Bank 1) is erased by a mass erase.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UR4 MEPAD_BANK1 LL_SYSCFG_IsFlashB1ProtectedAreaErasable

LL_SYSCFG_IsFlashB1SecuredAreaErasable

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsFlashB1SecuredAreaErasable (void)

Function description

Indicates if the flash secured area (Bank 1) is erased by a mass erase.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UR5 MESAD_BANK1 LL_SYSCFG_IsFlashB1SecuredAreaErasable

LL_SYSCFG_IsFlashB1Sector0WriteProtected

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsFlashB1Sector0WriteProtected (void)

Function description

Indicates if the sector 0 of the Flash memory bank 1 is write protected.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UR5 WRPN_BANK1 LL_SYSCFG_IsFlashB1Sector0WriteProtected

LL_SYSCFG_IsFlashB1Sector1WriteProtected

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsFlashB1Sector1WriteProtected (void)

Function description

Indicates if the sector 1 of the Flash memory bank 1 is write protected.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UR5 WRPN_BANK1 LL_SYSCFG_IsFlashB1Sector1WriteProtected

LL_SYSCFG_IsFlashB1Sector2WriteProtected

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsFlashB1Sector2WriteProtected (void)

Function description

Indicates if the sector 2 of the Flash memory bank 1 is write protected.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UR5 WRPN_BANK1 LL_SYSCFG_IsFlashB1Sector2WriteProtected

LL_SYSCFG_IsFlashB1Sector3WriteProtected

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsFlashB1Sector3WriteProtected (void)

Function description

Indicates if the sector 3 of the Flash memory bank 1 is write protected.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UR5 WRPN_BANK1 LL_SYSCFG_IsFlashB1Sector3WriteProtected

LL_SYSCFG_IsFlashB1Sector4WriteProtected

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsFlashB1Sector4WriteProtected (void)

Function description

Indicates if the sector 4 of the Flash memory bank 1 is write protected.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UR5 WRPN_BANK1 LL_SYSCFG_IsFlashB1Sector4WriteProtected

LL_SYSCFG_IsFlashB1Sector5WriteProtected

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsFlashB1Sector5WriteProtected (void)

Function description

Indicates if the sector 5 of the Flash memory bank 1 is write protected.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UR5 WRPN_BANK1 LL_SYSCFG_IsFlashB1Sector5WriteProtected

LL_SYSCFG_IsFlashB1Sector6WriteProtected

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsFlashB1Sector6WriteProtected (void)

Function description

Indicates if the sector 6 of the Flash memory bank 1 is write protected.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UR5 WRPN_BANK1 LL_SYSCFG_IsFlashB1Sector6WriteProtected

LL_SYSCFG_IsFlashB1Sector7WriteProtected

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsFlashB1Sector7WriteProtected (void)

Function description

Indicates if the sector 7 of the Flash memory bank 1 is write protected.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UR5 WRPN_BANK1 LL_SYSCFG_IsFlashB1Sector7WriteProtected

LL_SYSCFG_GetFlashB1ProtectedAreaStartAddress

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetFlashB1ProtectedAreaStartAddress (void)

Function description

Get the protected area start address for Flash bank 1.

Return values

- **Returned:** the protected area start address for Flash bank 1

Reference Manual to LL API cross reference:

- UR6 PABEG_BANK1 LL_SYSCFG_GetFlashB1ProtectedAreaStartAddress

LL_SYSCFG_GetFlashB1ProtectedAreaEndAddress

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetFlashB1ProtectedAreaEndAddress (void)

Function description

Get the protected area end address for Flash bank 1.

Return values

- **Returned:** the protected area end address for Flash bank 1

Reference Manual to LL API cross reference:

- UR6 PAEND_BANK1 LL_SYSCFG_GetFlashB1ProtectedAreaEndAddress

LL_SYSCFG_GetFlashB1SecuredAreaStartAddress

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetFlashB1SecuredAreaStartAddress (void)

Function description

Get the secured area start address for Flash bank 1.

Return values

- **Returned:** the secured area start address for Flash bank 1

Reference Manual to LL API cross reference:

- UR7 SABEG_BANK1 LL_SYSCFG_GetFlashB1SecuredAreaStartAddress

LL_SYSCFG_GetFlashB1SecuredAreaEndAddress

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetFlashB1SecuredAreaEndAddress (void)

Function description

Get the secured area end address for Flash bank 1.

Return values

- **Returned:** the secured area end address for Flash bank 1

Reference Manual to LL API cross reference:

- UR7 SAEND_BANK1 LL_SYSCFG_GetFlashB1SecuredAreaEndAddress

LL_SYSCFG_IsFlashB2ProtectedAreaErasable

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsFlashB2ProtectedAreaErasable (void)

Function description

Indicates if the flash protected area (Bank 2) is erased by a mass erase.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UR8 MEPAD_BANK2 LL_SYSCFG_IsFlashB2ProtectedAreaErasable

LL_SYSCFG_IsFlashB2SecuredAreaErasable

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsFlashB2SecuredAreaErasable (void)

Function description

Indicates if the flash secured area (Bank 2) is erased by a mass erase.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UR8 MESAD_BANK2 LL_SYSCFG_IsFlashB2SecuredAreaErasable

LL_SYSCFG_IsFlashB2Sector0WriteProtected

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsFlashB2Sector0WriteProtected (void)

Function description

Indicates if the sector 0 of the Flash memory bank 2 is write protected.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UR9 WRPN_BANK2 LL_SYSCFG_IsFlashB2Sector0WriteProtected

LL_SYSCFG_IsFlashB2Sector1WriteProtected

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsFlashB2Sector1WriteProtected (void)

Function description

Indicates if the sector 1 of the Flash memory bank 2 is write protected.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UR9 WRPN_BANK2 LL_SYSCFG_IsFlashB2Sector1WriteProtected

LL_SYSCFG_IsFlashB2Sector2WriteProtected

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsFlashB2Sector2WriteProtected (void)

Function description

Indicates if the sector 2 of the Flash memory bank 2 is write protected.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UR9 WRPN_BANK2 LL_SYSCFG_IsFlashB2Sector2WriteProtected

LL_SYSCFG_IsFlashB2Sector3WriteProtected

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsFlashB2Sector3WriteProtected (void)

Function description

Indicates if the sector 3 of the Flash memory bank 2 is write protected.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UR9 WRPN_BANK2 LL_SYSCFG_IsFlashB2Sector3WriteProtected

LL_SYSCFG_IsFlashB2Sector4WriteProtected

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsFlashB2Sector4WriteProtected (void)

Function description

Indicates if the sector 4 of the Flash memory bank 2 is write protected.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UR9 WRPN_BANK2 LL_SYSCFG_IsFlashB2Sector4WriteProtected

LL_SYSCFG_IsFlashB2Sector5WriteProtected

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsFlashB2Sector5WriteProtected (void)

Function description

Indicates if the sector 5 of the Flash memory bank 2 is write protected.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UR9 WRPN_BANK2 LL_SYSCFG_IsFlashB2Sector5WriteProtected

LL_SYSCFG_IsFlashB2Sector6WriteProtected

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsFlashB2Sector6WriteProtected (void)

Function description

Indicates if the sector 6 of the Flash memory bank 2 is write protected.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UR9 WRPN_BANK2 LL_SYSCFG_IsFlashB2Sector6WriteProtected

LL_SYSCFG_IsFlashB2Sector7WriteProtected

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsFlashB2Sector7WriteProtected (void)

Function description

Indicates if the sector 7 of the Flash memory bank 2 is write protected.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UR9 WRPN_BANK2 LL_SYSCFG_IsFlashB2Sector7WriteProtected

LL_SYSCFG_GetFlashB2ProtectedAreaStartAddress

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetFlashB2ProtectedAreaStartAddress (void)

Function description

Get the protected area start address for Flash bank 2.

Return values

- **Returned:** the protected area start address for Flash bank 2

Reference Manual to LL API cross reference:

- UR9 PABEG_BANK2 LL_SYSCFG_GetFlashB2ProtectedAreaStartAddress

LL_SYSCFG_GetFlashB2ProtectedAreaEndAddress

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetFlashB2ProtectedAreaEndAddress (void)

Function description

Get the protected area end address for Flash bank 2.

Return values

- **Returned:** the protected area end address for Flash bank 2

Reference Manual to LL API cross reference:

- UR10 PAEND_BANK2 LL_SYSCFG_GetFlashB2ProtectedAreaEndAddress

LL_SYSCFG_GetFlashB2SecuredAreaStartAddress

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetFlashB2SecuredAreaStartAddress (void)

Function description

Get the secured area start address for Flash bank 2.

Return values

- **Returned:** the secured area start address for Flash bank 2

Reference Manual to LL API cross reference:

- UR10 SABEG_BANK2 LL_SYSCFG_GetFlashB2SecuredAreaStartAddress

LL_SYSCFG_GetFlashB2SecuredAreaEndAddress

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetFlashB2SecuredAreaEndAddress (void)

Function description

Get the secured area end address for Flash bank 2.

Return values

- **Returned:** the secured area end address for Flash bank 2

Reference Manual to LL API cross reference:

- UR11 SAEND_BANK2 LL_SYSCFG_GetFlashB2SecuredAreaEndAddress

LL_SYSCFG_GetIWDG1ControlMode

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetIWDG1ControlMode (void)

Function description

Get the Independent Watchdog 1 control mode (Software or Hardware)

Return values

- **Returned:** value can be one of the following values:
  - LL_SYSCFG_IWDG1_SW_CONTROL_MODE
  - LL_SYSCFG_IWDG1_HW_CONTROL_MODE

Reference Manual to LL API cross reference:

- UR11 IWDG1M LL_SYSCFG_GetIWDG1ControlMode

LL_SYSCFG_GetIWDG2ControlMode

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetIWDG2ControlMode (void)

Function description

Get the Independent Watchdog 2 control mode (Software or Hardware)

Return values

- **Returned:** value can be one of the following values:
  - LL_SYSCFG_IWDG2_SW_CONTROL_MODE
  - LL_SYSCFG_IWDG2_HW_CONTROL_MODE

Reference Manual to LL API cross reference:

- UR12 IWDG2M LL_SYSCFG_GetIWDG2ControlMode

LL_SYSCFG_IsSecureModeEnabled

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsSecureModeEnabled (void)

Function description

Indicates the Secure mode status.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UR12 SECURE LL_SYSCFG_IsSecureModeEnabled

LL_SYSCFG_IsD1StandbyGenerateReset

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsD1StandbyGenerateReset (void)

Function description

Indicates if a reset is generated when D1 domain enters DStandby mode.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UR13 D1SBRST LL_SYSCFG_IsD1StandbyGenerateReset

LL_SYSCFG_GetSecuredDTCMSize

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetSecuredDTCMSize (void)

Function description

Get the secured DTCM RAM size.

Return values

- **Returned:** value can be one of the following values:
  - LL_SYSCFG_DTCM_RAM_SIZE_2KB
  - LL_SYSCFG_DTCM_RAM_SIZE_4KB
  - LL_SYSCFG_DTCM_RAM_SIZE_8KB
  - LL_SYSCFG_DTCM_RAM_SIZE_16KB

Reference Manual to LL API cross reference:

- UR13 SDRS LL_SYSCFG_GetSecuredDTCMSize

LL_SYSCFG_IsD1StopGenerateReset

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsD1StopGenerateReset (void)

Function description

Indicates if a reset is generated when D1 domain enters DStop mode.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UR14 D1STPRST LL_SYSCFG_IsD1StopGenerateReset

LL_SYSCFG_IsD2StandbyGenerateReset

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsD2StandbyGenerateReset (void)

Function description

Indicates if a reset is generated when D2 domain enters DStandby mode.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UR14 D2SBRST LL_SYSCFG_IsD2StandbyGenerateReset

LL_SYSCFG_IsD2StopGenerateReset

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsD2StopGenerateReset (void)

Function description

Indicates if a reset is generated when D2 domain enters DStop mode.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UR15 D2STPRST LL_SYSCFG_IsD2StopGenerateReset

LL_SYSCFG_IsIWDGFrozenInStandbyMode

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsIWDGFrozenInStandbyMode (void)

Function description

Indicates if the independent watchdog is frozen in Standby mode.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UR15 FZIWDGSTB LL_SYSCFG_IsIWDGFrozenInStandbyMode

LL_SYSCFG_IsIWDGFrozenInStopMode

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsIWDGFrozenInStopMode (void)

Function description

Indicates if the independent watchdog is frozen in Stop mode.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UR16 FZIWDGSTP LL_SYSCFG_IsIWDGFrozenInStopMode

LL_SYSCFG_IsPrivateKeyProgrammed

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsPrivateKeyProgrammed (void)

Function description

Indicates if the device private key is programmed.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UR16 PKP LL_SYSCFG_IsPrivateKeyProgrammed

LL_SYSCFG_IsActiveFlag_IOHSLV

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_IOHSLV (void)

Function description

Indicates if the Product is working on the full voltage range or not.

Return values

- **State:** of bit (1 or 0).

Notes

- When the IOHSLV option bit is set the Product is working below 2.7 V. When the IOHSLV option bit is reset the Product is working on the full voltage range.

Reference Manual to LL API cross reference:

- UR17 IOHSLV LL_SYSCFG_IsActiveFlag_IOHSLV

LL_DBGMCU_GetDeviceID

Function name

__STATIC_INLINE uint32_t LL_DBGMCU_GetDeviceID (void)

Function description

Return the device identifier.

Return values

- **Values:** between Min_Data=0x00 and Max_Data=0xFFF

Reference Manual to LL API cross reference:

- DBGMCU_IDCODE DEV_ID LL_DBGMCU_GetDeviceID

LL_DBGMCU_GetRevisionID

Function name

__STATIC_INLINE uint32_t LL_DBGMCU_GetRevisionID (void)

Function description

Return the device revision identifier.

Return values

- **Values:** between Min_Data=0x00 and Max_Data=0xFFFF

Notes

- This field indicates the revision of the device. For example, it is read as RevA -> 0x1000, Cat 2 revZ -> 0x1001

Reference Manual to LL API cross reference:

- DBGMCU_IDCODE REV_ID LL_DBGMCU_GetRevisionID

LL_DBGMCU_EnableD1DebugInSleepMode

Function name

__STATIC_INLINE void LL_DBGMCU_EnableD1DebugInSleepMode (void)

Function description

Enable D1 Domain/CDomain debug during SLEEP mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR DBGSLEEP_D1/DBGSLEEP_CD LL_DBGMCU_EnableD1DebugInSleepMode

LL_DBGMCU_DisableD1DebugInSleepMode

Function name

__STATIC_INLINE void LL_DBGMCU_DisableD1DebugInSleepMode (void)

Function description

Disable D1 Domain/CDomain debug during SLEEP mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR DBGSLEEP_D1/DBGSLEEP_CD LL_DBGMCU_DisableD1DebugInSleepMode

LL_DBGMCU_EnableD1DebugInStopMode

Function name

__STATIC_INLINE void LL_DBGMCU_EnableD1DebugInStopMode (void)

Function description

Enable D1 Domain/CDomain debug during STOP mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR DBGSTOP_D1/DBGSLEEP_CD LL_DBGMCU_EnableD1DebugInStopMode

LL_DBGMCU_DisableD1DebugInStopMode

Function name

__STATIC_INLINE void LL_DBGMCU_DisableD1DebugInStopMode (void)

Function description

Disable D1 Domain/CDomain debug during STOP mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR DBGSTOP_D1/DBGSLEEP_CD LL_DBGMCU_DisableD1DebugInStopMode

LL_DBGMCU_EnableD1DebugInStandbyMode

Function name

__STATIC_INLINE void LL_DBGMCU_EnableD1DebugInStandbyMode (void)

Function description

Enable D1 Domain/CDomain debug during STANDBY mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR DBGSTBY_D1/DBGSLEEP_CD LL_DBGMCU_EnableD1DebugInStandbyMode

LL_DBGMCU_DisableD1DebugInStandbyMode

Function name

__STATIC_INLINE void LL_DBGMCU_DisableD1DebugInStandbyMode (void)

Function description

Disable D1 Domain/CDomain debug during STANDBY mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR DBGSTBY_D1/DBGSLEEP_CD LL_DBGMCU_DisableD1DebugInStandbyMode

LL_DBGMCU_EnableD2DebugInSleepMode

Function name

__STATIC_INLINE void LL_DBGMCU_EnableD2DebugInSleepMode (void)

Function description

Enable D2 Domain debug during SLEEP mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR DBGSLEEP_D2 LL_DBGMCU_EnableD2DebugInSleepMode

LL_DBGMCU_DisableD2DebugInSleepMode

Function name

__STATIC_INLINE void LL_DBGMCU_DisableD2DebugInSleepMode (void)

Function description

Disable D2 Domain debug during SLEEP mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR DBGSLEEP_D2 LL_DBGMCU_DisableD2DebugInSleepMode

LL_DBGMCU_EnableD2DebugInStopMode

Function name

__STATIC_INLINE void LL_DBGMCU_EnableD2DebugInStopMode (void)

Function description

Enable D2 Domain debug during STOP mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR DBGSTOP_D2 LL_DBGMCU_EnableD2DebugInStopMode

LL_DBGMCU_DisableD2DebugInStopMode

Function name

__STATIC_INLINE void LL_DBGMCU_DisableD2DebugInStopMode (void)

Function description

Disable D2 Domain debug during STOP mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR DBGSTOP_D2 LL_DBGMCU_DisableD2DebugInStopMode

LL_DBGMCU_EnableD2DebugInStandbyMode

Function name

__STATIC_INLINE void LL_DBGMCU_EnableD2DebugInStandbyMode (void)

Function description

Enable D2 Domain debug during STANDBY mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR DBGSTBY_D2 LL_DBGMCU_EnableD2DebugInStandbyMode

LL_DBGMCU_DisableD2DebugInStandbyMode

Function name

__STATIC_INLINE void LL_DBGMCU_DisableD2DebugInStandbyMode (void)

Function description

Disable D2 Domain debug during STANDBY mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR DBGSTBY_D2 LL_DBGMCU_DisableD2DebugInStandbyMode

LL_DBGMCU_EnableTracePortClock

Function name

__STATIC_INLINE void LL_DBGMCU_EnableTracePortClock (void)

Function description

Enable the trace port clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR TRACECKEN LL_DBGMCU_EnableTracePortClock

LL_DBGMCU_DisableTracePortClock

Function name

__STATIC_INLINE void LL_DBGMCU_DisableTracePortClock (void)

Function description

Disable the trace port clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR TRACECKEN LL_DBGMCU_DisableTracePortClock

LL_DBGMCU_EnableD1DebugClock

Function name

__STATIC_INLINE void LL_DBGMCU_EnableD1DebugClock (void)

Function description

Enable the Domain1/CDomain debug clock enable.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR CKD1EN/CKCDEN LL_DBGMCU_EnableD1DebugClock

LL_DBGMCU_DisableD1DebugClock

Function name

__STATIC_INLINE void LL_DBGMCU_DisableD1DebugClock (void)

Function description

Disable the Domain1/CDomain debug clock enable.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR CKD1EN/CKCDEN LL_DBGMCU_DisableD1DebugClock

LL_DBGMCU_EnableD3DebugClock

Function name

__STATIC_INLINE void LL_DBGMCU_EnableD3DebugClock (void)

Function description

Enable the Domain3/SRDomain debug clock enable.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR CKD3EN/CKSRDEN LL_DBGMCU_EnableD3DebugClock

LL_DBGMCU_DisableD3DebugClock

Function name

__STATIC_INLINE void LL_DBGMCU_DisableD3DebugClock (void)

Function description

Disable the Domain3/SRDomain debug clock enable.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR CKD3EN/CKSRDEN LL_DBGMCU_DisableD3DebugClock

LL_DBGMCU_SetExternalTriggerPinDirection

Function name

__STATIC_INLINE void LL_DBGMCU_SetExternalTriggerPinDirection (uint32_t PinDirection)

Function description

Set the direction of the bi-directional trigger pin TRGIO.

Parameters

- **PinDirection:** This parameter can be one of the following values:
  - LL_DBGMCU_TRGIO_INPUT_DIRECTION
  - LL_DBGMCU_TRGIO_OUTPUT_DIRECTION

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR TRGOEN LL_DBGMCU_SetExternalTriggerPinDirection
-

LL_DBGMCU_GetExternalTriggerPinDirection

Function name

__STATIC_INLINE uint32_t LL_DBGMCU_GetExternalTriggerPinDirection (void)

Function description

Get the direction of the bi-directional trigger pin TRGIO.

Return values

- **Returned:** value can be one of the following values:
  - LL_DBGMCU_TRGIO_INPUT_DIRECTION
  - LL_DBGMCU_TRGIO_OUTPUT_DIRECTION

Reference Manual to LL API cross reference:

- DBGMCU_CR TRGOEN LL_DBGMCU_GetExternalTriggerPinDirection
-

LL_DBGMCU_APB1_GRP1_FreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB1_GRP1_FreezePeriph (uint32_t Periphs)

Function description

Freeze APB1 group1 peripherals.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB1_GRP1_TIM2_STOP
  - LL_DBGMCU_APB1_GRP1_TIM3_STOP
  - LL_DBGMCU_APB1_GRP1_TIM4_STOP
  - LL_DBGMCU_APB1_GRP1_TIM5_STOP
  - LL_DBGMCU_APB1_GRP1_TIM6_STOP
  - LL_DBGMCU_APB1_GRP1_TIM7_STOP
  - LL_DBGMCU_APB1_GRP1_TIM12_STOP
  - LL_DBGMCU_APB1_GRP1_TIM13_STOP
  - LL_DBGMCU_APB1_GRP1_TIM14_STOP
  - LL_DBGMCU_APB1_GRP1_LPTIM1_STOP
  - LL_DBGMCU_APB1_GRP1_I2C1_STOP
  - LL_DBGMCU_APB1_GRP1_I2C2_STOP
  - LL_DBGMCU_APB1_GRP1_I2C3_STOP
  - LL_DBGMCU_APB1_GRP1_I2C5_STOP (*)

  (*) value not defined in all devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB1LFZ1 TIM2 LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_APB1LFZ1 TIM3 LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_APB1LFZ1 TIM4 LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_APB1LFZ1 TIM5 LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_APB1LFZ1 TIM6 LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_APB1LFZ1 TIM7 LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_APB1LFZ1 TIM12 LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_APB1LFZ1 TIM13 LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_APB1LFZ1 TIM14 LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_APB1LFZ1 LPTIM1 LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_APB1LFZ1 I2C1 LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_APB1LFZ1 I2C2 LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_APB1LFZ1 I2C3 LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_APB1LFZ1 I2C5 LL_DBGMCU_APB1_GRP1_FreezePeriph
- (*)

LL_DBGMCU_APB1_GRP1_UnFreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB1_GRP1_UnFreezePeriph (uint32_t Periphs)

Function description

Unfreeze APB1 peripherals (group1 peripherals)

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB1_GRP1_TIM2_STOP
  - LL_DBGMCU_APB1_GRP1_TIM3_STOP
  - LL_DBGMCU_APB1_GRP1_TIM4_STOP
  - LL_DBGMCU_APB1_GRP1_TIM5_STOP
  - LL_DBGMCU_APB1_GRP1_TIM6_STOP
  - LL_DBGMCU_APB1_GRP1_TIM7_STOP
  - LL_DBGMCU_APB1_GRP1_TIM12_STOP
  - LL_DBGMCU_APB1_GRP1_TIM13_STOP
  - LL_DBGMCU_APB1_GRP1_TIM14_STOP
  - LL_DBGMCU_APB1_GRP1_LPTIM1_STOP
  - LL_DBGMCU_APB1_GRP1_I2C1_STOP
  - LL_DBGMCU_APB1_GRP1_I2C2_STOP
  - LL_DBGMCU_APB1_GRP1_I2C3_STOP
  - LL_DBGMCU_APB1_GRP1_I2C5_STOP (*)

  (*) value not defined in all devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB1LFZ1 TIM2 LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_APB1LFZ1 TIM3 LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_APB1LFZ1 TIM4 LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_APB1LFZ1 TIM5 LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_APB1LFZ1 TIM6 LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_APB1LFZ1 TIM7 LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_APB1LFZ1 TIM12 LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_APB1LFZ1 TIM13 LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_APB1LFZ1 TIM14 LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_APB1LFZ1 LPTIM1 LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_APB1LFZ1 I2C1 LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_APB1LFZ1 I2C2 LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_APB1LFZ1 I2C3 LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBGMCU_APB1LFZ1 I2C5 LL_DBGMCU_APB1_GRP1_FreezePeriph
-

LL_DBGMCU_APB1_GRP2_FreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB1_GRP2_FreezePeriph (uint32_t Periphs)

Function description

Freeze APB1 group2 peripherals.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB1_GRP2_FDCAN_STOP

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB1HFZ1 FDCAN LL_DBGMCU_APB1_GRP2_FreezePeriph
-

LL_DBGMCU_APB1_GRP2_UnFreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB1_GRP2_UnFreezePeriph (uint32_t Periphs)

Function description

Unfreeze APB1 group2 peripherals.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB1_GRP2_FDCAN_STOP

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB1HFZ1 FDCAN LL_DBGMCU_APB1_GRP2_UnFreezePeriph
-

LL_DBGMCU_APB2_GRP1_FreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB2_GRP1_FreezePeriph (uint32_t Periphs)

Function description

Freeze APB2 peripherals.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB2_GRP1_TIM1_STOP
  - LL_DBGMCU_APB2_GRP1_TIM8_STOP
  - LL_DBGMCU_APB2_GRP1_TIM15_STOP
  - LL_DBGMCU_APB2_GRP1_TIM16_STOP
  - LL_DBGMCU_APB2_GRP1_TIM17_STOP
  - LL_DBGMCU_APB2_GRP1_HRTIM_STOP (*)

  (*) value not defined in all devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB2FZ1 TIM1 LL_DBGMCU_APB2_GRP1_FreezePeriph
- DBGMCU_APB2FZ1 TIM8 LL_DBGMCU_APB2_GRP1_FreezePeriph
- DBGMCU_APB2FZ1 TIM15 LL_DBGMCU_APB2_GRP1_FreezePeriph
- DBGMCU_APB2FZ1 TIM16 LL_DBGMCU_APB2_GRP1_FreezePeriph
- DBGMCU_APB2FZ1 TIM17 LL_DBGMCU_APB2_GRP1_FreezePeriph DBGMCU_APB2FZ1 HRTIM LL_DBGMCU_APB2_GRP1_FreezePeriph

LL_DBGMCU_APB2_GRP1_UnFreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB2_GRP1_UnFreezePeriph (uint32_t Periphs)

Function description

Unfreeze APB2 peripherals.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB2_GRP1_TIM1_STOP
  - LL_DBGMCU_APB2_GRP1_TIM8_STOP
  - LL_DBGMCU_APB2_GRP1_TIM15_STOP
  - LL_DBGMCU_APB2_GRP1_TIM16_STOP
  - LL_DBGMCU_APB2_GRP1_TIM17_STOP
  - LL_DBGMCU_APB2_GRP1_HRTIM_STOP (*)

  (*) value not defined in all devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB2FZ1 TIM1 LL_DBGMCU_APB2_GRP1_FreezePeriph
- DBGMCU_APB2FZ1 TIM8 LL_DBGMCU_APB2_GRP1_FreezePeriph
- DBGMCU_APB2FZ1 TIM15 LL_DBGMCU_APB2_GRP1_FreezePeriph
- DBGMCU_APB2FZ1 TIM16 LL_DBGMCU_APB2_GRP1_FreezePeriph
- DBGMCU_APB2FZ1 TIM17 LL_DBGMCU_APB2_GRP1_FreezePeriph DBGMCU_APB2FZ1 HRTIM LL_DBGMCU_APB2_GRP1_FreezePeriph

LL_DBGMCU_APB3_GRP1_FreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB3_GRP1_FreezePeriph (uint32_t Periphs)

Function description

Freeze APB3 peripherals.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB3_GRP1_WWDG1_STOP

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB3FZ1 WWDG1 LL_DBGMCU_APB3_GRP1_FreezePeriph
-

LL_DBGMCU_APB3_GRP1_UnFreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB3_GRP1_UnFreezePeriph (uint32_t Periphs)

Function description

Unfreeze APB3 peripherals.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB3_GRP1_WWDG1_STOP

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB3FZ1 WWDG1 LL_DBGMCU_APB3_GRP1_UnFreezePeriph
-

LL_DBGMCU_APB4_GRP1_FreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB4_GRP1_FreezePeriph (uint32_t Periphs)

Function description

Freeze APB4 peripherals.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB4_GRP1_I2C4_STOP
  - LL_DBGMCU_APB4_GRP1_LPTIM2_STOP
  - LL_DBGMCU_APB4_GRP1_LPTIM3_STOP
  - LL_DBGMCU_APB4_GRP1_LPTIM4_STOP (*)
  - LL_DBGMCU_APB4_GRP1_LPTIM5_STOP (*)
  - LL_DBGMCU_APB4_GRP1_RTC_STOP
  - LL_DBGMCU_APB4_GRP1_IWDG1_STOP

  (*) value not defined in all devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB4FZ1 I2C4 LL_DBGMCU_APB4_GRP1_FreezePeriph
-
- DBGMCU_APB4FZ1 LPTIM2 LL_DBGMCU_APB4_GRP1_FreezePeriph
-
- DBGMCU_APB4FZ1 LPTIM3 LL_DBGMCU_APB4_GRP1_FreezePeriph
-
- DBGMCU_APB4FZ1 LPTIM4 LL_DBGMCU_APB4_GRP1_FreezePeriph
-
- DBGMCU_APB4FZ1 LPTIM5 LL_DBGMCU_APB4_GRP1_FreezePeriph
-
- DBGMCU_APB4FZ1 RTC LL_DBGMCU_APB4_GRP1_FreezePeriph
-
- DBGMCU_APB4FZ1 WDGLSD1 LL_DBGMCU_APB4_GRP1_FreezePeriph
-

LL_DBGMCU_APB4_GRP1_UnFreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB4_GRP1_UnFreezePeriph (uint32_t Periphs)

Function description

Unfreeze APB4 peripherals.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB4_GRP1_I2C4_STOP
  - LL_DBGMCU_APB4_GRP1_LPTIM2_STOP
  - LL_DBGMCU_APB4_GRP1_LPTIM3_STOP
  - LL_DBGMCU_APB4_GRP1_LPTIM4_STOP (*)
  - LL_DBGMCU_APB4_GRP1_LPTIM5_STOP (*)
  - LL_DBGMCU_APB4_GRP1_RTC_STOP
  - LL_DBGMCU_APB4_GRP1_IWDG1_STOP

  (*) value not defined in all devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB4FZ1 I2C4 LL_DBGMCU_APB4_GRP1_FreezePeriph
- DBGMCU_APB4FZ1 LPTIM2 LL_DBGMCU_APB4_GRP1_FreezePeriph
- DBGMCU_APB4FZ1 LPTIM3 LL_DBGMCU_APB4_GRP1_FreezePeriph
- DBGMCU_APB4FZ1 LPTIM4 LL_DBGMCU_APB4_GRP1_FreezePeriph
- DBGMCU_APB4FZ1 LPTIM5 LL_DBGMCU_APB4_GRP1_FreezePeriph
- DBGMCU_APB4FZ1 RTC LL_DBGMCU_APB4_GRP1_FreezePeriph
- DBGMCU_APB4FZ1 WDGLSD1 LL_DBGMCU_APB4_GRP1_FreezePeriph

LL_FLASH_SetLatency

Function name

__STATIC_INLINE void LL_FLASH_SetLatency (uint32_t Latency)

Function description

Set FLASH Latency.

Parameters

- **Latency:** This parameter can be one of the following values:
  - LL_FLASH_LATENCY_0
  - LL_FLASH_LATENCY_1
  - LL_FLASH_LATENCY_2
  - LL_FLASH_LATENCY_3
  - LL_FLASH_LATENCY_4
  - LL_FLASH_LATENCY_5
  - LL_FLASH_LATENCY_6
  - LL_FLASH_LATENCY_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLASH_ACR LATENCY LL_FLASH_SetLatency

LL_FLASH_GetLatency

Function name

__STATIC_INLINE uint32_t LL_FLASH_GetLatency (void)

Function description

Get FLASH Latency.

Return values

- **Returned:** value can be one of the following values:
  - LL_FLASH_LATENCY_0
  - LL_FLASH_LATENCY_1
  - LL_FLASH_LATENCY_2
  - LL_FLASH_LATENCY_3
  - LL_FLASH_LATENCY_4
  - LL_FLASH_LATENCY_5
  - LL_FLASH_LATENCY_6
  - LL_FLASH_LATENCY_7

Reference Manual to LL API cross reference:

- FLASH_ACR LATENCY LL_FLASH_GetLatency

LL_ART_Enable

Function name

__STATIC_INLINE void LL_ART_Enable (void)

Function description

Enable the Cortex-M4 ART cache.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ART_CTR EN LL_ART_Enable

LL_ART_Disable

Function name

__STATIC_INLINE void LL_ART_Disable (void)

Function description

Disable the Cortex-M4 ART cache.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ART_CTR EN LL_ART_Disable

LL_ART_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_ART_IsEnabled (void)

Function description

Check if the Cortex-M4 ART cache is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ART_CTR EN LL_ART_IsEnabled

LL_ART_SetBaseAddress

Function name

__STATIC_INLINE void LL_ART_SetBaseAddress (uint32_t BaseAddress)

Function description

Set the Cortex-M4 ART cache Base Address.

Parameters

- **BaseAddress:** Specifies the Base address of 1 Mbyte address page (cacheable page) from which the ART accelerator loads code to the cache.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ART_CTR PCACHEADDR LL_ART_SetBaseAddress

LL_ART_GetBaseAddress

Function name

__STATIC_INLINE uint32_t LL_ART_GetBaseAddress (void)

Function description

Get the Cortex-M4 ART cache Base Address.

Return values

- **the:** Base address of 1 Mbyte address page (cacheable page) from which the ART accelerator loads code to the cache

Reference Manual to LL API cross reference:

- ART_CTR PCACHEADDR LL_ART_GetBaseAddress

125.2 SYSTEM Firmware driver defines

The following section lists the various define and macros of the module.

125.2.1 SYSTEM

SYSTEM

SYSCFG DTCM RAM size configuration

LL_SYSCFG_DTCM_RAM_SIZE_2KB

LL_SYSCFG_DTCM_RAM_SIZE_4KB

LL_SYSCFG_DTCM_RAM_SIZE_8KB

LL_SYSCFG_DTCM_RAM_SIZE_16KB

Analog Switch control

LL_SYSCFG_ANALOG_SWITCH_BOOSTEN

I/O analog switch voltage booster enable

LL_SYSCFG_ANALOG_SWITCH_PA0

PA0 Switch Open

LL_SYSCFG_ANALOG_SWITCH_PA1

PA1 Switch Open

LL_SYSCFG_ANALOG_SWITCH_PC2

PC2 Switch Open

LL_SYSCFG_ANALOG_SWITCH_PC3

PC3 Switch Open

DBGMCU APB1 GRP1 STOP IP

LL_DBGMCU_APB1_GRP1_TIM2_STOP

TIM2 counter stopped when core is halted

LL_DBGMCU_APB1_GRP1_TIM3_STOP

TIM3 counter stopped when core is halted

LL_DBGMCU_APB1_GRP1_TIM4_STOP

TIM4 counter stopped when core is halted

LL_DBGMCU_APB1_GRP1_TIM5_STOP

TIM5 counter stopped when core is halted

LL_DBGMCU_APB1_GRP1_TIM6_STOP

TIM6 counter stopped when core is halted

LL_DBGMCU_APB1_GRP1_TIM7_STOP

TIM7 counter stopped when core is halted

LL_DBGMCU_APB1_GRP1_TIM12_STOP

TIM12 counter stopped when core is halted

LL_DBGMCU_APB1_GRP1_TIM13_STOP

TIM13 counter stopped when core is halted

LL_DBGMCU_APB1_GRP1_TIM14_STOP

TIM14 counter stopped when core is halted

LL_DBGMCU_APB1_GRP1_LPTIM1_STOP

LPTIM1 counter stopped when core is halted

LL_DBGMCU_APB1_GRP1_I2C1_STOP

I2C1 SMBUS timeout mode stopped when Core is halted

LL_DBGMCU_APB1_GRP1_I2C2_STOP

I2C2 SMBUS timeout mode stopped when Core is halted

LL_DBGMCU_APB1_GRP1_I2C3_STOP

I2C3 SMBUS timeout mode stopped when Core is halted

DBGMCU APB1 GRP2 STOP IP

LL_DBGMCU_APB1_GRP2_FDCAN_STOP

FDCAN is frozen while the core is in debug mode

DBGMCU APB2 GRP1 STOP IP

LL_DBGMCU_APB2_GRP1_TIM1_STOP

TIM1 counter stopped when core is halted

LL_DBGMCU_APB2_GRP1_TIM8_STOP

TIM8 counter stopped when core is halted

LL_DBGMCU_APB2_GRP1_TIM15_STOP

TIM15 counter stopped when core is halted

LL_DBGMCU_APB2_GRP1_TIM16_STOP

TIM16 counter stopped when core is halted

LL_DBGMCU_APB2_GRP1_TIM17_STOP

TIM17 counter stopped when core is halted

LL_DBGMCU_APB2_GRP1_HRTIM_STOP

HRTIM counter stopped when core is halted

DBGMCU APB3 GRP1 STOP IP

LL_DBGMCU_APB3_GRP1_WWDG1_STOP

WWDG1 is frozen while the core is in debug mode

DBGMCU APB4 GRP1 STOP IP

LL_DBGMCU_APB4_GRP1_I2C4_STOP

I2C4 is frozen while the core is in debug mode

LL_DBGMCU_APB4_GRP1_LPTIM2_STOP

LPTIM2 is frozen while the core is in debug mode

LL_DBGMCU_APB4_GRP1_LPTIM3_STOP

LPTIM3 is frozen while the core is in debug mode

LL_DBGMCU_APB4_GRP1_LPTIM4_STOP

LPTIM4 is frozen while the core is in debug mode

LL_DBGMCU_APB4_GRP1_LPTIM5_STOP

LPTIM5 is frozen while the core is in debug mode

LL_DBGMCU_APB4_GRP1_RTC_STOP

RTC is frozen while the core is in debug mode

LL_DBGMCU_APB4_GRP1_IWDG1_STOP

IWDG1 is frozen while the core is in debug mode

SYSCFG I/O compensation cell Code selection

LL_SYSCFG_CELL_CODE

LL_SYSCFG_REGISTER_CODE

Ethernet PHY Interface Selection

LL_SYSCFG_ETH_MII

ETH Media MII interface

LL_SYSCFG_ETH_RMII

ETH Media RMII interface

SYSCFG EXTI LINE

LL_SYSCFG_EXTI_LINE0

EXTI_POSITION_0 | EXTICR[0]

LL_SYSCFG_EXTI_LINE1

EXTI_POSITION_4 | EXTICR[0]

LL_SYSCFG_EXTI_LINE2

EXTI_POSITION_8 | EXTICR[0]

LL_SYSCFG_EXTI_LINE3

EXTI_POSITION_12 | EXTICR[0]

LL_SYSCFG_EXTI_LINE4

EXTI_POSITION_0 | EXTICR[1]

LL_SYSCFG_EXTI_LINE5

EXTI_POSITION_4 | EXTICR[1]

LL_SYSCFG_EXTI_LINE6

EXTI_POSITION_8 | EXTICR[1]

LL_SYSCFG_EXTI_LINE7

EXTI_POSITION_12 | EXTICR[1]

LL_SYSCFG_EXTI_LINE8

EXTI_POSITION_0 | EXTICR[2]

LL_SYSCFG_EXTI_LINE9

EXTI_POSITION_4 | EXTICR[2]

LL_SYSCFG_EXTI_LINE10

EXTI_POSITION_8 | EXTICR[2]

LL_SYSCFG_EXTI_LINE11

EXTI_POSITION_12 | EXTICR[2]

LL_SYSCFG_EXTI_LINE12

EXTI_POSITION_0 | EXTICR[3]

LL_SYSCFG_EXTI_LINE13

EXTI_POSITION_4 | EXTICR[3]

LL_SYSCFG_EXTI_LINE14

EXTI_POSITION_8 | EXTICR[3]

LL_SYSCFG_EXTI_LINE15

EXTI_POSITION_12 | EXTICR[3]

SYSCFG EXTI PORT

LL_SYSCFG_EXTI_PORTA

EXTI PORT A

LL_SYSCFG_EXTI_PORTB

EXTI PORT B

LL_SYSCFG_EXTI_PORTC

EXTI PORT C

LL_SYSCFG_EXTI_PORTD

EXTI PORT D

LL_SYSCFG_EXTI_PORTE

EXTI PORT E

LL_SYSCFG_EXTI_PORTF

EXTI PORT F

LL_SYSCFG_EXTI_PORTG

EXTI PORT G

LL_SYSCFG_EXTI_PORTH

EXTI PORT H

LL_SYSCFG_EXTI_PORTI

EXTI PORT I

LL_SYSCFG_EXTI_PORTJ

EXTI PORT J

LL_SYSCFG_EXTI_PORTK

EXTI PORT k

SYSCFG Flash Bank1 sectors bits status

LL_SYSCFG_FLASH_B1_SECTOR0_STATUS_BIT

LL_SYSCFG_FLASH_B1_SECTOR1_STATUS_BIT

LL_SYSCFG_FLASH_B1_SECTOR2_STATUS_BIT

LL_SYSCFG_FLASH_B1_SECTOR3_STATUS_BIT

LL_SYSCFG_FLASH_B1_SECTOR4_STATUS_BIT

LL_SYSCFG_FLASH_B1_SECTOR5_STATUS_BIT

LL_SYSCFG_FLASH_B1_SECTOR6_STATUS_BIT

LL_SYSCFG_FLASH_B1_SECTOR7_STATUS_BIT

SYSCFG Flash Bank2 sectors bits status

LL_SYSCFG_FLASH_B2_SECTOR0_STATUS_BIT

LL_SYSCFG_FLASH_B2_SECTOR1_STATUS_BIT

LL_SYSCFG_FLASH_B2_SECTOR2_STATUS_BIT

LL_SYSCFG_FLASH_B2_SECTOR3_STATUS_BIT

LL_SYSCFG_FLASH_B2_SECTOR4_STATUS_BIT

LL_SYSCFG_FLASH_B2_SECTOR5_STATUS_BIT

LL_SYSCFG_FLASH_B2_SECTOR6_STATUS_BIT

LL_SYSCFG_FLASH_B2_SECTOR7_STATUS_BIT

SYSCFG I2C FASTMODEPLUS

LL_SYSCFG_I2C_FASTMODEPLUS_I2C1

Enable Fast Mode Plus for I2C1

LL_SYSCFG_I2C_FASTMODEPLUS_I2C2

Enable Fast Mode Plus for I2C2

LL_SYSCFG_I2C_FASTMODEPLUS_I2C3

Enable Fast Mode Plus for I2C3

LL_SYSCFG_I2C_FASTMODEPLUS_I2C4

Enable Fast Mode Plus for I2C4

LL_SYSCFG_I2C_FASTMODEPLUS_PB6

Enable Fast Mode Plus on PB6

LL_SYSCFG_I2C_FASTMODEPLUS_PB7

Enable Fast Mode Plus on PB7

LL_SYSCFG_I2C_FASTMODEPLUS_PB8

Enable Fast Mode Plus on PB8

LL_SYSCFG_I2C_FASTMODEPLUS_PB9

Enable Fast Mode Plus on PB9

FLASH LATENCY

LL_FLASH_LATENCY_0

FLASH Zero wait state

LL_FLASH_LATENCY_1

FLASH One wait state

LL_FLASH_LATENCY_2

FLASH Two wait states

LL_FLASH_LATENCY_3

FLASH Three wait states

LL_FLASH_LATENCY_4

FLASH Four wait states

LL_FLASH_LATENCY_5

FLASH five wait state

LL_FLASH_LATENCY_6

FLASH six wait state

LL_FLASH_LATENCY_7

FLASH seven wait states

SYSCFG TIMER BREAK

LL_SYSCFG_TIMBREAK_AXISRAM_DBL_ECC

Enables and locks the AXIRAM double ECC error signal with Break Input of TIM1/8/15/16/17 and HRTIM

LL_SYSCFG_TIMBREAK_ITCM_DBL_ECC

Enables and locks the ITCM double ECC error signal with Break Input of TIM1/8/15/16/17 and HRTIM

LL_SYSCFG_TIMBREAK_DTCM_DBL_ECC

Enables and locks the DTCM double ECC error signal with Break Input of TIM1/8/15/16/17 and HRTIM

LL_SYSCFG_TIMBREAK_SRAM1_DBL_ECC

Enables and locks the SRAM1 double ECC error signal with Break Input of TIM1/8/15/16/17 and HRTIM

LL_SYSCFG_TIMBREAK_SRAM2_DBL_ECC

Enables and locks the SRAM2 double ECC error signal with Break Input of TIM1/8/15/16/17 and HRTIM

LL_SYSCFG_TIMBREAK_SRAM3_DBL_ECC

Enables and locks the SRAM3 double ECC error signal with Break Input of TIM1/8/15/16/17 and HRTIM

LL_SYSCFG_TIMBREAK_SRAM4_DBL_ECC

Enables and locks the SRAM4 double ECC error signal with Break Input of TIM1/8/15/16/17 and HRTIM

LL_SYSCFG_TIMBREAK_BKRAM_DBL_ECC

Enables and locks the BKRAM double ECC error signal with Break Input of TIM1/8/15/16/17 and HRTIM

LL_SYSCFG_TIMBREAK_CM7_LOCKUP

Enables and locks the Cortex-M7 LOCKUP signal with Break Input of TIM1/8/15/16/17 and HRTIM

LL_SYSCFG_TIMBREAK_FLASH_DBL_ECC

Enables and locks the FLASH double ECC error signal with Break Input of TIM1/8/15/16/17 and HRTIM

LL_SYSCFG_TIMBREAK_PVD

Enables and locks the PVD connection with TIM1/8/15/16/17 and HRTIM Break Input and also the PVDE and PLS bits of the Power Control Interface

LL_SYSCFG_TIMBREAK_CM4_LOCKUP

Enables and locks the Cortex-M4 LOCKUP signal with Break Input of TIM1/8/15/16/17 and HRTIM

DBGMCU TRACE Pin Assignment

LL_DBGMCU_TRACE_NONE

TRACE pins not assigned (default state)

LL_DBGMCU_TRACE_ASYNCH

TRACE pin assignment for Asynchronous Mode

LL_DBGMCU_TRACE_SYNCH_SIZE1

TRACE pin assignment for Synchronous Mode with a TRACEDATA size of 1

LL_DBGMCU_TRACE_SYNCH_SIZE2

TRACE pin assignment for Synchronous Mode with a TRACEDATA size of 2

LL_DBGMCU_TRACE_SYNCH_SIZE4

TRACE pin assignment for Synchronous Mode with a TRACEDATA size of 4

DBGMCU

LL_DBGMCU_TRGIO_INPUT_DIRECTION

LL_DBGMCU_TRGIO_OUTPUT_DIRECTION

SYSCFG IWDG1 control modes

LL_SYSCFG_IWDG1_SW_CONTROL_MODE

LL_SYSCFG_IWDG1_HW_CONTROL_MODE

SYSCFG IWDG2 control modes

LL_SYSCFG_IWDG2_SW_CONTROL_MODE

LL_SYSCFG_IWDG2_HW_CONTROL_MODE

SYSCFG device package

LL_SYSCFG_LQFP100_PACKAGE

LL_SYSCFG_TQFP144_PACKAGE

LL_SYSCFG_TQFP176_UFBGA176_PACKAGE

LL_SYSCFG_LQFP208_TFBGA240_PACKAGE

SYSCFG Brownout Reset Threshold Level

LL_SYSCFG_BOR_OFF_RESET_LEVEL

LL_SYSCFG_BOR_LOW_RESET_LEVEL

LL_SYSCFG_BOR_MEDIUM_RESET_LEVEL

LL_SYSCFG_BOR_HIGH_RESET_LEVEL

126 LL TIM Generic Driver

126.1 TIM Firmware driver registers structures

126.1.1 LL_TIM_InitTypeDef

LL_TIM_InitTypeDef is defined in the stm32h7xx_ll_tim.h

Data Fields

- ***uint16_t Prescaler***
- ***uint32_t CounterMode***
- ***uint32_t Autoreload***
- ***uint32_t ClockDivision***
- ***uint32_t RepetitionCounter***

Field Documentation

- ***uint16_t LL_TIM_InitTypeDef::Prescaler***

  Specifies the prescaler value used to divide the TIM clock. This parameter can be a number between Min_Data=0x0000 and Max_Data=0xFFFF.This feature can be modified afterwards using unitary function **LL_TIM_SetPrescaler()**.
- ***uint32_t LL_TIM_InitTypeDef::CounterMode***

  Specifies the counter mode. This parameter can be a value of ***TIM_LL_EC_COUNTERMODE***.This feature can be modified afterwards using unitary function **LL_TIM_SetCounterMode()**.
- ***uint32_t LL_TIM_InitTypeDef::Autoreload***

  Specifies the auto reload value to be loaded into the active Auto-Reload Register at the next update event. This parameter must be a number between Min_Data=0x0000 and Max_Data=0xFFFF. Some timer instances may support 32 bits counters. In that case this parameter must be a number between 0x0000 and 0xFFFFFFFF.This feature can be modified afterwards using unitary function **LL_TIM_SetAutoReload()**.
- ***uint32_t LL_TIM_InitTypeDef::ClockDivision***

  Specifies the clock division. This parameter can be a value of ***TIM_LL_EC_CLOCKDIVISION***.This feature can be modified afterwards using unitary function **LL_TIM_SetClockDivision()**.
- ***uint32_t LL_TIM_InitTypeDef::RepetitionCounter***

  Specifies the repetition counter value. Each time the RCR downcounter reaches zero, an update event is generated and counting restarts from the RCR value (N). This means in PWM mode that (N+1) corresponds to:
  - the number of PWM periods in edge-aligned mode
  - the number of half PWM period in center-aligned mode GP timers: this parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF. Advanced timers: this parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.

  This feature can be modified afterwards using unitary function **LL_TIM_SetRepetitionCounter()**.

126.1.2 LL_TIM_OC_InitTypeDef

LL_TIM_OC_InitTypeDef is defined in the stm32h7xx_ll_tim.h

Data Fields

- ***uint32_t OCMode***
- ***uint32_t OCState***
- ***uint32_t OCNState***
- ***uint32_t CompareValue***
- ***uint32_t OCPolarity***
- ***uint32_t OCNPolarity***
- ***uint32_t OCIdleState***

- ***uint32_t OCNIdleState***

Field Documentation

- ***uint32_t LL_TIM_OC_InitTypeDef::OCMode***

  Specifies the output mode. This parameter can be a value of ***TIM_LL_EC_OCMODE***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetMode()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::OCState***

  Specifies the TIM Output Compare state. This parameter can be a value of ***TIM_LL_EC_OCSTATE***.This feature can be modified afterwards using unitary functions **LL_TIM_CC_EnableChannel()** or **LL_TIM_CC_DisableChannel()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::OCNState***

  Specifies the TIM complementary Output Compare state. This parameter can be a value of ***TIM_LL_EC_OCSTATE***.This feature can be modified afterwards using unitary functions **LL_TIM_CC_EnableChannel()** or **LL_TIM_CC_DisableChannel()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::CompareValue***

  Specifies the Compare value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data=0x0000 and Max_Data=0xFFFF.This feature can be modified afterwards using unitary function LL_TIM_OC_SetCompareCHx (x=1..6).
- ***uint32_t LL_TIM_OC_InitTypeDef::OCPolarity***

  Specifies the output polarity. This parameter can be a value of ***TIM_LL_EC_OCPOLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetPolarity()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::OCNPolarity***

  Specifies the complementary output polarity. This parameter can be a value of ***TIM_LL_EC_OCPOLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetPolarity()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::OCIdleState***

  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_LL_EC_OCIDLESTATE***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetIdleState()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::OCNIdleState***

  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_LL_EC_OCIDLESTATE***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetIdleState()**.

126.1.3 LL_TIM_IC_InitTypeDef

LL_TIM_IC_InitTypeDef is defined in the stm32h7xx_ll_tim.h

Data Fields

- ***uint32_t ICPolarity***
- ***uint32_t ICActiveInput***
- ***uint32_t ICPrescaler***
- ***uint32_t ICFilter***

Field Documentation

- ***uint32_t LL_TIM_IC_InitTypeDef::ICPolarity***

  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_LL_EC_IC_POLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.
- ***uint32_t LL_TIM_IC_InitTypeDef::ICActiveInput***

  Specifies the input. This parameter can be a value of ***TIM_LL_EC_ACTIVEINPUT***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetActiveInput()**.
- ***uint32_t LL_TIM_IC_InitTypeDef::ICPrescaler***

  Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_LL_EC_ICPSC***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.

- ***uint32_t LL_TIM_IC_InitTypeDef::ICFilter***

  Specifies the input capture filter. This parameter can be a value of ***TIM_LL_EC_IC_FILTER***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.

126.1.4 LL_TIM_ENCODER_InitTypeDef

LL_TIM_ENCODER_InitTypeDef is defined in the stm32h7xx_ll_tim.h

Data Fields

- ***uint32_t EncoderMode***
- ***uint32_t IC1Polarity***
- ***uint32_t IC1ActiveInput***
- ***uint32_t IC1Prescaler***
- ***uint32_t IC1Filter***
- ***uint32_t IC2Polarity***
- ***uint32_t IC2ActiveInput***
- ***uint32_t IC2Prescaler***
- ***uint32_t IC2Filter***

Field Documentation

- ***uint32_t LL_TIM_ENCODER_InitTypeDef::EncoderMode***

  Specifies the encoder resolution (x2 or x4). This parameter can be a value of ***TIM_LL_EC_ENCODERMODE***.This feature can be modified afterwards using unitary function **LL_TIM_SetEncoderMode()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1Polarity***

  Specifies the active edge of TI1 input. This parameter can be a value of ***TIM_LL_EC_IC_POLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1ActiveInput***

  Specifies the TI1 input source This parameter can be a value of ***TIM_LL_EC_ACTIVEINPUT***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetActiveInput()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1Prescaler***

  Specifies the TI1 input prescaler value. This parameter can be a value of ***TIM_LL_EC_ICPSC***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1Filter***

  Specifies the TI1 input filter. This parameter can be a value of ***TIM_LL_EC_IC_FILTER***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2Polarity***

  Specifies the active edge of TI2 input. This parameter can be a value of ***TIM_LL_EC_IC_POLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2ActiveInput***

  Specifies the TI2 input source This parameter can be a value of ***TIM_LL_EC_ACTIVEINPUT***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetActiveInput()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2Prescaler***

  Specifies the TI2 input prescaler value. This parameter can be a value of ***TIM_LL_EC_ICPSC***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2Filter***

  Specifies the TI2 input filter. This parameter can be a value of ***TIM_LL_EC_IC_FILTER***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.

126.1.5 LL_TIM_HALLSENSOR_InitTypeDef

LL_TIM_HALLSENSOR_InitTypeDef is defined in the stm32h7xx_ll_tim.h

Data Fields

- ***uint32_t IC1Polarity***
- ***uint32_t IC1Prescaler***

- ***uint32_t IC1Filter***
- ***uint32_t CommutationDelay***

Field Documentation

- ***uint32_t LL_TIM_HALLSENSOR_InitTypeDef::IC1Polarity***

  Specifies the active edge of TI1 input. This parameter can be a value of ***TIM_LL_EC_IC_POLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.
- ***uint32_t LL_TIM_HALLSENSOR_InitTypeDef::IC1Prescaler***

  Specifies the TI1 input prescaler value. Prescaler must be set to get a maximum counter period longer than the time interval between 2 consecutive changes on the Hall inputs. This parameter can be a value of ***TIM_LL_EC_ICPSC***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.
- ***uint32_t LL_TIM_HALLSENSOR_InitTypeDef::IC1Filter***

  Specifies the TI1 input filter. This parameter can be a value of ***TIM_LL_EC_IC_FILTER***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.
- ***uint32_t LL_TIM_HALLSENSOR_InitTypeDef::CommutationDelay***

  Specifies the compare value to be loaded into the Capture Compare Register. A positive pulse (TRGO event) is generated with a programmable delay every time a change occurs on the Hall inputs. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetCompareCH2()**.

126.1.6 LL_TIM_BDTR_InitTypeDef

LL_TIM_BDTR_InitTypeDef is defined in the stm32h7xx_ll_tim.h

Data Fields

- ***uint32_t OSSRState***
- ***uint32_t OSSIState***
- ***uint32_t LockLevel***
- ***uint8_t DeadTime***
- ***uint16_t BreakState***
- ***uint32_t BreakPolarity***
- ***uint32_t BreakFilter***
- ***uint32_t Break2State***
- ***uint32_t Break2Polarity***
- ***uint32_t Break2Filter***
- ***uint32_t AutomaticOutput***

Field Documentation

- ***uint32_t LL_TIM_BDTR_InitTypeDef::OSSRState***

  Specifies the Off-State selection used in Run mode. This parameter can be a value of ***TIM_LL_EC_OSSR***This feature can be modified afterwards using unitary function **LL_TIM_SetOffStates()**

  Note:

  – This bit-field cannot be modified as long as LOCK level 2 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::OSSIState***

  Specifies the Off-State used in Idle state. This parameter can be a value of ***TIM_LL_EC_OSSI***This feature can be modified afterwards using unitary function **LL_TIM_SetOffStates()**

  Note:

  – This bit-field cannot be modified as long as LOCK level 2 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::LockLevel***

  Specifies the LOCK level parameters. This parameter can be a value of ***TIM_LL_EC_LOCKLEVEL***

  Note:

  – The LOCK bits can be written only once after the reset. Once the TIMx_BDTR register has been written, their content is frozen until the next reset.

- ***uint8_t LL_TIM_BDTR_InitTypeDef::DeadTime***

  Specifies the delay time between the switching-off and the switching-on of the outputs. This parameter can be a number between Min_Data = 0x00 and Max_Data = 0xFF.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetDeadTime()**

  Note:

  – This bit-field can not be modified as long as LOCK level 1, 2 or 3 has been programmed.

- ***uint16_t LL_TIM_BDTR_InitTypeDef::BreakState***

  Specifies whether the TIM Break input is enabled or not. This parameter can be a value of ***TIM_LL_EC_BREAK_ENABLE***This feature can be modified afterwards using unitary functions **LL_TIM_EnableBRK()** or **LL_TIM_DisableBRK()**

  Note:

  – This bit-field can not be modified as long as LOCK level 1 has been programmed.

- ***uint32_t LL_TIM_BDTR_InitTypeDef::BreakPolarity***

  Specifies the TIM Break Input pin polarity. This parameter can be a value of ***TIM_LL_EC_BREAK_POLARITY***This feature can be modified afterwards using unitary function **LL_TIM_ConfigBRK()**

  Note:

  – This bit-field can not be modified as long as LOCK level 1 has been programmed.

- ***uint32_t LL_TIM_BDTR_InitTypeDef::BreakFilter***

  Specifies the TIM Break Filter. This parameter can be a value of ***TIM_LL_EC_BREAK_FILTER***This feature can be modified afterwards using unitary function **LL_TIM_ConfigBRK()**

  Note:

  – This bit-field can not be modified as long as LOCK level 1 has been programmed.

- ***uint32_t LL_TIM_BDTR_InitTypeDef::Break2State***

  Specifies whether the TIM Break2 input is enabled or not. This parameter can be a value of ***TIM_LL_EC_BREAK2_ENABLE***This feature can be modified afterwards using unitary functions **LL_TIM_EnableBRK2()** or **LL_TIM_DisableBRK2()**

  Note:

  – This bit-field can not be modified as long as LOCK level 1 has been programmed.

- ***uint32_t LL_TIM_BDTR_InitTypeDef::Break2Polarity***

  Specifies the TIM Break2 Input pin polarity. This parameter can be a value of ***TIM_LL_EC_BREAK2_POLARITY***This feature can be modified afterwards using unitary function **LL_TIM_ConfigBRK2()**

  Note:

  – This bit-field can not be modified as long as LOCK level 1 has been programmed.

- ***uint32_t LL_TIM_BDTR_InitTypeDef::Break2Filter***

  Specifies the TIM Break2 Filter. This parameter can be a value of ***TIM_LL_EC_BREAK2_FILTER***This feature can be modified afterwards using unitary function **LL_TIM_ConfigBRK2()**

  Note:

  – This bit-field can not be modified as long as LOCK level 1 has been programmed.

- ***uint32_t LL_TIM_BDTR_InitTypeDef::AutomaticOutput***

  Specifies whether the TIM Automatic Output feature is enabled or not. This parameter can be a value of ***TIM_LL_EC_AUTOMATICOUTPUT_ENABLE***This feature can be modified afterwards using unitary functions **LL_TIM_EnableAutomaticOutput()** or **LL_TIM_DisableAutomaticOutput()**

  Note:

  – This bit-field can not be modified as long as LOCK level 1 has been programmed.

126.2 TIM Firmware driver API description

The following section lists the various functions of the TIM library.

126.2.1 Detailed description of functions

LL_TIM_EnableCounter

Function name

__STATIC_INLINE void LL_TIM_EnableCounter (TIM_TypeDef * TIMx)

Function description

Enable timer counter.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CEN LL_TIM_EnableCounter

LL_TIM_DisableCounter

Function name

__STATIC_INLINE void LL_TIM_DisableCounter (TIM_TypeDef * TIMx)

Function description

Disable timer counter.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CEN LL_TIM_DisableCounter

LL_TIM_IsEnabledCounter

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledCounter (const TIM_TypeDef * TIMx)

Function description

Indicates whether the timer counter is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 CEN LL_TIM_IsEnabledCounter

LL_TIM_EnableUpdateEvent

Function name

__STATIC_INLINE void LL_TIM_EnableUpdateEvent (TIM_TypeDef * TIMx)

Function description

Enable update event generation.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 UDIS LL_TIM_EnableUpdateEvent

LL_TIM_DisableUpdateEvent

Function name

__STATIC_INLINE void LL_TIM_DisableUpdateEvent (TIM_TypeDef * TIMx)

Function description

Disable update event generation.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 UDIS LL_TIM_DisableUpdateEvent

LL_TIM_IsEnabledUpdateEvent

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledUpdateEvent (const TIM_TypeDef * TIMx)

Function description

Indicates whether update event generation is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **Inverted:** state of bit (0 or 1).

Reference Manual to LL API cross reference:

- CR1 UDIS LL_TIM_IsEnabledUpdateEvent

LL_TIM_SetUpdateSource

Function name

__STATIC_INLINE void LL_TIM_SetUpdateSource (TIM_TypeDef * TIMx, uint32_t UpdateSource)

Function description

Set update event source.

Parameters

- **TIMx:** Timer instance
- **UpdateSource:** This parameter can be one of the following values:
  - LL_TIM_UPDATESOURCE_REGULAR
  - LL_TIM_UPDATESOURCE_COUNTER

Return values

- **None:**

Notes

- Update event source set to LL_TIM_UPDATESOURCE_REGULAR: any of the following events generate an update interrupt or DMA request if enabled: Counter overflow/underflowSetting the UG bitUpdate generation through the slave mode controller
- Update event source set to LL_TIM_UPDATESOURCE_COUNTER: only counter overflow/underflow generates an update interrupt or DMA request if enabled.

Reference Manual to LL API cross reference:

- CR1 URS LL_TIM_SetUpdateSource

LL_TIM_GetUpdateSource

Function name

__STATIC_INLINE uint32_t LL_TIM_GetUpdateSource (const TIM_TypeDef * TIMx)

Function description

Get actual event update source.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_UPDATESOURCE_REGULAR
  - LL_TIM_UPDATESOURCE_COUNTER

Reference Manual to LL API cross reference:

- CR1 URS LL_TIM_GetUpdateSource

LL_TIM_SetOnePulseMode

Function name

__STATIC_INLINE void LL_TIM_SetOnePulseMode (TIM_TypeDef * TIMx, uint32_t OnePulseMode)

Function description

Set one pulse mode (one shot v.s.

Parameters

- **TIMx:** Timer instance
- **OnePulseMode:** This parameter can be one of the following values:
  - LL_TIM_ONEPULSEMODE_SINGLE
  - LL_TIM_ONEPULSEMODE_REPETITIVE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 OPM LL_TIM_SetOnePulseMode

LL_TIM_GetOnePulseMode

Function name

__STATIC_INLINE uint32_t LL_TIM_GetOnePulseMode (const TIM_TypeDef * TIMx)

Function description

Get actual one pulse mode.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_ONEPULSEMODE_SINGLE
  - LL_TIM_ONEPULSEMODE_REPETITIVE

Reference Manual to LL API cross reference:

- CR1 OPM LL_TIM_GetOnePulseMode

LL_TIM_SetCounterMode

Function name

__STATIC_INLINE void LL_TIM_SetCounterMode (TIM_TypeDef * TIMx, uint32_t CounterMode)

Function description

Set the timer counter counting mode.

Parameters

- **TIMx:** Timer instance
- **CounterMode:** This parameter can be one of the following values:
  - LL_TIM_COUNTERMODE_UP
  - LL_TIM_COUNTERMODE_DOWN
  - LL_TIM_COUNTERMODE_CENTER_UP
  - LL_TIM_COUNTERMODE_CENTER_DOWN
  - LL_TIM_COUNTERMODE_CENTER_UP_DOWN

Return values

- **None:**

Notes

- Macro IS_TIM_COUNTER_MODE_SELECT_INSTANCE(TIMx) can be used to check whether or not the counter mode selection feature is supported by a timer instance.
- Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode.

Reference Manual to LL API cross reference:

- CR1 DIR LL_TIM_SetCounterMode
- CR1 CMS LL_TIM_SetCounterMode

LL_TIM_GetCounterMode

Function name

__STATIC_INLINE uint32_t LL_TIM_GetCounterMode (const TIM_TypeDef * TIMx)

Function description

Get actual counter mode.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_COUNTERMODE_UP
  - LL_TIM_COUNTERMODE_DOWN
  - LL_TIM_COUNTERMODE_CENTER_UP
  - LL_TIM_COUNTERMODE_CENTER_DOWN
  - LL_TIM_COUNTERMODE_CENTER_UP_DOWN

Notes

- Macro IS_TIM_COUNTER_MODE_SELECT_INSTANCE(TIMx) can be used to check whether or not the counter mode selection feature is supported by a timer instance.

Reference Manual to LL API cross reference:

- CR1 DIR LL_TIM_GetCounterMode
- CR1 CMS LL_TIM_GetCounterMode

LL_TIM_EnableARRPreload

Function name

__STATIC_INLINE void LL_TIM_EnableARRPreload (TIM_TypeDef * TIMx)

Function description

Enable auto-reload (ARR) preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 ARPE LL_TIM_EnableARRPreload

LL_TIM_DisableARRPreload

Function name

__STATIC_INLINE void LL_TIM_DisableARRPreload (TIM_TypeDef * TIMx)

Function description

Disable auto-reload (ARR) preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 ARPE LL_TIM_DisableARRPreload

LL_TIM_IsEnabledARRPreload

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledARRPreload (const TIM_TypeDef * TIMx)

Function description

Indicates whether auto-reload (ARR) preload is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 ARPE LL_TIM_IsEnabledARRPreload

LL_TIM_SetClockDivision

Function name

__STATIC_INLINE void LL_TIM_SetClockDivision (TIM_TypeDef * TIMx, uint32_t ClockDivision)

Function description

Set the division ratio between the timer clock and the sampling clock used by the dead-time generators (when supported) and the digital filters.

Parameters

- **TIMx:** Timer instance
- **ClockDivision:** This parameter can be one of the following values:
  - LL_TIM_CLOCKDIVISION_DIV1
  - LL_TIM_CLOCKDIVISION_DIV2
  - LL_TIM_CLOCKDIVISION_DIV4

Return values

- **None:**

Notes

- Macro IS_TIM_CLOCK_DIVISION_INSTANCE(TIMx) can be used to check whether or not the clock division feature is supported by the timer instance.

Reference Manual to LL API cross reference:

- CR1 CKD LL_TIM_SetClockDivision

LL_TIM_GetClockDivision

Function name

__STATIC_INLINE uint32_t LL_TIM_GetClockDivision (const TIM_TypeDef * TIMx)

Function description

Get the actual division ratio between the timer clock and the sampling clock used by the dead-time generators (when supported) and the digital filters.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_CLOCKDIVISION_DIV1
  - LL_TIM_CLOCKDIVISION_DIV2
  - LL_TIM_CLOCKDIVISION_DIV4

Notes

- Macro IS_TIM_CLOCK_DIVISION_INSTANCE(TIMx) can be used to check whether or not the clock division feature is supported by the timer instance.

Reference Manual to LL API cross reference:

- CR1 CKD LL_TIM_GetClockDivision

LL_TIM_SetCounter

Function name

__STATIC_INLINE void LL_TIM_SetCounter (TIM_TypeDef * TIMx, uint32_t Counter)

Function description

Set the counter value.

Parameters

- **TIMx:** Timer instance
- **Counter:** Counter value (between Min_Data=0 and Max_Data=0xFFFF or 0xFFFFFFFF)

Return values

- **None:**

Notes

- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.

Reference Manual to LL API cross reference:

- CNT CNT LL_TIM_SetCounter

LL_TIM_GetCounter

Function name

__STATIC_INLINE uint32_t LL_TIM_GetCounter (const TIM_TypeDef * TIMx)

Function description

Get the counter value.

Parameters

- **TIMx:** Timer instance

Return values

- **Counter:** value (between Min_Data=0 and Max_Data=0xFFFF or 0xFFFFFFFF)

Notes

- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.

Reference Manual to LL API cross reference:

- CNT CNT LL_TIM_GetCounter

LL_TIM_GetDirection

Function name

__STATIC_INLINE uint32_t LL_TIM_GetDirection (const TIM_TypeDef * TIMx)

Function description

Get the current direction of the counter.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_COUNTERDIRECTION_UP
  - LL_TIM_COUNTERDIRECTION_DOWN

Reference Manual to LL API cross reference:

- CR1 DIR LL_TIM_GetDirection

LL_TIM_SetPrescaler

Function name

__STATIC_INLINE void LL_TIM_SetPrescaler (TIM_TypeDef * TIMx, uint32_t Prescaler)

Function description

Set the prescaler value.

Parameters

- **TIMx:** Timer instance
- **Prescaler:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- The counter clock frequency CK_CNT is equal to fCK_PSC / (PSC[15:0] + 1).
- The prescaler can be changed on the fly as this control register is buffered. The new prescaler ratio is taken into account at the next update event.
- Helper macro __LL_TIM_CALC_PSC can be used to calculate the Prescaler parameter

Reference Manual to LL API cross reference:

- PSC PSC LL_TIM_SetPrescaler

LL_TIM_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_TIM_GetPrescaler (const TIM_TypeDef * TIMx)

Function description

Get the prescaler value.

Parameters

- **TIMx:** Timer instance

Return values

- **Prescaler:** value between Min_Data=0 and Max_Data=65535

Reference Manual to LL API cross reference:

- PSC PSC LL_TIM_GetPrescaler

LL_TIM_SetAutoReload

Function name

__STATIC_INLINE void LL_TIM_SetAutoReload (TIM_TypeDef * TIMx, uint32_t AutoReload)

Function description

Set the auto-reload value.

Parameters

- **TIMx:** Timer instance
- **AutoReload:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- The counter is blocked while the auto-reload value is null.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Helper macro __LL_TIM_CALC_ARR can be used to calculate the AutoReload parameter

Reference Manual to LL API cross reference:

- ARR ARR LL_TIM_SetAutoReload

LL_TIM_GetAutoReload

Function name

__STATIC_INLINE uint32_t LL_TIM_GetAutoReload (const TIM_TypeDef * TIMx)

Function description

Get the auto-reload value.

Parameters

- **TIMx:** Timer instance

Return values

- **Auto-reload:** value

Notes

- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.

Reference Manual to LL API cross reference:

- ARR ARR LL_TIM_GetAutoReload

LL_TIM_SetRepetitionCounter

Function name

__STATIC_INLINE void LL_TIM_SetRepetitionCounter (TIM_TypeDef * TIMx, uint32_t RepetitionCounter)

Function description

Set the repetition counter value.

Parameters

- **TIMx:** Timer instance
- **RepetitionCounter:** between Min_Data=0 and Max_Data=255 or 65535 for advanced timer.

Return values

- **None:**

Notes

- For advanced timer instances RepetitionCounter can be up to 65535.
- Macro IS_TIM_REPETITION_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a repetition counter.

Reference Manual to LL API cross reference:

- RCR REP LL_TIM_SetRepetitionCounter

LL_TIM_GetRepetitionCounter

Function name

__STATIC_INLINE uint32_t LL_TIM_GetRepetitionCounter (const TIM_TypeDef * TIMx)

Function description

Get the repetition counter value.

Parameters

- **TIMx:** Timer instance

Return values

- **Repetition:** counter value

Notes

- Macro IS_TIM_REPETITION_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a repetition counter.

Reference Manual to LL API cross reference:

- RCR REP LL_TIM_GetRepetitionCounter

LL_TIM_EnableUIFRemap

Function name

__STATIC_INLINE void LL_TIM_EnableUIFRemap (TIM_TypeDef * TIMx)

Function description

Force a continuous copy of the update interrupt flag (UIF) into the timer counter register (bit 31).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- This allows both the counter value and a potential roll-over condition signalled by the UIFCPY flag to be read in an atomic way.

Reference Manual to LL API cross reference:

- CR1 UIFREMAP LL_TIM_EnableUIFRemap

LL_TIM_DisableUIFRemap

Function name

__STATIC_INLINE void LL_TIM_DisableUIFRemap (TIM_TypeDef * TIMx)

Function description

Disable update interrupt flag (UIF) remapping.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 UIFREMAP LL_TIM_DisableUIFRemap

LL_TIM_IsActiveUIFCPY

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveUIFCPY (const uint32_t Counter)

Function description

Indicate whether update interrupt flag (UIF) copy is set.

Parameters

- **Counter:** Counter value

Return values

- **State:** of bit (1 or 0).

LL_TIM_CC_EnablePreload

Function name

__STATIC_INLINE void LL_TIM_CC_EnablePreload (TIM_TypeDef * TIMx)

Function description

Enable the capture/compare control bits (CCxE, CCxNE and OCxM) preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- CCxE, CCxNE and OCxM bits are preloaded, after having been written, they are updated only when a commutation event (COM) occurs.
- Only on channels that have a complementary output.
- Macro IS_TIM_COMMUTATION_EVENT_INSTANCE(TIMx) can be used to check whether or not a timer instance is able to generate a commutation event.

Reference Manual to LL API cross reference:

- CR2 CCPC LL_TIM_CC_EnablePreload

LL_TIM_CC_DisablePreload

Function name

__STATIC_INLINE void LL_TIM_CC_DisablePreload (TIM_TypeDef * TIMx)

Function description

Disable the capture/compare control bits (CCxE, CCxNE and OCxM) preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_COMMUTATION_EVENT_INSTANCE(TIMx) can be used to check whether or not a timer instance is able to generate a commutation event.

Reference Manual to LL API cross reference:

- CR2 CCPC LL_TIM_CC_DisablePreload

LL_TIM_CC_SetUpdate

Function name

__STATIC_INLINE void LL_TIM_CC_SetUpdate (TIM_TypeDef * TIMx, uint32_t CCUpdateSource)

Function description

Set the updated source of the capture/compare control bits (CCxE, CCxNE and OCxM).

Parameters

- **TIMx:** Timer instance
- **CCUpdateSource:** This parameter can be one of the following values:
  - LL_TIM_CCUPDATESOURCE_COMG_ONLY
  - LL_TIM_CCUPDATESOURCE_COMG_AND_TRGI

Return values

- **None:**

Notes

- Macro IS_TIM_COMMUTATION_EVENT_INSTANCE(TIMx) can be used to check whether or not a timer instance is able to generate a commutation event.

Reference Manual to LL API cross reference:

- CR2 CCUS LL_TIM_CC_SetUpdate

LL_TIM_CC_SetDMAReqTrigger

Function name

__STATIC_INLINE void LL_TIM_CC_SetDMAReqTrigger (TIM_TypeDef * TIMx, uint32_t DMAReqTrigger)

Function description

Set the trigger of the capture/compare DMA request.

Parameters

- **TIMx:** Timer instance
- **DMAReqTrigger:** This parameter can be one of the following values:
  - LL_TIM_CCDMAREQUEST_CC
  - LL_TIM_CCDMAREQUEST_UPDATE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 CCDS LL_TIM_CC_SetDMAReqTrigger

LL_TIM_CC_GetDMAReqTrigger

Function name

__STATIC_INLINE uint32_t LL_TIM_CC_GetDMAReqTrigger (const TIM_TypeDef * TIMx)

Function description

Get actual trigger of the capture/compare DMA request.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_CCDMAREQUEST_CC
  - LL_TIM_CCDMAREQUEST_UPDATE

Reference Manual to LL API cross reference:

- CR2 CCDS LL_TIM_CC_GetDMAReqTrigger

LL_TIM_CC_SetLockLevel

Function name

__STATIC_INLINE void LL_TIM_CC_SetLockLevel (TIM_TypeDef * TIMx, uint32_t LockLevel)

Function description

Set the lock level to freeze the configuration of several capture/compare parameters.

Parameters

- **TIMx:** Timer instance
- **LockLevel:** This parameter can be one of the following values:
  - LL_TIM_LOCKLEVEL_OFF
  - LL_TIM_LOCKLEVEL_1
  - LL_TIM_LOCKLEVEL_2
  - LL_TIM_LOCKLEVEL_3

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not the lock mechanism is supported by a timer instance.

Reference Manual to LL API cross reference:

- BDTR LOCK LL_TIM_CC_SetLockLevel

LL_TIM_CC_EnableChannel

Function name

__STATIC_INLINE void LL_TIM_CC_EnableChannel (TIM_TypeDef * TIMx, uint32_t Channels)

Function description

Enable capture/compare channels.

Parameters

- **TIMx:** Timer instance
- **Channels:** This parameter can be a combination of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCER CC1E LL_TIM_CC_EnableChannel
- CCER CC1NE LL_TIM_CC_EnableChannel
- CCER CC2E LL_TIM_CC_EnableChannel
- CCER CC2NE LL_TIM_CC_EnableChannel
- CCER CC3E LL_TIM_CC_EnableChannel
- CCER CC3NE LL_TIM_CC_EnableChannel
- CCER CC4E LL_TIM_CC_EnableChannel
- CCER CC5E LL_TIM_CC_EnableChannel
- CCER CC6E LL_TIM_CC_EnableChannel

LL_TIM_CC_DisableChannel

Function name

__STATIC_INLINE void LL_TIM_CC_DisableChannel (TIM_TypeDef * TIMx, uint32_t Channels)

Function description

Disable capture/compare channels.

Parameters

- **TIMx:** Timer instance
- **Channels:** This parameter can be a combination of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCER CC1E LL_TIM_CC_DisableChannel
- CCER CC1NE LL_TIM_CC_DisableChannel
- CCER CC2E LL_TIM_CC_DisableChannel
- CCER CC2NE LL_TIM_CC_DisableChannel
- CCER CC3E LL_TIM_CC_DisableChannel
- CCER CC3NE LL_TIM_CC_DisableChannel
- CCER CC4E LL_TIM_CC_DisableChannel
- CCER CC5E LL_TIM_CC_DisableChannel
- CCER CC6E LL_TIM_CC_DisableChannel

LL_TIM_CC_IsEnabledChannel

Function name

__STATIC_INLINE uint32_t LL_TIM_CC_IsEnabledChannel (TIM_TypeDef * TIMx, uint32_t Channels)

Function description

Indicate whether channel(s) is(are) enabled.

Parameters

- **TIMx:** Timer instance
- **Channels:** This parameter can be a combination of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCER CC1E LL_TIM_CC_IsEnabledChannel
- CCER CC1NE LL_TIM_CC_IsEnabledChannel
- CCER CC2E LL_TIM_CC_IsEnabledChannel
- CCER CC2NE LL_TIM_CC_IsEnabledChannel
- CCER CC3E LL_TIM_CC_IsEnabledChannel
- CCER CC3NE LL_TIM_CC_IsEnabledChannel
- CCER CC4E LL_TIM_CC_IsEnabledChannel
- CCER CC5E LL_TIM_CC_IsEnabledChannel
- CCER CC6E LL_TIM_CC_IsEnabledChannel

LL_TIM_OC_ConfigOutput

Function name

__STATIC_INLINE void LL_TIM_OC_ConfigOutput (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Configuration)

Function description

Configure an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_TIM_OCPOLARITY_HIGH or LL_TIM_OCPOLARITY_LOW
  - LL_TIM_OCIDLESTATE_LOW or LL_TIM_OCIDLESTATE_HIGH

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 CC1S LL_TIM_OC_ConfigOutput
- CCMR1 CC2S LL_TIM_OC_ConfigOutput
- CCMR2 CC3S LL_TIM_OC_ConfigOutput
- CCMR2 CC4S LL_TIM_OC_ConfigOutput
- CCMR3 CC5S LL_TIM_OC_ConfigOutput
- CCMR3 CC6S LL_TIM_OC_ConfigOutput
- CCER CC1P LL_TIM_OC_ConfigOutput
- CCER CC2P LL_TIM_OC_ConfigOutput
- CCER CC3P LL_TIM_OC_ConfigOutput
- CCER CC4P LL_TIM_OC_ConfigOutput
- CCER CC5P LL_TIM_OC_ConfigOutput
- CCER CC6P LL_TIM_OC_ConfigOutput
- CR2 OIS1 LL_TIM_OC_ConfigOutput
- CR2 OIS2 LL_TIM_OC_ConfigOutput
- CR2 OIS3 LL_TIM_OC_ConfigOutput
- CR2 OIS4 LL_TIM_OC_ConfigOutput
- CR2 OIS5 LL_TIM_OC_ConfigOutput
- CR2 OIS6 LL_TIM_OC_ConfigOutput

LL_TIM_OC_SetMode

Function name

__STATIC_INLINE void LL_TIM_OC_SetMode (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Mode)

Function description

Define the behavior of the output reference signal OCxREF from which OCx and OCxN (when relevant) are derived.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6
- **Mode:** This parameter can be one of the following values:
  - LL_TIM_OCMODE_FROZEN
  - LL_TIM_OCMODE_ACTIVE
  - LL_TIM_OCMODE_INACTIVE
  - LL_TIM_OCMODE_TOGGLE
  - LL_TIM_OCMODE_FORCED_INACTIVE
  - LL_TIM_OCMODE_FORCED_ACTIVE
  - LL_TIM_OCMODE_PWM1
  - LL_TIM_OCMODE_PWM2
  - LL_TIM_OCMODE_RETRIG_OPM1
  - LL_TIM_OCMODE_RETRIG_OPM2
  - LL_TIM_OCMODE_COMBINED_PWM1
  - LL_TIM_OCMODE_COMBINED_PWM2
  - LL_TIM_OCMODE_ASSYMETRIC_PWM1
  - LL_TIM_OCMODE_ASSYMETRIC_PWM2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 OC1M LL_TIM_OC_SetMode
- CCMR1 OC2M LL_TIM_OC_SetMode
- CCMR2 OC3M LL_TIM_OC_SetMode
- CCMR2 OC4M LL_TIM_OC_SetMode
- CCMR3 OC5M LL_TIM_OC_SetMode
- CCMR3 OC6M LL_TIM_OC_SetMode

LL_TIM_OC_GetMode

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetMode (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the output compare mode of an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_OCMODE_FROZEN
  - LL_TIM_OCMODE_ACTIVE
  - LL_TIM_OCMODE_INACTIVE
  - LL_TIM_OCMODE_TOGGLE
  - LL_TIM_OCMODE_FORCED_INACTIVE
  - LL_TIM_OCMODE_FORCED_ACTIVE
  - LL_TIM_OCMODE_PWM1
  - LL_TIM_OCMODE_PWM2
  - LL_TIM_OCMODE_RETRIG_OPM1
  - LL_TIM_OCMODE_RETRIG_OPM2
  - LL_TIM_OCMODE_COMBINED_PWM1
  - LL_TIM_OCMODE_COMBINED_PWM2
  - LL_TIM_OCMODE_ASSYMETRIC_PWM1
  - LL_TIM_OCMODE_ASSYMETRIC_PWM2

Reference Manual to LL API cross reference:

- CCMR1 OC1M LL_TIM_OC_GetMode
- CCMR1 OC2M LL_TIM_OC_GetMode
- CCMR2 OC3M LL_TIM_OC_GetMode
- CCMR2 OC4M LL_TIM_OC_GetMode
- CCMR3 OC5M LL_TIM_OC_GetMode
- CCMR3 OC6M LL_TIM_OC_GetMode

LL_TIM_OC_SetPolarity

Function name

__STATIC_INLINE void LL_TIM_OC_SetPolarity (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Polarity)

Function description

Set the polarity of an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6
- **Polarity:** This parameter can be one of the following values:
  - LL_TIM_OCPOLARITY_HIGH
  - LL_TIM_OCPOLARITY_LOW

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCER CC1P LL_TIM_OC_SetPolarity
- CCER CC1NP LL_TIM_OC_SetPolarity
- CCER CC2P LL_TIM_OC_SetPolarity
- CCER CC2NP LL_TIM_OC_SetPolarity
- CCER CC3P LL_TIM_OC_SetPolarity
- CCER CC3NP LL_TIM_OC_SetPolarity
- CCER CC4P LL_TIM_OC_SetPolarity
- CCER CC5P LL_TIM_OC_SetPolarity
- CCER CC6P LL_TIM_OC_SetPolarity

LL_TIM_OC_GetPolarity

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetPolarity (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the polarity of an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_OCPOLARITY_HIGH
  - LL_TIM_OCPOLARITY_LOW

Reference Manual to LL API cross reference:

- CCER CC1P LL_TIM_OC_GetPolarity
- CCER CC1NP LL_TIM_OC_GetPolarity
- CCER CC2P LL_TIM_OC_GetPolarity
- CCER CC2NP LL_TIM_OC_GetPolarity
- CCER CC3P LL_TIM_OC_GetPolarity
- CCER CC3NP LL_TIM_OC_GetPolarity
- CCER CC4P LL_TIM_OC_GetPolarity
- CCER CC5P LL_TIM_OC_GetPolarity
- CCER CC6P LL_TIM_OC_GetPolarity

LL_TIM_OC_SetIdleState

Function name

__STATIC_INLINE void LL_TIM_OC_SetIdleState (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t IdleState)

Function description

Set the IDLE state of an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6
- **IdleState:** This parameter can be one of the following values:
  - LL_TIM_OCIDLESTATE_LOW
  - LL_TIM_OCIDLESTATE_HIGH

Return values

- **None:**

Notes

- This function is significant only for the timer instances supporting the break feature. Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- CR2 OIS1 LL_TIM_OC_SetIdleState
- CR2 OIS2N LL_TIM_OC_SetIdleState
- CR2 OIS2 LL_TIM_OC_SetIdleState
- CR2 OIS2N LL_TIM_OC_SetIdleState
- CR2 OIS3 LL_TIM_OC_SetIdleState
- CR2 OIS3N LL_TIM_OC_SetIdleState
- CR2 OIS4 LL_TIM_OC_SetIdleState
- CR2 OIS5 LL_TIM_OC_SetIdleState
- CR2 OIS6 LL_TIM_OC_SetIdleState

LL_TIM_OC_GetIdleState

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetIdleState (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the IDLE state of an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_OCIDLESTATE_LOW
  - LL_TIM_OCIDLESTATE_HIGH

Reference Manual to LL API cross reference:

- CR2 OIS1 LL_TIM_OC_GetIdleState
- CR2 OIS2N LL_TIM_OC_GetIdleState
- CR2 OIS2 LL_TIM_OC_GetIdleState
- CR2 OIS2N LL_TIM_OC_GetIdleState
- CR2 OIS3 LL_TIM_OC_GetIdleState
- CR2 OIS3N LL_TIM_OC_GetIdleState
- CR2 OIS4 LL_TIM_OC_GetIdleState
- CR2 OIS5 LL_TIM_OC_GetIdleState
- CR2 OIS6 LL_TIM_OC_GetIdleState

LL_TIM_OC_EnableFast

Function name

__STATIC_INLINE void LL_TIM_OC_EnableFast (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Enable fast mode for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Notes

- Acts only if the channel is configured in PWM1 or PWM2 mode.

Reference Manual to LL API cross reference:

- CCMR1 OC1FE LL_TIM_OC_EnableFast
- CCMR1 OC2FE LL_TIM_OC_EnableFast
- CCMR2 OC3FE LL_TIM_OC_EnableFast
- CCMR2 OC4FE LL_TIM_OC_EnableFast
- CCMR3 OC5FE LL_TIM_OC_EnableFast
- CCMR3 OC6FE LL_TIM_OC_EnableFast

LL_TIM_OC_DisableFast

Function name

__STATIC_INLINE void LL_TIM_OC_DisableFast (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Disable fast mode for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 OC1FE LL_TIM_OC_DisableFast
- CCMR1 OC2FE LL_TIM_OC_DisableFast
- CCMR2 OC3FE LL_TIM_OC_DisableFast
- CCMR2 OC4FE LL_TIM_OC_DisableFast
- CCMR3 OC5FE LL_TIM_OC_DisableFast
- CCMR3 OC6FE LL_TIM_OC_DisableFast

LL_TIM_OC_IsEnabledFast

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_IsEnabledFast (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Indicates whether fast mode is enabled for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCMR1 OC1FE LL_TIM_OC_IsEnabledFast
- CCMR1 OC2FE LL_TIM_OC_IsEnabledFast
- CCMR2 OC3FE LL_TIM_OC_IsEnabledFast
- CCMR2 OC4FE LL_TIM_OC_IsEnabledFast
- CCMR3 OC5FE LL_TIM_OC_IsEnabledFast
- CCMR3 OC6FE LL_TIM_OC_IsEnabledFast

LL_TIM_OC_EnablePreload

Function name

__STATIC_INLINE void LL_TIM_OC_EnablePreload (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Enable compare register (TIMx_CCRx) preload for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 OC1PE LL_TIM_OC_EnablePreload
- CCMR1 OC2PE LL_TIM_OC_EnablePreload
- CCMR2 OC3PE LL_TIM_OC_EnablePreload
- CCMR2 OC4PE LL_TIM_OC_EnablePreload
- CCMR3 OC5PE LL_TIM_OC_EnablePreload
- CCMR3 OC6PE LL_TIM_OC_EnablePreload

LL_TIM_OC_DisablePreload

Function name

__STATIC_INLINE void LL_TIM_OC_DisablePreload (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Disable compare register (TIMx_CCRx) preload for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 OC1PE LL_TIM_OC_DisablePreload
- CCMR1 OC2PE LL_TIM_OC_DisablePreload
- CCMR2 OC3PE LL_TIM_OC_DisablePreload
- CCMR2 OC4PE LL_TIM_OC_DisablePreload
- CCMR3 OC5PE LL_TIM_OC_DisablePreload
- CCMR3 OC6PE LL_TIM_OC_DisablePreload

LL_TIM_OC_IsEnabledPreload

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_IsEnabledPreload (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Indicates whether compare register (TIMx_CCRx) preload is enabled for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCMR1 OC1PE LL_TIM_OC_IsEnabledPreload
- CCMR1 OC2PE LL_TIM_OC_IsEnabledPreload
- CCMR2 OC3PE LL_TIM_OC_IsEnabledPreload
- CCMR2 OC4PE LL_TIM_OC_IsEnabledPreload
- CCMR3 OC5PE LL_TIM_OC_IsEnabledPreload
- CCMR3 OC6PE LL_TIM_OC_IsEnabledPreload

LL_TIM_OC_EnableClear

Function name

__STATIC_INLINE void LL_TIM_OC_EnableClear (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Enable clearing the output channel on an external event.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Notes

- This function can only be used in Output compare and PWM modes. It does not work in Forced mode.
- Macro IS_TIM_OCXREF_CLEAR_INSTANCE(TIMx) can be used to check whether or not a timer instance can clear the OCxREF signal on an external event.

Reference Manual to LL API cross reference:

- CCMR1 OC1CE LL_TIM_OC_EnableClear
- CCMR1 OC2CE LL_TIM_OC_EnableClear
- CCMR2 OC3CE LL_TIM_OC_EnableClear
- CCMR2 OC4CE LL_TIM_OC_EnableClear
- CCMR3 OC5CE LL_TIM_OC_EnableClear
- CCMR3 OC6CE LL_TIM_OC_EnableClear

LL_TIM_OC_DisableClear

Function name

__STATIC_INLINE void LL_TIM_OC_DisableClear (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Disable clearing the output channel on an external event.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Notes

- Macro IS_TIM_OCXREF_CLEAR_INSTANCE(TIMx) can be used to check whether or not a timer instance can clear the OCxREF signal on an external event.

Reference Manual to LL API cross reference:

- CCMR1 OC1CE LL_TIM_OC_DisableClear
- CCMR1 OC2CE LL_TIM_OC_DisableClear
- CCMR2 OC3CE LL_TIM_OC_DisableClear
- CCMR2 OC4CE LL_TIM_OC_DisableClear
- CCMR3 OC5CE LL_TIM_OC_DisableClear
- CCMR3 OC6CE LL_TIM_OC_DisableClear

LL_TIM_OC_IsEnabledClear

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_IsEnabledClear (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Indicates clearing the output channel on an external event is enabled for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **State:** of bit (1 or 0).

Notes

- This function enables clearing the output channel on an external event.
- This function can only be used in Output compare and PWM modes. It does not work in Forced mode.
- Macro IS_TIM_OCXREF_CLEAR_INSTANCE(TIMx) can be used to check whether or not a timer instance can clear the OCxREF signal on an external event.

Reference Manual to LL API cross reference:

- CCMR1 OC1CE LL_TIM_OC_IsEnabledClear
- CCMR1 OC2CE LL_TIM_OC_IsEnabledClear
- CCMR2 OC3CE LL_TIM_OC_IsEnabledClear
- CCMR2 OC4CE LL_TIM_OC_IsEnabledClear
- CCMR3 OC5CE LL_TIM_OC_IsEnabledClear
- CCMR3 OC6CE LL_TIM_OC_IsEnabledClear

LL_TIM_OC_SetDeadTime

Function name

__STATIC_INLINE void LL_TIM_OC_SetDeadTime (TIM_TypeDef * TIMx, uint32_t DeadTime)

Function description

Set the dead-time delay (delay inserted between the rising edge of the OCxREF signal and the rising edge of the Ocx and OCxN signals).

Parameters

- **TIMx:** Timer instance
- **DeadTime:** between Min_Data=0 and Max_Data=255

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not dead-time insertion feature is supported by a timer instance.
- Helper macro __LL_TIM_CALC_DEADTIME can be used to calculate the DeadTime parameter

Reference Manual to LL API cross reference:

- BDTR DTG LL_TIM_OC_SetDeadTime

LL_TIM_OC_SetCompareCH1

Function name

__STATIC_INLINE void LL_TIM_OC_SetCompareCH1 (TIM_TypeDef * TIMx, uint32_t CompareValue)

Function description

Set compare value for output channel 1 (TIMx_CCR1).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- In 32-bit timer implementations compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC1_INSTANCE(TIMx) can be used to check whether or not output channel 1 is supported by a timer instance.

Reference Manual to LL API cross reference:

- CCR1 CCR1 LL_TIM_OC_SetCompareCH1

LL_TIM_OC_SetCompareCH2

Function name

__STATIC_INLINE void LL_TIM_OC_SetCompareCH2 (TIM_TypeDef * TIMx, uint32_t CompareValue)

Function description

Set compare value for output channel 2 (TIMx_CCR2).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- In 32-bit timer implementations compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC2_INSTANCE(TIMx) can be used to check whether or not output channel 2 is supported by a timer instance.

Reference Manual to LL API cross reference:

- CCR2 CCR2 LL_TIM_OC_SetCompareCH2

LL_TIM_OC_SetCompareCH3

Function name

__STATIC_INLINE void LL_TIM_OC_SetCompareCH3 (TIM_TypeDef * TIMx, uint32_t CompareValue)

Function description

Set compare value for output channel 3 (TIMx_CCR3).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- In 32-bit timer implementations compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC3_INSTANCE(TIMx) can be used to check whether or not output channel is supported by a timer instance.

Reference Manual to LL API cross reference:

- CCR3 CCR3 LL_TIM_OC_SetCompareCH3

LL_TIM_OC_SetCompareCH4

Function name

__STATIC_INLINE void LL_TIM_OC_SetCompareCH4 (TIM_TypeDef * TIMx, uint32_t CompareValue)

Function description

Set compare value for output channel 4 (TIMx_CCR4).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- In 32-bit timer implementations compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC4_INSTANCE(TIMx) can be used to check whether or not output channel 4 is supported by a timer instance.

Reference Manual to LL API cross reference:

- CCR4 CCR4 LL_TIM_OC_SetCompareCH4

LL_TIM_OC_SetCompareCH5

Function name

__STATIC_INLINE void LL_TIM_OC_SetCompareCH5 (TIM_TypeDef * TIMx, uint32_t CompareValue)

Function description

Set compare value for output channel 5 (TIMx_CCR5).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- Macro IS_TIM_CC5_INSTANCE(TIMx) can be used to check whether or not output channel 5 is supported by a timer instance.

Reference Manual to LL API cross reference:

- CCR5 CCR5 LL_TIM_OC_SetCompareCH5

LL_TIM_OC_SetCompareCH6

Function name

__STATIC_INLINE void LL_TIM_OC_SetCompareCH6 (TIM_TypeDef * TIMx, uint32_t CompareValue)

Function description

Set compare value for output channel 6 (TIMx_CCR6).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- Macro IS_TIM_CC6_INSTANCE(TIMx) can be used to check whether or not output channel 6 is supported by a timer instance.

Reference Manual to LL API cross reference:

- CCR6 CCR6 LL_TIM_OC_SetCompareCH6

LL_TIM_OC_GetCompareCH1

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH1 (const TIM_TypeDef * TIMx)

Function description

Get compare value (TIMx_CCR1) set for output channel 1.

Parameters

- **TIMx:** Timer instance

Return values

- **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC1_INSTANCE(TIMx) can be used to check whether or not output channel 1 is supported by a timer instance.

Reference Manual to LL API cross reference:

- CCR1 CCR1 LL_TIM_OC_GetCompareCH1

LL_TIM_OC_GetCompareCH2

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH2 (const TIM_TypeDef * TIMx)

Function description

Get compare value (TIMx_CCR2) set for output channel 2.

Parameters

- **TIMx:** Timer instance

Return values

- **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC2_INSTANCE(TIMx) can be used to check whether or not output channel 2 is supported by a timer instance.

Reference Manual to LL API cross reference:

- CCR2 CCR2 LL_TIM_OC_GetCompareCH2

LL_TIM_OC_GetCompareCH3

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH3 (const TIM_TypeDef * TIMx)

Function description

Get compare value (TIMx_CCR3) set for output channel 3.

Parameters

- **TIMx:** Timer instance

Return values

- **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC3_INSTANCE(TIMx) can be used to check whether or not output channel 3 is supported by a timer instance.

Reference Manual to LL API cross reference:

- CCR3 CCR3 LL_TIM_OC_GetCompareCH3

LL_TIM_OC_GetCompareCH4

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH4 (const TIM_TypeDef * TIMx)

Function description

Get compare value (TIMx_CCR4) set for output channel 4.

Parameters

- **TIMx:** Timer instance

Return values

- **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC4_INSTANCE(TIMx) can be used to check whether or not output channel 4 is supported by a timer instance.

Reference Manual to LL API cross reference:

- CCR4 CCR4 LL_TIM_OC_GetCompareCH4

LL_TIM_OC_GetCompareCH5

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH5 (const TIM_TypeDef * TIMx)

Function description

Get compare value (TIMx_CCR5) set for output channel 5.

Parameters

- **TIMx:** Timer instance

Return values

- **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- Macro IS_TIM_CC5_INSTANCE(TIMx) can be used to check whether or not output channel 5 is supported by a timer instance.

Reference Manual to LL API cross reference:

- CCR5 CCR5 LL_TIM_OC_GetCompareCH5

LL_TIM_OC_GetCompareCH6

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH6 (const TIM_TypeDef * TIMx)

Function description

Get compare value (TIMx_CCR6) set for output channel 6.

Parameters

- **TIMx:** Timer instance

Return values

- **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- Macro IS_TIM_CC6_INSTANCE(TIMx) can be used to check whether or not output channel 6 is supported by a timer instance.

Reference Manual to LL API cross reference:

- CCR6 CCR6 LL_TIM_OC_GetCompareCH6

LL_TIM_SetCH5CombinedChannels

Function name

__STATIC_INLINE void LL_TIM_SetCH5CombinedChannels (TIM_TypeDef * TIMx, uint32_t GroupCH5)

Function description

Select on which reference signal the OC5REF is combined to.

Parameters

- **TIMx:** Timer instance
- **GroupCH5:** This parameter can be a combination of the following values:
  - LL_TIM_GROUPCH5_NONE
  - LL_TIM_GROUPCH5_OC1REFC
  - LL_TIM_GROUPCH5_OC2REFC
  - LL_TIM_GROUPCH5_OC3REFC

Return values

- **None:**

Notes

- Macro IS_TIM_COMBINED3PHASEPWM_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the combined 3-phase PWM mode.

Reference Manual to LL API cross reference:

- CCR5 GC5C3 LL_TIM_SetCH5CombinedChannels
- CCR5 GC5C2 LL_TIM_SetCH5CombinedChannels
- CCR5 GC5C1 LL_TIM_SetCH5CombinedChannels

LL_TIM_IC_Config

Function name

__STATIC_INLINE void LL_TIM_IC_Config (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Configuration)

Function description

Configure input channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_TIM_ACTIVEINPUT_DIRECTTI or LL_TIM_ACTIVEINPUT_INDIRECTTI or LL_TIM_ACTIVEINPUT_TRC
  - LL_TIM_ICPSC_DIV1 or ... or LL_TIM_ICPSC_DIV8
  - LL_TIM_IC_FILTER_FDIV1 or ... or LL_TIM_IC_FILTER_FDIV32_N8
  - LL_TIM_IC_POLARITY_RISING or LL_TIM_IC_POLARITY_FALLING or LL_TIM_IC_POLARITY_BOTHEDGE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 CC1S LL_TIM_IC_Config
- CCMR1 IC1PSC LL_TIM_IC_Config
- CCMR1 IC1F LL_TIM_IC_Config
- CCMR1 CC2S LL_TIM_IC_Config
- CCMR1 IC2PSC LL_TIM_IC_Config
- CCMR1 IC2F LL_TIM_IC_Config
- CCMR2 CC3S LL_TIM_IC_Config
- CCMR2 IC3PSC LL_TIM_IC_Config
- CCMR2 IC3F LL_TIM_IC_Config
- CCMR2 CC4S LL_TIM_IC_Config
- CCMR2 IC4PSC LL_TIM_IC_Config
- CCMR2 IC4F LL_TIM_IC_Config
- CCER CC1P LL_TIM_IC_Config
- CCER CC1NP LL_TIM_IC_Config
- CCER CC2P LL_TIM_IC_Config
- CCER CC2NP LL_TIM_IC_Config
- CCER CC3P LL_TIM_IC_Config
- CCER CC3NP LL_TIM_IC_Config
- CCER CC4P LL_TIM_IC_Config
- CCER CC4NP LL_TIM_IC_Config

LL_TIM_IC_SetActiveInput

Function name

__STATIC_INLINE void LL_TIM_IC_SetActiveInput (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICActiveInput)

Function description

Set the active input.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **ICActiveInput:** This parameter can be one of the following values:
  - LL_TIM_ACTIVEINPUT_DIRECTTI
  - LL_TIM_ACTIVEINPUT_INDIRECTTI
  - LL_TIM_ACTIVEINPUT_TRC

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 CC1S LL_TIM_IC_SetActiveInput
- CCMR1 CC2S LL_TIM_IC_SetActiveInput
- CCMR2 CC3S LL_TIM_IC_SetActiveInput
- CCMR2 CC4S LL_TIM_IC_SetActiveInput

LL_TIM_IC_GetActiveInput

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetActiveInput (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the current active input.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_ACTIVEINPUT_DIRECTTI
  - LL_TIM_ACTIVEINPUT_INDIRECTTI
  - LL_TIM_ACTIVEINPUT_TRC

Reference Manual to LL API cross reference:

- CCMR1 CC1S LL_TIM_IC_GetActiveInput
- CCMR1 CC2S LL_TIM_IC_GetActiveInput
- CCMR2 CC3S LL_TIM_IC_GetActiveInput
- CCMR2 CC4S LL_TIM_IC_GetActiveInput

LL_TIM_IC_SetPrescaler

Function name

__STATIC_INLINE void LL_TIM_IC_SetPrescaler (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICPrescaler)

Function description

Set the prescaler of input channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **ICPrescaler:** This parameter can be one of the following values:
  - LL_TIM_ICPSC_DIV1
  - LL_TIM_ICPSC_DIV2
  - LL_TIM_ICPSC_DIV4
  - LL_TIM_ICPSC_DIV8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 IC1PSC LL_TIM_IC_SetPrescaler
- CCMR1 IC2PSC LL_TIM_IC_SetPrescaler
- CCMR2 IC3PSC LL_TIM_IC_SetPrescaler
- CCMR2 IC4PSC LL_TIM_IC_SetPrescaler

LL_TIM_IC_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetPrescaler (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the current prescaler value acting on an input channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_ICPSC_DIV1
  - LL_TIM_ICPSC_DIV2
  - LL_TIM_ICPSC_DIV4
  - LL_TIM_ICPSC_DIV8

Reference Manual to LL API cross reference:

- CCMR1 IC1PSC LL_TIM_IC_GetPrescaler
- CCMR1 IC2PSC LL_TIM_IC_GetPrescaler
- CCMR2 IC3PSC LL_TIM_IC_GetPrescaler
- CCMR2 IC4PSC LL_TIM_IC_GetPrescaler

LL_TIM_IC_SetFilter

Function name

__STATIC_INLINE void LL_TIM_IC_SetFilter (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICFilter)

Function description

Set the input filter duration.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **ICFilter:** This parameter can be one of the following values:
  - LL_TIM_IC_FILTER_FDIV1
  - LL_TIM_IC_FILTER_FDIV1_N2
  - LL_TIM_IC_FILTER_FDIV1_N4
  - LL_TIM_IC_FILTER_FDIV1_N8
  - LL_TIM_IC_FILTER_FDIV2_N6
  - LL_TIM_IC_FILTER_FDIV2_N8
  - LL_TIM_IC_FILTER_FDIV4_N6
  - LL_TIM_IC_FILTER_FDIV4_N8
  - LL_TIM_IC_FILTER_FDIV8_N6
  - LL_TIM_IC_FILTER_FDIV8_N8
  - LL_TIM_IC_FILTER_FDIV16_N5
  - LL_TIM_IC_FILTER_FDIV16_N6
  - LL_TIM_IC_FILTER_FDIV16_N8
  - LL_TIM_IC_FILTER_FDIV32_N5
  - LL_TIM_IC_FILTER_FDIV32_N6
  - LL_TIM_IC_FILTER_FDIV32_N8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 IC1F LL_TIM_IC_SetFilter
- CCMR1 IC2F LL_TIM_IC_SetFilter
- CCMR2 IC3F LL_TIM_IC_SetFilter
- CCMR2 IC4F LL_TIM_IC_SetFilter

LL_TIM_IC_GetFilter

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetFilter (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the input filter duration.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_IC_FILTER_FDIV1
  - LL_TIM_IC_FILTER_FDIV1_N2
  - LL_TIM_IC_FILTER_FDIV1_N4
  - LL_TIM_IC_FILTER_FDIV1_N8
  - LL_TIM_IC_FILTER_FDIV2_N6
  - LL_TIM_IC_FILTER_FDIV2_N8
  - LL_TIM_IC_FILTER_FDIV4_N6
  - LL_TIM_IC_FILTER_FDIV4_N8
  - LL_TIM_IC_FILTER_FDIV8_N6
  - LL_TIM_IC_FILTER_FDIV8_N8
  - LL_TIM_IC_FILTER_FDIV16_N5
  - LL_TIM_IC_FILTER_FDIV16_N6
  - LL_TIM_IC_FILTER_FDIV16_N8
  - LL_TIM_IC_FILTER_FDIV32_N5
  - LL_TIM_IC_FILTER_FDIV32_N6
  - LL_TIM_IC_FILTER_FDIV32_N8

Reference Manual to LL API cross reference:

- CCMR1 IC1F LL_TIM_IC_GetFilter
- CCMR1 IC2F LL_TIM_IC_GetFilter
- CCMR2 IC3F LL_TIM_IC_GetFilter
- CCMR2 IC4F LL_TIM_IC_GetFilter

LL_TIM_IC_SetPolarity

Function name

__STATIC_INLINE void LL_TIM_IC_SetPolarity (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICPolarity)

Function description

Set the input channel polarity.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **ICPolarity:** This parameter can be one of the following values:
  - LL_TIM_IC_POLARITY_RISING
  - LL_TIM_IC_POLARITY_FALLING
  - LL_TIM_IC_POLARITY_BOTHEDGE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCER CC1P LL_TIM_IC_SetPolarity
- CCER CC1NP LL_TIM_IC_SetPolarity
- CCER CC2P LL_TIM_IC_SetPolarity
- CCER CC2NP LL_TIM_IC_SetPolarity
- CCER CC3P LL_TIM_IC_SetPolarity
- CCER CC3NP LL_TIM_IC_SetPolarity
- CCER CC4P LL_TIM_IC_SetPolarity
- CCER CC4NP LL_TIM_IC_SetPolarity

LL_TIM_IC_GetPolarity

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetPolarity (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the current input channel polarity.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_IC_POLARITY_RISING
  - LL_TIM_IC_POLARITY_FALLING
  - LL_TIM_IC_POLARITY_BOTHEDGE

Reference Manual to LL API cross reference:

- CCER CC1P LL_TIM_IC_GetPolarity
- CCER CC1NP LL_TIM_IC_GetPolarity
- CCER CC2P LL_TIM_IC_GetPolarity
- CCER CC2NP LL_TIM_IC_GetPolarity
- CCER CC3P LL_TIM_IC_GetPolarity
- CCER CC3NP LL_TIM_IC_GetPolarity
- CCER CC4P LL_TIM_IC_GetPolarity
- CCER CC4NP LL_TIM_IC_GetPolarity

LL_TIM_IC_EnableXORCombination

Function name

__STATIC_INLINE void LL_TIM_IC_EnableXORCombination (TIM_TypeDef * TIMx)

Function description

Connect the TIMx_CH1, CH2 and CH3 pins to the TI1 input (XOR combination).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_XOR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an XOR input.

Reference Manual to LL API cross reference:

- CR2 TI1S LL_TIM_IC_EnableXORCombination

LL_TIM_IC_DisableXORCombination

Function name

__STATIC_INLINE void LL_TIM_IC_DisableXORCombination (TIM_TypeDef * TIMx)

Function description

Disconnect the TIMx_CH1, CH2 and CH3 pins from the TI1 input.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_XOR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an XOR input.

Reference Manual to LL API cross reference:

- CR2 TI1S LL_TIM_IC_DisableXORCombination

LL_TIM_IC_IsEnabledXORCombination

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_IsEnabledXORCombination (TIM_TypeDef * TIMx)

Function description

Indicates whether the TIMx_CH1, CH2 and CH3 pins are connectected to the TI1 input.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_XOR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an XOR input.

Reference Manual to LL API cross reference:

- CR2 TI1S LL_TIM_IC_IsEnabledXORCombination

LL_TIM_IC_GetCaptureCH1

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH1 (const TIM_TypeDef * TIMx)

Function description

Get captured value for input channel 1.

Parameters

- **TIMx:** Timer instance

Return values

- **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned captured value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC1_INSTANCE(TIMx) can be used to check whether or not input channel 1 is supported by a timer instance.

Reference Manual to LL API cross reference:

- CCR1 CCR1 LL_TIM_IC_GetCaptureCH1

LL_TIM_IC_GetCaptureCH2

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH2 (const TIM_TypeDef * TIMx)

Function description

Get captured value for input channel 2.

Parameters

- **TIMx:** Timer instance

Return values

- **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned captured value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC2_INSTANCE(TIMx) can be used to check whether or not input channel 2 is supported by a timer instance.

Reference Manual to LL API cross reference:

- CCR2 CCR2 LL_TIM_IC_GetCaptureCH2

LL_TIM_IC_GetCaptureCH3

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH3 (const TIM_TypeDef * TIMx)

Function description

Get captured value for input channel 3.

Parameters

- **TIMx:** Timer instance

Return values

- **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned captured value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC3_INSTANCE(TIMx) can be used to check whether or not input channel 3 is supported by a timer instance.

Reference Manual to LL API cross reference:

- CCR3 CCR3 LL_TIM_IC_GetCaptureCH3

LL_TIM_IC_GetCaptureCH4

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH4 (const TIM_TypeDef * TIMx)

Function description

Get captured value for input channel 4.

Parameters

- **TIMx:** Timer instance

Return values

- **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned captured value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC4_INSTANCE(TIMx) can be used to check whether or not input channel 4 is supported by a timer instance.

Reference Manual to LL API cross reference:

- CCR4 CCR4 LL_TIM_IC_GetCaptureCH4

LL_TIM_EnableExternalClock

Function name

__STATIC_INLINE void LL_TIM_EnableExternalClock (TIM_TypeDef * TIMx)

Function description

Enable external clock mode 2.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- When external clock mode 2 is enabled the counter is clocked by any active edge on the ETRF signal.
- Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.

Reference Manual to LL API cross reference:

- SMCR ECE LL_TIM_EnableExternalClock

LL_TIM_DisableExternalClock

Function name

__STATIC_INLINE void LL_TIM_DisableExternalClock (TIM_TypeDef * TIMx)

Function description

Disable external clock mode 2.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.

Reference Manual to LL API cross reference:

- SMCR ECE LL_TIM_DisableExternalClock

LL_TIM_IsEnabledExternalClock

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledExternalClock (const TIM_TypeDef * TIMx)

Function description

Indicate whether external clock mode 2 is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.

Reference Manual to LL API cross reference:

- SMCR ECE LL_TIM_IsEnabledExternalClock

LL_TIM_SetClockSource

Function name

__STATIC_INLINE void LL_TIM_SetClockSource (TIM_TypeDef * TIMx, uint32_t ClockSource)

Function description

Set the clock source of the counter clock.

Parameters

- **TIMx:** Timer instance
- **ClockSource:** This parameter can be one of the following values:
  - LL_TIM_CLOCKSOURCE_INTERNAL
  - LL_TIM_CLOCKSOURCE_EXT_MODE1
  - LL_TIM_CLOCKSOURCE_EXT_MODE2

Return values

- **None:**

Notes

- when selected clock source is external clock mode 1, the timer input the external clock is applied is selected by calling the LL_TIM_SetTriggerInput() function. This timer input must be configured by calling the LL_TIM_IC_Config() function.
- Macro IS_TIM_CLOCKSOURCE_ETRMODE1_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode1.
- Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.

Reference Manual to LL API cross reference:

- SMCR SMS LL_TIM_SetClockSource
- SMCR ECE LL_TIM_SetClockSource

LL_TIM_SetEncoderMode

Function name

__STATIC_INLINE void LL_TIM_SetEncoderMode (TIM_TypeDef * TIMx, uint32_t EncoderMode)

Function description

Set the encoder interface mode.

Parameters

- **TIMx:** Timer instance
- **EncoderMode:** This parameter can be one of the following values:
  - LL_TIM_ENCODERMODE_X2_TI1
  - LL_TIM_ENCODERMODE_X2_TI2
  - LL_TIM_ENCODERMODE_X4_TI12

Return values

- **None:**

Notes

- Macro IS_TIM_ENCODER_INTERFACE_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the encoder mode.

Reference Manual to LL API cross reference:

- SMCR SMS LL_TIM_SetEncoderMode

LL_TIM_SetTriggerOutput

Function name

__STATIC_INLINE void LL_TIM_SetTriggerOutput (TIM_TypeDef * TIMx, uint32_t TimerSynchronization)

Function description

Set the trigger output (TRGO) used for timer synchronization .

Parameters

- **TIMx:** Timer instance
- **TimerSynchronization:** This parameter can be one of the following values:
  - LL_TIM_TRGO_RESET
  - LL_TIM_TRGO_ENABLE
  - LL_TIM_TRGO_UPDATE
  - LL_TIM_TRGO_CC1IF
  - LL_TIM_TRGO_OC1REF
  - LL_TIM_TRGO_OC2REF
  - LL_TIM_TRGO_OC3REF
  - LL_TIM_TRGO_OC4REF

Return values

- **None:**

Notes

- Macro IS_TIM_MASTER_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a master timer.

Reference Manual to LL API cross reference:

- CR2 MMS LL_TIM_SetTriggerOutput

LL_TIM_SetTriggerOutput2

Function name

__STATIC_INLINE void LL_TIM_SetTriggerOutput2 (TIM_TypeDef * TIMx, uint32_t ADCSynchronization)

Function description

Set the trigger output 2 (TRGO2) used for ADC synchronization .

Parameters

- **TIMx:** Timer Instance
- **ADCSynchronization:** This parameter can be one of the following values:
  - LL_TIM_TRGO2_RESET
  - LL_TIM_TRGO2_ENABLE
  - LL_TIM_TRGO2_UPDATE
  - LL_TIM_TRGO2_CC1F
  - LL_TIM_TRGO2_OC1
  - LL_TIM_TRGO2_OC2
  - LL_TIM_TRGO2_OC3
  - LL_TIM_TRGO2_OC4
  - LL_TIM_TRGO2_OC5
  - LL_TIM_TRGO2_OC6
  - LL_TIM_TRGO2_OC4_RISINGFALLING
  - LL_TIM_TRGO2_OC6_RISINGFALLING
  - LL_TIM_TRGO2_OC4_RISING_OC6_RISING
  - LL_TIM_TRGO2_OC4_RISING_OC6_FALLING
  - LL_TIM_TRGO2_OC5_RISING_OC6_RISING
  - LL_TIM_TRGO2_OC5_RISING_OC6_FALLING

Return values

- **None:**

Notes

- Macro IS_TIM_TRGO2_INSTANCE(TIMx) can be used to check whether or not a timer instance can be used for ADC synchronization.

Reference Manual to LL API cross reference:

- CR2 MMS2 LL_TIM_SetTriggerOutput2

LL_TIM_SetSlaveMode

Function name

__STATIC_INLINE void LL_TIM_SetSlaveMode (TIM_TypeDef * TIMx, uint32_t SlaveMode)

Function description

Set the synchronization mode of a slave timer.

Parameters

- **TIMx:** Timer instance
- **SlaveMode:** This parameter can be one of the following values:
  - LL_TIM_SLAVEMODE_DISABLED
  - LL_TIM_SLAVEMODE_RESET
  - LL_TIM_SLAVEMODE_GATED
  - LL_TIM_SLAVEMODE_TRIGGER
  - LL_TIM_SLAVEMODE_COMBINED_RESETTRIGGER

Return values

- **None:**

Notes

- Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:

- SMCR SMS LL_TIM_SetSlaveMode

LL_TIM_SetTriggerInput

Function name

__STATIC_INLINE void LL_TIM_SetTriggerInput (TIM_TypeDef * TIMx, uint32_t TriggerInput)

Function description

Set the selects the trigger input to be used to synchronize the counter.

Parameters

- **TIMx:** Timer instance
- **TriggerInput:** This parameter can be one of the following values:
  - LL_TIM_TS_ITR0
  - LL_TIM_TS_ITR1
  - LL_TIM_TS_ITR2
  - LL_TIM_TS_ITR3
  - LL_TIM_TS_ITR4
  - LL_TIM_TS_ITR5
  - LL_TIM_TS_ITR6
  - LL_TIM_TS_ITR7
  - LL_TIM_TS_ITR8 (*)
  - LL_TIM_TS_ITR9 (*)
  - LL_TIM_TS_ITR10 (*)
  - LL_TIM_TS_ITR11 (*)
  - LL_TIM_TS_ITR12 (*)
  - LL_TIM_TS_ITR13 (*)
  - LL_TIM_TS_TI1F_ED
  - LL_TIM_TS_TI1FP1
  - LL_TIM_TS_TI2FP2
  - LL_TIM_TS_ETRF

  (*) Value not defined in all devices.

Return values

- **None:**

Notes

- Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:

- SMCR TS LL_TIM_SetTriggerInput

LL_TIM_EnableMasterSlaveMode

Function name

__STATIC_INLINE void LL_TIM_EnableMasterSlaveMode (TIM_TypeDef * TIMx)

Function description

Enable the Master/Slave mode.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:

- SMCR MSM LL_TIM_EnableMasterSlaveMode

LL_TIM_DisableMasterSlaveMode

Function name

__STATIC_INLINE void LL_TIM_DisableMasterSlaveMode (TIM_TypeDef * TIMx)

Function description

Disable the Master/Slave mode.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:

- SMCR MSM LL_TIM_DisableMasterSlaveMode

LL_TIM_IsEnabledMasterSlaveMode

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledMasterSlaveMode (const TIM_TypeDef * TIMx)

Function description

Indicates whether the Master/Slave mode is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:

- SMCR MSM LL_TIM_IsEnabledMasterSlaveMode

LL_TIM_ConfigETR

Function name

__STATIC_INLINE void LL_TIM_ConfigETR (TIM_TypeDef * TIMx, uint32_t ETRPolarity, uint32_t ETRPrescaler, uint32_t ETRFilter)

Function description

Configure the external trigger (ETR) input.

Parameters

- **TIMx:** Timer instance
- **ETRPolarity:** This parameter can be one of the following values:
  - LL_TIM_ETR_POLARITY_NONINVERTED
  - LL_TIM_ETR_POLARITY_INVERTED
- **ETRPrescaler:** This parameter can be one of the following values:
  - LL_TIM_ETR_PRESCALER_DIV1
  - LL_TIM_ETR_PRESCALER_DIV2
  - LL_TIM_ETR_PRESCALER_DIV4
  - LL_TIM_ETR_PRESCALER_DIV8
- **ETRFilter:** This parameter can be one of the following values:
  - LL_TIM_ETR_FILTER_FDIV1
  - LL_TIM_ETR_FILTER_FDIV1_N2
  - LL_TIM_ETR_FILTER_FDIV1_N4
  - LL_TIM_ETR_FILTER_FDIV1_N8
  - LL_TIM_ETR_FILTER_FDIV2_N6
  - LL_TIM_ETR_FILTER_FDIV2_N8
  - LL_TIM_ETR_FILTER_FDIV4_N6
  - LL_TIM_ETR_FILTER_FDIV4_N8
  - LL_TIM_ETR_FILTER_FDIV8_N6
  - LL_TIM_ETR_FILTER_FDIV8_N8
  - LL_TIM_ETR_FILTER_FDIV16_N5
  - LL_TIM_ETR_FILTER_FDIV16_N6
  - LL_TIM_ETR_FILTER_FDIV16_N8
  - LL_TIM_ETR_FILTER_FDIV32_N5
  - LL_TIM_ETR_FILTER_FDIV32_N6
  - LL_TIM_ETR_FILTER_FDIV32_N8

Return values

- **None:**

Notes

- Macro IS_TIM_ETR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an external trigger input.

Reference Manual to LL API cross reference:

- SMCR ETP LL_TIM_ConfigETR
- SMCR ETPS LL_TIM_ConfigETR
- SMCR ETF LL_TIM_ConfigETR

LL_TIM_SetETRSource

Function name

__STATIC_INLINE void LL_TIM_SetETRSource (TIM_TypeDef * TIMx, uint32_t ETRSource)

Function description

Select the external trigger (ETR) input source.

Parameters

- **TIMx:** Timer instance

- **ETRSource:** This parameter can be one of the following values: For TIM1, the parameter is one of the following values:
  - LL_TIM_TIM1_ETRSOURCE_GPIO: TIM1_ETR is connected to GPIO
  - LL_TIM_TIM1_ETRSOURCE_COMP1: TIM1_ETR is connected to COMP1 output
  - LL_TIM_TIM1_ETRSOURCE_COMP2: TIM1_ETR is connected to COMP2 output
  - LL_TIM_TIM1_ETRSOURCE_ADC1_AWD1: TIM1_ETR is connected to ADC1 AWD1
  - LL_TIM_TIM1_ETRSOURCE_ADC1_AWD2: TIM1_ETR is connected to ADC1 AWD2
  - LL_TIM_TIM1_ETRSOURCE_ADC1_AWD3: TIM1_ETR is connected to ADC1 AWD3
  - LL_TIM_TIM1_ETRSOURCE_ADC3_AWD1: TIM1_ETR is connected to ADC3 AWD1
  - LL_TIM_TIM1_ETRSOURCE_ADC3_AWD2: TIM1_ETR is connected to ADC3 AWD2
  - LL_TIM_TIM1_ETRSOURCE_ADC3_AWD3: TIM1_ETR is connected to ADC3 AWD3

  For TIM2, the parameter is one of the following values:
  - LL_TIM_TIM2_ETRSOURCE_GPIO: TIM2_ETR is connected to GPIO
  - LL_TIM_TIM2_ETRSOURCE_COMP1: TIM2_ETR is connected to COMP1 output
  - LL_TIM_TIM2_ETRSOURCE_COMP2: TIM2_ETR is connected to COMP2 output
  - LL_TIM_TIM2_ETRSOURCE_LSE: TIM2_ETR is connected to LSE
  - LL_TIM_TIM2_ETRSOURCE_SAI1_FSA: TIM2_ETR is connected to SAI1 FS_A
  - LL_TIM_TIM2_ETRSOURCE_SAI1_FSB: TIM2_ETR is connected to SAI1 FS_B

  For TIM3, the parameter is one of the following values:
  - LL_TIM_TIM3_ETRSOURCE_GPIO: TIM3_ETR is connected to GPIO
  - LL_TIM_TIM3_ETRSOURCE_COMP1: TIM3_ETR is connected to COMP1 output

  For TIM5, the parameter is one of the following values:
  - LL_TIM_TIM5_ETRSOURCE_GPIO: TIM5_ETR is connected to GPIO
  - LL_TIM_TIM5_ETRSOURCE_SAI2_FSA: TIM5_ETR is connected to SAI2 FS_A (*)
  - LL_TIM_TIM5_ETRSOURCE_SAI2_FSB: TIM5_ETR is connected to SAI2 FS_B (*)
  - LL_TIM_TIM5_ETRSOURCE_SAI4_FSA: TIM5_ETR is connected to SAI2 FS_A (*)
  - LL_TIM_TIM5_ETRSOURCE_SAI4_FSB: TIM5_ETR is connected to SAI2 FS_B (*)

  For TIM8, the parameter is one of the following values:
  - LL_TIM_TIM8_ETRSOURCE_GPIO: TIM8_ETR is connected to GPIO
  - LL_TIM_TIM8_ETRSOURCE_COMP1: TIM8_ETR is connected to COMP1 output
  - LL_TIM_TIM8_ETRSOURCE_COMP2: TIM8_ETR is connected to COMP2 output
  - LL_TIM_TIM8_ETRSOURCE_ADC2_AWD1: TIM8_ETR is connected to ADC2 AWD1
  - LL_TIM_TIM8_ETRSOURCE_ADC2_AWD2: TIM8_ETR is connected to ADC2 AWD2
  - LL_TIM_TIM8_ETRSOURCE_ADC2_AWD3: TIM8_ETR is connected to ADC2 AWD3
  - LL_TIM_TIM8_ETRSOURCE_ADC3_AWD1: TIM8_ETR is connected to ADC3 AWD1
  - LL_TIM_TIM8_ETRSOURCE_ADC3_AWD2: TIM8_ETR is connected to ADC3 AWD2
  - LL_TIM_TIM8_ETRSOURCE_ADC3_AWD3: TIM8_ETR is connected to ADC3 AWD3

  For TIM23, the parameter is one of the following values: (*)
  - LL_TIM_TIM23_ETRSOURCE_GPIO TIM23_ETR is connected to GPIO
  - LL_TIM_TIM23_ETRSOURCE_COMP1 TIM23_ETR is connected to COMP1 output
  - LL_TIM_TIM23_ETRSOURCE_COMP2 TIM23_ETR is connected to COMP2 output

  For TIM24, the parameter is one of the following values: (*)
  - LL_TIM_TIM24_ETRSOURCE_GPIO TIM24_ETR is connected to GPIO
  - LL_TIM_TIM24_ETRSOURCE_SAI4_FSA TIM24_ETR is connected to SAI4 FS_A
  - LL_TIM_TIM24_ETRSOURCE_SAI4_FSB TIM24_ETR is connected to SAI4 FS_B
  - LL_TIM_TIM24_ETRSOURCE_SAI1_FSA TIM24_ETR is connected to SAI1 FS_A
  - LL_TIM_TIM24_ETRSOURCE_SAI1_FSB TIM24_ETR is connected to SAI1 FS_B

  (*) Value not defined in all devices.

Return values

- **None:**

Notes

- Macro IS_TIM_ETRSEL_INSTANCE(TIMx) can be used to check whether or not a timer instance supports ETR source selection.

Reference Manual to LL API cross reference:

- AF1 ETRSEL LL_TIM_SetETRSource

LL_TIM_EnableBRK

Function name

__STATIC_INLINE void LL_TIM_EnableBRK (TIM_TypeDef * TIMx)

Function description

Enable the break function.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR BKE LL_TIM_EnableBRK

LL_TIM_DisableBRK

Function name

__STATIC_INLINE void LL_TIM_DisableBRK (TIM_TypeDef * TIMx)

Function description

Disable the break function.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR BKE LL_TIM_DisableBRK

LL_TIM_ConfigBRK

Function name

__STATIC_INLINE void LL_TIM_ConfigBRK (TIM_TypeDef * TIMx, uint32_t BreakPolarity, uint32_t BreakFilter)

Function description

Configure the break input.

Parameters

- **TIMx:** Timer instance
- **BreakPolarity:** This parameter can be one of the following values:
  - LL_TIM_BREAK_POLARITY_LOW
  - LL_TIM_BREAK_POLARITY_HIGH
- **BreakFilter:** This parameter can be one of the following values:
  - LL_TIM_BREAK_FILTER_FDIV1
  - LL_TIM_BREAK_FILTER_FDIV1_N2
  - LL_TIM_BREAK_FILTER_FDIV1_N4
  - LL_TIM_BREAK_FILTER_FDIV1_N8
  - LL_TIM_BREAK_FILTER_FDIV2_N6
  - LL_TIM_BREAK_FILTER_FDIV2_N8
  - LL_TIM_BREAK_FILTER_FDIV4_N6
  - LL_TIM_BREAK_FILTER_FDIV4_N8
  - LL_TIM_BREAK_FILTER_FDIV8_N6
  - LL_TIM_BREAK_FILTER_FDIV8_N8
  - LL_TIM_BREAK_FILTER_FDIV16_N5
  - LL_TIM_BREAK_FILTER_FDIV16_N6
  - LL_TIM_BREAK_FILTER_FDIV16_N8
  - LL_TIM_BREAK_FILTER_FDIV32_N5
  - LL_TIM_BREAK_FILTER_FDIV32_N6
  - LL_TIM_BREAK_FILTER_FDIV32_N8

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR BKP LL_TIM_ConfigBRK
- BDTR BKF LL_TIM_ConfigBRK

LL_TIM_EnableBRK2

Function name

__STATIC_INLINE void LL_TIM_EnableBRK2 (TIM_TypeDef * TIMx)

Function description

Enable the break 2 function.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BKIN2_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a second break input.

Reference Manual to LL API cross reference:

- BDTR BK2E LL_TIM_EnableBRK2

LL_TIM_DisableBRK2

Function name

__STATIC_INLINE void LL_TIM_DisableBRK2 (TIM_TypeDef * TIMx)

Function description

Disable the break 2 function.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BKIN2_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a second break input.

Reference Manual to LL API cross reference:

- BDTR BK2E LL_TIM_DisableBRK2

LL_TIM_ConfigBRK2

Function name

__STATIC_INLINE void LL_TIM_ConfigBRK2 (TIM_TypeDef * TIMx, uint32_t Break2Polarity, uint32_t Break2Filter)

Function description

Configure the break 2 input.

Parameters

- **TIMx:** Timer instance
- **Break2Polarity:** This parameter can be one of the following values:
  - LL_TIM_BREAK2_POLARITY_LOW
  - LL_TIM_BREAK2_POLARITY_HIGH
- **Break2Filter:** This parameter can be one of the following values:
  - LL_TIM_BREAK2_FILTER_FDIV1
  - LL_TIM_BREAK2_FILTER_FDIV1_N2
  - LL_TIM_BREAK2_FILTER_FDIV1_N4
  - LL_TIM_BREAK2_FILTER_FDIV1_N8
  - LL_TIM_BREAK2_FILTER_FDIV2_N6
  - LL_TIM_BREAK2_FILTER_FDIV2_N8
  - LL_TIM_BREAK2_FILTER_FDIV4_N6
  - LL_TIM_BREAK2_FILTER_FDIV4_N8
  - LL_TIM_BREAK2_FILTER_FDIV8_N6
  - LL_TIM_BREAK2_FILTER_FDIV8_N8
  - LL_TIM_BREAK2_FILTER_FDIV16_N5
  - LL_TIM_BREAK2_FILTER_FDIV16_N6
  - LL_TIM_BREAK2_FILTER_FDIV16_N8
  - LL_TIM_BREAK2_FILTER_FDIV32_N5
  - LL_TIM_BREAK2_FILTER_FDIV32_N6
  - LL_TIM_BREAK2_FILTER_FDIV32_N8

Return values

- **None:**

Notes

- Macro IS_TIM_BKIN2_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a second break input.

Reference Manual to LL API cross reference:

- BDTR BK2P LL_TIM_ConfigBRK2
- BDTR BK2F LL_TIM_ConfigBRK2

LL_TIM_SetOffStates

Function name

__STATIC_INLINE void LL_TIM_SetOffStates (TIM_TypeDef * TIMx, uint32_t OffStateIdle, uint32_t OffStateRun)

Function description

Select the outputs off state (enabled v.s.

Parameters

- **TIMx:** Timer instance
- **OffStateIdle:** This parameter can be one of the following values:
  - LL_TIM_OSSI_DISABLE
  - LL_TIM_OSSI_ENABLE
- **OffStateRun:** This parameter can be one of the following values:
  - LL_TIM_OSSR_DISABLE
  - LL_TIM_OSSR_ENABLE

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR OSSI LL_TIM_SetOffStates
- BDTR OSSR LL_TIM_SetOffStates

LL_TIM_EnableAutomaticOutput

Function name

__STATIC_INLINE void LL_TIM_EnableAutomaticOutput (TIM_TypeDef * TIMx)

Function description

Enable automatic output (MOE can be set by software or automatically when a break input is active).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR AOE LL_TIM_EnableAutomaticOutput

LL_TIM_DisableAutomaticOutput

Function name

__STATIC_INLINE void LL_TIM_DisableAutomaticOutput (TIM_TypeDef * TIMx)

Function description

Disable automatic output (MOE can be set only by software).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR AOE LL_TIM_DisableAutomaticOutput

LL_TIM_IsEnabledAutomaticOutput

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledAutomaticOutput (const TIM_TypeDef * TIMx)

Function description

Indicate whether automatic output is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR AOE LL_TIM_IsEnabledAutomaticOutput

LL_TIM_EnableAllOutputs

Function name

__STATIC_INLINE void LL_TIM_EnableAllOutputs (TIM_TypeDef * TIMx)

Function description

Enable the outputs (set the MOE bit in TIMx_BDTR register).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- The MOE bit in TIMx_BDTR register allows to enable /disable the outputs by software and is reset in case of break or break2 event
- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR MOE LL_TIM_EnableAllOutputs

LL_TIM_DisableAllOutputs

Function name

__STATIC_INLINE void LL_TIM_DisableAllOutputs (TIM_TypeDef * TIMx)

Function description

Disable the outputs (reset the MOE bit in TIMx_BDTR register).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- The MOE bit in TIMx_BDTR register allows to enable /disable the outputs by software and is reset in case of break or break2 event.
- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR MOE LL_TIM_DisableAllOutputs

LL_TIM_IsEnabledAllOutputs

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledAllOutputs (const TIM_TypeDef * TIMx)

Function description

Indicates whether outputs are enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR MOE LL_TIM_IsEnabledAllOutputs

LL_TIM_EnableBreakInputSource

Function name

__STATIC_INLINE void LL_TIM_EnableBreakInputSource (TIM_TypeDef * TIMx, uint32_t BreakInput, uint32_t Source)

Function description

Enable the signals connected to the designated timer break input.

Parameters

- **TIMx:** Timer instance
- **BreakInput:** This parameter can be one of the following values:
  - LL_TIM_BREAK_INPUT_BKIN
  - LL_TIM_BREAK_INPUT_BKIN2
- **Source:** This parameter can be one of the following values:
  - LL_TIM_BKIN_SOURCE_BKIN
  - LL_TIM_BKIN_SOURCE_BKCOMP1
  - LL_TIM_BKIN_SOURCE_BKCOMP2
  - LL_TIM_BKIN_SOURCE_DF1BK

Return values

- **None:**

Notes

- Macro IS_TIM_BREAKSOURCE_INSTANCE(TIMx) can be used to check whether or not a timer instance allows for break input selection.

Reference Manual to LL API cross reference:

- AF1 BKINE LL_TIM_EnableBreakInputSource
- AF1 BKCMP1E LL_TIM_EnableBreakInputSource
- AF1 BKCMP2E LL_TIM_EnableBreakInputSource
- AF1 BKDF1BK0E LL_TIM_EnableBreakInputSource
- AF2 BK2INE LL_TIM_EnableBreakInputSource
- AF2 BK2CMP1E LL_TIM_EnableBreakInputSource
- AF2 BK2CMP2E LL_TIM_EnableBreakInputSource
- AF2 BK2DF1BK1E LL_TIM_EnableBreakInputSource

LL_TIM_DisableBreakInputSource

Function name

__STATIC_INLINE void LL_TIM_DisableBreakInputSource (TIM_TypeDef * TIMx, uint32_t BreakInput, uint32_t Source)

Function description

Disable the signals connected to the designated timer break input.

Parameters

- **TIMx:** Timer instance
- **BreakInput:** This parameter can be one of the following values:
  - LL_TIM_BREAK_INPUT_BKIN
  - LL_TIM_BREAK_INPUT_BKIN2
- **Source:** This parameter can be one of the following values:
  - LL_TIM_BKIN_SOURCE_BKIN
  - LL_TIM_BKIN_SOURCE_BKCOMP1
  - LL_TIM_BKIN_SOURCE_BKCOMP2
  - LL_TIM_BKIN_SOURCE_DF1BK

Return values

- **None:**

Notes

- Macro IS_TIM_BREAKSOURCE_INSTANCE(TIMx) can be used to check whether or not a timer instance allows for break input selection.

Reference Manual to LL API cross reference:

- AF1 BKINE LL_TIM_DisableBreakInputSource
- AF1 BKCMP1E LL_TIM_DisableBreakInputSource
- AF1 BKCMP2E LL_TIM_DisableBreakInputSource
- AF1 BKDF1BK0E LL_TIM_DisableBreakInputSource
- AF2 BK2INE LL_TIM_DisableBreakInputSource
- AF2 BK2CMP1E LL_TIM_DisableBreakInputSource
- AF2 BK2CMP2E LL_TIM_DisableBreakInputSource
- AF2 BK2DF1BK1E LL_TIM_DisableBreakInputSource

LL_TIM_SetBreakInputSourcePolarity

Function name

__STATIC_INLINE void LL_TIM_SetBreakInputSourcePolarity (TIM_TypeDef * TIMx, uint32_t BreakInput, uint32_t Source, uint32_t Polarity)

Function description

Set the polarity of the break signal for the timer break input.

Parameters

- **TIMx:** Timer instance
- **BreakInput:** This parameter can be one of the following values:
  - LL_TIM_BREAK_INPUT_BKIN
  - LL_TIM_BREAK_INPUT_BKIN2
- **Source:** This parameter can be one of the following values:
  - LL_TIM_BKIN_SOURCE_BKIN
  - LL_TIM_BKIN_SOURCE_BKCOMP1
  - LL_TIM_BKIN_SOURCE_BKCOMP2
- **Polarity:** This parameter can be one of the following values:
  - LL_TIM_BKIN_POLARITY_LOW
  - LL_TIM_BKIN_POLARITY_HIGH

Return values

- **None:**

Notes

- Macro IS_TIM_BREAKSOURCE_INSTANCE(TIMx) can be used to check whether or not a timer instance allows for break input selection.

Reference Manual to LL API cross reference:

- AF1 BKINP LL_TIM_SetBreakInputSourcePolarity
- AF1 BKCMP1P LL_TIM_SetBreakInputSourcePolarity
- AF1 BKCMP2P LL_TIM_SetBreakInputSourcePolarity
- AF2 BK2INP LL_TIM_SetBreakInputSourcePolarity
- AF2 BK2CMP1P LL_TIM_SetBreakInputSourcePolarity
- AF2 BK2CMP2P LL_TIM_SetBreakInputSourcePolarity

LL_TIM_ConfigDMABurst

Function name

__STATIC_INLINE void LL_TIM_ConfigDMABurst (TIM_TypeDef * TIMx, uint32_t DMABurstBaseAddress, uint32_t DMABurstLength)

Function description

Configures the timer DMA burst feature.

Parameters

- **TIMx:** Timer instance
- **DMABurstBaseAddress:** This parameter can be one of the following values:
  - LL_TIM_DMABURST_BASEADDR_CR1
  - LL_TIM_DMABURST_BASEADDR_CR2
  - LL_TIM_DMABURST_BASEADDR_SMCR
  - LL_TIM_DMABURST_BASEADDR_DIER
  - LL_TIM_DMABURST_BASEADDR_SR
  - LL_TIM_DMABURST_BASEADDR_EGR
  - LL_TIM_DMABURST_BASEADDR_CCMR1
  - LL_TIM_DMABURST_BASEADDR_CCMR2
  - LL_TIM_DMABURST_BASEADDR_CCER
  - LL_TIM_DMABURST_BASEADDR_CNT
  - LL_TIM_DMABURST_BASEADDR_PSC
  - LL_TIM_DMABURST_BASEADDR_ARR
  - LL_TIM_DMABURST_BASEADDR_RCR
  - LL_TIM_DMABURST_BASEADDR_CCR1
  - LL_TIM_DMABURST_BASEADDR_CCR2
  - LL_TIM_DMABURST_BASEADDR_CCR3
  - LL_TIM_DMABURST_BASEADDR_CCR4
  - LL_TIM_DMABURST_BASEADDR_BDTR
  - LL_TIM_DMABURST_BASEADDR_CCMR3
  - LL_TIM_DMABURST_BASEADDR_CCR5
  - LL_TIM_DMABURST_BASEADDR_CCR6
  - LL_TIM_DMABURST_BASEADDR_AF1
  - LL_TIM_DMABURST_BASEADDR_AF2
  - LL_TIM_DMABURST_BASEADDR_TISEL
- **DMABurstLength:** This parameter can be one of the following values:
  - LL_TIM_DMABURST_LENGTH_1TRANSFER
  - LL_TIM_DMABURST_LENGTH_2TRANSFERS
  - LL_TIM_DMABURST_LENGTH_3TRANSFERS
  - LL_TIM_DMABURST_LENGTH_4TRANSFERS
  - LL_TIM_DMABURST_LENGTH_5TRANSFERS
  - LL_TIM_DMABURST_LENGTH_6TRANSFERS
  - LL_TIM_DMABURST_LENGTH_7TRANSFERS
  - LL_TIM_DMABURST_LENGTH_8TRANSFERS
  - LL_TIM_DMABURST_LENGTH_9TRANSFERS
  - LL_TIM_DMABURST_LENGTH_10TRANSFERS
  - LL_TIM_DMABURST_LENGTH_11TRANSFERS
  - LL_TIM_DMABURST_LENGTH_12TRANSFERS
  - LL_TIM_DMABURST_LENGTH_13TRANSFERS
  - LL_TIM_DMABURST_LENGTH_14TRANSFERS
  - LL_TIM_DMABURST_LENGTH_15TRANSFERS
  - LL_TIM_DMABURST_LENGTH_16TRANSFERS
  - LL_TIM_DMABURST_LENGTH_17TRANSFERS
  - LL_TIM_DMABURST_LENGTH_18TRANSFERS

Return values

- **None:**

Notes

- Macro IS_TIM_DMABURST_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the DMA burst mode.

Reference Manual to LL API cross reference:

- DCR DBL LL_TIM_ConfigDMABurst
- DCR DBA LL_TIM_ConfigDMABurst

LL_TIM_SetRemap

Function name

__STATIC_INLINE void LL_TIM_SetRemap (TIM_TypeDef * TIMx, uint32_t Remap)

Function description

Remap TIM inputs (input channel, internal/external triggers).

Return values

- **None:**

Notes

- Macro IS_TIM_REMAP_INSTANCE(TIMx) can be used to check whether or not a some timer inputs can be remapped. TIM1: one of the following values: LL_TIM_TIM1_TI1_RMP_GPIO: TIM1 TI1 is connected to GPIO LL_TIM_TIM1_TI1_RMP_COMP1: TIM1 TI1 is connected to COMP1 output TIM2: one of the following values: LL_TIM_TIM2_TI4_RMP_GPIO: TIM2 TI4 is connected to GPIO LL_TIM_TIM2_TI4_RMP_COMP1: TIM2 TI4 is connected to COMP1 output LL_TIM_TIM2_TI4_RMP_COMP2: TIM2 TI4 is connected to COMP2 output LL_TIM_TIM2_TI4_RMP_COMP1_COMP2: TIM2 TI4 is connected to logical OR between COMP1 and COMP2 output TIM3: one of the following values: LL_TIM_TIM3_TI1_RMP_GPIO: TIM3 TI1 is connected to GPIO LL_TIM_TIM3_TI1_RMP_COMP1: TIM3 TI1 is connected to COMP1 output LL_TIM_TIM3_TI1_RMP_COMP2: TIM3 TI1 is connected to COMP2 output LL_TIM_TIM3_TI1_RMP_COMP1_COMP2: TIM3 TI1 is connected to logical OR between COMP1 and COMP2 output TIM5: one of the following values: LL_TIM_TIM5_TI1_RMP_GPIO: TIM5 TI1 is connected to GPIO LL_TIM_TIM5_TI1_RMP_CAN_TMP: TIM5 TI1 is connected to CAN TMP LL_TIM_TIM5_TI1_RMP_CAN_RTP: TIM5 TI1 is connected to CAN RTP TIM8: one of the following values: LL_TIM_TIM8_TI1_RMP_GPIO: TIM8 TI1 is connected to GPIO LL_TIM_TIM8_TI1_RMP_COMP2: TIM8 TI1 is connected to COMP2 output TIM12: one of the following values: (*) LL_TIM_TIM12_TI1_RMP_GPIO: TIM12 TI1 is connected to GPIO LL_TIM_TIM12_TI1_RMP_SPDIF_FS: TIM12 TI1 is connected to SPDIF FS TIM15: one of the following values: LL_TIM_TIM15_TI1_RMP_GPIO: TIM15 TI1 is connected to GPIO LL_TIM_TIM15_TI1_RMP_TIM2: TIM15 TI1 is connected to TIM2 CH1 LL_TIM_TIM15_TI1_RMP_TIM3: TIM15 TI1 is connected to TIM3 CH1 LL_TIM_TIM15_TI1_RMP_TIM4: TIM15 TI1 is connected to TIM4 CH1 LL_TIM_TIM15_TI1_RMP_LSE: TIM15 TI1 is connected to LSE LL_TIM_TIM15_TI1_RMP_CSI: TIM15 TI1 is connected to CSI LL_TIM_TIM15_TI1_RMP_MCO2: TIM15 TI1 is connected to MCO2 LL_TIM_TIM15_TI2_RMP_GPIO: TIM15 TI2 is connected to GPIO LL_TIM_TIM15_TI2_RMP_TIM2: TIM15 TI2 is connected to TIM2 CH2 LL_TIM_TIM15_TI2_RMP_TIM3: TIM15 TI2 is connected to TIM3 CH2 LL_TIM_TIM15_TI2_RMP_TIM4: TIM15 TI2 is connected to TIM4 CH2 TIM16: one of the following values: LL_TIM_TIM16_TI1_RMP_GPIO: TIM16 TI1 is connected to GPIO LL_TIM_TIM16_TI1_RMP_LSI: TIM16 TI1 is connected to LSI LL_TIM_TIM16_TI1_RMP_LSE: TIM16 TI1 is connected to LSE LL_TIM_TIM16_TI1_RMP_RTC: TIM16 TI1 is connected to RTC wakeup interrupt TIM17: one of the following values: LL_TIM_TIM17_TI1_RMP_GPIO: TIM17 TI1 is connected to GPIO LL_TIM_TIM17_TI1_RMP_SPDIF_FS: TIM17 TI1 is connected to SPDIF FS (*) LL_TIM_TIM17_TI1_RMP_HSE_1MHZ: TIM17 TI1 is connected to HSE 1MHz LL_TIM_TIM17_TI1_RMP_MCO1: TIM17 TI1 is connected to MCO1 TIM23: one of the following values: (*) LL_TIM_TIM23_TI4_RMP_GPIO TIM23_TI4 is connected to GPIO LL_TIM_TIM23_TI4_RMP_COMP1 TIM23_TI4 is connected to COMP1 output LL_TIM_TIM23_TI4_RMP_COMP2 TIM23_TI4 is connected to COMP2 output LL_TIM_TIM23_TI4_RMP_COMP1_COMP2 TIM23_TI4 is connected to COMP2 output TIM24: one of the following values: (*) LL_TIM_TIM24_TI1_RMP_GPIO TIM24_TI1 is connected to GPIO LL_TIM_TIM24_TI1_RMP_CAN_TMP TIM24_TI1 is connected to CAN_TMP LL_TIM_TIM24_TI1_RMP_CAN_RTP TIM24_TI1 is connected to CAN_RTP LL_TIM_TIM24_TI1_RMP_CAN_SOC TIM24_TI1 is connected to CAN_SOC (*) Value not defined in all devices.

LL_TIM_ClearFlag_UPDATE

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_UPDATE (TIM_TypeDef * TIMx)

Function description

Clear the update interrupt flag (UIF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR UIF LL_TIM_ClearFlag_UPDATE

LL_TIM_IsActiveFlag_UPDATE

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_UPDATE (const TIM_TypeDef * TIMx)

Function description

Indicate whether update interrupt flag (UIF) is set (update interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR UIF LL_TIM_IsActiveFlag_UPDATE

LL_TIM_ClearFlag_CC1

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC1 (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 1 interrupt flag (CC1F).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC1IF LL_TIM_ClearFlag_CC1

LL_TIM_IsActiveFlag_CC1

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC1 (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 1 interrupt flag (CC1F) is set (Capture/Compare 1 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC1IF LL_TIM_IsActiveFlag_CC1

LL_TIM_ClearFlag_CC2

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC2 (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 2 interrupt flag (CC2F).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC2IF LL_TIM_ClearFlag_CC2

LL_TIM_IsActiveFlag_CC2

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC2 (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 2 interrupt flag (CC2F) is set (Capture/Compare 2 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC2IF LL_TIM_IsActiveFlag_CC2

LL_TIM_ClearFlag_CC3

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC3 (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 3 interrupt flag (CC3F).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC3IF LL_TIM_ClearFlag_CC3

LL_TIM_IsActiveFlag_CC3

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC3 (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 3 interrupt flag (CC3F) is set (Capture/Compare 3 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC3IF LL_TIM_IsActiveFlag_CC3

LL_TIM_ClearFlag_CC4

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC4 (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 4 interrupt flag (CC4F).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC4IF LL_TIM_ClearFlag_CC4

LL_TIM_IsActiveFlag_CC4

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC4 (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 4 interrupt flag (CC4F) is set (Capture/Compare 4 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC4IF LL_TIM_IsActiveFlag_CC4

LL_TIM_ClearFlag_CC5

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC5 (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 5 interrupt flag (CC5F).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC5IF LL_TIM_ClearFlag_CC5

LL_TIM_IsActiveFlag_CC5

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC5 (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 5 interrupt flag (CC5F) is set (Capture/Compare 5 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC5IF LL_TIM_IsActiveFlag_CC5

LL_TIM_ClearFlag_CC6

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC6 (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 6 interrupt flag (CC6F).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC6IF LL_TIM_ClearFlag_CC6

LL_TIM_IsActiveFlag_CC6

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC6 (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 6 interrupt flag (CC6F) is set (Capture/Compare 6 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC6IF LL_TIM_IsActiveFlag_CC6

LL_TIM_ClearFlag_COM

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_COM (TIM_TypeDef * TIMx)

Function description

Clear the commutation interrupt flag (COMIF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR COMIF LL_TIM_ClearFlag_COM

LL_TIM_IsActiveFlag_COM

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_COM (const TIM_TypeDef * TIMx)

Function description

Indicate whether commutation interrupt flag (COMIF) is set (commutation interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR COMIF LL_TIM_IsActiveFlag_COM

LL_TIM_ClearFlag_TRIG

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_TRIG (TIM_TypeDef * TIMx)

Function description

Clear the trigger interrupt flag (TIF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR TIF LL_TIM_ClearFlag_TRIG

LL_TIM_IsActiveFlag_TRIG

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_TRIG (const TIM_TypeDef * TIMx)

Function description

Indicate whether trigger interrupt flag (TIF) is set (trigger interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TIF LL_TIM_IsActiveFlag_TRIG

LL_TIM_ClearFlag_BRK

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_BRK (TIM_TypeDef * TIMx)

Function description

Clear the break interrupt flag (BIF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR BIF LL_TIM_ClearFlag_BRK

LL_TIM_IsActiveFlag_BRK

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_BRK (const TIM_TypeDef * TIMx)

Function description

Indicate whether break interrupt flag (BIF) is set (break interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BIF LL_TIM_IsActiveFlag_BRK

LL_TIM_ClearFlag_BRK2

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_BRK2 (TIM_TypeDef * TIMx)

Function description

Clear the break 2 interrupt flag (B2IF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR B2IF LL_TIM_ClearFlag_BRK2

LL_TIM_IsActiveFlag_BRK2

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_BRK2 (const TIM_TypeDef * TIMx)

Function description

Indicate whether break 2 interrupt flag (B2IF) is set (break 2 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR B2IF LL_TIM_IsActiveFlag_BRK2

LL_TIM_ClearFlag_CC1OVR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC1OVR (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 1 over-capture interrupt flag (CC1OF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC1OF LL_TIM_ClearFlag_CC1OVR

LL_TIM_IsActiveFlag_CC1OVR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC1OVR (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 1 over-capture interrupt flag (CC1OF) is set (Capture/Compare 1 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC1OF LL_TIM_IsActiveFlag_CC1OVR

LL_TIM_ClearFlag_CC2OVR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC2OVR (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 2 over-capture interrupt flag (CC2OF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC2OF LL_TIM_ClearFlag_CC2OVR

LL_TIM_IsActiveFlag_CC2OVR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC2OVR (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 2 over-capture interrupt flag (CC2OF) is set (Capture/Compare 2 over-capture interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC2OF LL_TIM_IsActiveFlag_CC2OVR

LL_TIM_ClearFlag_CC3OVR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC3OVR (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 3 over-capture interrupt flag (CC3OF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC3OF LL_TIM_ClearFlag_CC3OVR

LL_TIM_IsActiveFlag_CC3OVR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC3OVR (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 3 over-capture interrupt flag (CC3OF) is set (Capture/Compare 3 over-capture interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC3OF LL_TIM_IsActiveFlag_CC3OVR

LL_TIM_ClearFlag_CC4OVR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC4OVR (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 4 over-capture interrupt flag (CC4OF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC4OF LL_TIM_ClearFlag_CC4OVR

LL_TIM_IsActiveFlag_CC4OVR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC4OVR (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 4 over-capture interrupt flag (CC4OF) is set (Capture/Compare 4 over-capture interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC4OF LL_TIM_IsActiveFlag_CC4OVR

LL_TIM_ClearFlag_SYSBRK

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_SYSBRK (TIM_TypeDef * TIMx)

Function description

Clear the system break interrupt flag (SBIF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR SBIF LL_TIM_ClearFlag_SYSBRK

LL_TIM_IsActiveFlag_SYSBRK

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_SYSBRK (const TIM_TypeDef * TIMx)

Function description

Indicate whether system break interrupt flag (SBIF) is set (system break interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR SBIF LL_TIM_IsActiveFlag_SYSBRK

LL_TIM_EnableIT_UPDATE

Function name

__STATIC_INLINE void LL_TIM_EnableIT_UPDATE (TIM_TypeDef * TIMx)

Function description

Enable update interrupt (UIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UIE LL_TIM_EnableIT_UPDATE

LL_TIM_DisableIT_UPDATE

Function name

__STATIC_INLINE void LL_TIM_DisableIT_UPDATE (TIM_TypeDef * TIMx)

Function description

Disable update interrupt (UIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UIE LL_TIM_DisableIT_UPDATE

LL_TIM_IsEnabledIT_UPDATE

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_UPDATE (const TIM_TypeDef * TIMx)

Function description

Indicates whether the update interrupt (UIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER UIE LL_TIM_IsEnabledIT_UPDATE

LL_TIM_EnableIT_CC1

Function name

__STATIC_INLINE void LL_TIM_EnableIT_CC1 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 1 interrupt (CC1IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1IE LL_TIM_EnableIT_CC1

LL_TIM_DisableIT_CC1

Function name

__STATIC_INLINE void LL_TIM_DisableIT_CC1 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 1 interrupt (CC1IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1IE LL_TIM_DisableIT_CC1

LL_TIM_IsEnabledIT_CC1

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC1 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 1 interrupt (CC1IE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC1IE LL_TIM_IsEnabledIT_CC1

LL_TIM_EnableIT_CC2

Function name

__STATIC_INLINE void LL_TIM_EnableIT_CC2 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 2 interrupt (CC2IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2IE LL_TIM_EnableIT_CC2

LL_TIM_DisableIT_CC2

Function name

__STATIC_INLINE void LL_TIM_DisableIT_CC2 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 2 interrupt (CC2IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2IE LL_TIM_DisableIT_CC2

LL_TIM_IsEnabledIT_CC2

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC2 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 2 interrupt (CC2IE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC2IE LL_TIM_IsEnabledIT_CC2

LL_TIM_EnableIT_CC3

Function name

__STATIC_INLINE void LL_TIM_EnableIT_CC3 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 3 interrupt (CC3IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC3IE LL_TIM_EnableIT_CC3

LL_TIM_DisableIT_CC3

Function name

__STATIC_INLINE void LL_TIM_DisableIT_CC3 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 3 interrupt (CC3IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC3IE LL_TIM_DisableIT_CC3

LL_TIM_IsEnabledIT_CC3

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC3 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 3 interrupt (CC3IE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC3IE LL_TIM_IsEnabledIT_CC3

LL_TIM_EnableIT_CC4

Function name

__STATIC_INLINE void LL_TIM_EnableIT_CC4 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 4 interrupt (CC4IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC4IE LL_TIM_EnableIT_CC4

LL_TIM_DisableIT_CC4

Function name

__STATIC_INLINE void LL_TIM_DisableIT_CC4 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 4 interrupt (CC4IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC4IE LL_TIM_DisableIT_CC4

LL_TIM_IsEnabledIT_CC4

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC4 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 4 interrupt (CC4IE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC4IE LL_TIM_IsEnabledIT_CC4

LL_TIM_EnableIT_COM

Function name

__STATIC_INLINE void LL_TIM_EnableIT_COM (TIM_TypeDef * TIMx)

Function description

Enable commutation interrupt (COMIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER COMIE LL_TIM_EnableIT_COM

LL_TIM_DisableIT_COM

Function name

__STATIC_INLINE void LL_TIM_DisableIT_COM (TIM_TypeDef * TIMx)

Function description

Disable commutation interrupt (COMIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER COMIE LL_TIM_DisableIT_COM

LL_TIM_IsEnabledIT_COM

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_COM (const TIM_TypeDef * TIMx)

Function description

Indicates whether the commutation interrupt (COMIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER COMIE LL_TIM_IsEnabledIT_COM

LL_TIM_EnableIT_TRIG

Function name

__STATIC_INLINE void LL_TIM_EnableIT_TRIG (TIM_TypeDef * TIMx)

Function description

Enable trigger interrupt (TIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER TIE LL_TIM_EnableIT_TRIG

LL_TIM_DisableIT_TRIG

Function name

__STATIC_INLINE void LL_TIM_DisableIT_TRIG (TIM_TypeDef * TIMx)

Function description

Disable trigger interrupt (TIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER TIE LL_TIM_DisableIT_TRIG

LL_TIM_IsEnabledIT_TRIG

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_TRIG (const TIM_TypeDef * TIMx)

Function description

Indicates whether the trigger interrupt (TIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER TIE LL_TIM_IsEnabledIT_TRIG

LL_TIM_EnableIT_BRK

Function name

__STATIC_INLINE void LL_TIM_EnableIT_BRK (TIM_TypeDef * TIMx)

Function description

Enable break interrupt (BIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER BIE LL_TIM_EnableIT_BRK

LL_TIM_DisableIT_BRK

Function name

__STATIC_INLINE void LL_TIM_DisableIT_BRK (TIM_TypeDef * TIMx)

Function description

Disable break interrupt (BIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER BIE LL_TIM_DisableIT_BRK

LL_TIM_IsEnabledIT_BRK

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_BRK (const TIM_TypeDef * TIMx)

Function description

Indicates whether the break interrupt (BIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER BIE LL_TIM_IsEnabledIT_BRK

LL_TIM_EnableDMAReq_UPDATE

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_UPDATE (TIM_TypeDef * TIMx)

Function description

Enable update DMA request (UDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UDE LL_TIM_EnableDMAReq_UPDATE

LL_TIM_DisableDMAReq_UPDATE

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_UPDATE (TIM_TypeDef * TIMx)

Function description

Disable update DMA request (UDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UDE LL_TIM_DisableDMAReq_UPDATE

LL_TIM_IsEnabledDMAReq_UPDATE

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_UPDATE (const TIM_TypeDef * TIMx)

Function description

Indicates whether the update DMA request (UDE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER UDE LL_TIM_IsEnabledDMAReq_UPDATE

LL_TIM_EnableDMAReq_CC1

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_CC1 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 1 DMA request (CC1DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1DE LL_TIM_EnableDMAReq_CC1

LL_TIM_DisableDMAReq_CC1

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_CC1 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 1 DMA request (CC1DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1DE LL_TIM_DisableDMAReq_CC1

LL_TIM_IsEnabledDMAReq_CC1

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC1 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 1 DMA request (CC1DE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC1DE LL_TIM_IsEnabledDMAReq_CC1

LL_TIM_EnableDMAReq_CC2

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_CC2 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 2 DMA request (CC2DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2DE LL_TIM_EnableDMAReq_CC2

LL_TIM_DisableDMAReq_CC2

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_CC2 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 2 DMA request (CC2DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2DE LL_TIM_DisableDMAReq_CC2

LL_TIM_IsEnabledDMAReq_CC2

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC2 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 2 DMA request (CC2DE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC2DE LL_TIM_IsEnabledDMAReq_CC2

LL_TIM_EnableDMAReq_CC3

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_CC3 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 3 DMA request (CC3DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC3DE LL_TIM_EnableDMAReq_CC3

LL_TIM_DisableDMAReq_CC3

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_CC3 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 3 DMA request (CC3DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC3DE LL_TIM_DisableDMAReq_CC3

LL_TIM_IsEnabledDMAReq_CC3

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC3 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 3 DMA request (CC3DE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC3DE LL_TIM_IsEnabledDMAReq_CC3

LL_TIM_EnableDMAReq_CC4

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_CC4 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 4 DMA request (CC4DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC4DE LL_TIM_EnableDMAReq_CC4

LL_TIM_DisableDMAReq_CC4

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_CC4 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 4 DMA request (CC4DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC4DE LL_TIM_DisableDMAReq_CC4

LL_TIM_IsEnabledDMAReq_CC4

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC4 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 4 DMA request (CC4DE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC4DE LL_TIM_IsEnabledDMAReq_CC4

LL_TIM_EnableDMAReq_COM

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_COM (TIM_TypeDef * TIMx)

Function description

Enable commutation DMA request (COMDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER COMDE LL_TIM_EnableDMAReq_COM

LL_TIM_DisableDMAReq_COM

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_COM (TIM_TypeDef * TIMx)

Function description

Disable commutation DMA request (COMDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER COMDE LL_TIM_DisableDMAReq_COM

LL_TIM_IsEnabledDMAReq_COM

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_COM (const TIM_TypeDef * TIMx)

Function description

Indicates whether the commutation DMA request (COMDE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER COMDE LL_TIM_IsEnabledDMAReq_COM

LL_TIM_EnableDMAReq_TRIG

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_TRIG (TIM_TypeDef * TIMx)

Function description

Enable trigger interrupt (TDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER TDE LL_TIM_EnableDMAReq_TRIG

LL_TIM_DisableDMAReq_TRIG

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_TRIG (TIM_TypeDef * TIMx)

Function description

Disable trigger interrupt (TDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER TDE LL_TIM_DisableDMAReq_TRIG

LL_TIM_IsEnabledDMAReq_TRIG

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_TRIG (const TIM_TypeDef * TIMx)

Function description

Indicates whether the trigger interrupt (TDE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER TDE LL_TIM_IsEnabledDMAReq_TRIG

LL_TIM_GenerateEvent_UPDATE

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_UPDATE (TIM_TypeDef * TIMx)

Function description

Generate an update event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR UG LL_TIM_GenerateEvent_UPDATE

LL_TIM_GenerateEvent_CC1

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_CC1 (TIM_TypeDef * TIMx)

Function description

Generate Capture/Compare 1 event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR CC1G LL_TIM_GenerateEvent_CC1

LL_TIM_GenerateEvent_CC2

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_CC2 (TIM_TypeDef * TIMx)

Function description

Generate Capture/Compare 2 event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR CC2G LL_TIM_GenerateEvent_CC2

LL_TIM_GenerateEvent_CC3

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_CC3 (TIM_TypeDef * TIMx)

Function description

Generate Capture/Compare 3 event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR CC3G LL_TIM_GenerateEvent_CC3

LL_TIM_GenerateEvent_CC4

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_CC4 (TIM_TypeDef * TIMx)

Function description

Generate Capture/Compare 4 event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR CC4G LL_TIM_GenerateEvent_CC4

LL_TIM_GenerateEvent_COM

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_COM (TIM_TypeDef * TIMx)

Function description

Generate commutation event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR COMG LL_TIM_GenerateEvent_COM

LL_TIM_GenerateEvent_TRIG

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_TRIG (TIM_TypeDef * TIMx)

Function description

Generate trigger event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR TG LL_TIM_GenerateEvent_TRIG

LL_TIM_GenerateEvent_BRK

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_BRK (TIM_TypeDef * TIMx)

Function description

Generate break event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR BG LL_TIM_GenerateEvent_BRK

LL_TIM_GenerateEvent_BRK2

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_BRK2 (TIM_TypeDef * TIMx)

Function description

Generate break 2 event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR B2G LL_TIM_GenerateEvent_BRK2

LL_TIM_DeInit

Function name

ErrorStatus LL_TIM_DeInit (TIM_TypeDef * TIMx)

Function description

Set TIMx registers to their reset values.

Parameters

- **TIMx:** Timer instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: TIMx registers are de-initialized
  - ERROR: invalid TIMx instance

LL_TIM_StructInit

Function name

void LL_TIM_StructInit (LL_TIM_InitTypeDef * TIM_InitStruct)

Function description

Set the fields of the time base unit configuration data structure to their default values.

Parameters

- **TIM_InitStruct:** pointer to a LL_TIM_InitTypeDef structure (time base unit configuration data structure)

Return values

- **None:**

LL_TIM_Init

Function name

ErrorStatus LL_TIM_Init (TIM_TypeDef * TIMx, const LL_TIM_InitTypeDef * TIM_InitStruct)

Function description

Configure the TIMx time base unit.

Parameters

- **TIMx:** Timer Instance
- **TIM_InitStruct:** pointer to a LL_TIM_InitTypeDef structure (TIMx time base unit configuration data structure)

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: TIMx registers are de-initialized
  - ERROR: not applicable

LL_TIM_OC_StructInit

Function name

void LL_TIM_OC_StructInit (LL_TIM_OC_InitTypeDef * TIM_OC_InitStruct)

Function description

Set the fields of the TIMx output channel configuration data structure to their default values.

Parameters

- **TIM_OC_InitStruct:** pointer to a LL_TIM_OC_InitTypeDef structure (the output channel configuration data structure)

Return values

- **None:**

LL_TIM_OC_Init

Function name

ErrorStatus LL_TIM_OC_Init (TIM_TypeDef * TIMx, uint32_t Channel, const LL_TIM_OC_InitTypeDef * TIM_OC_InitStruct)

Function description

Configure the TIMx output channel.

Parameters

- **TIMx:** Timer Instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6
- **TIM_OC_InitStruct:** pointer to a LL_TIM_OC_InitTypeDef structure (TIMx output channel configuration data structure)

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: TIMx output channel is initialized
  - ERROR: TIMx output channel is not initialized

LL_TIM_IC_StructInit

Function name

void LL_TIM_IC_StructInit (LL_TIM_IC_InitTypeDef * TIM_ICInitStruct)

Function description

Set the fields of the TIMx input channel configuration data structure to their default values.

Parameters

- **TIM_ICInitStruct:** pointer to a LL_TIM_IC_InitTypeDef structure (the input channel configuration data structure)

Return values

- **None:**

LL_TIM_IC_Init

Function name

ErrorStatus LL_TIM_IC_Init (TIM_TypeDef * TIMx, uint32_t Channel, const LL_TIM_IC_InitTypeDef * TIM_IC_InitStruct)

Function description

Configure the TIMx input channel.

Parameters

- **TIMx:** Timer Instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **TIM_IC_InitStruct:** pointer to a LL_TIM_IC_InitTypeDef structure (TIMx input channel configuration data structure)

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: TIMx output channel is initialized
  - ERROR: TIMx output channel is not initialized

LL_TIM_ENCODER_StructInit

Function name

void LL_TIM_ENCODER_StructInit (LL_TIM_ENCODER_InitTypeDef * TIM_EncoderInitStruct)

Function description

Fills each TIM_EncoderInitStruct field with its default value.

Parameters

- **TIM_EncoderInitStruct:** pointer to a LL_TIM_ENCODER_InitTypeDef structure (encoder interface configuration data structure)

Return values

- **None:**

LL_TIM_ENCODER_Init

Function name

ErrorStatus LL_TIM_ENCODER_Init (TIM_TypeDef * TIMx, const LL_TIM_ENCODER_InitTypeDef * TIM_EncoderInitStruct)

Function description

Configure the encoder interface of the timer instance.

Parameters

- **TIMx:** Timer Instance
- **TIM_EncoderInitStruct:** pointer to a LL_TIM_ENCODER_InitTypeDef structure (TIMx encoder interface configuration data structure)

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: TIMx registers are de-initialized
  - ERROR: not applicable

LL_TIM_HALLSENSOR_StructInit

Function name

void LL_TIM_HALLSENSOR_StructInit (LL_TIM_HALLSENSOR_InitTypeDef * TIM_HallSensorInitStruct)

Function description

Set the fields of the TIMx Hall sensor interface configuration data structure to their default values.

Parameters

- **TIM_HallSensorInitStruct:** pointer to a LL_TIM_HALLSENSOR_InitTypeDef structure (HALL sensor interface configuration data structure)

Return values

- **None:**

LL_TIM_HALLSENSOR_Init

Function name

ErrorStatus LL_TIM_HALLSENSOR_Init (TIM_TypeDef * TIMx, const LL_TIM_HALLSENSOR_InitTypeDef * TIM_HallSensorInitStruct)

Function description

Configure the Hall sensor interface of the timer instance.

Parameters

- **TIMx:** Timer Instance
- **TIM_HallSensorInitStruct:** pointer to a LL_TIM_HALLSENSOR_InitTypeDef structure (TIMx HALL sensor interface configuration data structure)

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: TIMx registers are de-initialized
  - ERROR: not applicable

Notes

- TIMx CH1, CH2 and CH3 inputs connected through a XOR to the TI1 input channel
- TIMx slave mode controller is configured in reset mode. Selected internal trigger is TI1F_ED.
- Channel 1 is configured as input, IC1 is mapped on TRC.
- Captured value stored in TIMx_CCR1 correspond to the time elapsed between 2 changes on the inputs. It gives information about motor speed.
- Channel 2 is configured in output PWM 2 mode.
- Compare value stored in TIMx_CCR2 corresponds to the commutation delay.
- OC2REF is selected as trigger output on TRGO.
- LL_TIM_IC_POLARITY_BOTHEDGE must not be used for TI1 when it is used when TIMx operates in Hall sensor interface mode.

LL_TIM_BDTR_StructInit

Function name

void LL_TIM_BDTR_StructInit (LL_TIM_BDTR_InitTypeDef * TIM_BDTRInitStruct)

Function description

Set the fields of the Break and Dead Time configuration data structure to their default values.

Parameters

- **TIM_BDTRInitStruct:** pointer to a LL_TIM_BDTR_InitTypeDef structure (Break and Dead Time configuration data structure)

Return values

- **None:**

LL_TIM_BDTR_Init

Function name

ErrorStatus LL_TIM_BDTR_Init (TIM_TypeDef * TIMx, const LL_TIM_BDTR_InitTypeDef * TIM_BDTRInitStruct)

Function description

Configure the Break and Dead Time feature of the timer instance.

Parameters

- **TIMx:** Timer Instance
- **TIM_BDTRInitStruct:** pointer to a LL_TIM_BDTR_InitTypeDef structure (Break and Dead Time configuration data structure)

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: Break and Dead Time is initialized
  - ERROR: not applicable

Notes

- As the bits BK2P, BK2E, BK2F[3:0], BKF[3:0], AOE, BKP, BKE, OSSI, OSSR and DTG[7:0] can be write-locked depending on the LOCK configuration, it can be necessary to configure all of them during the first write access to the TIMx_BDTR register.
- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.
- Macro IS_TIM_BKIN2_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a second break input.

126.3 TIM Firmware driver defines

The following section lists the various define and macros of the module.

126.3.1 TIM

TIM

Active Input Selection

LL_TIM_ACTIVEINPUT_DIRECTTI

ICx is mapped on TIx

LL_TIM_ACTIVEINPUT_INDIRECTTI

ICx is mapped on TIy

LL_TIM_ACTIVEINPUT_TRC

ICx is mapped on TRC

Automatic output enable

LL_TIM_AUTOMATICOUTPUT_DISABLE

MOE can be set only by software

LL_TIM_AUTOMATICOUTPUT_ENABLE

MOE can be set by software or automatically at the next update event

BKIN POLARITY

LL_TIM_BKIN_POLARITY_LOW

BRK BKIN input is active low

LL_TIM_BKIN_POLARITY_HIGH

BRK BKIN input is active high

BKIN SOURCE

LL_TIM_BKIN_SOURCE_BKIN

BKIN input from AF controller

LL_TIM_BKIN_SOURCE_BKCOMP1

internal signal: COMP1 output

LL_TIM_BKIN_SOURCE_BKCOMP2

internal signal: COMP2 output

LL_TIM_BKIN_SOURCE_DF1BK

internal signal: DFSDM1 break output

Break2 Enable

LL_TIM_BREAK2_DISABLE

Break2 function disabled

LL_TIM_BREAK2_ENABLE

Break2 function enabled

BREAK2 FILTER

LL_TIM_BREAK2_FILTER_FDIV1

No filter, BRK acts asynchronously

LL_TIM_BREAK2_FILTER_FDIV1_N2

fSAMPLING=fCK_INT, N=2

LL_TIM_BREAK2_FILTER_FDIV1_N4

fSAMPLING=fCK_INT, N=4

LL_TIM_BREAK2_FILTER_FDIV1_N8

fSAMPLING=fCK_INT, N=8

LL_TIM_BREAK2_FILTER_FDIV2_N6

fSAMPLING=fDTS/2, N=6

LL_TIM_BREAK2_FILTER_FDIV2_N8

fSAMPLING=fDTS/2, N=8

LL_TIM_BREAK2_FILTER_FDIV4_N6

fSAMPLING=fDTS/4, N=6

LL_TIM_BREAK2_FILTER_FDIV4_N8

fSAMPLING=fDTS/4, N=8

LL_TIM_BREAK2_FILTER_FDIV8_N6

fSAMPLING=fDTS/8, N=6

LL_TIM_BREAK2_FILTER_FDIV8_N8

fSAMPLING=fDTS/8, N=8

LL_TIM_BREAK2_FILTER_FDIV16_N5

fSAMPLING=fDTS/16, N=5

LL_TIM_BREAK2_FILTER_FDIV16_N6

fSAMPLING=fDTS/16, N=6

LL_TIM_BREAK2_FILTER_FDIV16_N8

fSAMPLING=fDTS/16, N=8

LL_TIM_BREAK2_FILTER_FDIV32_N5

fSAMPLING=fDTS/32, N=5

LL_TIM_BREAK2_FILTER_FDIV32_N6

fSAMPLING=fDTS/32, N=6

LL_TIM_BREAK2_FILTER_FDIV32_N8

fSAMPLING=fDTS/32, N=8

BREAK2 POLARITY

LL_TIM_BREAK2_POLARITY_LOW

Break input BRK2 is active low

LL_TIM_BREAK2_POLARITY_HIGH

Break input BRK2 is active high

Break Enable

LL_TIM_BREAK_DISABLE

Break function disabled

LL_TIM_BREAK_ENABLE

Break function enabled

break filter

LL_TIM_BREAK_FILTER_FDIV1

No filter, BRK acts asynchronously

LL_TIM_BREAK_FILTER_FDIV1_N2

fSAMPLING=fCK_INT, N=2

LL_TIM_BREAK_FILTER_FDIV1_N4

fSAMPLING=fCK_INT, N=4

LL_TIM_BREAK_FILTER_FDIV1_N8

fSAMPLING=fCK_INT, N=8

LL_TIM_BREAK_FILTER_FDIV2_N6

fSAMPLING=fDTS/2, N=6

LL_TIM_BREAK_FILTER_FDIV2_N8

fSAMPLING=fDTS/2, N=8

LL_TIM_BREAK_FILTER_FDIV4_N6

fSAMPLING=fDTS/4, N=6

LL_TIM_BREAK_FILTER_FDIV4_N8

fSAMPLING=fDTS/4, N=8

LL_TIM_BREAK_FILTER_FDIV8_N6

fSAMPLING=fDTS/8, N=6

LL_TIM_BREAK_FILTER_FDIV8_N8

fSAMPLING=fDTS/8, N=8

LL_TIM_BREAK_FILTER_FDIV16_N5

fSAMPLING=fDTS/16, N=5

LL_TIM_BREAK_FILTER_FDIV16_N6

fSAMPLING=fDTS/16, N=6

LL_TIM_BREAK_FILTER_FDIV16_N8

fSAMPLING=fDTS/16, N=8

LL_TIM_BREAK_FILTER_FDIV32_N5

fSAMPLING=fDTS/32, N=5

LL_TIM_BREAK_FILTER_FDIV32_N6

fSAMPLING=fDTS/32, N=6

LL_TIM_BREAK_FILTER_FDIV32_N8

fSAMPLING=fDTS/32, N=8

BREAK INPUT

LL_TIM_BREAK_INPUT_BKIN

TIMx_BKIN input

LL_TIM_BREAK_INPUT_BKIN2

TIMx_BKIN2 input

break polarity

LL_TIM_BREAK_POLARITY_LOW

Break input BRK is active low

LL_TIM_BREAK_POLARITY_HIGH

Break input BRK is active high

Capture Compare DMA Request

LL_TIM_CCDMAREQUEST_CC

CCx DMA request sent when CCx event occurs

LL_TIM_CCDMAREQUEST_UPDATE

CCx DMA requests sent when update event occurs

Capture Compare Update Source

LL_TIM_CCUPDATESOURCE_COMG_ONLY

Capture/compare control bits are updated by setting the COMG bit only

LL_TIM_CCUPDATESOURCE_COMG_AND_TRGI

Capture/compare control bits are updated by setting the COMG bit or when a rising edge occurs on trigger input (TRGI)

Channel

LL_TIM_CHANNEL_CH1

Timer input/output channel 1

LL_TIM_CHANNEL_CH1N

Timer complementary output channel 1

LL_TIM_CHANNEL_CH2

Timer input/output channel 2

LL_TIM_CHANNEL_CH2N

Timer complementary output channel 2

LL_TIM_CHANNEL_CH3

Timer input/output channel 3

LL_TIM_CHANNEL_CH3N

Timer complementary output channel 3

LL_TIM_CHANNEL_CH4

Timer input/output channel 4

LL_TIM_CHANNEL_CH5

Timer output channel 5

LL_TIM_CHANNEL_CH6

Timer output channel 6

Clock Division

LL_TIM_CLOCKDIVISION_DIV1

tDTS=tCK_INT

LL_TIM_CLOCKDIVISION_DIV2

tDTS=2*tCK_INT

LL_TIM_CLOCKDIVISION_DIV4

tDTS=4*tCK_INT

Clock Source

LL_TIM_CLOCKSOURCE_INTERNAL

The timer is clocked by the internal clock provided from the RCC

LL_TIM_CLOCKSOURCE_EXT_MODE1

Counter counts at each rising or falling edge on a selected input

LL_TIM_CLOCKSOURCE_EXT_MODE2

Counter counts at each rising or falling edge on the external trigger input ETR

Counter Direction

LL_TIM_COUNTERDIRECTION_UP

Timer counter counts up

LL_TIM_COUNTERDIRECTION_DOWN

Timer counter counts down

Counter Mode

LL_TIM_COUNTERMODE_UP

Counter used as upcounter

LL_TIM_COUNTERMODE_DOWN

Counter used as downcounter

LL_TIM_COUNTERMODE_CENTER_DOWN

The counter counts up and down alternatively. Output compare interrupt flags of output channels are set only when the counter is counting down.

LL_TIM_COUNTERMODE_CENTER_UP

The counter counts up and down alternatively. Output compare interrupt flags of output channels are set only when the counter is counting up

LL_TIM_COUNTERMODE_CENTER_UP_DOWN

The counter counts up and down alternatively. Output compare interrupt flags of output channels are set only when the counter is counting up or down.

DMA Burst Base Address

LL_TIM_DMABURST_BASEADDR_CR1

TIMx_CR1 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CR2

TIMx_CR2 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_SMCR

TIMx_SMCR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_DIER

TIMx_DIER register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_SR

TIMx_SR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_EGR

TIMx_EGR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCMR1

TIMx_CCMR1 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCMR2

TIMx_CCMR2 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCER

TIMx_CCER register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CNT

TIMx_CNT register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_PSC

TIMx_PSC register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_ARR

TIMx_ARR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_RCR

TIMx_RCR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCR1

TIMx_CCR1 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCR2

TIMx_CCR2 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCR3

TIMx_CCR3 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCR4

TIMx_CCR4 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_BDTR

TIMx_BDTR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCMR3

TIMx_CCMR3 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCR5

TIMx_CCR5 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCR6

TIMx_CCR6 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_TISEL

TIMx_TISEL register is the DMA base address for DMA burst

DMA Burst Length

LL_TIM_DMABURST_LENGTH_1TRANSFER

Transfer is done to 1 register starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_2TRANSFERS

Transfer is done to 2 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_3TRANSFERS

Transfer is done to 3 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_4TRANSFERS

Transfer is done to 4 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_5TRANSFERS

Transfer is done to 5 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_6TRANSFERS

Transfer is done to 6 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_7TRANSFERS

Transfer is done to 7 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_8TRANSFERS

Transfer is done to 1 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_9TRANSFERS

Transfer is done to 9 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_10TRANSFERS

Transfer is done to 10 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_11TRANSFERS

Transfer is done to 11 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_12TRANSFERS

Transfer is done to 12 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_13TRANSFERS

Transfer is done to 13 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_14TRANSFERS

Transfer is done to 14 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_15TRANSFERS

Transfer is done to 15 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_16TRANSFERS

Transfer is done to 16 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_17TRANSFERS

Transfer is done to 17 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_18TRANSFERS

Transfer is done to 18 registers starting from the DMA burst base address

Encoder Mode

LL_TIM_ENCODERMODE_X2_TI1

Quadrature encoder mode 1, x2 mode - Counter counts up/down on TI1FP1 edge depending on TI2FP2 level

LL_TIM_ENCODERMODE_X2_TI2

Quadrature encoder mode 2, x2 mode - Counter counts up/down on TI2FP2 edge depending on TI1FP1 level

LL_TIM_ENCODERMODE_X4_TI12

Quadrature encoder mode 3, x4 mode - Counter counts up/down on both TI1FP1 and TI2FP2 edges depending on the level of the other input

External Trigger Filter

LL_TIM_ETR_FILTER_FDIV1

No filter, sampling is done at fDTS

LL_TIM_ETR_FILTER_FDIV1_N2

fSAMPLING=fCK_INT, N=2

LL_TIM_ETR_FILTER_FDIV1_N4

fSAMPLING=fCK_INT, N=4

LL_TIM_ETR_FILTER_FDIV1_N8

fSAMPLING=fCK_INT, N=8

LL_TIM_ETR_FILTER_FDIV2_N6

fSAMPLING=fDTS/2, N=6

LL_TIM_ETR_FILTER_FDIV2_N8

fSAMPLING=fDTS/2, N=8

LL_TIM_ETR_FILTER_FDIV4_N6

fSAMPLING=fDTS/4, N=6

LL_TIM_ETR_FILTER_FDIV4_N8

fSAMPLING=fDTS/4, N=8

LL_TIM_ETR_FILTER_FDIV8_N6

fSAMPLING=fDTS/8, N=8

LL_TIM_ETR_FILTER_FDIV8_N8

fSAMPLING=fDTS/16, N=5

LL_TIM_ETR_FILTER_FDIV16_N5

fSAMPLING=fDTS/16, N=6

LL_TIM_ETR_FILTER_FDIV16_N6

fSAMPLING=fDTS/16, N=8

LL_TIM_ETR_FILTER_FDIV16_N8

fSAMPLING=fDTS/16, N=5

LL_TIM_ETR_FILTER_FDIV32_N5

fSAMPLING=fDTS/32, N=5

LL_TIM_ETR_FILTER_FDIV32_N6

fSAMPLING=fDTS/32, N=6

LL_TIM_ETR_FILTER_FDIV32_N8

fSAMPLING=fDTS/32, N=8

External Trigger Polarity

LL_TIM_ETR_POLARITY_NONINVERTED

ETR is non-inverted, active at high level or rising edge

LL_TIM_ETR_POLARITY_INVERTED

ETR is inverted, active at low level or falling edge

External Trigger Prescaler

LL_TIM_ETR_PRESCALER_DIV1

ETR prescaler OFF

LL_TIM_ETR_PRESCALER_DIV2

ETR frequency is divided by 2

LL_TIM_ETR_PRESCALER_DIV4

ETR frequency is divided by 4

LL_TIM_ETR_PRESCALER_DIV8

ETR frequency is divided by 8

Get Flags Defines

LL_TIM_SR_UIF

Update interrupt flag

LL_TIM_SR_CC1IF

Capture/compare 1 interrupt flag

LL_TIM_SR_CC2IF

Capture/compare 2 interrupt flag

LL_TIM_SR_CC3IF

Capture/compare 3 interrupt flag

LL_TIM_SR_CC4IF

Capture/compare 4 interrupt flag

LL_TIM_SR_CC5IF

Capture/compare 5 interrupt flag

LL_TIM_SR_CC6IF

Capture/compare 6 interrupt flag

LL_TIM_SR_COMIF

COM interrupt flag

LL_TIM_SR_TIF

Trigger interrupt flag

LL_TIM_SR_BIF

Break interrupt flag

LL_TIM_SR_B2IF

Second break interrupt flag

LL_TIM_SR_CC1OF

Capture/Compare 1 overcapture flag

LL_TIM_SR_CC2OF

Capture/Compare 2 overcapture flag

LL_TIM_SR_CC3OF

Capture/Compare 3 overcapture flag

LL_TIM_SR_CC4OF

Capture/Compare 4 overcapture flag

LL_TIM_SR_SBIF

System Break interrupt flag

GROUPCH5

LL_TIM_GROUPCH5_NONE

No effect of OC5REF on OC1REFC, OC2REFC and OC3REFC

LL_TIM_GROUPCH5_OC1REFC

OC1REFC is the logical AND of OC1REFC and OC5REF

LL_TIM_GROUPCH5_OC2REFC

OC2REFC is the logical AND of OC2REFC and OC5REF

LL_TIM_GROUPCH5_OC3REFC

OC3REFC is the logical AND of OC3REFC and OC5REF

Input Configuration Prescaler

LL_TIM_ICPSC_DIV1

No prescaler, capture is done each time an edge is detected on the capture input

LL_TIM_ICPSC_DIV2

Capture is done once every 2 events

LL_TIM_ICPSC_DIV4

Capture is done once every 4 events

LL_TIM_ICPSC_DIV8

Capture is done once every 8 events

Input Configuration Filter

LL_TIM_IC_FILTER_FDIV1

No filter, sampling is done at fDTS

LL_TIM_IC_FILTER_FDIV1_N2

fSAMPLING=fCK_INT, N=2

LL_TIM_IC_FILTER_FDIV1_N4

fSAMPLING=fCK_INT, N=4

LL_TIM_IC_FILTER_FDIV1_N8

fSAMPLING=fCK_INT, N=8

LL_TIM_IC_FILTER_FDIV2_N6

fSAMPLING=fDTS/2, N=6

LL_TIM_IC_FILTER_FDIV2_N8

fSAMPLING=fDTS/2, N=8

LL_TIM_IC_FILTER_FDIV4_N6

fSAMPLING=fDTS/4, N=6

LL_TIM_IC_FILTER_FDIV4_N8

fSAMPLING=fDTS/4, N=8

LL_TIM_IC_FILTER_FDIV8_N6

fSAMPLING=fDTS/8, N=6

LL_TIM_IC_FILTER_FDIV8_N8

fSAMPLING=fDTS/8, N=8

LL_TIM_IC_FILTER_FDIV16_N5

fSAMPLING=fDTS/16, N=5

LL_TIM_IC_FILTER_FDIV16_N6

fSAMPLING=fDTS/16, N=6

LL_TIM_IC_FILTER_FDIV16_N8

fSAMPLING=fDTS/16, N=8

LL_TIM_IC_FILTER_FDIV32_N5

fSAMPLING=fDTS/32, N=5

LL_TIM_IC_FILTER_FDIV32_N6

fSAMPLING=fDTS/32, N=6

LL_TIM_IC_FILTER_FDIV32_N8

fSAMPLING=fDTS/32, N=8

Input Configuration Polarity

LL_TIM_IC_POLARITY_RISING

The circuit is sensitive to TIxFP1 rising edge, TIxFP1 is not inverted

LL_TIM_IC_POLARITY_FALLING

The circuit is sensitive to TIxFP1 falling edge, TIxFP1 is inverted

LL_TIM_IC_POLARITY_BOTHEDGE

The circuit is sensitive to both TIxFP1 rising and falling edges, TIxFP1 is not inverted

IT Defines

LL_TIM_DIER_UIE

Update interrupt enable

LL_TIM_DIER_CC1IE

Capture/compare 1 interrupt enable

LL_TIM_DIER_CC2IE

Capture/compare 2 interrupt enable

LL_TIM_DIER_CC3IE

Capture/compare 3 interrupt enable

LL_TIM_DIER_CC4IE

Capture/compare 4 interrupt enable

LL_TIM_DIER_COMIE

COM interrupt enable

LL_TIM_DIER_TIE

Trigger interrupt enable

LL_TIM_DIER_BIE

Break interrupt enable

Lock Level

LL_TIM_LOCKLEVEL_OFF

LOCK OFF - No bit is write protected

LL_TIM_LOCKLEVEL_1

LOCK Level 1

LL_TIM_LOCKLEVEL_2

LOCK Level 2

LL_TIM_LOCKLEVEL_3

LOCK Level 3

Output Configuration Idle State

LL_TIM_OCIDLESTATE_LOW

OCx=0 (after a dead-time if OC is implemented) when MOE=0

LL_TIM_OCIDLESTATE_HIGH

OCx=1 (after a dead-time if OC is implemented) when MOE=0

Output Configuration Mode

LL_TIM_OCMODE_FROZEN

The comparison between the output compare register TIMx_CCRy and the counter TIMx_CNT has no effect on the output channel level

LL_TIM_OCMODE_ACTIVE

OCyREF is forced high on compare match

LL_TIM_OCMODE_INACTIVE

OCyREF is forced low on compare match

LL_TIM_OCMODE_TOGGLE

OCyREF toggles on compare match

LL_TIM_OCMODE_FORCED_INACTIVE

OCyREF is forced low

LL_TIM_OCMODE_FORCED_ACTIVE

OCyREF is forced high

LL_TIM_OCMODE_PWM1

In upcounting, channel y is active as long as TIMx_CNT<TIMx_CCRy else inactive. In downcounting, channel y is inactive as long as TIMx_CNT>TIMx_CCRy else active.

LL_TIM_OCMODE_PWM2

In upcounting, channel y is inactive as long as TIMx_CNT<TIMx_CCRy else active. In downcounting, channel y is active as long as TIMx_CNT>TIMx_CCRy else inactive

LL_TIM_OCMODE_RETRIG_OPM1

Retrigerrable OPM mode 1

LL_TIM_OCMODE_RETRIG_OPM2

Retrigerrable OPM mode 2

LL_TIM_OCMODE_COMBINED_PWM1

Combined PWM mode 1

LL_TIM_OCMODE_COMBINED_PWM2

Combined PWM mode 2

LL_TIM_OCMODE_ASSYMETRIC_PWM1

Asymmetric PWM mode 1

LL_TIM_OCMODE_ASSYMETRIC_PWM2

Asymmetric PWM mode 2

Output Configuration Polarity

LL_TIM_OCPOLARITY_HIGH

OCxactive high

LL_TIM_OCPOLARITY_LOW

OCxactive low

Output Configuration State

LL_TIM_OCSTATE_DISABLE

OCx is not active

LL_TIM_OCSTATE_ENABLE

OCx signal is output on the corresponding output pin

One Pulse Mode

LL_TIM_ONEPULSEMODE_SINGLE

Counter stops counting at the next update event

LL_TIM_ONEPULSEMODE_REPETITIVE

Counter is not stopped at update event

OSSI

LL_TIM_OSSI_DISABLE

When inactive, OCx/OCxN outputs are disabled

LL_TIM_OSSI_ENABLE

When inactive, OxC/OCxN outputs are first forced with their inactive level then forced to their idle level after the deadtime

OSSR

LL_TIM_OSSR_DISABLE

When inactive, OCx/OCxN outputs are disabled

LL_TIM_OSSR_ENABLE

When inactive, OC/OCN outputs are enabled with their inactive level as soon as CCxE=1 or CCxNE=1

Slave Mode

LL_TIM_SLAVEMODE_DISABLED

Slave mode disabled

LL_TIM_SLAVEMODE_RESET

Reset Mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter

LL_TIM_SLAVEMODE_GATED

Gated Mode - The counter clock is enabled when the trigger input (TRGI) is high

LL_TIM_SLAVEMODE_TRIGGER

Trigger Mode - The counter starts at a rising edge of the trigger TRGI

LL_TIM_SLAVEMODE_COMBINED_RESETTRIGGER

Combined reset + trigger mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter, generates an update of the registers and starts the counter

TIM12 Timer Input Ch1 Remap

LL_TIM_TIM12_TI1_RMP_GPIO

TIM12 input 1 is connected to GPIO

LL_TIM_TIM12_TI1_RMP_SPDIF_FS

TIM12 input 1 is connected to SPDIF FS

TIM15 Timer Input Ch1 Remap

LL_TIM_TIM15_TI1_RMP_GPIO

TIM15 input 1 is connected to GPIO

LL_TIM_TIM15_TI1_RMP_TIM2_CH1

TIM15 input 1 is connected to TIM2 CH1

LL_TIM_TIM15_TI1_RMP_TIM3_CH1

TIM15 input 1 is connected to TIM3 CH1

LL_TIM_TIM15_TI1_RMP_TIM4_CH1

TIM15 input 1 is connected to TIM4 CH1

LL_TIM_TIM15_TI1_RMP_RCC_LSE

TIM15 input 1 is connected to RCC LSE

LL_TIM_TIM15_TI1_RMP_RCC_CSI

TIM15 input 1 is connected to RCC CSI

LL_TIM_TIM15_TI1_RMP_RCC_MCO2

TIM15 input 1 is connected to RCC MCO2

TIM15 Timer Input Ch2 Remap

LL_TIM_TIM15_TI2_RMP_GPIO

TIM15 input 2 is connected to GPIO

LL_TIM_TIM15_TI2_RMP_TIM2_CH2

TIM15 input 2 is connected to TIM2 CH2

LL_TIM_TIM15_TI2_RMP_TIM3_CH2

TIM15 input 2 is connected to TIM3 CH2

LL_TIM_TIM15_TI2_RMP_TIM4_CH2

TIM15 input 2 is connected to TIM4 CH2

TIM16 Timer Input Ch1 Remap

LL_TIM_TIM16_TI1_RMP_GPIO

TIM16 input 1 is connected to GPIO

LL_TIM_TIM16_TI1_RMP_RCC_LSI

TIM16 input 1 is connected to RCC LSI

LL_TIM_TIM16_TI1_RMP_RCC_LSE

TIM16 input 1 is connected to RCC LSE

LL_TIM_TIM16_TI1_RMP_WKUP_IT

TIM16 input 1 is connected to WKUP_IT

TIM17 Timer Input Ch1 Remap

LL_TIM_TIM17_TI1_RMP_GPIO

TIM17 input 1 is connected to GPIO

LL_TIM_TIM17_TI1_RMP_SPDIF_FS

TIM17 input 1 is connected to SPDIF FS

LL_TIM_TIM17_TI1_RMP_RCC_HSE1MHZ

TIM17 input 1 is connected to RCC HSE 1Mhz

LL_TIM_TIM17_TI1_RMP_RCC_MCO1

TIM17 input 1 is connected to RCC MCO1

TIM1 Timer Input Ch1 Remap

LL_TIM_TIM1_TI1_RMP_GPIO

TIM1 input 1 is connected to GPIO

LL_TIM_TIM1_TI1_RMP_COMP1

TIM1 input 1 is connected to COMP1 OUT

TIM23 Timer Input Ch4 Remap

LL_TIM_TIM23_TI4_RMP_GPIO

TIM23 input 4 is connected to GPIO

LL_TIM_TIM23_TI4_RMP_COMP1

TIM23 input 4 is connected to COMP1 OUT

LL_TIM_TIM23_TI4_RMP_COMP2

TIM23 input 4 is connected to COMP2 OUT

LL_TIM_TIM23_TI4_RMP_COMP1_COMP2

TIM23 input 4 is connected to COMP1 OUT or COMP2 OUT

TIM24 Timer Input Ch1 Remap

LL_TIM_TIM24_TI1_RMP_GPIO

TIM24 input 1 is connected to GPIO

LL_TIM_TIM24_TI1_RMP_CAN_TMP

TIM24 input 1 is connected to CAN TMP

LL_TIM_TIM24_TI1_RMP_CAN_RTP

TIM24 input 1 is connected to CAN RTP

LL_TIM_TIM24_TI1_RMP_CAN_SOC

TIM24 input 1 is connected to CAN SOC

TIM2 Timer Input Ch4 Remap

LL_TIM_TIM2_TI4_RMP_GPIO

TIM2 input 4 is connected to GPIO

LL_TIM_TIM2_TI4_RMP_COMP1

TIM2 input 4 is connected to COMP1 OUT

LL_TIM_TIM2_TI4_RMP_COMP2

TIM2 input 4 is connected to COMP2 OUT

LL_TIM_TIM2_TI4_RMP_COMP1_COMP2

TIM2 input 4 is connected to COMP2 OUT OR COMP2 OUT

TIM3 Timer Input Ch1 Remap

LL_TIM_TIM3_TI1_RMP_GPIO

TIM3 input 1 is connected to GPIO

LL_TIM_TIM3_TI1_RMP_COMP1

TIM3 input 1 is connected to COMP1 OUT

LL_TIM_TIM3_TI1_RMP_COMP2

TIM3 input 1 is connected to COMP2 OUT

LL_TIM_TIM3_TI1_RMP_COMP1_COMP2

TIM3 input 1 is connected to COMP1 OUT or COMP2 OUT

TIM5 Timer Input Ch1 Remap

LL_TIM_TIM5_TI1_RMP_GPIO

TIM5 input 1 is connected to GPIO

LL_TIM_TIM5_TI1_RMP_CAN_TMP

TIM5 input 1 is connected to CAN TMP

LL_TIM_TIM5_TI1_RMP_CAN_RTP

TIM5 input 1 is connected to CAN RTP

TIM8 Timer Input Ch1 Remap

LL_TIM_TIM8_TI1_RMP_GPIO

TIM8 input 1 is connected to GPIO

LL_TIM_TIM8_TI1_RMP_COMP2

TIM8 input 1 is connected to COMP2 OUT

Trigger Output

LL_TIM_TRGO_RESET

UG bit from the TIMx_EGR register is used as trigger output

LL_TIM_TRGO_ENABLE

Counter Enable signal (CNT_EN) is used as trigger output

LL_TIM_TRGO_UPDATE

Update event is used as trigger output

LL_TIM_TRGO_CC1IF

CC1 capture or a compare match is used as trigger output

LL_TIM_TRGO_OC1REF

OC1REF signal is used as trigger output

LL_TIM_TRGO_OC2REF

OC2REF signal is used as trigger output

LL_TIM_TRGO_OC3REF

OC3REF signal is used as trigger output

LL_TIM_TRGO_OC4REF

OC4REF signal is used as trigger output

Trigger Output 2

LL_TIM_TRGO2_RESET

UG bit from the TIMx_EGR register is used as trigger output 2

LL_TIM_TRGO2_ENABLE

Counter Enable signal (CNT_EN) is used as trigger output 2

LL_TIM_TRGO2_UPDATE

Update event is used as trigger output 2

LL_TIM_TRGO2_CC1F

CC1 capture or a compare match is used as trigger output 2

LL_TIM_TRGO2_OC1

OC1REF signal is used as trigger output 2

LL_TIM_TRGO2_OC2

OC2REF signal is used as trigger output 2

LL_TIM_TRGO2_OC3

OC3REF signal is used as trigger output 2

LL_TIM_TRGO2_OC4

OC4REF signal is used as trigger output 2

LL_TIM_TRGO2_OC5

OC5REF signal is used as trigger output 2

LL_TIM_TRGO2_OC6

OC6REF signal is used as trigger output 2

LL_TIM_TRGO2_OC4_RISINGFALLING

OC4REF rising or falling edges are used as trigger output 2

LL_TIM_TRGO2_OC6_RISINGFALLING

OC6REF rising or falling edges are used as trigger output 2

LL_TIM_TRGO2_OC4_RISING_OC6_RISING

OC4REF or OC6REF rising edges are used as trigger output 2

LL_TIM_TRGO2_OC4_RISING_OC6_FALLING

OC4REF rising or OC6REF falling edges are used as trigger output 2

LL_TIM_TRGO2_OC5_RISING_OC6_RISING

OC5REF or OC6REF rising edges are used as trigger output 2

LL_TIM_TRGO2_OC5_RISING_OC6_FALLING

OC5REF rising or OC6REF falling edges are used as trigger output 2

Trigger Selection

LL_TIM_TS_ITR0

Internal Trigger 0 (ITR0) is used as trigger input

LL_TIM_TS_ITR1

Internal Trigger 1 (ITR1) is used as trigger input

LL_TIM_TS_ITR2

Internal Trigger 2 (ITR2) is used as trigger input

LL_TIM_TS_ITR3

Internal Trigger 3 (ITR3) is used as trigger input

LL_TIM_TS_ITR4

Internal Trigger 4 (ITR4) is used as trigger input

LL_TIM_TS_ITR5

Internal Trigger 5 (ITR5) is used as trigger input

LL_TIM_TS_ITR6

Internal Trigger 6 (ITR6) is used as trigger input

LL_TIM_TS_ITR7

Internal Trigger 7 (ITR7) is used as trigger input

LL_TIM_TS_ITR8

Internal Trigger 8 (ITR8) is used as trigger input

LL_TIM_TS_ITR9

Internal Trigger 9 (ITR9) is used as trigger input

LL_TIM_TS_ITR10

Internal Trigger 10 (ITR10) is used as trigger input

LL_TIM_TS_ITR11

Internal Trigger 11 (ITR11) is used as trigger input

LL_TIM_TS_ITR12

Internal Trigger 12 (ITR12) is used as trigger input

LL_TIM_TS_ITR13

Internal Trigger 13 (ITR13) is used as trigger input

LL_TIM_TS_TI1F_ED

TI1 Edge Detector (TI1F_ED) is used as trigger input

LL_TIM_TS_TI1FP1

Filtered Timer Input 1 (TI1FP1) is used as trigger input

LL_TIM_TS_TI2FP2

Filtered Timer Input 2 (TI12P2) is used as trigger input

LL_TIM_TS_ETRF

Filtered external Trigger (ETRF) is used as trigger input

Update Source

LL_TIM_UPDATESOURCE_REGULAR

Counter overflow/underflow, Setting the UG bit or Update generation through the slave mode controller generates an update request

LL_TIM_UPDATESOURCE_COUNTER

Only counter overflow/underflow generates an update request

Common Write and read registers Macros

LL_TIM_WriteReg

Description:

- Write a value in TIM register.

Parameters:

- __INSTANCE__: TIM Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_TIM_ReadReg

Description:

- Read a value in TIM register.

Parameters:

- __INSTANCE__: TIM Instance
- __REG__: Register to be read

Return value:

- Register: value

TIM Exported Constants

LL_TIM_TIM1_ETRSOURCE_GPIO

TIM1_ETR is connected to GPIO

LL_TIM_TIM1_ETRSOURCE_COMP1

TIM1_ETR is connected to COMP1 OUT

LL_TIM_TIM1_ETRSOURCE_COMP2

TIM1_ETR is connected to COMP2 OUT

LL_TIM_TIM1_ETRSOURCE_ADC1_AWD1

TIM1_ETR is connected to ADC1 AWD1

LL_TIM_TIM1_ETRSOURCE_ADC1_AWD2

TIM1_ETR is connected to ADC1 AWD2

LL_TIM_TIM1_ETRSOURCE_ADC1_AWD3

TIM1_ETR is connected to ADC1 AWD3

LL_TIM_TIM1_ETRSOURCE_ADC3_AWD1

TIM1_ETR is connected to ADC3 AWD1

LL_TIM_TIM1_ETRSOURCE_ADC3_AWD2

TIM1_ETR is connected to ADC3 AWD2

LL_TIM_TIM1_ETRSOURCE_ADC3_AWD3

TIM1_ETR is connected to ADC3 AWD3

LL_TIM_TIM8_ETRSOURCE_GPIO

TIM8_ETR is connected to GPIO

LL_TIM_TIM8_ETRSOURCE_COMP1

TIM8_ETR is connected to COMP1 OUT

LL_TIM_TIM8_ETRSOURCE_COMP2

TIM8_ETR is connected to COMP2 OUT

LL_TIM_TIM8_ETRSOURCE_ADC2_AWD1

TIM8_ETR is connected to ADC2 AWD1

LL_TIM_TIM8_ETRSOURCE_ADC2_AWD2

TIM8_ETR is connected to ADC2 AWD2

LL_TIM_TIM8_ETRSOURCE_ADC2_AWD3

TIM8_ETR is connected to ADC2 AWD3

LL_TIM_TIM8_ETRSOURCE_ADC3_AWD1

TIM8_ETR is connected to ADC3 AWD1

LL_TIM_TIM8_ETRSOURCE_ADC3_AWD2

TIM8_ETR is connected to ADC3 AWD2

LL_TIM_TIM8_ETRSOURCE_ADC3_AWD3

TIM8_ETR is connected to ADC3 AWD3

LL_TIM_TIM2_ETRSOURCE_GPIO

TIM2_ETR is connected to GPIO

LL_TIM_TIM2_ETRSOURCE_COMP1

TIM2_ETR is connected to COMP1 OUT

LL_TIM_TIM2_ETRSOURCE_COMP2

TIM2_ETR is connected to COMP2 OUT

LL_TIM_TIM2_ETRSOURCE_RCC_LSE

TIM2_ETR is connected to RCC LSE

LL_TIM_TIM2_ETRSOURCE_SAI1_FSA

TIM2_ETR is connected to SAI1 FS_A

LL_TIM_TIM2_ETRSOURCE_SAI1_FSB

TIM2_ETR is connected to SAI1 FS_B

LL_TIM_TIM3_ETRSOURCE_GPIO

TIM3_ETR is connected to GPIO

LL_TIM_TIM3_ETRSOURCE_COMP1

TIM3_ETR is connected to COMP1 OUT

LL_TIM_TIM5_ETRSOURCE_GPIO

TIM5_ETR is connected to GPIO

LL_TIM_TIM5_ETRSOURCE_SAI2_FSA

TIM5_ETR is connected to SAI2 FS_A

LL_TIM_TIM5_ETRSOURCE_SAI2_FSB

TIM5_ETR is connected to SAI2 FS_B

LL_TIM_TIM5_ETRSOURCE_SAI4_FSA

TIM5_ETR is connected to SAI4 FS_A

LL_TIM_TIM5_ETRSOURCE_SAI4_FSB

TIM5_ETR is connected to SAI4 FS_B

LL_TIM_TIM23_ETRSOURCE_GPIO

TIM23_ETR is connected to GPIO

LL_TIM_TIM23_ETRSOURCE_COMP1

TIM23_ETR is connected to COMP1 OUT

LL_TIM_TIM23_ETRSOURCE_COMP2

TIM23_ETR is connected to COMP2 OUT

LL_TIM_TIM24_ETRSOURCE_GPIO

TIM24_ETR is connected to GPIO

LL_TIM_TIM24_ETRSOURCE_SAI4_FSA

TIM24_ETR is connected to SAI4 FS_A

LL_TIM_TIM24_ETRSOURCE_SAI4_FSB

TIM24_ETR is connected to SAI4 FS_B

LL_TIM_TIM24_ETRSOURCE_SAI1_FSA

TIM24_ETR is connected to SAI1 FS_A

LL_TIM_TIM24_ETRSOURCE_SAI1_FSB

TIM24_ETR is connected to SAI1 FS_B

TIM Exported Macros

__LL_TIM_GETFLAG_UIFCPY

Description:

- HELPER macro retrieving the UIFCPY flag from the counter value.

Parameters:

- __CNT__: Counter value

Return value:

- UIF: status bit

Notes:

- ex: __LL_TIM_GETFLAG_UIFCPY (LL_TIM_GetCounter ()); Relevant only if UIF flag remapping has been enabled (UIF status bit is copied to TIMx_CNT register bit 31)

__LL_TIM_CALC_DEADTIME

Description:

- HELPER macro calculating DTG[0:7] in the TIMx_BDTR register to achieve the requested dead time duration.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __CKD__: This parameter can be one of the following values:
  - LL_TIM_CLOCKDIVISION_DIV1
  - LL_TIM_CLOCKDIVISION_DIV2
  - LL_TIM_CLOCKDIVISION_DIV4
- __DT__: deadtime duration (in ns)

Return value:

- DTG[0:7]

Notes:

- ex: __LL_TIM_CALC_DEADTIME (80000000, LL_TIM_GetClockDivision (), 120);

__LL_TIM_CALC_PSC

Description:

- HELPER macro calculating the prescaler value to achieve the required counter clock frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __CNTCLK__: counter clock frequency (in Hz)

Return value:

- Prescaler: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_PSC (80000000, 1000000);

__LL_TIM_CALC_ARR

Description:

- HELPER macro calculating the auto-reload value to achieve the required output signal frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __FREQ__: output signal frequency (in Hz)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_ARR (1000000, LL_TIM_GetPrescaler (), 10000);

__LL_TIM_CALC_DELAY

Description:

- HELPER macro calculating the compare value required to achieve the required timer output compare active/inactive delay.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)

Return value:

- Compare: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_DELAY (1000000, LL_TIM_GetPrescaler (), 10);

__LL_TIM_CALC_PULSE

Description:

- HELPER macro calculating the auto-reload value to achieve the required pulse duration (when the timer operates in one pulse mode).

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)
- __PULSE__: pulse duration (in us)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_PULSE (1000000, LL_TIM_GetPrescaler (), 10, 20);

__LL_TIM_GET_ICPSC_RATIO

Description:

- HELPER macro retrieving the ratio of the input capture prescaler.

Parameters:

- __ICPSC__: This parameter can be one of the following values:
  - LL_TIM_ICPSC_DIV1
  - LL_TIM_ICPSC_DIV2
  - LL_TIM_ICPSC_DIV4
  - LL_TIM_ICPSC_DIV8

Return value:

- Input: capture prescaler ratio (1, 2, 4 or 8)

Notes:

- ex: __LL_TIM_GET_ICPSC_RATIO (LL_TIM_IC_GetPrescaler ());

127 LL USART Generic Driver

127.1 USART Firmware driver registers structures

127.1.1 LL_USART_InitTypeDef

LL_USART_InitTypeDef is defined in the stm32h7xx_ll_usart.h

Data Fields

- ***uint32_t PrescalerValue***
- ***uint32_t BaudRate***
- ***uint32_t DataWidth***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t TransferDirection***
- ***uint32_t HardwareFlowControl***
- ***uint32_t OverSampling***

Field Documentation

- ***uint32_t LL_USART_InitTypeDef::PrescalerValue***

  Specifies the Prescaler to compute the communication baud rate. This parameter can be a value of ***USART_LL_EC_PRESCALER***.This feature can be modified afterwards using unitary function **LL_USART_SetPrescaler()**.
- ***uint32_t LL_USART_InitTypeDef::BaudRate***

  This field defines expected Usart communication baud rate.This feature can be modified afterwards using unitary function **LL_USART_SetBaudRate()**.
- ***uint32_t LL_USART_InitTypeDef::DataWidth***

  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***USART_LL_EC_DATAWIDTH***.This feature can be modified afterwards using unitary function **LL_USART_SetDataWidth()**.
- ***uint32_t LL_USART_InitTypeDef::StopBits***

  Specifies the number of stop bits transmitted. This parameter can be a value of ***USART_LL_EC_STOPBITS***.This feature can be modified afterwards using unitary function **LL_USART_SetStopBitsLength()**.
- ***uint32_t LL_USART_InitTypeDef::Parity***

  Specifies the parity mode. This parameter can be a value of ***USART_LL_EC_PARITY***.This feature can be modified afterwards using unitary function **LL_USART_SetParity()**.
- ***uint32_t LL_USART_InitTypeDef::TransferDirection***

  Specifies whether the Receive and/or Transmit mode is enabled or disabled. This parameter can be a value of ***USART_LL_EC_DIRECTION***.This feature can be modified afterwards using unitary function **LL_USART_SetTransferDirection()**.
- ***uint32_t LL_USART_InitTypeDef::HardwareFlowControl***

  Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of ***USART_LL_EC_HWCONTROL***.This feature can be modified afterwards using unitary function **LL_USART_SetHWFlowCtrl()**.
- ***uint32_t LL_USART_InitTypeDef::OverSampling***

  Specifies whether USART oversampling mode is 16 or 8. This parameter can be a value of ***USART_LL_EC_OVERSAMPLING***.This feature can be modified afterwards using unitary function **LL_USART_SetOverSampling()**.

127.1.2 LL_USART_ClockInitTypeDef

LL_USART_ClockInitTypeDef is defined in the stm32h7xx_ll_usart.h

Data Fields

- ***uint32_t ClockOutput***
- ***uint32_t ClockPolarity***
- ***uint32_t ClockPhase***
- ***uint32_t LastBitClockPulse***

Field Documentation

- ***uint32_t LL_USART_ClockInitTypeDef::ClockOutput***

  Specifies whether the USART clock is enabled or disabled. This parameter can be a value of ***USART_LL_EC_CLOCK***.USART HW configuration can be modified afterwards using unitary functions **LL_USART_EnableSCLKOutput()** or **LL_USART_DisableSCLKOutput()**. For more details, refer to description of this function.
- ***uint32_t LL_USART_ClockInitTypeDef::ClockPolarity***

  Specifies the steady state of the serial clock. This parameter can be a value of ***USART_LL_EC_POLARITY***.USART HW configuration can be modified afterwards using unitary functions **LL_USART_SetClockPolarity()**. For more details, refer to description of this function.
- ***uint32_t LL_USART_ClockInitTypeDef::ClockPhase***

  Specifies the clock transition on which the bit capture is made. This parameter can be a value of ***USART_LL_EC_PHASE***.USART HW configuration can be modified afterwards using unitary functions **LL_USART_SetClockPhase()**. For more details, refer to description of this function.
- ***uint32_t LL_USART_ClockInitTypeDef::LastBitClockPulse***

  Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of ***USART_LL_EC_LASTCLKPULSE***.USART HW configuration can be modified afterwards using unitary functions **LL_USART_SetLastClkPulseOutput()**. For more details, refer to description of this function.

127.2 USART Firmware driver API description

The following section lists the various functions of the USART library.

127.2.1 Detailed description of functions

LL_USART_Enable

Function name

__STATIC_INLINE void LL_USART_Enable (USART_TypeDef * USARTx)

Function description

USART Enable.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 UE LL_USART_Enable

LL_USART_Disable

Function name

__STATIC_INLINE void LL_USART_Disable (USART_TypeDef * USARTx)

Function description

USART Disable (all USART prescalers and outputs are disabled)

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- When USART is disabled, USART prescalers and outputs are stopped immediately, and current operations are discarded. The configuration of the USART is kept, but all the status flags, in the USARTx_ISR are set to their default values.

Reference Manual to LL API cross reference:

- CR1 UE LL_USART_Disable

LL_USART_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabled (const USART_TypeDef * USARTx)

Function description

Indicate if USART is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 UE LL_USART_IsEnabled

LL_USART_EnableFIFO

Function name

__STATIC_INLINE void LL_USART_EnableFIFO (USART_TypeDef * USARTx)

Function description

FIFO Mode Enable.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_USART_EnableFIFO

LL_USART_DisableFIFO

Function name

__STATIC_INLINE void LL_USART_DisableFIFO (USART_TypeDef * USARTx)

Function description

FIFO Mode Disable.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_USART_DisableFIFO

LL_USART_IsEnabledFIFO

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledFIFO (const USART_TypeDef * USARTx)

Function description

Indicate if FIFO Mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_USART_IsEnabledFIFO

LL_USART_SetTXFIFOThreshold

Function name

__STATIC_INLINE void LL_USART_SetTXFIFOThreshold (USART_TypeDef * USARTx, uint32_t Threshold)

Function description

Configure TX FIFO Threshold.

Parameters

- **USARTx:** USART Instance
- **Threshold:** This parameter can be one of the following values:
  - LL_USART_FIFOTHRESHOLD_1_8
  - LL_USART_FIFOTHRESHOLD_1_4
  - LL_USART_FIFOTHRESHOLD_1_2
  - LL_USART_FIFOTHRESHOLD_3_4
  - LL_USART_FIFOTHRESHOLD_7_8
  - LL_USART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_USART_SetTXFIFOThreshold

LL_USART_GetTXFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_USART_GetTXFIFOThreshold (const USART_TypeDef * USARTx)

Function description

Return TX FIFO Threshold Configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_FIFOTHRESHOLD_1_8
  - LL_USART_FIFOTHRESHOLD_1_4
  - LL_USART_FIFOTHRESHOLD_1_2
  - LL_USART_FIFOTHRESHOLD_3_4
  - LL_USART_FIFOTHRESHOLD_7_8
  - LL_USART_FIFOTHRESHOLD_8_8

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_USART_GetTXFIFOThreshold

LL_USART_SetRXFIFOThreshold

Function name

__STATIC_INLINE void LL_USART_SetRXFIFOThreshold (USART_TypeDef * USARTx, uint32_t Threshold)

Function description

Configure RX FIFO Threshold.

Parameters

- **USARTx:** USART Instance
- **Threshold:** This parameter can be one of the following values:
  - LL_USART_FIFOTHRESHOLD_1_8
  - LL_USART_FIFOTHRESHOLD_1_4
  - LL_USART_FIFOTHRESHOLD_1_2
  - LL_USART_FIFOTHRESHOLD_3_4
  - LL_USART_FIFOTHRESHOLD_7_8
  - LL_USART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RXFTCFG LL_USART_SetRXFIFOThreshold

LL_USART_GetRXFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_USART_GetRXFIFOThreshold (const USART_TypeDef * USARTx)

Function description

Return RX FIFO Threshold Configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_FIFOTHRESHOLD_1_8
  - LL_USART_FIFOTHRESHOLD_1_4
  - LL_USART_FIFOTHRESHOLD_1_2
  - LL_USART_FIFOTHRESHOLD_3_4
  - LL_USART_FIFOTHRESHOLD_7_8
  - LL_USART_FIFOTHRESHOLD_8_8

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RXFTCFG LL_USART_GetRXFIFOThreshold

LL_USART_ConfigFIFOsThreshold

Function name

__STATIC_INLINE void LL_USART_ConfigFIFOsThreshold (USART_TypeDef * USARTx, uint32_t TXThreshold, uint32_t RXThreshold)

Function description

Configure TX and RX FIFOs Threshold.

Parameters

- **USARTx:** USART Instance
- **TXThreshold:** This parameter can be one of the following values:
  - LL_USART_FIFOTHRESHOLD_1_8
  - LL_USART_FIFOTHRESHOLD_1_4
  - LL_USART_FIFOTHRESHOLD_1_2
  - LL_USART_FIFOTHRESHOLD_3_4
  - LL_USART_FIFOTHRESHOLD_7_8
  - LL_USART_FIFOTHRESHOLD_8_8
- **RXThreshold:** This parameter can be one of the following values:
  - LL_USART_FIFOTHRESHOLD_1_8
  - LL_USART_FIFOTHRESHOLD_1_4
  - LL_USART_FIFOTHRESHOLD_1_2
  - LL_USART_FIFOTHRESHOLD_3_4
  - LL_USART_FIFOTHRESHOLD_7_8
  - LL_USART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_USART_ConfigFIFOsThreshold
- CR3 RXFTCFG LL_USART_ConfigFIFOsThreshold

LL_USART_EnableInStopMode

Function name

__STATIC_INLINE void LL_USART_EnableInStopMode (USART_TypeDef * USARTx)

Function description

USART enabled in STOP Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- When this function is enabled, USART is able to wake up the MCU from Stop mode, provided that USART clock selection is HSI or LSE in RCC.
- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 UESM LL_USART_EnableInStopMode

LL_USART_DisableInStopMode

Function name

__STATIC_INLINE void LL_USART_DisableInStopMode (USART_TypeDef * USARTx)

Function description

USART disabled in STOP Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- When this function is disabled, USART is not able to wake up the MCU from Stop mode
- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 UESM LL_USART_DisableInStopMode

LL_USART_IsEnabledInStopMode

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledInStopMode (const USART_TypeDef * USARTx)

Function description

Indicate if USART is enabled in STOP Mode (able to wake up MCU from Stop mode or not)

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 UESM LL_USART_IsEnabledInStopMode

LL_USART_EnableDirectionRx

Function name

__STATIC_INLINE void LL_USART_EnableDirectionRx (USART_TypeDef * USARTx)

Function description

Receiver Enable (Receiver is enabled and begins searching for a start bit)

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_USART_EnableDirectionRx

LL_USART_DisableDirectionRx

Function name

__STATIC_INLINE void LL_USART_DisableDirectionRx (USART_TypeDef * USARTx)

Function description

Receiver Disable.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_USART_DisableDirectionRx

LL_USART_EnableDirectionTx

Function name

__STATIC_INLINE void LL_USART_EnableDirectionTx (USART_TypeDef * USARTx)

Function description

Transmitter Enable.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TE LL_USART_EnableDirectionTx

LL_USART_DisableDirectionTx

Function name

__STATIC_INLINE void LL_USART_DisableDirectionTx (USART_TypeDef * USARTx)

Function description

Transmitter Disable.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TE LL_USART_DisableDirectionTx

LL_USART_SetTransferDirection

Function name

__STATIC_INLINE void LL_USART_SetTransferDirection (USART_TypeDef * USARTx, uint32_t TransferDirection)

Function description

Configure simultaneously enabled/disabled states of Transmitter and Receiver.

Parameters

- **USARTx:** USART Instance
- **TransferDirection:** This parameter can be one of the following values:
  - LL_USART_DIRECTION_NONE
  - LL_USART_DIRECTION_RX
  - LL_USART_DIRECTION_TX
  - LL_USART_DIRECTION_TX_RX

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_USART_SetTransferDirection
- CR1 TE LL_USART_SetTransferDirection

LL_USART_GetTransferDirection

Function name

__STATIC_INLINE uint32_t LL_USART_GetTransferDirection (const USART_TypeDef * USARTx)

Function description

Return enabled/disabled states of Transmitter and Receiver.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_DIRECTION_NONE
  - LL_USART_DIRECTION_RX
  - LL_USART_DIRECTION_TX
  - LL_USART_DIRECTION_TX_RX

Reference Manual to LL API cross reference:

- CR1 RE LL_USART_GetTransferDirection
- CR1 TE LL_USART_GetTransferDirection

LL_USART_SetParity

Function name

__STATIC_INLINE void LL_USART_SetParity (USART_TypeDef * USARTx, uint32_t Parity)

Function description

Configure Parity (enabled/disabled and parity mode if enabled).

Parameters

- **USARTx:** USART Instance
- **Parity:** This parameter can be one of the following values:
  - LL_USART_PARITY_NONE
  - LL_USART_PARITY_EVEN
  - LL_USART_PARITY_ODD

Return values

- **None:**

Notes

- This function selects if hardware parity control (generation and detection) is enabled or disabled. When the parity control is enabled (Odd or Even), computed parity bit is inserted at the MSB position (9th or 8th bit depending on data width) and parity is checked on the received data.

Reference Manual to LL API cross reference:

- CR1 PS LL_USART_SetParity
- CR1 PCE LL_USART_SetParity

LL_USART_GetParity

Function name

__STATIC_INLINE uint32_t LL_USART_GetParity (const USART_TypeDef * USARTx)

Function description

Return Parity configuration (enabled/disabled and parity mode if enabled)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_PARITY_NONE
  - LL_USART_PARITY_EVEN
  - LL_USART_PARITY_ODD

Reference Manual to LL API cross reference:

- CR1 PS LL_USART_GetParity
- CR1 PCE LL_USART_GetParity

LL_USART_SetWakeUpMethod

Function name

__STATIC_INLINE void LL_USART_SetWakeUpMethod (USART_TypeDef * USARTx, uint32_t Method)

Function description

Set Receiver Wake Up method from Mute mode.

Parameters

- **USARTx:** USART Instance
- **Method:** This parameter can be one of the following values:
  - LL_USART_WAKEUP_IDLELINE
  - LL_USART_WAKEUP_ADDRESSMARK

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 WAKE LL_USART_SetWakeUpMethod

LL_USART_GetWakeUpMethod

Function name

__STATIC_INLINE uint32_t LL_USART_GetWakeUpMethod (const USART_TypeDef * USARTx)

Function description

Return Receiver Wake Up method from Mute mode.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_WAKEUP_IDLELINE
  - LL_USART_WAKEUP_ADDRESSMARK

Reference Manual to LL API cross reference:

- CR1 WAKE LL_USART_GetWakeUpMethod

LL_USART_SetDataWidth

Function name

__STATIC_INLINE void LL_USART_SetDataWidth (USART_TypeDef * USARTx, uint32_t DataWidth)

Function description

Set Word length (i.e.

Parameters

- **USARTx:** USART Instance
- **DataWidth:** This parameter can be one of the following values:
  - LL_USART_DATAWIDTH_7B
  - LL_USART_DATAWIDTH_8B
  - LL_USART_DATAWIDTH_9B

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 M0 LL_USART_SetDataWidth
- CR1 M1 LL_USART_SetDataWidth

LL_USART_GetDataWidth

Function name

__STATIC_INLINE uint32_t LL_USART_GetDataWidth (const USART_TypeDef * USARTx)

Function description

Return Word length (i.e.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_DATAWIDTH_7B
  - LL_USART_DATAWIDTH_8B
  - LL_USART_DATAWIDTH_9B

Reference Manual to LL API cross reference:

- CR1 M0 LL_USART_GetDataWidth
- CR1 M1 LL_USART_GetDataWidth

LL_USART_EnableMuteMode

Function name

__STATIC_INLINE void LL_USART_EnableMuteMode (USART_TypeDef * USARTx)

Function description

Allow switch between Mute Mode and Active mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 MME LL_USART_EnableMuteMode

LL_USART_DisableMuteMode

Function name

__STATIC_INLINE void LL_USART_DisableMuteMode (USART_TypeDef * USARTx)

Function description

Prevent Mute Mode use.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 MME LL_USART_DisableMuteMode

LL_USART_IsEnabledMuteMode

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledMuteMode (const USART_TypeDef * USARTx)

Function description

Indicate if switch between Mute Mode and Active mode is allowed.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 MME LL_USART_IsEnabledMuteMode

LL_USART_SetOverSampling

Function name

__STATIC_INLINE void LL_USART_SetOverSampling (USART_TypeDef * USARTx, uint32_t OverSampling)

Function description

Set Oversampling to 8-bit or 16-bit mode.

Parameters

- **USARTx:** USART Instance
- **OverSampling:** This parameter can be one of the following values:
  - LL_USART_OVERSAMPLING_16
  - LL_USART_OVERSAMPLING_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 OVER8 LL_USART_SetOverSampling

LL_USART_GetOverSampling

Function name

__STATIC_INLINE uint32_t LL_USART_GetOverSampling (const USART_TypeDef * USARTx)

Function description

Return Oversampling mode.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_OVERSAMPLING_16
  - LL_USART_OVERSAMPLING_8

Reference Manual to LL API cross reference:

- CR1 OVER8 LL_USART_GetOverSampling

LL_USART_SetLastClkPulseOutput

Function name

__STATIC_INLINE void LL_USART_SetLastClkPulseOutput (USART_TypeDef * USARTx, uint32_t LastBitClockPulse)

Function description

Configure if Clock pulse of the last data bit is output to the SCLK pin or not.

Parameters

- **USARTx:** USART Instance
- **LastBitClockPulse:** This parameter can be one of the following values:
  - LL_USART_LASTCLKPULSE_NO_OUTPUT
  - LL_USART_LASTCLKPULSE_OUTPUT

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBCL LL_USART_SetLastClkPulseOutput

LL_USART_GetLastClkPulseOutput

Function name

__STATIC_INLINE uint32_t LL_USART_GetLastClkPulseOutput (const USART_TypeDef * USARTx)

Function description

Retrieve Clock pulse of the last data bit output configuration (Last bit Clock pulse output to the SCLK pin or not)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_LASTCLKPULSE_NO_OUTPUT
  - LL_USART_LASTCLKPULSE_OUTPUT

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBCL LL_USART_GetLastClkPulseOutput

LL_USART_SetClockPhase

Function name

__STATIC_INLINE void LL_USART_SetClockPhase (USART_TypeDef * USARTx, uint32_t ClockPhase)

Function description

Select the phase of the clock output on the SCLK pin in synchronous mode.

Parameters

- **USARTx:** USART Instance
- **ClockPhase:** This parameter can be one of the following values:
  - LL_USART_PHASE_1EDGE
  - LL_USART_PHASE_2EDGE

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CPHA LL_USART_SetClockPhase

LL_USART_GetClockPhase

Function name

__STATIC_INLINE uint32_t LL_USART_GetClockPhase (const USART_TypeDef * USARTx)

Function description

Return phase of the clock output on the SCLK pin in synchronous mode.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_PHASE_1EDGE
  - LL_USART_PHASE_2EDGE

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CPHA LL_USART_GetClockPhase

LL_USART_SetClockPolarity

Function name

__STATIC_INLINE void LL_USART_SetClockPolarity (USART_TypeDef * USARTx, uint32_t ClockPolarity)

Function description

Select the polarity of the clock output on the SCLK pin in synchronous mode.

Parameters

- **USARTx:** USART Instance
- **ClockPolarity:** This parameter can be one of the following values:
  - LL_USART_POLARITY_LOW
  - LL_USART_POLARITY_HIGH

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CPOL LL_USART_SetClockPolarity

LL_USART_GetClockPolarity

Function name

__STATIC_INLINE uint32_t LL_USART_GetClockPolarity (const USART_TypeDef * USARTx)

Function description

Return polarity of the clock output on the SCLK pin in synchronous mode.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_POLARITY_LOW
  - LL_USART_POLARITY_HIGH

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CPOL LL_USART_GetClockPolarity

LL_USART_ConfigClock

Function name

__STATIC_INLINE void LL_USART_ConfigClock (USART_TypeDef * USARTx, uint32_t Phase, uint32_t Polarity, uint32_t LBCPOutput)

Function description

Configure Clock signal format (Phase Polarity and choice about output of last bit clock pulse)

Parameters

- **USARTx:** USART Instance
- **Phase:** This parameter can be one of the following values:
  - LL_USART_PHASE_1EDGE
  - LL_USART_PHASE_2EDGE
- **Polarity:** This parameter can be one of the following values:
  - LL_USART_POLARITY_LOW
  - LL_USART_POLARITY_HIGH
- **LBCPOutput:** This parameter can be one of the following values:
  - LL_USART_LASTCLKPULSE_NO_OUTPUT
  - LL_USART_LASTCLKPULSE_OUTPUT

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clock Phase configuration using LL_USART_SetClockPhase() functionClock Polarity configuration using LL_USART_SetClockPolarity() functionOutput of Last bit Clock pulse configuration using LL_USART_SetLastClkPulseOutput() function

Reference Manual to LL API cross reference:

- CR2 CPHA LL_USART_ConfigClock
- CR2 CPOL LL_USART_ConfigClock
- CR2 LBCL LL_USART_ConfigClock

LL_USART_SetPrescaler

Function name

__STATIC_INLINE void LL_USART_SetPrescaler (USART_TypeDef * USARTx, uint32_t PrescalerValue)

Function description

Configure Clock source prescaler for baudrate generator and oversampling.

Parameters

- **USARTx:** USART Instance
- **PrescalerValue:** This parameter can be one of the following values:
  - LL_USART_PRESCALER_DIV1
  - LL_USART_PRESCALER_DIV2
  - LL_USART_PRESCALER_DIV4
  - LL_USART_PRESCALER_DIV6
  - LL_USART_PRESCALER_DIV8
  - LL_USART_PRESCALER_DIV10
  - LL_USART_PRESCALER_DIV12
  - LL_USART_PRESCALER_DIV16
  - LL_USART_PRESCALER_DIV32
  - LL_USART_PRESCALER_DIV64
  - LL_USART_PRESCALER_DIV128
  - LL_USART_PRESCALER_DIV256

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- PRESC PRESCALER LL_USART_SetPrescaler

LL_USART_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_USART_GetPrescaler (const USART_TypeDef * USARTx)

Function description

Retrieve the Clock source prescaler for baudrate generator and oversampling.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_PRESCALER_DIV1
  - LL_USART_PRESCALER_DIV2
  - LL_USART_PRESCALER_DIV4
  - LL_USART_PRESCALER_DIV6
  - LL_USART_PRESCALER_DIV8
  - LL_USART_PRESCALER_DIV10
  - LL_USART_PRESCALER_DIV12
  - LL_USART_PRESCALER_DIV16
  - LL_USART_PRESCALER_DIV32
  - LL_USART_PRESCALER_DIV64
  - LL_USART_PRESCALER_DIV128
  - LL_USART_PRESCALER_DIV256

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- PRESC PRESCALER LL_USART_GetPrescaler

LL_USART_EnableSCLKOutput

Function name

__STATIC_INLINE void LL_USART_EnableSCLKOutput (USART_TypeDef * USARTx)

Function description

Enable Clock output on SCLK pin.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CLKEN LL_USART_EnableSCLKOutput

LL_USART_DisableSCLKOutput

Function name

__STATIC_INLINE void LL_USART_DisableSCLKOutput (USART_TypeDef * USARTx)

Function description

Disable Clock output on SCLK pin.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CLKEN LL_USART_DisableSCLKOutput

LL_USART_IsEnabledSCLKOutput

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledSCLKOutput (const USART_TypeDef * USARTx)

Function description

Indicate if Clock output on SCLK pin is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CLKEN LL_USART_IsEnabledSCLKOutput

LL_USART_SetStopBitsLength

Function name

__STATIC_INLINE void LL_USART_SetStopBitsLength (USART_TypeDef * USARTx, uint32_t StopBits)

Function description

Set the length of the stop bits.

Parameters

- **USARTx:** USART Instance
- **StopBits:** This parameter can be one of the following values:
  - LL_USART_STOPBITS_0_5
  - LL_USART_STOPBITS_1
  - LL_USART_STOPBITS_1_5
  - LL_USART_STOPBITS_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 STOP LL_USART_SetStopBitsLength

LL_USART_GetStopBitsLength

Function name

__STATIC_INLINE uint32_t LL_USART_GetStopBitsLength (const USART_TypeDef * USARTx)

Function description

Retrieve the length of the stop bits.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_STOPBITS_0_5
  - LL_USART_STOPBITS_1
  - LL_USART_STOPBITS_1_5
  - LL_USART_STOPBITS_2

Reference Manual to LL API cross reference:

- CR2 STOP LL_USART_GetStopBitsLength

LL_USART_ConfigCharacter

Function name

__STATIC_INLINE void LL_USART_ConfigCharacter (USART_TypeDef * USARTx, uint32_t DataWidth, uint32_t Parity, uint32_t StopBits)

Function description

Configure Character frame format (Datawidth, Parity control, Stop Bits)

Parameters

- **USARTx:** USART Instance
- **DataWidth:** This parameter can be one of the following values:
  - LL_USART_DATAWIDTH_7B
  - LL_USART_DATAWIDTH_8B
  - LL_USART_DATAWIDTH_9B
- **Parity:** This parameter can be one of the following values:
  - LL_USART_PARITY_NONE
  - LL_USART_PARITY_EVEN
  - LL_USART_PARITY_ODD
- **StopBits:** This parameter can be one of the following values:
  - LL_USART_STOPBITS_0_5
  - LL_USART_STOPBITS_1
  - LL_USART_STOPBITS_1_5
  - LL_USART_STOPBITS_2

Return values

- **None:**

Notes

- Call of this function is equivalent to following function call sequence : Data Width configuration using LL_USART_SetDataWidth() functionParity Control and mode configuration using LL_USART_SetParity() functionStop bits configuration using LL_USART_SetStopBitsLength() function

Reference Manual to LL API cross reference:

- CR1 PS LL_USART_ConfigCharacter
- CR1 PCE LL_USART_ConfigCharacter
- CR1 M0 LL_USART_ConfigCharacter
- CR1 M1 LL_USART_ConfigCharacter
- CR2 STOP LL_USART_ConfigCharacter

LL_USART_SetTXRXSwap

Function name

__STATIC_INLINE void LL_USART_SetTXRXSwap (USART_TypeDef * USARTx, uint32_t SwapConfig)

Function description

Configure TX/RX pins swapping setting.

Parameters

- **USARTx:** USART Instance
- **SwapConfig:** This parameter can be one of the following values:
  - LL_USART_TXRX_STANDARD
  - LL_USART_TXRX_SWAPPED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 SWAP LL_USART_SetTXRXSwap

LL_USART_GetTXRXSwap

Function name

__STATIC_INLINE uint32_t LL_USART_GetTXRXSwap (const USART_TypeDef * USARTx)

Function description

Retrieve TX/RX pins swapping configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_TXRX_STANDARD
  - LL_USART_TXRX_SWAPPED

Reference Manual to LL API cross reference:

- CR2 SWAP LL_USART_GetTXRXSwap

LL_USART_SetRXPinLevel

Function name

__STATIC_INLINE void LL_USART_SetRXPinLevel (USART_TypeDef * USARTx, uint32_t PinInvMethod)

Function description

Configure RX pin active level logic.

Parameters

- **USARTx:** USART Instance
- **PinInvMethod:** This parameter can be one of the following values:
  - LL_USART_RXPIN_LEVEL_STANDARD
  - LL_USART_RXPIN_LEVEL_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RXINV LL_USART_SetRXPinLevel

LL_USART_GetRXPinLevel

Function name

__STATIC_INLINE uint32_t LL_USART_GetRXPinLevel (const USART_TypeDef * USARTx)

Function description

Retrieve RX pin active level logic configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_RXPIN_LEVEL_STANDARD
  - LL_USART_RXPIN_LEVEL_INVERTED

Reference Manual to LL API cross reference:

- CR2 RXINV LL_USART_GetRXPinLevel

LL_USART_SetTXPinLevel

Function name

__STATIC_INLINE void LL_USART_SetTXPinLevel (USART_TypeDef * USARTx, uint32_t PinInvMethod)

Function description

Configure TX pin active level logic.

Parameters

- **USARTx:** USART Instance
- **PinInvMethod:** This parameter can be one of the following values:
  - LL_USART_TXPIN_LEVEL_STANDARD
  - LL_USART_TXPIN_LEVEL_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 TXINV LL_USART_SetTXPinLevel

LL_USART_GetTXPinLevel

Function name

__STATIC_INLINE uint32_t LL_USART_GetTXPinLevel (const USART_TypeDef * USARTx)

Function description

Retrieve TX pin active level logic configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_TXPIN_LEVEL_STANDARD
  - LL_USART_TXPIN_LEVEL_INVERTED

Reference Manual to LL API cross reference:

- CR2 TXINV LL_USART_GetTXPinLevel

LL_USART_SetBinaryDataLogic

Function name

__STATIC_INLINE void LL_USART_SetBinaryDataLogic (USART_TypeDef * USARTx, uint32_t DataLogic)

Function description

Configure Binary data logic.

Parameters

- **USARTx:** USART Instance
- **DataLogic:** This parameter can be one of the following values:
  - LL_USART_BINARY_LOGIC_POSITIVE
  - LL_USART_BINARY_LOGIC_NEGATIVE

Return values

- **None:**

Notes

- Allow to define how Logical data from the data register are send/received : either in positive/direct logic (1=H, 0=L) or in negative/inverse logic (1=L, 0=H)

Reference Manual to LL API cross reference:

- CR2 DATAINV LL_USART_SetBinaryDataLogic

LL_USART_GetBinaryDataLogic

Function name

__STATIC_INLINE uint32_t LL_USART_GetBinaryDataLogic (const USART_TypeDef * USARTx)

Function description

Retrieve Binary data configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_BINARY_LOGIC_POSITIVE
  - LL_USART_BINARY_LOGIC_NEGATIVE

Reference Manual to LL API cross reference:

- CR2 DATAINV LL_USART_GetBinaryDataLogic

LL_USART_SetTransferBitOrder

Function name

__STATIC_INLINE void LL_USART_SetTransferBitOrder (USART_TypeDef * USARTx, uint32_t BitOrder)

Function description

Configure transfer bit order (either Less or Most Significant Bit First)

Parameters

- **USARTx:** USART Instance
- **BitOrder:** This parameter can be one of the following values:
  - LL_USART_BITORDER_LSBFIRST
  - LL_USART_BITORDER_MSBFIRST

Return values

- **None:**

Notes

- MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.

Reference Manual to LL API cross reference:

- CR2 MSBFIRST LL_USART_SetTransferBitOrder

LL_USART_GetTransferBitOrder

Function name

__STATIC_INLINE uint32_t LL_USART_GetTransferBitOrder (const USART_TypeDef * USARTx)

Function description

Return transfer bit order (either Less or Most Significant Bit First)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_BITORDER_LSBFIRST
  - LL_USART_BITORDER_MSBFIRST

Notes

- MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.

Reference Manual to LL API cross reference:

- CR2 MSBFIRST LL_USART_GetTransferBitOrder

LL_USART_EnableAutoBaudRate

Function name

__STATIC_INLINE void LL_USART_EnableAutoBaudRate (USART_TypeDef * USARTx)

Function description

Enable Auto Baud-Rate Detection.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 ABREN LL_USART_EnableAutoBaudRate

LL_USART_DisableAutoBaudRate

Function name

__STATIC_INLINE void LL_USART_DisableAutoBaudRate (USART_TypeDef * USARTx)

Function description

Disable Auto Baud-Rate Detection.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 ABREN LL_USART_DisableAutoBaudRate

LL_USART_IsEnabledAutoBaud

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledAutoBaud (const USART_TypeDef * USARTx)

Function description

Indicate if Auto Baud-Rate Detection mechanism is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 ABREN LL_USART_IsEnabledAutoBaud

LL_USART_SetAutoBaudRateMode

Function name

__STATIC_INLINE void LL_USART_SetAutoBaudRateMode (USART_TypeDef * USARTx, uint32_t AutoBaudRateMode)

Function description

Set Auto Baud-Rate mode bits.

Parameters

- **USARTx:** USART Instance
- **AutoBaudRateMode:** This parameter can be one of the following values:
  - LL_USART_AUTOBAUD_DETECT_ON_STARTBIT
  - LL_USART_AUTOBAUD_DETECT_ON_FALLINGEDGE
  - LL_USART_AUTOBAUD_DETECT_ON_7F_FRAME
  - LL_USART_AUTOBAUD_DETECT_ON_55_FRAME

Return values

- **None:**

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 ABRMODE LL_USART_SetAutoBaudRateMode

LL_USART_GetAutoBaudRateMode

Function name

__STATIC_INLINE uint32_t LL_USART_GetAutoBaudRateMode (USART_TypeDef * USARTx)

Function description

Return Auto Baud-Rate mode.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_AUTOBAUD_DETECT_ON_STARTBIT
  - LL_USART_AUTOBAUD_DETECT_ON_FALLINGEDGE
  - LL_USART_AUTOBAUD_DETECT_ON_7F_FRAME
  - LL_USART_AUTOBAUD_DETECT_ON_55_FRAME

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 ABRMODE LL_USART_GetAutoBaudRateMode

LL_USART_EnableRxTimeout

Function name

__STATIC_INLINE void LL_USART_EnableRxTimeout (USART_TypeDef * USARTx)

Function description

Enable Receiver Timeout.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RTOEN LL_USART_EnableRxTimeout

LL_USART_DisableRxTimeout

Function name

__STATIC_INLINE void LL_USART_DisableRxTimeout (USART_TypeDef * USARTx)

Function description

Disable Receiver Timeout.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RTOEN LL_USART_DisableRxTimeout

LL_USART_IsEnabledRxTimeout

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledRxTimeout (const USART_TypeDef * USARTx)

Function description

Indicate if Receiver Timeout feature is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 RTOEN LL_USART_IsEnabledRxTimeout

LL_USART_ConfigNodeAddress

Function name

__STATIC_INLINE void LL_USART_ConfigNodeAddress (USART_TypeDef * USARTx, uint32_t AddressLen, uint32_t NodeAddress)

Function description

Set Address of the USART node.

Parameters

- **USARTx:** USART Instance
- **AddressLen:** This parameter can be one of the following values:
  - LL_USART_ADDRESS_DETECT_4B
  - LL_USART_ADDRESS_DETECT_7B
- **NodeAddress:** 4 or 7 bit Address of the USART node.

Return values

- **None:**

Notes

- This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with address mark detection.
- 4bits address node is used when 4-bit Address Detection is selected in ADDM7. (b7-b4 should be set to 0) 8bits address node is used when 7-bit Address Detection is selected in ADDM7. (This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with 7-bit address mark detection. The MSB of the character sent by the transmitter should be equal to 1. It may also be used for character detection during normal reception, Mute mode inactive (for example, end of block detection in ModBus protocol). In this case, the whole received character (8-bit) is compared to the ADD[7:0] value and CMF flag is set on match)

Reference Manual to LL API cross reference:

- CR2 ADD LL_USART_ConfigNodeAddress
- CR2 ADDM7 LL_USART_ConfigNodeAddress

LL_USART_GetNodeAddress

Function name

__STATIC_INLINE uint32_t LL_USART_GetNodeAddress (const USART_TypeDef * USARTx)

Function description

Return 8 bit Address of the USART node as set in ADD field of CR2.

Parameters

- **USARTx:** USART Instance

Return values

- **Address:** of the USART node (Value between Min_Data=0 and Max_Data=255)

Notes

- If 4-bit Address Detection is selected in ADDM7, only 4bits (b3-b0) of returned value are relevant (b31-b4 are not relevant) If 7-bit Address Detection is selected in ADDM7, only 8bits (b7-b0) of returned value are relevant (b31-b8 are not relevant)

Reference Manual to LL API cross reference:

- CR2 ADD LL_USART_GetNodeAddress

LL_USART_GetNodeAddressLen

Function name

__STATIC_INLINE uint32_t LL_USART_GetNodeAddressLen (const USART_TypeDef * USARTx)

Function description

Return Length of Node Address used in Address Detection mode (7-bit or 4-bit)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_ADDRESS_DETECT_4B
  - LL_USART_ADDRESS_DETECT_7B

Reference Manual to LL API cross reference:

- CR2 ADDM7 LL_USART_GetNodeAddressLen

LL_USART_EnableRTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_USART_EnableRTSHWFlowCtrl (USART_TypeDef * USARTx)

Function description

Enable RTS HW Flow Control.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RTSE LL_USART_EnableRTSHWFlowCtrl

LL_USART_DisableRTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_USART_DisableRTSHWFlowCtrl (USART_TypeDef * USARTx)

Function description

Disable RTS HW Flow Control.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RTSE LL_USART_DisableRTSHWFlowCtrl

LL_USART_EnableCTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_USART_EnableCTSHWFlowCtrl (USART_TypeDef * USARTx)

Function description

Enable CTS HW Flow Control.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 CTSE LL_USART_EnableCTSHWFlowCtrl

LL_USART_DisableCTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_USART_DisableCTSHWFlowCtrl (USART_TypeDef * USARTx)

Function description

Disable CTS HW Flow Control.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 CTSE LL_USART_DisableCTSHWFlowCtrl

LL_USART_SetHWFlowCtrl

Function name

__STATIC_INLINE void LL_USART_SetHWFlowCtrl (USART_TypeDef * USARTx, uint32_t HardwareFlowControl)

Function description

Configure HW Flow Control mode (both CTS and RTS)

Parameters

- **USARTx:** USART Instance
- **HardwareFlowControl:** This parameter can be one of the following values:
  - LL_USART_HWCONTROL_NONE
  - LL_USART_HWCONTROL_RTS
  - LL_USART_HWCONTROL_CTS
  - LL_USART_HWCONTROL_RTS_CTS

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RTSE LL_USART_SetHWFlowCtrl
- CR3 CTSE LL_USART_SetHWFlowCtrl

LL_USART_GetHWFlowCtrl

Function name

__STATIC_INLINE uint32_t LL_USART_GetHWFlowCtrl (const USART_TypeDef * USARTx)

Function description

Return HW Flow Control configuration (both CTS and RTS)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_HWCONTROL_NONE
  - LL_USART_HWCONTROL_RTS
  - LL_USART_HWCONTROL_CTS
  - LL_USART_HWCONTROL_RTS_CTS

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RTSE LL_USART_GetHWFlowCtrl
- CR3 CTSE LL_USART_GetHWFlowCtrl

LL_USART_EnableOneBitSamp

Function name

__STATIC_INLINE void LL_USART_EnableOneBitSamp (USART_TypeDef * USARTx)

Function description

Enable One bit sampling method.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 ONEBIT LL_USART_EnableOneBitSamp

LL_USART_DisableOneBitSamp

Function name

__STATIC_INLINE void LL_USART_DisableOneBitSamp (USART_TypeDef * USARTx)

Function description

Disable One bit sampling method.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 ONEBIT LL_USART_DisableOneBitSamp

LL_USART_IsEnabledOneBitSamp

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledOneBitSamp (const USART_TypeDef * USARTx)

Function description

Indicate if One bit sampling method is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 ONEBIT LL_USART_IsEnabledOneBitSamp

LL_USART_EnableOverrunDetect

Function name

__STATIC_INLINE void LL_USART_EnableOverrunDetect (USART_TypeDef * USARTx)

Function description

Enable Overrun detection.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_USART_EnableOverrunDetect

LL_USART_DisableOverrunDetect

Function name

__STATIC_INLINE void LL_USART_DisableOverrunDetect (USART_TypeDef * USARTx)

Function description

Disable Overrun detection.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_USART_DisableOverrunDetect

LL_USART_IsEnabledOverrunDetect

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledOverrunDetect (const USART_TypeDef * USARTx)

Function description

Indicate if Overrun detection is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_USART_IsEnabledOverrunDetect

LL_USART_SetWKUPType

Function name

__STATIC_INLINE void LL_USART_SetWKUPType (USART_TypeDef * USARTx, uint32_t Type)

Function description

Select event type for Wake UP Interrupt Flag (WUS[1:0] bits)

Parameters

- **USARTx:** USART Instance
- **Type:** This parameter can be one of the following values:
  - LL_USART_WAKEUP_ON_ADDRESS
  - LL_USART_WAKEUP_ON_STARTBIT
  - LL_USART_WAKEUP_ON_RXNE

Return values

- **None:**

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 WUS LL_USART_SetWKUPType

LL_USART_GetWKUPType

Function name

__STATIC_INLINE uint32_t LL_USART_GetWKUPType (const USART_TypeDef * USARTx)

Function description

Return event type for Wake UP Interrupt Flag (WUS[1:0] bits)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_WAKEUP_ON_ADDRESS
  - LL_USART_WAKEUP_ON_STARTBIT
  - LL_USART_WAKEUP_ON_RXNE

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 WUS LL_USART_GetWKUPType

LL_USART_SetBaudRate

Function name

__STATIC_INLINE void LL_USART_SetBaudRate (USART_TypeDef * USARTx, uint32_t PeriphClk, uint32_t PrescalerValue, uint32_t OverSampling, uint32_t BaudRate)

Function description

Configure USART BRR register for achieving expected Baud Rate value.

Parameters

- **USARTx:** USART Instance
- **PeriphClk:** Peripheral Clock
- **PrescalerValue:** This parameter can be one of the following values:
  - LL_USART_PRESCALER_DIV1
  - LL_USART_PRESCALER_DIV2
  - LL_USART_PRESCALER_DIV4
  - LL_USART_PRESCALER_DIV6
  - LL_USART_PRESCALER_DIV8
  - LL_USART_PRESCALER_DIV10
  - LL_USART_PRESCALER_DIV12
  - LL_USART_PRESCALER_DIV16
  - LL_USART_PRESCALER_DIV32
  - LL_USART_PRESCALER_DIV64
  - LL_USART_PRESCALER_DIV128
  - LL_USART_PRESCALER_DIV256
- **OverSampling:** This parameter can be one of the following values:
  - LL_USART_OVERSAMPLING_16
  - LL_USART_OVERSAMPLING_8
- **BaudRate:** Baud Rate

Return values

- **None:**

Notes

- Compute and set USARTDIV value in BRR Register (full BRR content) according to used Peripheral Clock, Oversampling mode, and expected Baud Rate values
- Peripheral clock and Baud rate values provided as function parameters should be valid (Baud rate value != 0)
- In case of oversampling by 16 and 8, BRR content must be greater than or equal to 16d.

Reference Manual to LL API cross reference:

- BRR BRR LL_USART_SetBaudRate

LL_USART_GetBaudRate

Function name

__STATIC_INLINE uint32_t LL_USART_GetBaudRate (const USART_TypeDef * USARTx, uint32_t PeriphClk, uint32_t PrescalerValue, uint32_t OverSampling)

Function description

Return current Baud Rate value, according to USARTDIV present in BRR register (full BRR content), and to used Peripheral Clock and Oversampling mode values.

Parameters

- **USARTx:** USART Instance
- **PeriphClk:** Peripheral Clock
- **PrescalerValue:** This parameter can be one of the following values:
  - LL_USART_PRESCALER_DIV1
  - LL_USART_PRESCALER_DIV2
  - LL_USART_PRESCALER_DIV4
  - LL_USART_PRESCALER_DIV6
  - LL_USART_PRESCALER_DIV8
  - LL_USART_PRESCALER_DIV10
  - LL_USART_PRESCALER_DIV12
  - LL_USART_PRESCALER_DIV16
  - LL_USART_PRESCALER_DIV32
  - LL_USART_PRESCALER_DIV64
  - LL_USART_PRESCALER_DIV128
  - LL_USART_PRESCALER_DIV256
- **OverSampling:** This parameter can be one of the following values:
  - LL_USART_OVERSAMPLING_16
  - LL_USART_OVERSAMPLING_8

Return values

- **Baud:** Rate

Notes

- In case of non-initialized or invalid value stored in BRR register, value 0 will be returned.
- In case of oversampling by 16 and 8, BRR content must be greater than or equal to 16d.

Reference Manual to LL API cross reference:

- BRR BRR LL_USART_GetBaudRate

LL_USART_SetRxTimeout

Function name

__STATIC_INLINE void LL_USART_SetRxTimeout (USART_TypeDef * USARTx, uint32_t Timeout)

Function description

Set Receiver Time Out Value (expressed in nb of bits duration)

Parameters

- **USARTx:** USART Instance
- **Timeout:** Value between Min_Data=0x00 and Max_Data=0x00FFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTOR RTO LL_USART_SetRxTimeout

LL_USART_GetRxTimeout

Function name

__STATIC_INLINE uint32_t LL_USART_GetRxTimeout (const USART_TypeDef * USARTx)

Function description

Get Receiver Time Out Value (expressed in nb of bits duration)

Parameters

- **USARTx:** USART Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x00FFFFFF

Reference Manual to LL API cross reference:

- RTOR RTO LL_USART_GetRxTimeout

LL_USART_SetBlockLength

Function name

__STATIC_INLINE void LL_USART_SetBlockLength (USART_TypeDef * USARTx, uint32_t BlockLength)

Function description

Set Block Length value in reception.

Parameters

- **USARTx:** USART Instance
- **BlockLength:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTOR BLEN LL_USART_SetBlockLength

LL_USART_GetBlockLength

Function name

__STATIC_INLINE uint32_t LL_USART_GetBlockLength (const USART_TypeDef * USARTx)

Function description

Get Block Length value in reception.

Parameters

- **USARTx:** USART Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- RTOR BLEN LL_USART_GetBlockLength

LL_USART_EnableIrda

Function name

__STATIC_INLINE void LL_USART_EnableIrda (USART_TypeDef * USARTx)

Function description

Enable IrDA mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 IREN LL_USART_EnableIrda

LL_USART_DisableIrda

Function name

__STATIC_INLINE void LL_USART_DisableIrda (USART_TypeDef * USARTx)

Function description

Disable IrDA mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 IREN LL_USART_DisableIrda

LL_USART_IsEnabledIrda

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIrda (const USART_TypeDef * USARTx)

Function description

Indicate if IrDA mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 IREN LL_USART_IsEnabledIrda

LL_USART_SetIrdaPowerMode

Function name

__STATIC_INLINE void LL_USART_SetIrdaPowerMode (USART_TypeDef * USARTx, uint32_t PowerMode)

Function description

Configure IrDA Power Mode (Normal or Low Power)

Parameters

- **USARTx:** USART Instance
- **PowerMode:** This parameter can be one of the following values:
  - LL_USART_IRDA_POWER_NORMAL
  - LL_USART_IRDA_POWER_LOW

Return values

- **None:**

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 IRLP LL_USART_SetIrdaPowerMode

LL_USART_GetIrdaPowerMode

Function name

__STATIC_INLINE uint32_t LL_USART_GetIrdaPowerMode (const USART_TypeDef * USARTx)

Function description

Retrieve IrDA Power Mode configuration (Normal or Low Power)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_IRDA_POWER_NORMAL
  - LL_USART_PHASE_2EDGE

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 IRLP LL_USART_GetIrdaPowerMode

LL_USART_SetIrdaPrescaler

Function name

__STATIC_INLINE void LL_USART_SetIrdaPrescaler (USART_TypeDef * USARTx, uint32_t PrescalerValue)

Function description

Set Irda prescaler value, used for dividing the USART clock source to achieve the Irda Low Power frequency (8 bits value)

Parameters

- **USARTx:** USART Instance
- **PrescalerValue:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- GTPR PSC LL_USART_SetIrdaPrescaler

LL_USART_GetIrdaPrescaler

Function name

__STATIC_INLINE uint32_t LL_USART_GetIrdaPrescaler (const USART_TypeDef * USARTx)

Function description

Return Irda prescaler value, used for dividing the USART clock source to achieve the Irda Low Power frequency (8 bits value)

Parameters

- **USARTx:** USART Instance

Return values

- **Irda:** prescaler value (Value between Min_Data=0x00 and Max_Data=0xFF)

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- GTPR PSC LL_USART_GetIrdaPrescaler

LL_USART_EnableSmartcardNACK

Function name

__STATIC_INLINE void LL_USART_EnableSmartcardNACK (USART_TypeDef * USARTx)

Function description

Enable Smartcard NACK transmission.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 NACK LL_USART_EnableSmartcardNACK

LL_USART_DisableSmartcardNACK

Function name

__STATIC_INLINE void LL_USART_DisableSmartcardNACK (USART_TypeDef * USARTx)

Function description

Disable Smartcard NACK transmission.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 NACK LL_USART_DisableSmartcardNACK

LL_USART_IsEnabledSmartcardNACK

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledSmartcardNACK (const USART_TypeDef * USARTx)

Function description

Indicate if Smartcard NACK transmission is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 NACK LL_USART_IsEnabledSmartcardNACK

LL_USART_EnableSmartcard

Function name

__STATIC_INLINE void LL_USART_EnableSmartcard (USART_TypeDef * USARTx)

Function description

Enable Smartcard mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 SCEN LL_USART_EnableSmartcard

LL_USART_DisableSmartcard

Function name

__STATIC_INLINE void LL_USART_DisableSmartcard (USART_TypeDef * USARTx)

Function description

Disable Smartcard mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 SCEN LL_USART_DisableSmartcard

LL_USART_IsEnabledSmartcard

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledSmartcard (const USART_TypeDef * USARTx)

Function description

Indicate if Smartcard mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 SCEN LL_USART_IsEnabledSmartcard

LL_USART_SetSmartcardAutoRetryCount

Function name

__STATIC_INLINE void LL_USART_SetSmartcardAutoRetryCount (USART_TypeDef * USARTx, uint32_t AutoRetryCount)

Function description

Set Smartcard Auto-Retry Count value (SCARCNT[2:0] bits)

Parameters

- **USARTx:** USART Instance
- **AutoRetryCount:** Value between Min_Data=0 and Max_Data=7

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
- This bit-field specifies the number of retries in transmit and receive, in Smartcard mode. In transmission mode, it specifies the number of automatic retransmission retries, before generating a transmission error (FE bit set). In reception mode, it specifies the number or erroneous reception trials, before generating a reception error (RXNE and PE bits set)

Reference Manual to LL API cross reference:

- CR3 SCARCNT LL_USART_SetSmartcardAutoRetryCount

LL_USART_GetSmartcardAutoRetryCount

Function name

__STATIC_INLINE uint32_t LL_USART_GetSmartcardAutoRetryCount (const USART_TypeDef * USARTx)

Function description

Return Smartcard Auto-Retry Count value (SCARCNT[2:0] bits)

Parameters

- **USARTx:** USART Instance

Return values

- **Smartcard:** Auto-Retry Count value (Value between Min_Data=0 and Max_Data=7)

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 SCARCNT LL_USART_GetSmartcardAutoRetryCount

LL_USART_SetSmartcardPrescaler

Function name

__STATIC_INLINE void LL_USART_SetSmartcardPrescaler (USART_TypeDef * USARTx, uint32_t PrescalerValue)

Function description

Set Smartcard prescaler value, used for dividing the USART clock source to provide the SMARTCARD Clock (5 bits value)

Parameters

- **USARTx:** USART Instance
- **PrescalerValue:** Value between Min_Data=0 and Max_Data=31

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- GTPR PSC LL_USART_SetSmartcardPrescaler

LL_USART_GetSmartcardPrescaler

Function name

__STATIC_INLINE uint32_t LL_USART_GetSmartcardPrescaler (const USART_TypeDef * USARTx)

Function description

Return Smartcard prescaler value, used for dividing the USART clock source to provide the SMARTCARD Clock (5 bits value)

Parameters

- **USARTx:** USART Instance

Return values

- **Smartcard:** prescaler value (Value between Min_Data=0 and Max_Data=31)

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- GTPR PSC LL_USART_GetSmartcardPrescaler

LL_USART_SetSmartcardGuardTime

Function name

__STATIC_INLINE void LL_USART_SetSmartcardGuardTime (USART_TypeDef * USARTx, uint32_t GuardTime)

Function description

Set Smartcard Guard time value, expressed in nb of baud clocks periods (GT[7:0] bits : Guard time value)

Parameters

- **USARTx:** USART Instance
- **GuardTime:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- GTPR GT LL_USART_SetSmartcardGuardTime

LL_USART_GetSmartcardGuardTime

Function name

__STATIC_INLINE uint32_t LL_USART_GetSmartcardGuardTime (const USART_TypeDef * USARTx)

Function description

Return Smartcard Guard time value, expressed in nb of baud clocks periods (GT[7:0] bits : Guard time value)

Parameters

- **USARTx:** USART Instance

Return values

- **Smartcard:** Guard time value (Value between Min_Data=0x00 and Max_Data=0xFF)

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- GTPR GT LL_USART_GetSmartcardGuardTime

LL_USART_EnableHalfDuplex

Function name

__STATIC_INLINE void LL_USART_EnableHalfDuplex (USART_TypeDef * USARTx)

Function description

Enable Single Wire Half-Duplex mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_USART_EnableHalfDuplex

LL_USART_DisableHalfDuplex

Function name

__STATIC_INLINE void LL_USART_DisableHalfDuplex (USART_TypeDef * USARTx)

Function description

Disable Single Wire Half-Duplex mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_USART_DisableHalfDuplex

LL_USART_IsEnabledHalfDuplex

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledHalfDuplex (const USART_TypeDef * USARTx)

Function description

Indicate if Single Wire Half-Duplex mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_USART_IsEnabledHalfDuplex

LL_USART_EnableSPISlave

Function name

__STATIC_INLINE void LL_USART_EnableSPISlave (USART_TypeDef * USARTx)

Function description

Enable SPI Synchronous Slave mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 SLVEN LL_USART_EnableSPISlave

LL_USART_DisableSPISlave

Function name

__STATIC_INLINE void LL_USART_DisableSPISlave (USART_TypeDef * USARTx)

Function description

Disable SPI Synchronous Slave mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 SLVEN LL_USART_DisableSPISlave

LL_USART_IsEnabledSPISlave

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledSPISlave (const USART_TypeDef * USARTx)

Function description

Indicate if SPI Synchronous Slave mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 SLVEN LL_USART_IsEnabledSPISlave

LL_USART_EnableSPISlaveSelect

Function name

__STATIC_INLINE void LL_USART_EnableSPISlaveSelect (USART_TypeDef * USARTx)

Function description

Enable SPI Slave Selection using NSS input pin.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.
- SPI Slave Selection depends on NSS input pin (The slave is selected when NSS is low and deselected when NSS is high).

Reference Manual to LL API cross reference:

- CR2 DIS_NSS LL_USART_EnableSPISlaveSelect

LL_USART_DisableSPISlaveSelect

Function name

__STATIC_INLINE void LL_USART_DisableSPISlaveSelect (USART_TypeDef * USARTx)

Function description

Disable SPI Slave Selection using NSS input pin.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.
- SPI Slave will be always selected and NSS input pin will be ignored.

Reference Manual to LL API cross reference:

- CR2 DIS_NSS LL_USART_DisableSPISlaveSelect

LL_USART_IsEnabledSPISlaveSelect

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledSPISlaveSelect (const USART_TypeDef * USARTx)

Function description

Indicate if SPI Slave Selection depends on NSS input pin.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 DIS_NSS LL_USART_IsEnabledSPISlaveSelect

LL_USART_SetLINBrkDetectionLen

Function name

__STATIC_INLINE void LL_USART_SetLINBrkDetectionLen (USART_TypeDef * USARTx, uint32_t LINBDLength)

Function description

Set LIN Break Detection Length.

Parameters

- **USARTx:** USART Instance
- **LINBDLength:** This parameter can be one of the following values:
  - LL_USART_LINBREAK_DETECT_10B
  - LL_USART_LINBREAK_DETECT_11B

Return values

- **None:**

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBDL LL_USART_SetLINBrkDetectionLen

LL_USART_GetLINBrkDetectionLen

Function name

__STATIC_INLINE uint32_t LL_USART_GetLINBrkDetectionLen (const USART_TypeDef * USARTx)

Function description

Return LIN Break Detection Length.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_LINBREAK_DETECT_10B
  - LL_USART_LINBREAK_DETECT_11B

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBDL LL_USART_GetLINBrkDetectionLen

LL_USART_EnableLIN

Function name

__STATIC_INLINE void LL_USART_EnableLIN (USART_TypeDef * USARTx)

Function description

Enable LIN mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_EnableLIN

LL_USART_DisableLIN

Function name

__STATIC_INLINE void LL_USART_DisableLIN (USART_TypeDef * USARTx)

Function description

Disable LIN mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_DisableLIN

LL_USART_IsEnabledLIN

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledLIN (const USART_TypeDef * USARTx)

Function description

Indicate if LIN mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_IsEnabledLIN

LL_USART_SetDEDeassertionTime

Function name

__STATIC_INLINE void LL_USART_SetDEDeassertionTime (USART_TypeDef * USARTx, uint32_t Time)

Function description

Set DEDT (Driver Enable De-Assertion Time), Time value expressed on 5 bits ([4:0] bits).

Parameters

- **USARTx:** USART Instance
- **Time:** Value between Min_Data=0 and Max_Data=31

Return values

- **None:**

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 DEDT LL_USART_SetDEDeassertionTime

LL_USART_GetDEDeassertionTime

Function name

__STATIC_INLINE uint32_t LL_USART_GetDEDeassertionTime (const USART_TypeDef * USARTx)

Function description

Return DEDT (Driver Enable De-Assertion Time)

Parameters

- **USARTx:** USART Instance

Return values

- **Time:** value expressed on 5 bits ([4:0] bits) : Value between Min_Data=0 and Max_Data=31

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 DEDT LL_USART_GetDEDeassertionTime

LL_USART_SetDEAssertionTime

Function name

__STATIC_INLINE void LL_USART_SetDEAssertionTime (USART_TypeDef * USARTx, uint32_t Time)

Function description

Set DEAT (Driver Enable Assertion Time), Time value expressed on 5 bits ([4:0] bits).

Parameters

- **USARTx:** USART Instance
- **Time:** Value between Min_Data=0 and Max_Data=31

Return values

- **None:**

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 DEAT LL_USART_SetDEAssertionTime

LL_USART_GetDEAssertionTime

Function name

__STATIC_INLINE uint32_t LL_USART_GetDEAssertionTime (const USART_TypeDef * USARTx)

Function description

Return DEAT (Driver Enable Assertion Time)

Parameters

- **USARTx:** USART Instance

Return values

- **Time:** value expressed on 5 bits ([4:0] bits) : Value between Min_Data=0 and Max_Data=31

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 DEAT LL_USART_GetDEAssertionTime

LL_USART_EnableDEMode

Function name

__STATIC_INLINE void LL_USART_EnableDEMode (USART_TypeDef * USARTx)

Function description

Enable Driver Enable (DE) Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 DEM LL_USART_EnableDEMode

LL_USART_DisableDEMode

Function name

__STATIC_INLINE void LL_USART_DisableDEMode (USART_TypeDef * USARTx)

Function description

Disable Driver Enable (DE) Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 DEM LL_USART_DisableDEMode

LL_USART_IsEnabledDEMode

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledDEMode (const USART_TypeDef * USARTx)

Function description

Indicate if Driver Enable (DE) Mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 DEM LL_USART_IsEnabledDEMode

LL_USART_SetDESignalPolarity

Function name

__STATIC_INLINE void LL_USART_SetDESignalPolarity (USART_TypeDef * USARTx, uint32_t Polarity)

Function description

Select Driver Enable Polarity.

Parameters

- **USARTx:** USART Instance
- **Polarity:** This parameter can be one of the following values:
  - LL_USART_DE_POLARITY_HIGH
  - LL_USART_DE_POLARITY_LOW

Return values

- **None:**

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 DEP LL_USART_SetDESignalPolarity

LL_USART_GetDESignalPolarity

Function name

__STATIC_INLINE uint32_t LL_USART_GetDESignalPolarity (const USART_TypeDef * USARTx)

Function description

Return Driver Enable Polarity.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_DE_POLARITY_HIGH
  - LL_USART_DE_POLARITY_LOW

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 DEP LL_USART_GetDESignalPolarity

LL_USART_ConfigAsyncMode

Function name

__STATIC_INLINE void LL_USART_ConfigAsyncMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Asynchronous Mode (UART)

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In UART mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,CLKEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() function
- Other remaining configurations items related to Asynchronous Mode (as Baud Rate, Word length, Parity, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigAsyncMode
- CR2 CLKEN LL_USART_ConfigAsyncMode
- CR3 SCEN LL_USART_ConfigAsyncMode
- CR3 IREN LL_USART_ConfigAsyncMode
- CR3 HDSEL LL_USART_ConfigAsyncMode

LL_USART_ConfigSyncMode

Function name

__STATIC_INLINE void LL_USART_ConfigSyncMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Synchronous Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In Synchronous mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also sets the USART in Synchronous mode.
- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionSet CLKEN in CR2 using LL_USART_EnableSCLKOutput() function
- Other remaining configurations items related to Synchronous Mode (as Baud Rate, Word length, Parity, Clock Polarity, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigSyncMode
- CR2 CLKEN LL_USART_ConfigSyncMode
- CR3 SCEN LL_USART_ConfigSyncMode
- CR3 IREN LL_USART_ConfigSyncMode
- CR3 HDSEL LL_USART_ConfigSyncMode

LL_USART_ConfigLINMode

Function name

__STATIC_INLINE void LL_USART_ConfigLINMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in LIN Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In LIN mode, the following bits must be kept cleared: STOP and CLKEN bits in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also set the UART/USART in LIN mode.
- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear STOP in CR2 using LL_USART_SetStopBitsLength() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionSet LINEN in CR2 using LL_USART_EnableLIN() function
- Other remaining configurations items related to LIN Mode (as Baud Rate, Word length, LIN Break Detection Length, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 CLKEN LL_USART_ConfigLINMode
- CR2 STOP LL_USART_ConfigLINMode
- CR2 LINEN LL_USART_ConfigLINMode
- CR3 IREN LL_USART_ConfigLINMode
- CR3 SCEN LL_USART_ConfigLINMode
- CR3 HDSEL LL_USART_ConfigLINMode

LL_USART_ConfigHalfDuplexMode

Function name

__STATIC_INLINE void LL_USART_ConfigHalfDuplexMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Half Duplex Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In Half Duplex mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,CLKEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register, This function also sets the UART/USART in Half Duplex mode.
- Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionSet HDSEL in CR3 using LL_USART_EnableHalfDuplex() function
- Other remaining configurations items related to Half Duplex Mode (as Baud Rate, Word length, Parity, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigHalfDuplexMode
- CR2 CLKEN LL_USART_ConfigHalfDuplexMode
- CR3 HDSEL LL_USART_ConfigHalfDuplexMode
- CR3 SCEN LL_USART_ConfigHalfDuplexMode
- CR3 IREN LL_USART_ConfigHalfDuplexMode

LL_USART_ConfigSmartcardMode

Function name

__STATIC_INLINE void LL_USART_ConfigSmartcardMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Smartcard Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In Smartcard mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also configures Stop bits to 1.5 bits and sets the USART in Smartcard mode (SCEN bit). Clock Output is also enabled (CLKEN).
- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionConfigure STOP in CR2 using LL_USART_SetStopBitsLength() functionSet CLKEN in CR2 using LL_USART_EnableSCLKOutput() functionSet SCEN in CR3 using LL_USART_EnableSmartcard() function
- Other remaining configurations items related to Smartcard Mode (as Baud Rate, Word length, Parity, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigSmartcardMode
- CR2 STOP LL_USART_ConfigSmartcardMode
- CR2 CLKEN LL_USART_ConfigSmartcardMode
- CR3 HDSEL LL_USART_ConfigSmartcardMode
- CR3 SCEN LL_USART_ConfigSmartcardMode

LL_USART_ConfigIrdaMode

Function name

__STATIC_INLINE void LL_USART_ConfigIrdaMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Irda Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In IRDA mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,STOP and CLKEN bits in the USART_CR2 register,SCEN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also sets the UART/USART in IRDA mode (IREN bit).
- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionConfigure STOP in CR2 using LL_USART_SetStopBitsLength() functionSet IREN in CR3 using LL_USART_EnableIrda() function
- Other remaining configurations items related to Irda Mode (as Baud Rate, Word length, Power mode, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigIrdaMode
- CR2 CLKEN LL_USART_ConfigIrdaMode
- CR2 STOP LL_USART_ConfigIrdaMode
- CR3 SCEN LL_USART_ConfigIrdaMode
- CR3 HDSEL LL_USART_ConfigIrdaMode
- CR3 IREN LL_USART_ConfigIrdaMode

LL_USART_ConfigMultiProcessMode

Function name

__STATIC_INLINE void LL_USART_ConfigMultiProcessMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Multi processor Mode (several USARTs connected in a network, one of the USARTs can be the master, its TX output connected to the RX inputs of the other slaves USARTs).

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In MultiProcessor mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,CLKEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() function
- Other remaining configurations items related to Multi processor Mode (as Baud Rate, Wake Up Method, Node address, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigMultiProcessMode
- CR2 CLKEN LL_USART_ConfigMultiProcessMode
- CR3 SCEN LL_USART_ConfigMultiProcessMode
- CR3 HDSEL LL_USART_ConfigMultiProcessMode
- CR3 IREN LL_USART_ConfigMultiProcessMode

LL_USART_IsActiveFlag_PE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_PE (const USART_TypeDef * USARTx)

Function description

Check if the USART Parity Error Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR PE LL_USART_IsActiveFlag_PE

LL_USART_IsActiveFlag_FE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_FE (const USART_TypeDef * USARTx)

Function description

Check if the USART Framing Error Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR FE LL_USART_IsActiveFlag_FE

LL_USART_IsActiveFlag_NE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_NE (const USART_TypeDef * USARTx)

Function description

Check if the USART Noise error detected Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR NE LL_USART_IsActiveFlag_NE

LL_USART_IsActiveFlag_ORE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_ORE (const USART_TypeDef * USARTx)

Function description

Check if the USART OverRun Error Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ORE LL_USART_IsActiveFlag_ORE

LL_USART_IsActiveFlag_IDLE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_IDLE (const USART_TypeDef * USARTx)

Function description

Check if the USART IDLE line detected Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR IDLE LL_USART_IsActiveFlag_IDLE

LL_USART_IsActiveFlag_RXNE_RXFNE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RXNE_RXFNE (const USART_TypeDef * USARTx)

Function description

Check if the USART Read Data Register or USART RX FIFO Not Empty Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR RXNE_RXFNE LL_USART_IsActiveFlag_RXNE_RXFNE

LL_USART_IsActiveFlag_TC

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TC (const USART_TypeDef * USARTx)

Function description

Check if the USART Transmission Complete Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TC LL_USART_IsActiveFlag_TC

LL_USART_IsActiveFlag_TXE_TXFNF

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TXE_TXFNF (const USART_TypeDef * USARTx)

Function description

Check if the USART Transmit Data Register Empty or USART TX FIFO Not Full Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR TXE_TXFNF LL_USART_IsActiveFlag_TXE_TXFNF

LL_USART_IsActiveFlag_LBD

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_LBD (const USART_TypeDef * USARTx)

Function description

Check if the USART LIN Break Detection Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR LBDF LL_USART_IsActiveFlag_LBD

LL_USART_IsActiveFlag_nCTS

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_nCTS (const USART_TypeDef * USARTx)

Function description

Check if the USART CTS interrupt Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR CTSIF LL_USART_IsActiveFlag_nCTS

LL_USART_IsActiveFlag_CTS

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_CTS (const USART_TypeDef * USARTx)

Function description

Check if the USART CTS Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR CTS LL_USART_IsActiveFlag_CTS

LL_USART_IsActiveFlag_RTO

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RTO (const USART_TypeDef * USARTx)

Function description

Check if the USART Receiver Time Out Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RTOF LL_USART_IsActiveFlag_RTO

LL_USART_IsActiveFlag_EOB

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_EOB (const USART_TypeDef * USARTx)

Function description

Check if the USART End Of Block Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR EOBF LL_USART_IsActiveFlag_EOB

LL_USART_IsActiveFlag_UDR

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_UDR (const USART_TypeDef * USARTx)

Function description

Check if the SPI Slave Underrun error flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR UDR LL_USART_IsActiveFlag_UDR

LL_USART_IsActiveFlag_ABRE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_ABRE (const USART_TypeDef * USARTx)

Function description

Check if the USART Auto-Baud Rate Error Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR ABRE LL_USART_IsActiveFlag_ABRE

LL_USART_IsActiveFlag_ABR

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_ABR (const USART_TypeDef * USARTx)

Function description

Check if the USART Auto-Baud Rate Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR ABRF LL_USART_IsActiveFlag_ABR

LL_USART_IsActiveFlag_BUSY

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_BUSY (const USART_TypeDef * USARTx)

Function description

Check if the USART Busy Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR BUSY LL_USART_IsActiveFlag_BUSY

LL_USART_IsActiveFlag_CM

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_CM (const USART_TypeDef * USARTx)

Function description

Check if the USART Character Match Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CMF LL_USART_IsActiveFlag_CM

LL_USART_IsActiveFlag_SBK

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_SBK (const USART_TypeDef * USARTx)

Function description

Check if the USART Send Break Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR SBKF LL_USART_IsActiveFlag_SBK

LL_USART_IsActiveFlag_RWU

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RWU (const USART_TypeDef * USARTx)

Function description

Check if the USART Receive Wake Up from mute mode Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RWU LL_USART_IsActiveFlag_RWU

LL_USART_IsActiveFlag_WKUP

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_WKUP (const USART_TypeDef * USARTx)

Function description

Check if the USART Wake Up from stop mode Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR WUF LL_USART_IsActiveFlag_WKUP

LL_USART_IsActiveFlag_TEACK

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TEACK (const USART_TypeDef * USARTx)

Function description

Check if the USART Transmit Enable Acknowledge Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEACK LL_USART_IsActiveFlag_TEACK

LL_USART_IsActiveFlag_REACK

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_REACK (const USART_TypeDef * USARTx)

Function description

Check if the USART Receive Enable Acknowledge Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR REACK LL_USART_IsActiveFlag_REACK

LL_USART_IsActiveFlag_TXFE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TXFE (const USART_TypeDef * USARTx)

Function description

Check if the USART TX FIFO Empty Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR TXFE LL_USART_IsActiveFlag_TXFE

LL_USART_IsActiveFlag_RXFF

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RXFF (const USART_TypeDef * USARTx)

Function description

Check if the USART RX FIFO Full Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR RXFF LL_USART_IsActiveFlag_RXFF

LL_USART_IsActiveFlag_TCBGT

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TCBGT (const USART_TypeDef * USARTx)

Function description

Check if the Smartcard Transmission Complete Before Guard Time Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCBGT LL_USART_IsActiveFlag_TCBGT

LL_USART_IsActiveFlag_TXFT

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TXFT (const USART_TypeDef * USARTx)

Function description

Check if the USART TX FIFO Threshold Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR TXFT LL_USART_IsActiveFlag_TXFT

LL_USART_IsActiveFlag_RXFT

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RXFT (const USART_TypeDef * USARTx)

Function description

Check if the USART RX FIFO Threshold Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR RXFT LL_USART_IsActiveFlag_RXFT

LL_USART_ClearFlag_PE

Function name

__STATIC_INLINE void LL_USART_ClearFlag_PE (USART_TypeDef * USARTx)

Function description

Clear Parity Error Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR PECF LL_USART_ClearFlag_PE

LL_USART_ClearFlag_FE

Function name

__STATIC_INLINE void LL_USART_ClearFlag_FE (USART_TypeDef * USARTx)

Function description

Clear Framing Error Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR FECF LL_USART_ClearFlag_FE

LL_USART_ClearFlag_NE

Function name

__STATIC_INLINE void LL_USART_ClearFlag_NE (USART_TypeDef * USARTx)

Function description

Clear Noise Error detected Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR NECF LL_USART_ClearFlag_NE

LL_USART_ClearFlag_ORE

Function name

__STATIC_INLINE void LL_USART_ClearFlag_ORE (USART_TypeDef * USARTx)

Function description

Clear OverRun Error Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ORECF LL_USART_ClearFlag_ORE

LL_USART_ClearFlag_IDLE

Function name

__STATIC_INLINE void LL_USART_ClearFlag_IDLE (USART_TypeDef * USARTx)

Function description

Clear IDLE line detected Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR IDLECF LL_USART_ClearFlag_IDLE

LL_USART_ClearFlag_TXFE

Function name

__STATIC_INLINE void LL_USART_ClearFlag_TXFE (USART_TypeDef * USARTx)

Function description

Clear TX FIFO Empty Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ICR TXFECF LL_USART_ClearFlag_TXFE

LL_USART_ClearFlag_TC

Function name

__STATIC_INLINE void LL_USART_ClearFlag_TC (USART_TypeDef * USARTx)

Function description

Clear Transmission Complete Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TCCF LL_USART_ClearFlag_TC

LL_USART_ClearFlag_TCBGT

Function name

__STATIC_INLINE void LL_USART_ClearFlag_TCBGT (USART_TypeDef * USARTx)

Function description

Clear Smartcard Transmission Complete Before Guard Time Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TCBGTCF LL_USART_ClearFlag_TCBGT

LL_USART_ClearFlag_LBD

Function name

__STATIC_INLINE void LL_USART_ClearFlag_LBD (USART_TypeDef * USARTx)

Function description

Clear LIN Break Detection Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ICR LBDCF LL_USART_ClearFlag_LBD

LL_USART_ClearFlag_nCTS

Function name

__STATIC_INLINE void LL_USART_ClearFlag_nCTS (USART_TypeDef * USARTx)

Function description

Clear CTS Interrupt Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ICR CTSCF LL_USART_ClearFlag_nCTS

LL_USART_ClearFlag_RTO

Function name

__STATIC_INLINE void LL_USART_ClearFlag_RTO (USART_TypeDef * USARTx)

Function description

Clear Receiver Time Out Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR RTOCF LL_USART_ClearFlag_RTO

LL_USART_ClearFlag_EOB

Function name

__STATIC_INLINE void LL_USART_ClearFlag_EOB (USART_TypeDef * USARTx)

Function description

Clear End Of Block Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ICR EOBCF LL_USART_ClearFlag_EOB

LL_USART_ClearFlag_UDR

Function name

__STATIC_INLINE void LL_USART_ClearFlag_UDR (USART_TypeDef * USARTx)

Function description

Clear SPI Slave Underrun Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ICR UDRCF LL_USART_ClearFlag_UDR

LL_USART_ClearFlag_CM

Function name

__STATIC_INLINE void LL_USART_ClearFlag_CM (USART_TypeDef * USARTx)

Function description

Clear Character Match Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CMCF LL_USART_ClearFlag_CM

LL_USART_ClearFlag_WKUP

Function name

__STATIC_INLINE void LL_USART_ClearFlag_WKUP (USART_TypeDef * USARTx)

Function description

Clear Wake Up from stop mode Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ICR WUCF LL_USART_ClearFlag_WKUP

LL_USART_EnableIT_IDLE

Function name

__STATIC_INLINE void LL_USART_EnableIT_IDLE (USART_TypeDef * USARTx)

Function description

Enable IDLE Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_USART_EnableIT_IDLE

LL_USART_EnableIT_RXNE_RXFNE

Function name

__STATIC_INLINE void LL_USART_EnableIT_RXNE_RXFNE (USART_TypeDef * USARTx)

Function description

Enable RX Not Empty and RX FIFO Not Empty Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_USART_EnableIT_RXNE_RXFNE

LL_USART_EnableIT_TC

Function name

__STATIC_INLINE void LL_USART_EnableIT_TC (USART_TypeDef * USARTx)

Function description

Enable Transmission Complete Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TCIE LL_USART_EnableIT_TC

LL_USART_EnableIT_TXE_TXFNF

Function name

__STATIC_INLINE void LL_USART_EnableIT_TXE_TXFNF (USART_TypeDef * USARTx)

Function description

Enable TX Empty and TX FIFO Not Full Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_USART_EnableIT_TXE_TXFNF

LL_USART_EnableIT_PE

Function name

__STATIC_INLINE void LL_USART_EnableIT_PE (USART_TypeDef * USARTx)

Function description

Enable Parity Error Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 PEIE LL_USART_EnableIT_PE

LL_USART_EnableIT_CM

Function name

__STATIC_INLINE void LL_USART_EnableIT_CM (USART_TypeDef * USARTx)

Function description

Enable Character Match Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CMIE LL_USART_EnableIT_CM

LL_USART_EnableIT_RTO

Function name

__STATIC_INLINE void LL_USART_EnableIT_RTO (USART_TypeDef * USARTx)

Function description

Enable Receiver Timeout Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RTOIE LL_USART_EnableIT_RTO

LL_USART_EnableIT_EOB

Function name

__STATIC_INLINE void LL_USART_EnableIT_EOB (USART_TypeDef * USARTx)

Function description

Enable End Of Block Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 EOBIE LL_USART_EnableIT_EOB

LL_USART_EnableIT_TXFE

Function name

__STATIC_INLINE void LL_USART_EnableIT_TXFE (USART_TypeDef * USARTx)

Function description

Enable TX FIFO Empty Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_USART_EnableIT_TXFE

LL_USART_EnableIT_RXFF

Function name

__STATIC_INLINE void LL_USART_EnableIT_RXFF (USART_TypeDef * USARTx)

Function description

Enable RX FIFO Full Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_USART_EnableIT_RXFF

LL_USART_EnableIT_LBD

Function name

__STATIC_INLINE void LL_USART_EnableIT_LBD (USART_TypeDef * USARTx)

Function description

Enable LIN Break Detection Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBDIE LL_USART_EnableIT_LBD

LL_USART_EnableIT_ERROR

Function name

__STATIC_INLINE void LL_USART_EnableIT_ERROR (USART_TypeDef * USARTx)

Function description

Enable Error Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the USARTx_ISR register). 0: Interrupt is inhibited 1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the USARTx_ISR register.

Reference Manual to LL API cross reference:

- CR3 EIE LL_USART_EnableIT_ERROR

LL_USART_EnableIT_CTS

Function name

__STATIC_INLINE void LL_USART_EnableIT_CTS (USART_TypeDef * USARTx)

Function description

Enable CTS Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_USART_EnableIT_CTS

LL_USART_EnableIT_WKUP

Function name

__STATIC_INLINE void LL_USART_EnableIT_WKUP (USART_TypeDef * USARTx)

Function description

Enable Wake Up from Stop Mode Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 WUFIE LL_USART_EnableIT_WKUP

LL_USART_EnableIT_TXFT

Function name

__STATIC_INLINE void LL_USART_EnableIT_TXFT (USART_TypeDef * USARTx)

Function description

Enable TX FIFO Threshold Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_USART_EnableIT_TXFT

LL_USART_EnableIT_TCBGT

Function name

__STATIC_INLINE void LL_USART_EnableIT_TCBGT (USART_TypeDef * USARTx)

Function description

Enable Smartcard Transmission Complete Before Guard Time Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TCBGTIE LL_USART_EnableIT_TCBGT

LL_USART_EnableIT_RXFT

Function name

__STATIC_INLINE void LL_USART_EnableIT_RXFT (USART_TypeDef * USARTx)

Function description

Enable RX FIFO Threshold Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_USART_EnableIT_RXFT

LL_USART_DisableIT_IDLE

Function name

__STATIC_INLINE void LL_USART_DisableIT_IDLE (USART_TypeDef * USARTx)

Function description

Disable IDLE Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_USART_DisableIT_IDLE

LL_USART_DisableIT_RXNE_RXFNE

Function name

__STATIC_INLINE void LL_USART_DisableIT_RXNE_RXFNE (USART_TypeDef * USARTx)

Function description

Disable RX Not Empty and RX FIFO Not Empty Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_USART_DisableIT_RXNE_RXFNE

LL_USART_DisableIT_TC

Function name

__STATIC_INLINE void LL_USART_DisableIT_TC (USART_TypeDef * USARTx)

Function description

Disable Transmission Complete Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TCIE LL_USART_DisableIT_TC

LL_USART_DisableIT_TXE_TXFNF

Function name

__STATIC_INLINE void LL_USART_DisableIT_TXE_TXFNF (USART_TypeDef * USARTx)

Function description

Disable TX Empty and TX FIFO Not Full Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_USART_DisableIT_TXE_TXFNF

LL_USART_DisableIT_PE

Function name

__STATIC_INLINE void LL_USART_DisableIT_PE (USART_TypeDef * USARTx)

Function description

Disable Parity Error Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 PEIE LL_USART_DisableIT_PE

LL_USART_DisableIT_CM

Function name

__STATIC_INLINE void LL_USART_DisableIT_CM (USART_TypeDef * USARTx)

Function description

Disable Character Match Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CMIE LL_USART_DisableIT_CM

LL_USART_DisableIT_RTO

Function name

__STATIC_INLINE void LL_USART_DisableIT_RTO (USART_TypeDef * USARTx)

Function description

Disable Receiver Timeout Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RTOIE LL_USART_DisableIT_RTO

LL_USART_DisableIT_EOB

Function name

__STATIC_INLINE void LL_USART_DisableIT_EOB (USART_TypeDef * USARTx)

Function description

Disable End Of Block Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 EOBIE LL_USART_DisableIT_EOB

LL_USART_DisableIT_TXFE

Function name

__STATIC_INLINE void LL_USART_DisableIT_TXFE (USART_TypeDef * USARTx)

Function description

Disable TX FIFO Empty Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_USART_DisableIT_TXFE

LL_USART_DisableIT_RXFF

Function name

__STATIC_INLINE void LL_USART_DisableIT_RXFF (USART_TypeDef * USARTx)

Function description

Disable RX FIFO Full Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_USART_DisableIT_RXFF

LL_USART_DisableIT_LBD

Function name

__STATIC_INLINE void LL_USART_DisableIT_LBD (USART_TypeDef * USARTx)

Function description

Disable LIN Break Detection Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBDIE LL_USART_DisableIT_LBD

LL_USART_DisableIT_ERROR

Function name

__STATIC_INLINE void LL_USART_DisableIT_ERROR (USART_TypeDef * USARTx)

Function description

Disable Error Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the USARTx_ISR register). 0: Interrupt is inhibited 1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the USARTx_ISR register.

Reference Manual to LL API cross reference:

- CR3 EIE LL_USART_DisableIT_ERROR

LL_USART_DisableIT_CTS

Function name

__STATIC_INLINE void LL_USART_DisableIT_CTS (USART_TypeDef * USARTx)

Function description

Disable CTS Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_USART_DisableIT_CTS

LL_USART_DisableIT_WKUP

Function name

__STATIC_INLINE void LL_USART_DisableIT_WKUP (USART_TypeDef * USARTx)

Function description

Disable Wake Up from Stop Mode Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 WUFIE LL_USART_DisableIT_WKUP

LL_USART_DisableIT_TXFT

Function name

__STATIC_INLINE void LL_USART_DisableIT_TXFT (USART_TypeDef * USARTx)

Function description

Disable TX FIFO Threshold Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_USART_DisableIT_TXFT

LL_USART_DisableIT_TCBGT

Function name

__STATIC_INLINE void LL_USART_DisableIT_TCBGT (USART_TypeDef * USARTx)

Function description

Disable Smartcard Transmission Complete Before Guard Time Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TCBGTIE LL_USART_DisableIT_TCBGT

LL_USART_DisableIT_RXFT

Function name

__STATIC_INLINE void LL_USART_DisableIT_RXFT (USART_TypeDef * USARTx)

Function description

Disable RX FIFO Threshold Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_USART_DisableIT_RXFT

LL_USART_IsEnabledIT_IDLE

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_IDLE (const USART_TypeDef * USARTx)

Function description

Check if the USART IDLE Interrupt source is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_USART_IsEnabledIT_IDLE

LL_USART_IsEnabledIT_RXNE_RXFNE

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RXNE_RXFNE (const USART_TypeDef * USARTx)

Function description

Check if the USART RX Not Empty and USART RX FIFO Not Empty Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_USART_IsEnabledIT_RXNE_RXFNE

LL_USART_IsEnabledIT_TC

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TC (const USART_TypeDef * USARTx)

Function description

Check if the USART Transmission Complete Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TCIE LL_USART_IsEnabledIT_TC

LL_USART_IsEnabledIT_TXE_TXFNF

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TXE_TXFNF (const USART_TypeDef * USARTx)

Function description

Check if the USART TX Empty and USART TX FIFO Not Full Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_USART_IsEnabledIT_TXE_TXFNF

LL_USART_IsEnabledIT_PE

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_PE (const USART_TypeDef * USARTx)

Function description

Check if the USART Parity Error Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 PEIE LL_USART_IsEnabledIT_PE

LL_USART_IsEnabledIT_CM

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_CM (const USART_TypeDef * USARTx)

Function description

Check if the USART Character Match Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 CMIE LL_USART_IsEnabledIT_CM

LL_USART_IsEnabledIT_RTO

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RTO (const USART_TypeDef * USARTx)

Function description

Check if the USART Receiver Timeout Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 RTOIE LL_USART_IsEnabledIT_RTO

LL_USART_IsEnabledIT_EOB

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_EOB (const USART_TypeDef * USARTx)

Function description

Check if the USART End Of Block Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 EOBIE LL_USART_IsEnabledIT_EOB

LL_USART_IsEnabledIT_TXFE

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TXFE (const USART_TypeDef * USARTx)

Function description

Check if the USART TX FIFO Empty Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_USART_IsEnabledIT_TXFE

LL_USART_IsEnabledIT_RXFF

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RXFF (const USART_TypeDef * USARTx)

Function description

Check if the USART RX FIFO Full Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_USART_IsEnabledIT_RXFF

LL_USART_IsEnabledIT_LBD

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_LBD (const USART_TypeDef * USARTx)

Function description

Check if the USART LIN Break Detection Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBDIE LL_USART_IsEnabledIT_LBD

LL_USART_IsEnabledIT_ERROR

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_ERROR (const USART_TypeDef * USARTx)

Function description

Check if the USART Error Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 EIE LL_USART_IsEnabledIT_ERROR

LL_USART_IsEnabledIT_CTS

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_CTS (const USART_TypeDef * USARTx)

Function description

Check if the USART CTS Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_USART_IsEnabledIT_CTS

LL_USART_IsEnabledIT_WKUP

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_WKUP (const USART_TypeDef * USARTx)

Function description

Check if the USART Wake Up from Stop Mode Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 WUFIE LL_USART_IsEnabledIT_WKUP

LL_USART_IsEnabledIT_TXFT

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TXFT (const USART_TypeDef * USARTx)

Function description

Check if USART TX FIFO Threshold Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_USART_IsEnabledIT_TXFT

LL_USART_IsEnabledIT_TCBGT

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TCBGT (const USART_TypeDef * USARTx)

Function description

Check if the Smartcard Transmission Complete Before Guard Time Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TCBGTIE LL_USART_IsEnabledIT_TCBGT

LL_USART_IsEnabledIT_RXFT

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RXFT (const USART_TypeDef * USARTx)

Function description

Check if USART RX FIFO Threshold Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_USART_IsEnabledIT_RXFT

LL_USART_EnableDMAReq_RX

Function name

__STATIC_INLINE void LL_USART_EnableDMAReq_RX (USART_TypeDef * USARTx)

Function description

Enable DMA Mode for reception.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAR LL_USART_EnableDMAReq_RX

LL_USART_DisableDMAReq_RX

Function name

__STATIC_INLINE void LL_USART_DisableDMAReq_RX (USART_TypeDef * USARTx)

Function description

Disable DMA Mode for reception.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAR LL_USART_DisableDMAReq_RX

LL_USART_IsEnabledDMAReq_RX

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledDMAReq_RX (const USART_TypeDef * USARTx)

Function description

Check if DMA Mode is enabled for reception.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DMAR LL_USART_IsEnabledDMAReq_RX

LL_USART_EnableDMAReq_TX

Function name

__STATIC_INLINE void LL_USART_EnableDMAReq_TX (USART_TypeDef * USARTx)

Function description

Enable DMA Mode for transmission.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAT LL_USART_EnableDMAReq_TX

LL_USART_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_USART_DisableDMAReq_TX (USART_TypeDef * USARTx)

Function description

Disable DMA Mode for transmission.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAT LL_USART_DisableDMAReq_TX

LL_USART_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledDMAReq_TX (const USART_TypeDef * USARTx)

Function description

Check if DMA Mode is enabled for transmission.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DMAT LL_USART_IsEnabledDMAReq_TX

LL_USART_EnableDMADeactOnRxErr

Function name

__STATIC_INLINE void LL_USART_EnableDMADeactOnRxErr (USART_TypeDef * USARTx)

Function description

Enable DMA Disabling on Reception Error.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DDRE LL_USART_EnableDMADeactOnRxErr

LL_USART_DisableDMADeactOnRxErr

Function name

__STATIC_INLINE void LL_USART_DisableDMADeactOnRxErr (USART_TypeDef * USARTx)

Function description

Disable DMA Disabling on Reception Error.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DDRE LL_USART_DisableDMADeactOnRxErr

LL_USART_IsEnabledDMADeactOnRxErr

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledDMADeactOnRxErr (const USART_TypeDef * USARTx)

Function description

Indicate if DMA Disabling on Reception Error is disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DDRE LL_USART_IsEnabledDMADeactOnRxErr

LL_USART_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_USART_DMA_GetRegAddr (const USART_TypeDef * USARTx, uint32_t Direction)

Function description

Get the data register address used for DMA transfer.

Parameters

- **USARTx:** USART Instance
- **Direction:** This parameter can be one of the following values:
  - LL_USART_DMA_REG_DATA_TRANSMIT
  - LL_USART_DMA_REG_DATA_RECEIVE

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- RDR RDR LL_USART_DMA_GetRegAddr
-
- TDR TDR LL_USART_DMA_GetRegAddr

LL_USART_ReceiveData8

Function name

__STATIC_INLINE uint8_t LL_USART_ReceiveData8 (const USART_TypeDef * USARTx)

Function description

Read Receiver Data register (Receive Data value, 8 bits)

Parameters

- **USARTx:** USART Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- RDR RDR LL_USART_ReceiveData8

LL_USART_ReceiveData9

Function name

__STATIC_INLINE uint16_t LL_USART_ReceiveData9 (const USART_TypeDef * USARTx)

Function description

Read Receiver Data register (Receive Data value, 9 bits)

Parameters

- **USARTx:** USART Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x1FF

Reference Manual to LL API cross reference:

- RDR RDR LL_USART_ReceiveData9

LL_USART_TransmitData8

Function name

__STATIC_INLINE void LL_USART_TransmitData8 (USART_TypeDef * USARTx, uint8_t Value)

Function description

Write in Transmitter Data Register (Transmit Data value, 8 bits)

Parameters

- **USARTx:** USART Instance
- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TDR TDR LL_USART_TransmitData8

LL_USART_TransmitData9

Function name

__STATIC_INLINE void LL_USART_TransmitData9 (USART_TypeDef * USARTx, uint16_t Value)

Function description

Write in Transmitter Data Register (Transmit Data value, 9 bits)

Parameters

- **USARTx:** USART Instance
- **Value:** between Min_Data=0x00 and Max_Data=0x1FF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TDR TDR LL_USART_TransmitData9

LL_USART_RequestAutoBaudRate

Function name

__STATIC_INLINE void LL_USART_RequestAutoBaudRate (USART_TypeDef * USARTx)

Function description

Request an Automatic Baud Rate measurement on next received data frame.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- RQR ABRRQ LL_USART_RequestAutoBaudRate

LL_USART_RequestBreakSending

Function name

__STATIC_INLINE void LL_USART_RequestBreakSending (USART_TypeDef * USARTx)

Function description

Request Break sending.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RQR SBKRQ LL_USART_RequestBreakSending

LL_USART_RequestEnterMuteMode

Function name

__STATIC_INLINE void LL_USART_RequestEnterMuteMode (USART_TypeDef * USARTx)

Function description

Put USART in mute mode and set the RWU flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RQR MMRQ LL_USART_RequestEnterMuteMode

LL_USART_RequestRxDataFlush

Function name

__STATIC_INLINE void LL_USART_RequestRxDataFlush (USART_TypeDef * USARTx)

Function description

Request a Receive Data and FIFO flush.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
- Allows to discard the received data without reading them, and avoid an overrun condition.

Reference Manual to LL API cross reference:

- RQR RXFRQ LL_USART_RequestRxDataFlush

LL_USART_RequestTxDataFlush

Function name

__STATIC_INLINE void LL_USART_RequestTxDataFlush (USART_TypeDef * USARTx)

Function description

Request a Transmit data and FIFO flush.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- RQR TXFRQ LL_USART_RequestTxDataFlush

LL_USART_DeInit

Function name

ErrorStatus LL_USART_DeInit (const USART_TypeDef * USARTx)

Function description

De-initialize USART registers (Registers restored to their default values).

Parameters

- **USARTx:** USART Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: USART registers are de-initialized
  - ERROR: USART registers are not de-initialized

LL_USART_Init

Function name

ErrorStatus LL_USART_Init (USART_TypeDef * USARTx, const LL_USART_InitTypeDef * USART_InitStruct)

Function description

Initialize USART registers according to the specified parameters in USART_InitStruct.

Parameters

- **USARTx:** USART Instance
- **USART_InitStruct:** pointer to a LL_USART_InitTypeDef structure that contains the configuration information for the specified USART peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: USART registers are initialized according to USART_InitStruct content
  - ERROR: Problem occurred during USART Registers initialization

Notes

- As some bits in USART configuration registers can only be written when the USART is disabled (USART_CR1_UE bit =0), USART Peripheral should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.
- Baud rate value stored in USART_InitStruct BaudRate field, should be valid (different from 0).

LL_USART_StructInit

Function name

void LL_USART_StructInit (LL_USART_InitTypeDef * USART_InitStruct)

Function description

Set each LL_USART_InitTypeDef field to default value.

Parameters

- **USART_InitStruct:** pointer to a LL_USART_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_USART_ClockInit

Function name

ErrorStatus LL_USART_ClockInit (USART_TypeDef * USARTx, const LL_USART_ClockInitTypeDef * USART_ClockInitStruct)

Function description

Initialize USART Clock related settings according to the specified parameters in the USART_ClockInitStruct.

Parameters

- **USARTx:** USART Instance
- **USART_ClockInitStruct:** pointer to a LL_USART_ClockInitTypeDef structure that contains the Clock configuration information for the specified USART peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: USART registers related to Clock settings are initialized according to USART_ClockInitStruct content
  - ERROR: Problem occurred during USART Registers initialization

Notes

- As some bits in USART configuration registers can only be written when the USART is disabled (USART_CR1_UE bit =0), USART Peripheral should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.

LL_USART_ClockStructInit

Function name

void LL_USART_ClockStructInit (LL_USART_ClockInitTypeDef * USART_ClockInitStruct)

Function description

Set each field of a LL_USART_ClockInitTypeDef type structure to default value.

Parameters

- **USART_ClockInitStruct:** pointer to a LL_USART_ClockInitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

127.3 USART Firmware driver defines

The following section lists the various define and macros of the module.

127.3.1 USART

USART

Address Length Detection

LL_USART_ADDRESS_DETECT_4B

4-bit address detection method selected

LL_USART_ADDRESS_DETECT_7B

7-bit address detection (in 8-bit data mode) method selected

Autobaud Detection

LL_USART_AUTOBAUD_DETECT_ON_STARTBIT

Measurement of the start bit is used to detect the baud rate

LL_USART_AUTOBAUD_DETECT_ON_FALLINGEDGE

Falling edge to falling edge measurement. Received frame must start with a single bit = 1 -> Frame = Start10xxxxxx

LL_USART_AUTOBAUD_DETECT_ON_7F_FRAME

0x7F frame detection

LL_USART_AUTOBAUD_DETECT_ON_55_FRAME

0x55 frame detection

Binary Data Inversion

LL_USART_BINARY_LOGIC_POSITIVE

Logical data from the data register are send/received in positive/direct logic. (1=H, 0=L)

LL_USART_BINARY_LOGIC_NEGATIVE

Logical data from the data register are send/received in negative/inverse logic. (1=L, 0=H). The parity bit is also inverted.

Bit Order

LL_USART_BITORDER_LSBFIRST

data is transmitted/received with data bit 0 first, following the start bit

LL_USART_BITORDER_MSBFIRST

data is transmitted/received with the MSB first, following the start bit

Clear Flags Defines

LL_USART_ICR_PECF

Parity error clear flag

LL_USART_ICR_FECF

Framing error clear flag

LL_USART_ICR_NECF

Noise error detected clear flag

LL_USART_ICR_ORECF

Overrun error clear flag

LL_USART_ICR_IDLECF

Idle line detected clear flag

LL_USART_ICR_TXFECF

TX FIFO Empty clear flag

LL_USART_ICR_TCCF

Transmission complete clear flag

LL_USART_ICR_TCBGTCF

Transmission completed before guard time clear flag

LL_USART_ICR_LBDCF

LIN break detection clear flag

LL_USART_ICR_CTSCF

CTS clear flag

LL_USART_ICR_RTOCF

Receiver timeout clear flag

LL_USART_ICR_EOBCF

End of block clear flag

LL_USART_ICR_UDRCF

SPI Slave Underrun clear flag

LL_USART_ICR_CMCF

Character match clear flag

LL_USART_ICR_WUCF

Wakeup from Stop mode clear flag

Clock Signal

LL_USART_CLOCK_DISABLE

Clock signal not provided

LL_USART_CLOCK_ENABLE

Clock signal provided

Datawidth

LL_USART_DATAWIDTH_7B

7 bits word length : Start bit, 7 data bits, n stop bits

LL_USART_DATAWIDTH_8B

8 bits word length : Start bit, 8 data bits, n stop bits

LL_USART_DATAWIDTH_9B

9 bits word length : Start bit, 9 data bits, n stop bits

Driver Enable Polarity

LL_USART_DE_POLARITY_HIGH

DE signal is active high

LL_USART_DE_POLARITY_LOW

DE signal is active low

Communication Direction

LL_USART_DIRECTION_NONE

Transmitter and Receiver are disabled

LL_USART_DIRECTION_RX

Transmitter is disabled and Receiver is enabled

LL_USART_DIRECTION_TX

Transmitter is enabled and Receiver is disabled

LL_USART_DIRECTION_TX_RX

Transmitter and Receiver are enabled

DMA Register Data

LL_USART_DMA_REG_DATA_TRANSMIT

Get address of data register used for transmission

LL_USART_DMA_REG_DATA_RECEIVE

Get address of data register used for reception

FIFO Threshold

LL_USART_FIFOTHRESHOLD_1_8

FIFO reaches 1/8 of its depth

LL_USART_FIFOTHRESHOLD_1_4

FIFO reaches 1/4 of its depth

LL_USART_FIFOTHRESHOLD_1_2

FIFO reaches 1/2 of its depth

LL_USART_FIFOTHRESHOLD_3_4

FIFO reaches 3/4 of its depth

LL_USART_FIFOTHRESHOLD_7_8

FIFO reaches 7/8 of its depth

LL_USART_FIFOTHRESHOLD_8_8

FIFO becomes empty for TX and full for RX

Get Flags Defines

LL_USART_ISR_PE

Parity error flag

LL_USART_ISR_FE

Framing error flag

LL_USART_ISR_NE

Noise detected flag

LL_USART_ISR_ORE

Overrun error flag

LL_USART_ISR_IDLE

Idle line detected flag

LL_USART_ISR_RXNE_RXFNE

Read data register or RX FIFO not empty flag

LL_USART_ISR_TC

Transmission complete flag

LL_USART_ISR_TXE_TXFNF

Transmit data register empty or TX FIFO Not Full flag

LL_USART_ISR_LBDF

LIN break detection flag

LL_USART_ISR_CTSIF

CTS interrupt flag

LL_USART_ISR_CTS

CTS flag

LL_USART_ISR_RTOF

Receiver timeout flag

LL_USART_ISR_EOBF

End of block flag

LL_USART_ISR_UDR

SPI Slave underrun error flag

LL_USART_ISR_ABRE

Auto baud rate error flag

LL_USART_ISR_ABRF

Auto baud rate flag

LL_USART_ISR_BUSY

Busy flag

LL_USART_ISR_CMF

Character match flag

LL_USART_ISR_SBKF

Send break flag

LL_USART_ISR_RWU

Receiver wakeup from Mute mode flag

LL_USART_ISR_WUF

Wakeup from Stop mode flag

LL_USART_ISR_TEACK

Transmit enable acknowledge flag

LL_USART_ISR_REACK

Receive enable acknowledge flag

LL_USART_ISR_TXFE

TX FIFO empty flag

LL_USART_ISR_RXFF

RX FIFO full flag

LL_USART_ISR_TCBGT

Transmission complete before guard time completion flag

LL_USART_ISR_RXFT

RX FIFO threshold flag

LL_USART_ISR_TXFT

TX FIFO threshold flag

Hardware Control

LL_USART_HWCONTROL_NONE

CTS and RTS hardware flow control disabled

LL_USART_HWCONTROL_RTS

RTS output enabled, data is only requested when there is space in the receive buffer

LL_USART_HWCONTROL_CTS

CTS mode enabled, data is only transmitted when the nCTS input is asserted (tied to 0)

LL_USART_HWCONTROL_RTS_CTS

CTS and RTS hardware flow control enabled

IrDA Power

LL_USART_IRDA_POWER_NORMAL

IrDA normal power mode

LL_USART_IRDA_POWER_LOW

IrDA low power mode

IT Defines

LL_USART_CR1_IDLEIE

IDLE interrupt enable

LL_USART_CR1_RXNEIE_RXFNEIE

Read data register and RXFIFO not empty interrupt enable

LL_USART_CR1_TCIE

Transmission complete interrupt enable

LL_USART_CR1_TXEIE_TXFNFIE

Transmit data register empty and TX FIFO not full interrupt enable

LL_USART_CR1_PEIE

Parity error

LL_USART_CR1_CMIE

Character match interrupt enable

LL_USART_CR1_RTOIE

Receiver timeout interrupt enable

LL_USART_CR1_EOBIE

End of Block interrupt enable

LL_USART_CR1_TXFEIE

TX FIFO empty interrupt enable

LL_USART_CR1_RXFFIE

RX FIFO full interrupt enable

LL_USART_CR2_LBDIE

LIN break detection interrupt enable

LL_USART_CR3_EIE

Error interrupt enable

LL_USART_CR3_CTSIE

CTS interrupt enable

LL_USART_CR3_WUFIE

Wakeup from Stop mode interrupt enable

LL_USART_CR3_TXFTIE

TX FIFO threshold interrupt enable

LL_USART_CR3_TCBGTIE

Transmission complete before guard time interrupt enable

LL_USART_CR3_RXFTIE

RX FIFO threshold interrupt enable

Last Clock Pulse

LL_USART_LASTCLKPULSE_NO_OUTPUT

The clock pulse of the last data bit is not output to the SCLK pin

LL_USART_LASTCLKPULSE_OUTPUT

The clock pulse of the last data bit is output to the SCLK pin

LIN Break Detection Length

LL_USART_LINBREAK_DETECT_10B

10-bit break detection method selected

LL_USART_LINBREAK_DETECT_11B

11-bit break detection method selected

Oversampling

LL_USART_OVERSAMPLING_16

Oversampling by 16

LL_USART_OVERSAMPLING_8

Oversampling by 8

Parity Control

LL_USART_PARITY_NONE

Parity control disabled

LL_USART_PARITY_EVEN

Parity control enabled and Even Parity is selected

LL_USART_PARITY_ODD

Parity control enabled and Odd Parity is selected

Clock Phase

LL_USART_PHASE_1EDGE

The first clock transition is the first data capture edge

LL_USART_PHASE_2EDGE

The second clock transition is the first data capture edge

Clock Polarity

LL_USART_POLARITY_LOW

Steady low value on SCLK pin outside transmission window

LL_USART_POLARITY_HIGH

Steady high value on SCLK pin outside transmission window

Clock Source Prescaler

LL_USART_PRESCALER_DIV1

Input clock not divided

LL_USART_PRESCALER_DIV2

Input clock divided by 2

LL_USART_PRESCALER_DIV4

Input clock divided by 4

LL_USART_PRESCALER_DIV6

Input clock divided by 6

LL_USART_PRESCALER_DIV8

Input clock divided by 8

LL_USART_PRESCALER_DIV10

Input clock divided by 10

LL_USART_PRESCALER_DIV12

Input clock divided by 12

LL_USART_PRESCALER_DIV16

Input clock divided by 16

LL_USART_PRESCALER_DIV32

Input clock divided by 32

LL_USART_PRESCALER_DIV64

Input clock divided by 64

LL_USART_PRESCALER_DIV128

Input clock divided by 128

LL_USART_PRESCALER_DIV256

Input clock divided by 256

RX Pin Active Level Inversion

LL_USART_RXPIN_LEVEL_STANDARD

RX pin signal works using the standard logic levels

LL_USART_RXPIN_LEVEL_INVERTED

RX pin signal values are inverted.

Stop Bits

LL_USART_STOPBITS_0_5

0.5 stop bit

LL_USART_STOPBITS_1

1 stop bit

LL_USART_STOPBITS_1_5

1.5 stop bits

LL_USART_STOPBITS_2

2 stop bits

TX Pin Active Level Inversion

LL_USART_TXPIN_LEVEL_STANDARD

TX pin signal works using the standard logic levels

LL_USART_TXPIN_LEVEL_INVERTED

TX pin signal values are inverted.

TX RX Pins Swap

LL_USART_TXRX_STANDARD

TX/RX pins are used as defined in standard pinout

LL_USART_TXRX_SWAPPED

TX and RX pins functions are swapped.

Wakeup

LL_USART_WAKEUP_IDLELINE

USART wake up from Mute mode on Idle Line

LL_USART_WAKEUP_ADDRESSMARK

USART wake up from Mute mode on Address Mark

Wakeup Activation

LL_USART_WAKEUP_ON_ADDRESS

Wake up active on address match

LL_USART_WAKEUP_ON_STARTBIT

Wake up active on Start bit detection

LL_USART_WAKEUP_ON_RXNE

Wake up active on RXNE

FLAG_Management

LL_USART_IsActiveFlag_RXNE

LL_USART_IsActiveFlag_TXE

IT_Management

LL_USART_EnableIT_RXNE

LL_USART_EnableIT_TXE

LL_USART_DisableIT_RXNE

LL_USART_DisableIT_TXE

LL_USART_IsEnabledIT_RXNE

LL_USART_IsEnabledIT_TXE

Exported_Macros_Helper

__LL_USART_DIV_SAMPLING8

Description:

- Compute USARTDIV value according to Peripheral Clock and expected Baud Rate in 8 bits sampling mode (32 bits value of USARTDIV is returned)

Parameters:

- __PERIPHCLK__: Peripheral Clock frequency used for USART instance
- __PRESCALER__: This parameter can be one of the following values:
  - LL_USART_PRESCALER_DIV1
  - LL_USART_PRESCALER_DIV2
  - LL_USART_PRESCALER_DIV4
  - LL_USART_PRESCALER_DIV6
  - LL_USART_PRESCALER_DIV8
  - LL_USART_PRESCALER_DIV10
  - LL_USART_PRESCALER_DIV12
  - LL_USART_PRESCALER_DIV16
  - LL_USART_PRESCALER_DIV32
  - LL_USART_PRESCALER_DIV64
  - LL_USART_PRESCALER_DIV128
  - LL_USART_PRESCALER_DIV256
- __BAUDRATE__: Baud rate value to achieve

Return value:

- USARTDIV: value to be used for BRR register filling in OverSampling_8 case

__LL_USART_DIV_SAMPLING16

Description:

- Compute USARTDIV value according to Peripheral Clock and expected Baud Rate in 16 bits sampling mode (32 bits value of USARTDIV is returned)

Parameters:

- __PERIPHCLK__: Peripheral Clock frequency used for USART instance
- __PRESCALER__: This parameter can be one of the following values:
  - LL_USART_PRESCALER_DIV1
  - LL_USART_PRESCALER_DIV2
  - LL_USART_PRESCALER_DIV4
  - LL_USART_PRESCALER_DIV6
  - LL_USART_PRESCALER_DIV8
  - LL_USART_PRESCALER_DIV10
  - LL_USART_PRESCALER_DIV12
  - LL_USART_PRESCALER_DIV16
  - LL_USART_PRESCALER_DIV32
  - LL_USART_PRESCALER_DIV64
  - LL_USART_PRESCALER_DIV128
  - LL_USART_PRESCALER_DIV256
- __BAUDRATE__: Baud rate value to achieve

Return value:

- USARTDIV: value to be used for BRR register filling in OverSampling_16 case

Common Write and read registers Macros

LL_USART_WriteReg

Description:

- Write a value in USART register.

Parameters:

- __INSTANCE__: USART Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_USART_ReadReg

Description:

- Read a value in USART register.

Parameters:

- __INSTANCE__: USART Instance
- __REG__: Register to be read

Return value:

- Register: value

128 LL UTILS Generic Driver

128.1 UTILS Firmware driver registers structures

128.1.1 LL_UTILS_PLLInitTypeDef

LL_UTILS_PLLInitTypeDef is defined in the stm32h7xx_ll_utils.h

Data Fields

- ***uint32_t PLLM***
- ***uint32_t PLLN***
- ***uint32_t PLLP***
- ***uint32_t FRACN***
- ***uint32_t VCO_Input***
- ***uint32_t VCO_Output***

Field Documentation

- ***uint32_t LL_UTILS_PLLInitTypeDef::PLLM***

  Division factor for PLL VCO input clock. This parameter must be a number between Min_Data = 0 and Max_Data = 63This feature can be modified afterwards using unitary function **LL_RCC_PLL1_SetM()**.
- ***uint32_t LL_UTILS_PLLInitTypeDef::PLLN***

  Multiplication factor for PLL VCO output clock. This parameter must be a number between Min_Data = 4 and Max_Data = 512This feature can be modified afterwards using unitary function **LL_RCC_PLL1_SetN()**.
- ***uint32_t LL_UTILS_PLLInitTypeDef::PLLP***

  Division for the main system clock. This parameter must be a number between Min_Data = 2 and Max_Data = 128 odd division factors are not allowedThis feature can be modified afterwards using unitary function **LL_RCC_PLL1_SetP()**.
- ***uint32_t LL_UTILS_PLLInitTypeDef::FRACN***

  Fractional part of the multiplication factor for PLL VCO. This parameter can be a value between 0 and 8191This feature can be modified afterwards using unitary function **LL_RCC_PLL1_SetFRACN()**.
- ***uint32_t LL_UTILS_PLLInitTypeDef::VCO_Input***

  PLL clock Input range. This parameter can be a value of ***RCC_LL_EC_PLLINPUTRANGE***This feature can be modified afterwards using unitary function **LL_RCC_PLL1_SetVCOInputRange()**.
- ***uint32_t LL_UTILS_PLLInitTypeDef::VCO_Output***

  PLL clock Output range. This parameter can be a value of ***RCC_LL_EC_PLLVCORANGE***This feature can be modified afterwards using unitary function **LL_RCC_PLL1_SetVCOOutputRange()**.

128.1.2 LL_UTILS_ClkInitTypeDef

LL_UTILS_ClkInitTypeDef is defined in the stm32h7xx_ll_utils.h

Data Fields

- ***uint32_t SYSCLKDivider***
- ***uint32_t AHBCLKDivider***
- ***uint32_t APB1CLKDivider***
- ***uint32_t APB2CLKDivider***
- ***uint32_t APB3CLKDivider***
- ***uint32_t APB4CLKDivider***

Field Documentation

- ***uint32_t LL_UTILS_ClkInitTypeDef::SYSCLKDivider***

  The System clock (SYSCLK) divider. This clock is derived from the PLL output. This parameter can be a value of ***RCC_LL_EC_SYSCLK_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetSysPrescaler()**.

- ***uint32_t LL_UTILS_ClkInitTypeDef::AHBCLKDivider***

  The AHB clock (HCLK) divider. This clock is derived from the system clock (SYSCLK). This parameter can be a value of ***RCC_LL_EC_AHB_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetAHBPrescaler()**.
- ***uint32_t LL_UTILS_ClkInitTypeDef::APB1CLKDivider***

  The APB1 clock (PCLK1) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_LL_EC_APB1_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetAPB1Prescaler()**.
- ***uint32_t LL_UTILS_ClkInitTypeDef::APB2CLKDivider***

  The APB2 clock (PCLK2) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_LL_EC_APB2_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetAPB2Prescaler()**.
- ***uint32_t LL_UTILS_ClkInitTypeDef::APB3CLKDivider***

  The APB2 clock (PCLK3) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_LL_EC_APB3_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetAPB3Prescaler()**.
- ***uint32_t LL_UTILS_ClkInitTypeDef::APB4CLKDivider***

  The APB4 clock (PCLK4) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_LL_EC_APB4_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetAPB4Prescaler()**.

128.2 UTILS Firmware driver API description

The following section lists the various functions of the UTILS library.

128.2.1 System Configuration functions

System, AHB and APB buses clocks configuration

- The maximum frequency of the SYSCLK is 480 MHz(*) and HCLK is 240 MHz.
- The maximum frequency of the PCLK1, PCLK2, PCLK3 and PCLK4 is 120 MHz.

This section contains the following APIs:

- ***LL_SetSystemCoreClock()***
- ***LL_PLL_ConfigSystemClock_HSI()***
- ***LL_PLL_ConfigSystemClock_HSE()***
- ***LL_SetFlashLatency()***

128.2.2 Detailed description of functions

LL_GetUID_Word0

Function name

__STATIC_INLINE uint32_t LL_GetUID_Word0 (void)

Function description

Get Word0 of the unique device identifier (UID based on 96 bits)

Return values

- **UID[31:0]:**

LL_GetUID_Word1

Function name

__STATIC_INLINE uint32_t LL_GetUID_Word1 (void)

Function description

Get Word1 of the unique device identifier (UID based on 96 bits)

Return values

- **UID[63:32]:**

LL_GetUID_Word2

Function name

__STATIC_INLINE uint32_t LL_GetUID_Word2 (void)

Function description

Get Word2 of the unique device identifier (UID based on 96 bits)

Return values

- **UID[95:64]:**

LL_GetFlashSize

Function name

__STATIC_INLINE uint32_t LL_GetFlashSize (void)

Function description

Get Flash memory size.

Return values

- **FLASH_SIZE[15:0]:** Flash memory size

Notes

- This bitfield indicates the size of the device Flash memory expressed in Kbytes. As an example, 0x040 corresponds to 64 Kbytes.

LL_GetPackageType

Function name

__STATIC_INLINE uint32_t LL_GetPackageType (void)

Function description

Get Package type.

Return values

- **Returned:** value can be one of the following values:
  - LL_UTILS_PACKAGETYPE_LQFP100
  - LL_UTILS_PACKAGETYPE_TQFP144
  - LL_UTILS_PACKAGETYPE_TQFP176_UFBGA176
  - LL_UTILS_PACKAGETYPE_LQFP208_TFBGA240
  - LL_UTILS_PACKAGETYPE_LQFP64 (*)
  - LL_UTILS_PACKAGETYPE_TFBGA100_LQFP100 (*)
  - LL_UTILS_PACKAGETYPE_LQFP100_SMPS (*)
  - LL_UTILS_PACKAGETYPE_TFBGA100_SMPS (*)
  - LL_UTILS_PACKAGETYPE_WLCSP132_SMPS (*)
  - LL_UTILS_PACKAGETYPE_LQFP144 (*)
  - LL_UTILS_PACKAGETYPE_LQFP144_SMPS (*)
  - LL_UTILS_PACKAGETYPE_UFBGA169 (*)
  - LL_UTILS_PACKAGETYPE_UFBGA176_LQFP176 (*)
  - LL_UTILS_PACKAGETYPE_LQFP176_SMPS (*)
  - LL_UTILS_PACKAGETYPE_UFBGA176_SMPS (*)
  - LL_UTILS_PACKAGETYPE_TFBGA216 (*)
  - LL_UTILS_PACKAGETYPE_TFBGA225 (*)
  - LL_UTILS_PACKAGETYPE_VFQFPN68_INDUS (*)
  - LL_UTILS_PACKAGETYPE_LQFP100_INDUS (*)
  - LL_UTILS_PACKAGETYPE_TFBGA100_INDUS (*)
  - LL_UTILS_PACKAGETYPE_WLCSP115_INDUS (*)
  - LL_UTILS_PACKAGETYPE_UFBGA144 (*)
  - LL_UTILS_PACKAGETYPE_LQFP144_INDUS (*)
  - LL_UTILS_PACKAGETYPE_UFBGA169_INDUS (*)
  - LL_UTILS_PACKAGETYPE_UFBGA176+25_INDUS (*)
  - LL_UTILS_PACKAGETYPE_LQFP176_INDUS (*)

  (*) Packages available on some STM32H7 lines only.

Notes

- For some SM32H7 lines, enabling the SYSCFG clock is mandatory. the SYSCFG clock enabling is ensured by LL_APB4_GRP1_EnableClock

LL_InitTick

Function name

__STATIC_INLINE void LL_InitTick (uint32_t HCLKFrequency, uint32_t Ticks)

Function description

This function configures the Cortex-M SysTick source of the time base.

Parameters

- **HCLKFrequency:** HCLK frequency in Hz (can be calculated thanks to RCC helper macro)
- **Ticks:** Number of ticks

Return values

- **None:**

Notes

- When a RTOS is used, it is recommended to avoid changing the SysTick configuration by calling this function, for a delay use rather osDelay RTOS service.

LL_Init1msTick

Function name

void LL_Init1msTick (uint32_t CPU_Frequency)

Function description

This function configures the Cortex-M SysTick source to have 1ms time base.

Parameters

- **CPU_Frequency:** Core frequency in Hz

Return values

- **None:**

Notes

- When a RTOS is used, it is recommended to avoid changing the Systick configuration by calling this function, for a delay use rather osDelay RTOS service.
- CPU_Frequency can be calculated thanks to RCC helper macro or function LL_RCC_GetSystemClocksFreq LL_RCC_GetSystemClocksFreq() is used to calculate the CM7 clock frequency and __LL_RCC_CALC_HCLK_FREQ is used to calculate the CM4 clock frequency.

LL_mDelay

Function name

void LL_mDelay (uint32_t Delay)

Function description

This function provides accurate delay (in milliseconds) based on SysTick counter flag.

Parameters

- **Delay:** specifies the delay time length, in milliseconds.

Return values

- **None:**

Notes

- When a RTOS is used, it is recommended to avoid using blocking delay and use rather osDelay service.
- To respect 1ms timebase, user should call LL_Init1msTick function which will configure Systick to 1ms

LL_SetSystemCoreClock

Function name

void LL_SetSystemCoreClock (uint32_t CPU_Frequency)

Function description

This function sets directly SystemCoreClock CMSIS variable.

Parameters

- **CPU_Frequency:** Core frequency in Hz

Return values

- **None:**

Notes

- Variable can be calculated also through SystemCoreClockUpdate function.
- CPU_Frequency can be calculated thanks to RCC helper macro or function LL_RCC_GetSystemClocksFreq LL_RCC_GetSystemClocksFreq() is used to calculate the CM7 clock frequency and __LL_RCC_CALC_HCLK_FREQ is used to calculate the CM4 clock frequency.

LL_PLL_ConfigSystemClock_HSI

Function name

ErrorStatus LL_PLL_ConfigSystemClock_HSI (LL_UTILS_PLLInitTypeDef * UTILS_PLLInitStruct, LL_UTILS_ClkInitTypeDef * UTILS_ClkInitStruct)

Function description

This function configures system clock at maximum frequency with HSI as clock source of the PLL.

Parameters

- **UTILS_PLLInitStruct:** pointer to a LL_UTILS_PLLInitTypeDef structure that contains the configuration information for the PLL.
- **UTILS_ClkInitStruct:** pointer to a LL_UTILS_ClkInitTypeDef structure that contains the configuration information for the BUS prescalers.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: Max frequency configuration done
  - ERROR: Max frequency configuration not done

Notes

- The application need to ensure that PLL is disabled.
- Function is based on the following formula: PLL output frequency = (((HSI frequency / PLLM) * PLLN) / PLLP)PLLM: ensure that the VCO input frequency ranges from 1 to 16 MHz (PLLVCO_input = HSI frequency / PLLM)PLLN: ensure that the VCO output frequency is between 150 and 836 MHz or 128 to 560 MHz(***) (PLLVCO_output = PLLVCO_input * PLLN)PLLP: ensure that max frequency at 550000000 Hz(*), 480000000 Hz(**) or 280000000 Hz(***) is reach (PLLVCO_output / PLLP)

LL_PLL_ConfigSystemClock_HSE

Function name

ErrorStatus LL_PLL_ConfigSystemClock_HSE (uint32_t HSEFrequency, uint32_t HSEBypass, LL_UTILS_PLLInitTypeDef * UTILS_PLLInitStruct, LL_UTILS_ClkInitTypeDef * UTILS_ClkInitStruct)

Function description

This function configures system clock with HSE as clock source of the PLL.

Parameters

- **HSEFrequency:** Value between Min_Data = 4000000 and Max_Data = 48000000
- **HSEBypass:** This parameter can be one of the following values:
  - LL_UTILS_HSEBYPASS_ON
  - LL_UTILS_HSEBYPASS_OFF
- **UTILS_PLLInitStruct:** pointer to a LL_UTILS_PLLInitTypeDef structure that contains the configuration information for the PLL.
- **UTILS_ClkInitStruct:** pointer to a LL_UTILS_ClkInitTypeDef structure that contains the configuration information for the BUS prescalers.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: Max frequency configuration done
  - ERROR: Max frequency configuration not done

Notes

- The application need to ensure that PLL is disabled.
- Function is based on the following formula: PLL output frequency = (((HSE frequency / PLLM) * PLLN) / PLLP)PLLM: ensure that the VCO input frequency ranges from 0.95 to 2.10 MHz (PLLVCO_input = HSE frequency / PLLM)PLLN: ensure that the VCO output frequency is between 150 and 836 MHz or 128 to 560 MHz(***) (PLLVCO_output = PLLVCO_input * PLLN)PLLP: ensure that max frequency at 550000000 Hz(*), 480000000 Hz(**) or 280000000 Hz(***) is reached (PLLVCO_output / PLLP)

LL_SetFlashLatency

Function name

ErrorStatus LL_SetFlashLatency (uint32_t HCLK_Frequency)

Function description

Update number of Flash wait states in line with new frequency and current voltage range.

Parameters

- **HCLK_Frequency:** HCLK frequency

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: Latency has been modified
  - ERROR: Latency cannot be modified

128.3 UTILS Firmware driver defines

The following section lists the various define and macros of the module.

128.3.1 UTILS

UTILS

HSE Bypass activation

LL_UTILS_HSEBYPASS_OFF

HSE Bypass is not enabled

LL_UTILS_HSEBYPASS_ON

HSE Bypass is enabled

PACKAGE TYPE

LL_UTILS_PACKAGETYPE_LQFP100

LQFP100 package type

LL_UTILS_PACKAGETYPE_TQFP144

TQFP144 package type

LL_UTILS_PACKAGETYPE_TQFP176_UFBGA176

TQFP176 or UFBGA176 package type

LL_UTILS_PACKAGETYPE_LQFP208_TFBGA240

LQFP208 or TFBGA240 package type

129 LL WWDG Generic Driver

129.1 WWDG Firmware driver API description

The following section lists the various functions of the WWDG library.

129.1.1 Detailed description of functions

LL_WWDG_Enable

Function name

__STATIC_INLINE void LL_WWDG_Enable (WWDG_TypeDef * WWDGx)

Function description

Enable Window Watchdog.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **None:**

Notes

- It is enabled by setting the WDGA bit in the WWDG_CR register, then it cannot be disabled again except by a reset. This bit is set by software and only cleared by hardware after a reset. When WDGA = 1, the watchdog can generate a reset.

Reference Manual to LL API cross reference:

- CR WDGA LL_WWDG_Enable

LL_WWDG_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_WWDG_IsEnabled (WWDG_TypeDef * WWDGx)

Function description

Checks if Window Watchdog is enabled.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR WDGA LL_WWDG_IsEnabled

LL_WWDG_SetCounter

Function name

__STATIC_INLINE void LL_WWDG_SetCounter (WWDG_TypeDef * WWDGx, uint32_t Counter)

Function description

Set the Watchdog counter value to provided value (7-bits T[6:0])

Parameters

- **WWDGx:** WWDG Instance
- **Counter:** 0..0x7F (7 bit counter value)

Return values

- **None:**

Notes

- When writing to the WWDG_CR register, always write 1 in the MSB b6 to avoid generating an immediate reset This counter is decremented every (4096 x 2expWDGTB) PCLK cycles A reset is produced when it rolls over from 0x40 to 0x3F (bit T6 becomes cleared) Setting the counter lower then 0x40 causes an immediate reset (if WWDG enabled)

Reference Manual to LL API cross reference:

- CR T LL_WWDG_SetCounter

LL_WWDG_GetCounter

Function name

__STATIC_INLINE uint32_t LL_WWDG_GetCounter (WWDG_TypeDef * WWDGx)

Function description

Return current Watchdog Counter Value (7 bits counter value)

Parameters

- **WWDGx:** WWDG Instance

Return values

- **7:** bit Watchdog Counter value

Reference Manual to LL API cross reference:

- CR T LL_WWDG_GetCounter

LL_WWDG_SetPrescaler

Function name

__STATIC_INLINE void LL_WWDG_SetPrescaler (WWDG_TypeDef * WWDGx, uint32_t Prescaler)

Function description

Set the time base of the prescaler (WDGTB).

Parameters

- **WWDGx:** WWDG Instance
- **Prescaler:** This parameter can be one of the following values:
  - LL_WWDG_PRESCALER_1
  - LL_WWDG_PRESCALER_2
  - LL_WWDG_PRESCALER_4
  - LL_WWDG_PRESCALER_8
  - LL_WWDG_PRESCALER_16
  - LL_WWDG_PRESCALER_32
  - LL_WWDG_PRESCALER_64
  - LL_WWDG_PRESCALER_128

Return values

- **None:**

Notes

- Prescaler is used to apply ratio on PCLK clock, so that Watchdog counter is decremented every (4096 x 2expWDGTB) PCLK cycles

Reference Manual to LL API cross reference:

- CFR WDGTB LL_WWDG_SetPrescaler

LL_WWDG_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_WWDG_GetPrescaler (WWDG_TypeDef * WWDGx)

Function description

Return current Watchdog Prescaler Value.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_WWDG_PRESCALER_1
  - LL_WWDG_PRESCALER_2
  - LL_WWDG_PRESCALER_4
  - LL_WWDG_PRESCALER_8
  - LL_WWDG_PRESCALER_16
  - LL_WWDG_PRESCALER_32
  - LL_WWDG_PRESCALER_64
  - LL_WWDG_PRESCALER_128

Reference Manual to LL API cross reference:

- CFR WDGTB LL_WWDG_GetPrescaler

LL_WWDG_SetWindow

Function name

__STATIC_INLINE void LL_WWDG_SetWindow (WWDG_TypeDef * WWDGx, uint32_t Window)

Function description

Set the Watchdog Window value to be compared to the downcounter (7-bits W[6:0]).

Parameters

- **WWDGx:** WWDG Instance
- **Window:** 0x00..0x7F (7 bit Window value)

Return values

- **None:**

Notes

- This window value defines when write in the WWDG_CR register to program Watchdog counter is allowed. Watchdog counter value update must occur only when the counter value is lower than the Watchdog window register value. Otherwise, a MCU reset is generated if the 7-bit Watchdog counter value (in the control register) is refreshed before the downcounter has reached the watchdog window register value. Physically is possible to set the Window lower then 0x40 but it is not recommended. To generate an immediate reset, it is possible to set the Counter lower than 0x40.

Reference Manual to LL API cross reference:

- CFR W LL_WWDG_SetWindow

LL_WWDG_GetWindow

Function name

__STATIC_INLINE uint32_t LL_WWDG_GetWindow (WWDG_TypeDef * WWDGx)

Function description

Return current Watchdog Window Value (7 bits value)

Parameters

- **WWDGx:** WWDG Instance

Return values

- **7:** bit Watchdog Window value

Reference Manual to LL API cross reference:

- CFR W LL_WWDG_GetWindow

LL_WWDG_IsActiveFlag_EWKUP

Function name

__STATIC_INLINE uint32_t LL_WWDG_IsActiveFlag_EWKUP (WWDG_TypeDef * WWDGx)

Function description

Indicates if the WWDG Early Wakeup Interrupt Flag is set or not.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **State:** of bit (1 or 0).

Notes

- This bit is set by hardware when the counter has reached the value 0x40. It must be cleared by software by writing 0. A write of 1 has no effect. This bit is also set if the interrupt is not enabled.

Reference Manual to LL API cross reference:

- SR EWIF LL_WWDG_IsActiveFlag_EWKUP

LL_WWDG_ClearFlag_EWKUP

Function name

__STATIC_INLINE void LL_WWDG_ClearFlag_EWKUP (WWDG_TypeDef * WWDGx)

Function description

Clear WWDG Early Wakeup Interrupt Flag (EWIF)

Parameters

- **WWDGx:** WWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR EWIF LL_WWDG_ClearFlag_EWKUP

LL_WWDG_EnableIT_EWKUP

Function name

__STATIC_INLINE void LL_WWDG_EnableIT_EWKUP (WWDG_TypeDef * WWDGx)

Function description

Enable the Early Wakeup Interrupt.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **None:**

Notes

- When set, an interrupt occurs whenever the counter reaches value 0x40. This interrupt is only cleared by hardware after a reset

Reference Manual to LL API cross reference:

- CFR EWI LL_WWDG_EnableIT_EWKUP

LL_WWDG_IsEnabledIT_EWKUP

Function name

__STATIC_INLINE uint32_t LL_WWDG_IsEnabledIT_EWKUP (WWDG_TypeDef * WWDGx)

Function description

Check if Early Wakeup Interrupt is enabled.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFR EWI LL_WWDG_IsEnabledIT_EWKUP

129.2 WWDG Firmware driver defines

The following section lists the various define and macros of the module.

129.2.1 WWDG

WWDG

IT Defines

LL_WWDG_CFR_EWI

PRESCALER

LL_WWDG_PRESCALER_1

WWDG counter clock = (PCLK1/4096)/1

LL_WWDG_PRESCALER_2

WWDG counter clock = (PCLK1/4096)/2

LL_WWDG_PRESCALER_4

WWDG counter clock = (PCLK1/4096)/4

LL_WWDG_PRESCALER_8

WWDG counter clock = (PCLK1/4096)/8

LL_WWDG_PRESCALER_16

WWDG counter clock = (PCLK1/4096)/16

LL_WWDG_PRESCALER_32

WWDG counter clock = (PCLK1/4096)/32

LL_WWDG_PRESCALER_64

WWDG counter clock = (PCLK1/4096)/64

LL_WWDG_PRESCALER_128

WWDG counter clock = (PCLK1/4096)/128

Common Write and read registers macros

LL_WWDG_WriteReg

Description:

- Write a value in WWDG register.

Parameters:

- __INSTANCE__: WWDG Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_WWDG_ReadReg

Description:

- Read a value in WWDG register.

Parameters:

- __INSTANCE__: WWDG Instance
- __REG__: Register to be read

Return value:

- Register: value

130 FAQs

General subjects

Why should I use the HAL drivers?

There are many advantages in using the HAL drivers:

- Ease of use: you can use the HAL drivers to configure and control any peripheral embedded within your STM32 MCU without prior in-depth knowledge of the product.
- HAL drivers provide intuitive and ready-to-use APIs to configure the peripherals and support polling, interrupt and DMA programming model to accommodate all application requirements, thus allowing the end-user to build a complete application by calling a few APIs.
- Higher level of abstraction than a standard peripheral library allowing to transparently manage:
  - Data transfers and processing using blocking mode (polling) or non-blocking mode (interrupt or DMA)
  - Error management through peripheral error detection and timeout mechanism.
- Generic architecture speeding up initialization and porting, thus allowing customers to focus on innovation.
- Generic set of APIs with full compatibility across the STM32 Series/lines, to ease the porting task between STM32 MCUs.
- The APIs provided within the HAL drivers are feature-oriented and do not require in-depth knowledge of peripheral operation.
- The APIs provided are modular. They include initialization, IO operation and control functions. The end-user has to call init function, then start the process by calling one IO operation functions (write, read, transmit, receive, …). Most of the peripherals have the same architecture.
- The number of functions required to build a complete and useful application is very reduced. As an example, to build a UART communication process, the user only has to call HAL_UART_Init() then HAL_UART_Transmit() or HAL_UART_Receive().

Which devices are supported by the HAL drivers?

The HAL drivers are developed to support all STM32H7 devices. To ensure compatibility between all devices and portability with others Series and lines, the API is split into the generic and the extension APIs . For more details, please refer to *section* ***Devices supported by the HAL drivers***.

What is the cost of using HAL drivers in term of code size and performance?

Like generic architecture drivers, the HAL drivers may induce firmware overhead.

This is due to the high abstraction level and ready-to-use APIs which allow data transfers, errors management and offloads the user application from implementation details.

Architecture

How many files should I modify to configure the HAL drivers?

Only one file needs to be modified: stm32h7xx_hal_conf.h. You can modify this file by disabling unused modules, or adjusting some parameters (i.e. HSE value, System configuration…)

A template is provided in the HAL drivers folders (stm32h7xx_hal_conf_template.c).

Which header files should I include in my application to use the HAL drivers?

Only stm32h7xx_hal.h file has to be included.

What is the difference between xx_hal_ppp.c/.h and xx_hal_ppp_ *ex* .c/.h?

The HAL driver architecture supports common features across STM32 Series/lines. To support specific features, the drivers are split into two groups.

- The generic APIs (stm32h7xx_hal_ppp.c): It includes the common set of APIs across all the STM32 product lines

- The extension APIs (stm32h7xx_hal_ppp_ex.c): It includes the specific APIs for specific device part number or family.

Initialization and I/O operation functions

How do I configure the system clock?

Unlike the standard library, the system clock configuration is not performed in CMSIS drivers file (system_stm32h7xx.c) but in the main user application by calling the two main functions, HAL_RCC_OscConfig() and HAL_RCC_ClockConfig(). It can be modified in any user application section.

What is the purpose of the *PPP_HandleTypeDef *pHandle* structure located in each driver in addition to the Initialization structure

PPP_HandleTypeDef *pHandle is the main structure implemented in the HAL drivers. It handles the peripheral configuration and registers, and embeds all the structures and variables required to follow the peripheral device flow (pointer to buffer, Error code, State,...)

However, this structure is not required to service peripherals such as GPIO, SYSTICK, PWR, and RCC.

What is the purpose of HAL_PPP_MspInit() and HAL_PPP_MspDeInit() functions?

These function are called within HAL_PPP_Init() and HAL_PPP_DeInit(), respectively. They are used to perform the low level Initialization/de-initialization related to the additional hardware resources (RCC, GPIO, NVIC and DMA).

These functions are declared in stm32h7xx_hal_msp.c. A template is provided in the HAL driver folders (stm32h7xx_hal_msp_template.c).

When and how should I use callbacks functions (functions declared with the attribute *__weak*)?

Use callback functions for the I/O operations used in DMA or interrupt mode. The PPP process complete callbacks are called to inform the user about process completion in real-time event mode (interrupts).

The Errors callbacks are called when a processing error occurs in DMA or interrupt mode. These callbacks are customized by the user to add user proprietary code. They can be declared in the application. Note that the same process completion callbacks are used for DMA and interrupt mode.

Is it mandatory to use HAL_Init() function at the beginning of the user application?

It is mandatory to use HAL_Init() function to enable the system configuration (Prefetch, Data instruction cache,…), configure the systTick and the NVIC priority grouping and the hardware low level initialization.

The SysTick configuration shall be adjusted by calling ***HAL_RCC_ClockConfig()*** function, to obtain 1 ms whatever the system clock.

Why do I need to configure the SysTick timer to use the HAL drivers?

The SysTick timer is configured to be used to generate variable increments by calling *HAL_IncTick()* function in SysTick ISR and retrieve the value of this variable by calling *HAL_GetTick()* function.

The call HAL_GetTick() function is mandatory when using HAL drivers with Polling Process or when using HAL_Delay().

Why is the SysTick timer configured to have 1 ms?

This is mandatory to ensure correct IO operation in particular for polling mode operation where the 1 ms is required as timebase.

Could HAL_Delay() function block my application under certain conditions?

Care must be taken when using HAL_Delay() since this function provides accurate delay based on a variable incremented in SysTick ISR. This implies that if HAL_Delay() is called from a peripheral ISR process, then the SysTick interrupt must have higher priority (numerically lower) than the peripheral interrupt, otherwise the caller ISR process will be blocked. Use HAL_NVIC_SetPriority() function to change the SysTick interrupt priority.

What programming model sequence should I follow to use HAL drivers ?

Follow the sequence below to use the APIs provided in the HAL drivers:

1. Call HAL_Init() function to initialize the system (data cache, NVIC priority,…).
2. Initialize the system clock by calling HAL_RCC_OscConfig() followed by HAL_RCC_ClockConfig().
3. Add HAL_IncTick() function under SysTick_Handler() ISR function to enable polling process when using HAL_Delay() function
4. Start initializing your peripheral by calling HAL_PPP_Init().
5. Implement the hardware low level initialization (Peripheral clock, GPIO, DMA,..) by calling HAL_PPP_MspInit() instm32h7xx_hal_msp.c
6. Start your process operation by calling IO operation functions.

What is the purpose of HAL_PPP_IRQHandler() function and when should I use it?

HAL_PPP_IRQHandler() is used to handle interrupt process. It is called under PPP_IRQHandler() function in stm32h7xx_it.c. In this case, the end-user has to implement only the callbacks functions (prefixed by __weak) to perform the appropriate action when an interrupt is detected. Advanced users can implement their own code in PPP_IRQHandler() without calling HAL_PPP_IRQHandler().

Can I use directly the macros defined in xx_hal_ppp.h ?

Yes, you can: a set of macros is provided with the APIs. They allow accessing directly some specific features using peripheral flags.

Where must PPP_HandleTypedef structure peripheral handler be declared?

PPP_HandleTypedef structure peripheral handler must be declared as a global variable, so that all the structure fields are set to 0 by default. In this way, the peripheral handler default state are set to HAL_PPP_STATE_RESET, which is the default state for each peripheral after a system reset.

When should I use HAL versus LL drivers?

HAL drivers offer high-level and function-oriented APIs, with a high level of portability. Product/IPs complexity is hidden for end users. LL drivers offer low-level APIs at registers level, with a better optimization but less portability. They require a deep knowledge of product/IPs specifications.

How can I include LL drivers in my environment? Is there any LL configuration file as for HAL?

There is no configuration file. Source code shall directly include the necessary stm32h7xx_ll_ppp.h file(s).

Can I use HAL and LL drivers together? If yes, what are the constraints?

It is possible to use both HAL and LL drivers. One can handle the IP initialization phase with HAL and then manage the I/O operations with LL drivers. The major difference between HAL and LL is that HAL drivers require to create and use handles for operation management while LL drivers operates directly on peripheral registers. Mixing HAL and LL is illustrated in Examples_MIX example.

Is there any LL APIs which are not available with HAL?

Yes, there are. A few Cortex® APIs have been added in stm32h7xx_ll_cortex.h e.g. for accessing SCB or SysTick registers.

Why are SysTick interrupts not enabled on LL drivers?

When using LL drivers in standalone mode, you do not need to enable SysTick interrupts because they are not used in LL APIs, while HAL functions requires SysTick interrupts to manage timeouts.

Revision history

Table 25. **Document revision history**

Date	Revision	Changes
11-May-2017	1	Initial release.
12-Jan-2018	2	Changed ARM into Arm in the whole document. Updated typical clock configuration sequence in System clock initialization. Updated DMA generic and extension drivers, Flash generic and extension drivers, System driver, NAND Flash generic driver, PCD generic and extension drivers, RCC generic and extension drivers, SPDIFRX generic driver, SPI generic and extension drivers.
05-Apr-2019	3	Added STM32CubeH7 low-layer drivers. Added dual-core STM32H7 microcontrollers.
20-Dec-2019	4	Added support for STM32H7A3/7B3 line and STM32H7B0 Value line. Added DTS, GFXMMU, OCTOSPI, OTFDEC and PSSI in Section 2 Acronyms and definitions. Updated Section 4.4 Devices supported by HAL drivers. Updated list of stm32h7xxx.h files in Section 4.5.1 HAL API naming rules. Added caution note related to firmware power configuration versus hardware board configuration in Section 4.12.2.2 System clock initialization. Added FAQs section.
02-Jul-2020	5	Updated Section 2 Acronyms and definitions. Updated list of stm32h7xxx.h files in Section 4.5.1 HAL API naming rules. Added STM32H723/H733, STM32H725/H735 and STM32H730 Value line part numbers in Table 5. List of devices supported by HAL drivers.
14-Dec-2022	6	Added the following HAL drivers: Section 7 HAL System Driver, Section 66 HAL QSPI Generic Driver, and Section 82 HAL SMBUS Extension Driver. Updated Section 27 HAL ETH Generic Driver, Section 28 HAL ETH Extension Driver, and Section 11 HAL COMP Generic Driver.

Contents

1 General information . . . 3
2 Acronyms and definitions . . . 4
3 Support of dual-core architectures . . . 7
4 Overview of HAL drivers . . . 10
4.1 HAL and user-application files . . . 11
4.1.1 HAL driver files . . . 11
4.1.2 User-application files . . . 11
4.2 HAL data structures . . . 12
4.2.1 Peripheral handle structures . . . 13
4.2.2 Initialization and configuration structure . . . 14
4.2.3 Specific process structures . . . 14
4.3 API classification . . . 15
4.4 Devices supported by HAL drivers . . . 16
4.5 HAL driver rules . . . 21
4.5.1 HAL API naming rules . . . 21
4.5.2 HAL general naming rules . . . 22
4.5.3 HAL interrupt handler and callback functions . . . 23
4.6 HAL generic APIs . . . 23
4.7 HAL extension APIs . . . 24
4.7.1 HAL extension model overview . . . 24
4.7.2 HAL extension model cases . . . 25
4.8 File inclusion model . . . 27
4.9 HAL common resources . . . 28
4.10 HAL configuration . . . 28
4.11 HAL system peripheral handling . . . 29
4.11.1 Clocks . . . 29
4.11.2 GPIOs . . . 29
4.11.3 Cortex® NVIC and SysTick timer . . . 31
4.11.4 PWR . . . 31
4.11.5 EXTI . . . 31

4.11.6 DMA . . . 32
4.12 How to use HAL drivers . . . 33
4.12.1 HAL usage models . . . 33
4.12.2 HAL initialization . . . 34
4.12.3 HAL I/O operation process . . . 38
4.12.4 Timeout and error management . . . 41
5 Overview of low-layer drivers . . . 45
5.1 Low-layer files . . . 45
5.2 Overview of low-layer APIs and naming rules . . . 47
5.2.1 Peripheral initialization functions . . . 47
5.2.2 Peripheral register-level configuration functions . . . 49
6 Cohabiting of HAL and LL . . . 51
6.1 Low-layer driver used in Standalone mode . . . 51
6.2 Mixed use of low-layer APIs and HAL drivers . . . 51
7 HAL System Driver . . . 52
7.1 HAL Firmware driver API description . . . 52
7.1.1 How to use this driver . . . 52
7.1.2 Initialization and de-initialization functions . . . 52
7.1.3 HAL Control functions . . . 52
7.1.4 Detailed description of functions . . . 54
7.2 HAL Firmware driver defines . . . 69
7.2.1 HAL . . . 69
8 HAL ADC Generic Driver . . . 76
8.1 ADC Firmware driver registers structures . . . 76
8.1.1 ADC_OversamplingTypeDef . . . 76
8.1.2 ADC_InitTypeDef . . . 76
8.1.3 ADC_ChannelConfTypeDef . . . 78
8.1.4 ADC_AnalogWDGConfTypeDef . . . 79
8.1.5 ADC_InjectionConfigTypeDef . . . 80
8.1.6 __ADC_HandleTypeDef . . . 81
8.2 ADC Firmware driver API description . . . 82
8.2.1 ADC peripheral features . . . 82

8.2.2 How to use this driver . . . 82
8.2.3 Peripheral Control functions . . . 85
8.2.4 Peripheral state and errors functions . . . 85
8.2.5 Detailed description of functions . . . 86
8.3 ADC Firmware driver defines . . . 97
8.3.1 ADC . . . 97
9 HAL ADC Extension Driver . . . 126
9.1 ADCEx Firmware driver registers structures . . . 126
9.1.1 ADC_InjOversamplingTypeDef . . . 126
9.1.2 ADC_InjectionConfTypeDef . . . 126
9.1.3 ADC_MultiModeTypeDef . . . 128
9.2 ADCEx Firmware driver API description . . . 129
9.2.1 IO operation functions . . . 129
9.2.2 Peripheral Control functions . . . 130
9.2.3 Detailed description of functions . . . 130
9.3 ADCEx Firmware driver defines . . . 141
9.3.1 ADCEx . . . 141
10 HAL CEC Generic Driver . . . 143
10.1 CEC Firmware driver registers structures . . . 143
10.1.1 CEC_InitTypeDef . . . 143
10.1.2 __CEC_HandleTypeDef . . . 144
10.2 CEC Firmware driver API description . . . 145
10.2.1 How to use this driver . . . 145
10.2.2 Initialization and Configuration functions . . . 145
10.2.3 IO operation functions . . . 146
10.2.4 Peripheral Control function . . . 146
10.2.5 Detailed description of functions . . . 146
10.3 CEC Firmware driver defines . . . 151
10.3.1 CEC . . . 151
11 HAL COMP Generic Driver . . . 161
11.1 COMP Firmware driver registers structures . . . 161
11.1.1 COMP_InitTypeDef . . . 161

11.1.2 __COMP_HandleTypeDef . . . 161
11.2 COMP Firmware driver API description . . . 162
11.2.1 COMP Peripheral features . . . 162
11.2.2 How to use this driver . . . 162
11.2.3 Initialization and de-initialization functions . . . 164
11.2.4 IO operation functions . . . 164
11.2.5 Peripheral Control functions . . . 164
11.2.6 Peripheral State functions . . . 164
11.2.7 Detailed description of functions . . . 164
11.3 COMP Firmware driver defines . . . 169
11.3.1 COMP . . . 169
12 HAL CORDIC Generic Driver . . . 182
12.1 CORDIC Firmware driver registers structures . . . 182
12.1.1 __CORDIC_HandleTypeDef . . . 182
12.1.2 CORDIC_ConfigTypeDef . . . 183
12.2 CORDIC Firmware driver API description . . . 183
12.2.1 How to use this driver . . . 183
12.2.2 Initialization and de-initialization functions . . . 184
12.2.3 Peripheral Control functions . . . 185
12.2.4 Callback functions . . . 185
12.2.5 IRQ handler management . . . 185
12.2.6 Peripheral State functions . . . 185
12.2.7 Detailed description of functions . . . 185
12.3 CORDIC Firmware driver defines . . . 190
12.3.1 CORDIC . . . 190
13 HAL CORTEX Generic Driver . . . 196
13.1 CORTEX Firmware driver API description . . . 196
13.1.1 How to use this driver . . . 196
13.1.2 Initialization and de-initialization functions . . . 196
13.1.3 Peripheral Control functions . . . 197
13.1.4 Detailed description of functions . . . 197
13.2 CORTEX Firmware driver defines . . . 201

13.2.1 CORTEX . . . 202
14 HAL CRC Generic Driver . . . 203
14.1 CRC Firmware driver registers structures . . . 203
14.1.1 CRC_InitTypeDef . . . 203
14.1.2 CRC_HandleTypeDef . . . 204
14.2 CRC Firmware driver API description . . . 204
14.2.1 How to use this driver . . . 204
14.2.2 Initialization and de-initialization functions . . . 204
14.2.3 Peripheral Control functions . . . 205
14.2.4 Peripheral State functions . . . 205
14.2.5 Detailed description of functions . . . 205
14.3 CRC Firmware driver defines . . . 207
14.3.1 CRC . . . 207
15 HAL CRC Extension Driver . . . 210
15.1 CRCEx Firmware driver API description . . . 210
15.1.1 How to use this driver . . . 210
15.1.2 Extended configuration functions . . . 210
15.1.3 Detailed description of functions . . . 210
15.2 CRCEx Firmware driver defines . . . 211
15.2.1 CRCEx . . . 211
16 HAL CRYP Generic Driver . . . 213
16.1 CRYP Firmware driver registers structures . . . 213
16.1.1 CRYP_ConfigTypeDef . . . 213
16.1.2 __CRYP_HandleTypeDef . . . 214
16.2 CRYP Firmware driver API description . . . 215
16.2.1 How to use this driver . . . 215
16.2.2 Initialization, de-initialization and Set and Get configuration functions . . . 218
16.2.3 Encrypt Decrypt functions . . . 218
16.2.4 CRYP IRQ handler management . . . 219
16.2.5 Detailed description of functions . . . 219
16.3 CRYP Firmware driver defines . . . 225
16.3.1 CRYP . . . 226

17 HAL CRYP Extension Driver 230
17.1 CRYPEx Firmware driver API description 230
17.1.1 How to use this driver 230
17.1.2 Extended AES processing functions 230
17.1.3 Detailed description of functions 230
18 HAL DAC Generic Driver 232
18.1 DAC Firmware driver registers structures 232
18.1.1 __DAC_HandleTypeDef 232
18.1.2 DAC_SampleAndHoldConfTypeDef 233
18.1.3 DAC_ChannelConfTypeDef 233
18.2 DAC Firmware driver API description 234
18.2.1 DAC Peripheral features 234
18.2.2 How to use this driver 236
18.2.3 Initialization and de-initialization functions 237
18.2.4 IO operation functions 237
18.2.5 Peripheral Control functions 238
18.2.6 Peripheral State and Errors functions 238
18.2.7 Detailed description of functions 238
18.3 DAC Firmware driver defines 246
18.3.1 DAC 246
19 HAL DAC Extension Driver 251
19.1 DACEx Firmware driver API description 251
19.1.1 How to use this driver 251
19.1.2 Extended features functions 251
19.1.3 Peripheral Control functions 252
19.1.4 Detailed description of functions 252
19.2 DACEx Firmware driver defines 258
19.2.1 DACEx 258
20 HAL DCMI Generic Driver 261
20.1 DCMI Firmware driver registers structures 261
20.1.1 DCMI_CodesInitTypeDef 261
20.1.2 DCMI_SyncUnmaskTypeDef 261

20.1.3 DCMI_InitTypeDef . . . 261
20.1.4 __DCMI_HandleTypeDef . . . 262
20.2 DCMI Firmware driver API description . . . 263
20.2.1 How to use this driver . . . 263
20.2.2 Initialization and Configuration functions . . . 264
20.2.3 IO operation functions . . . 264
20.2.4 Peripheral Control functions . . . 265
20.2.5 Peripheral State and Errors functions . . . 265
20.2.6 Detailed description of functions . . . 265
20.3 DCMI Firmware driver defines . . . 271
20.3.1 DCMI . . . 271
21 HAL DFSDM Generic Driver . . . 278
21.1 DFSDM Firmware driver registers structures . . . 278
21.1.1 DFSDM_Channel_OutputClockTypeDef . . . 278
21.1.2 DFSDM_Channel_InputTypeDef . . . 278
21.1.3 DFSDM_Channel_SerialInterfaceTypeDef . . . 278
21.1.4 DFSDM_Channel_AwdTypeDef . . . 279
21.1.5 DFSDM_Channel_InitTypeDef . . . 279
21.1.6 __DFSDM_Channel_HandleTypeDef . . . 279
21.1.7 DFSDM_Filter_RegularParamTypeDef . . . 280
21.1.8 DFSDM_Filter_InjectedParamTypeDef . . . 280
21.1.9 DFSDM_Filter_FilterParamTypeDef . . . 281
21.1.10 DFSDM_Filter_InitTypeDef . . . 281
21.1.11 __DFSDM_Filter_HandleTypeDef . . . 281
21.1.12 DFSDM_Filter_AwdParamTypeDef . . . 283
21.2 DFSDM Firmware driver API description . . . 283
21.2.1 How to use this driver . . . 283
21.2.2 Channel initialization and de-initialization functions . . . 286
21.2.3 Channel operation functions . . . 287
21.2.4 Channel state function . . . 287
21.2.5 Filter initialization and de-initialization functions . . . 287
21.2.6 Filter control functions . . . 287

21.2.7 Filter operation functions . . . 288
21.2.8 Filter state functions . . . 289
21.2.9 Detailed description of functions . . . 289
21.3 DFSDM Firmware driver defines . . . 309
21.3.1 DFSDM . . . 309
22 HAL DMA2D Generic Driver . . . 314
22.1 DMA2D Firmware driver registers structures . . . 314
22.1.1 DMA2D_CLUTCfgTypeDef . . . 314
22.1.2 DMA2D_InitTypeDef . . . 314
22.1.3 DMA2D_LayerCfgTypeDef . . . 315
22.1.4 __DMA2D_HandleTypeDef . . . 315
22.2 DMA2D Firmware driver API description . . . 316
22.2.1 How to use this driver . . . 316
22.2.2 Initialization and Configuration functions . . . 318
22.2.3 IO operation functions . . . 319
22.2.4 Peripheral Control functions . . . 320
22.2.5 Peripheral State and Errors functions . . . 320
22.2.6 Detailed description of functions . . . 320
22.3 DMA2D Firmware driver defines . . . 331
22.3.1 DMA2D . . . 331
23 HAL DMA Generic Driver . . . 338
23.1 DMA Firmware driver registers structures . . . 338
23.1.1 DMA_InitTypeDef . . . 338
23.1.2 __DMA_HandleTypeDef . . . 339
23.2 DMA Firmware driver API description . . . 340
23.2.1 How to use this driver . . . 340
23.2.2 Initialization and de-initialization functions . . . 341
23.2.3 IO operation functions . . . 341
23.2.4 State and Errors functions . . . 342
23.2.5 Detailed description of functions . . . 342
23.3 DMA Firmware driver defines . . . 346
23.3.1 DMA . . . 346

24 HAL DMA Extension Driver 362
24.1 DMAEx Firmware driver registers structures 362
24.1.1 HAL_DMA_MuxSyncConfigTypeDef 362
24.1.2 HAL_DMA_MuxRequestGeneratorConfigTypeDef 362
24.2 DMAEx Firmware driver API description 363
24.2.1 How to use this driver 363
24.2.2 Extended features functions 363
24.2.3 Detailed description of functions 364
24.3 DMAEx Firmware driver defines 366
24.3.1 DMAEx 366
25 HAL DSI Generic Driver 371
25.1 DSI Firmware driver registers structures 371
25.1.1 DSI_InitTypeDef 371
25.1.2 DSI_PLLInitTypeDef 371
25.1.3 DSI_VidCfgTypeDef 371
25.1.4 DSI_CmdCfgTypeDef 373
25.1.5 DSI_LPCmdTypeDef 374
25.1.6 DSI_PHY_TimerTypeDef 375
25.1.7 DSI_HOST_TimeoutTypeDef 375
25.1.8 __DSI_HandleTypeDef 376
25.2 DSI Firmware driver API description 377
25.2.1 How to use this driver 377
25.2.2 Initialization and Configuration functions 379
25.2.3 IO operation functions 379
25.2.4 Peripheral Control functions 379
25.2.5 Peripheral State and Errors functions 380
25.2.6 Detailed description of functions 380
25.3 DSI Firmware driver defines 393
25.3.1 DSI 393
26 HAL DTS Generic Driver 406
26.1 DTS Firmware driver registers structures 406
26.1.1 DTS_InitTypeDef 406

26.1.2 __DTS_HandleTypeDef . . . 406
26.2 DTS Firmware driver API description . . . 407
26.2.1 DTS Peripheral features . . . 407
26.2.2 How to use this driver . . . 407
26.2.3 Initialization and de-initialization functions . . . 407
26.2.4 DTS Start Stop operation functions . . . 407
26.2.5 Peripheral State functions . . . 408
26.2.6 Detailed description of functions . . . 408
26.3 DTS Firmware driver defines . . . 413
26.3.1 DTS . . . 413
27 HAL ETH Generic Driver . . . 420
27.1 ETH Firmware driver registers structures . . . 420
27.1.1 ETH_DMADescTypeDef . . . 420
27.1.2 __ETH_BufferTypeDef . . . 420
27.1.3 ETH_TxDescListTypeDef . . . 420
27.1.4 ETH_TxPacketConfig . . . 421
27.1.5 ETH_TimeStampTypeDef . . . 422
27.1.6 ETH_TimeTypeDef . . . 422
27.1.7 ETH_RxDescListTypeDef . . . 422
27.1.8 ETH_MACConfigTypeDef . . . 423
27.1.9 ETH_DMAConfigTypeDef . . . 426
27.1.10 ETH_InitTypeDef . . . 426
27.1.11 ETH_PTP_ConfigTypeDef . . . 427
27.1.12 __ETH_HandleTypeDef . . . 428
27.1.13 ETH_MACFilterConfigTypeDef . . . 430
27.1.14 ETH_PowerDownConfigTypeDef . . . 430
27.2 ETH Firmware driver API description . . . 431
27.2.1 How to use this driver . . . 431
27.2.2 Initialization and Configuration functions . . . 432
27.2.3 IO operation functions . . . 432
27.2.4 Peripheral Control functions . . . 433
27.2.5 Peripheral State and Errors functions . . . 433

27.2.6 Detailed description of functions . . . 434
27.3 ETH Firmware driver defines . . . 448
27.3.1 ETH . . . 448
28 HAL ETH Extension Driver . . . 470
28.1 ETHEx Firmware driver registers structures . . . 470
28.1.1 ETH_RxVLANConfigTypeDef . . . 470
28.1.2 ETH_TxVLANConfigTypeDef . . . 470
28.1.3 ETH_L3FilterConfigTypeDef . . . 471
28.1.4 ETH_L4FilterConfigTypeDef . . . 471
28.2 ETHEx Firmware driver API description . . . 472
28.2.1 Extended features functions . . . 472
28.2.2 Detailed description of functions . . . 472
28.3 ETHEx Firmware driver defines . . . 478
28.3.1 ETHEx . . . 478
29 HAL EXTI Generic Driver . . . 481
29.1 EXTI Firmware driver registers structures . . . 481
29.1.1 EXTI_HandleTypeDef . . . 481
29.1.2 EXTI_ConfigTypeDef . . . 481
29.2 EXTI Firmware driver API description . . . 481
29.2.1 EXTI Peripheral features . . . 481
29.2.2 How to use this driver . . . 482
29.2.3 Configuration functions . . . 483
29.2.4 Detailed description of functions . . . 483
29.3 EXTI Firmware driver defines . . . 485
29.3.1 EXTI . . . 485
30 HAL FDCAN Generic Driver . . . 495
30.1 FDCAN Firmware driver registers structures . . . 495
30.1.1 FDCAN_InitTypeDef . . . 495
30.1.2 FDCAN_ClkCalUnitTypeDef . . . 497
30.1.3 FDCAN_FilterTypeDef . . . 497
30.1.4 FDCAN_TxHeaderTypeDef . . . 498
30.1.5 FDCAN_RxHeaderTypeDef . . . 499

30.1.6 FDCAN_TxEventFifoTypeDef . . . 500
30.1.7 FDCAN_HpMsgStatusTypeDef . . . 501
30.1.8 FDCAN_ProtocolStatusTypeDef . . . 501
30.1.9 FDCAN_ErrorCountersTypeDef . . . 502
30.1.10 FDCAN_TT_ConfigTypeDef . . . 503
30.1.11 FDCAN_TriggerTypeDef . . . 504
30.1.12 FDCAN_TTOperationStatusTypeDef . . . 505
30.1.13 FDCAN_MsgRamAddressTypeDef . . . 506
30.1.14 __FDCAN_HandleTypeDef . . . 507
30.2 FDCAN Firmware driver API description . . . 509
30.2.1 How to use this driver . . . 509
30.2.2 Initialization and de-initialization functions . . . 511
30.2.3 Configuration functions . . . 511
30.2.4 Control functions . . . 513
30.2.5 TT Configuration and control functions . . . 514
30.2.6 Interrupts management . . . 514
30.2.7 Callback functions . . . 515
30.2.8 Peripheral State functions . . . 516
30.2.9 Detailed description of functions . . . 516
30.3 FDCAN Firmware driver defines . . . 553
30.3.1 FDCAN . . . 553
31 HAL FLASH Generic Driver . . . 580
31.1 FLASH Firmware driver registers structures . . . 580
31.1.1 FLASH_ProcessTypeDef . . . 580
31.2 FLASH Firmware driver API description . . . 580
31.2.1 FLASH peripheral features . . . 580
31.2.2 How to use this driver . . . 580
31.2.3 Programming operation functions . . . 581
31.2.4 Peripheral Control functions . . . 581
31.2.5 Peripheral Errors functions . . . 581
31.2.6 Detailed description of functions . . . 582
31.3 FLASH Firmware driver defines . . . 586

31.3.1 FLASH . . . 586
32 HAL FLASH Extension Driver . . . 599
32.1 FLASHEx Firmware driver registers structures . . . 599
32.1.1 FLASH_EraseInitTypeDef . . . 599
32.1.2 FLASH_OBProgramInitTypeDef . . . 599
32.1.3 FLASH_CRCInitTypeDef . . . 601
32.2 FLASHEx Firmware driver API description . . . 601
32.2.1 Flash Extension features . . . 601
32.2.2 How to use this driver . . . 602
32.2.3 Extended programming operation functions . . . 602
32.2.4 Detailed description of functions . . . 602
32.3 FLASHEx Firmware driver defines . . . 605
32.3.1 FLASHEx . . . 605
33 HAL FMAC Generic Driver . . . 615
33.1 FMAC Firmware driver registers structures . . . 615
33.1.1 __FMAC_HandleTypeDef . . . 615
33.1.2 FMAC_FilterConfigTypeDef . . . 617
33.2 FMAC Firmware driver API description . . . 618
33.2.1 How to use this driver . . . 618
33.2.2 Callback registration . . . 620
33.2.3 Initialization and de-initialization functions . . . 620
33.2.4 Peripheral Control functions . . . 621
33.2.5 Callback functions . . . 621
33.2.6 IRQ handler management . . . 621
33.2.7 Peripheral State and Error functions . . . 621
33.2.8 Detailed description of functions . . . 622
33.3 FMAC Firmware driver defines . . . 630
33.3.1 FMAC . . . 630
34 HAL GFXMMU Generic Driver . . . 636
34.1 GFXMMU Firmware driver registers structures . . . 636
34.1.1 GFXMMU_BuffersTypeDef . . . 636
34.1.2 GFXMMU_CachePrefetchTypeDef . . . 636

34.1.3 GFXMMU_InterruptsTypeDef . . . 637
34.1.4 GFXMMU_InitTypeDef . . . 637
34.1.5 __GFXMMU_HandleTypeDef . . . 637
34.1.6 GFXMMU_LutLineTypeDef . . . 638
34.2 GFXMMU Firmware driver API description . . . 638
34.2.1 How to use this driver . . . 639
34.2.2 Initialization and de-initialization functions . . . 640
34.2.3 Operation functions . . . 640
34.2.4 State functions . . . 640
34.2.5 Detailed description of functions . . . 641
34.3 GFXMMU Firmware driver defines . . . 645
34.3.1 GFXMMU . . . 645
35 HAL GPIO Generic Driver . . . 648
35.1 GPIO Firmware driver registers structures . . . 648
35.1.1 GPIO_InitTypeDef . . . 648
35.2 GPIO Firmware driver API description . . . 648
35.2.1 GPIO Peripheral features . . . 648
35.2.2 How to use this driver . . . 649
35.2.3 Initialization and de-initialization functions . . . 649
35.2.4 IO operation functions . . . 649
35.2.5 Detailed description of functions . . . 649
35.3 GPIO Firmware driver defines . . . 652
35.3.1 GPIO . . . 652
36 HAL GPIO Extension Driver . . . 661
36.1 GPIOEx Firmware driver defines . . . 661
36.1.1 GPIOEx . . . 661
37 HAL HASH Generic Driver . . . 662
37.1 HASH Firmware driver registers structures . . . 662
37.1.1 HASH_InitTypeDef . . . 662
37.1.2 __HASH_HandleTypeDef . . . 662
37.2 HASH Firmware driver API description . . . 664
37.2.1 How to use this driver . . . 664

37.2.2 Initialization and de-initialization functions . . . 666
37.2.3 Polling mode HASH processing functions . . . 667
37.2.4 Interruption mode HASH processing functions . . . 667
37.2.5 DMA mode HASH processing functions . . . 668
37.2.6 Polling mode HMAC processing functions . . . 668
37.2.7 Interrupt mode HMAC processing functions . . . 668
37.2.8 DMA mode HMAC processing functions . . . 668
37.2.9 Peripheral State methods . . . 669
37.2.10 Detailed description of functions . . . 669
37.3 HASH Firmware driver defines . . . 688
37.3.1 HASH . . . 688
38 HAL HASH Extension Driver . . . 693
38.1 HASHEx Firmware driver API description . . . 693
38.1.1 HASH peripheral extended features . . . 693
38.1.2 Polling mode HASH extended processing functions . . . 693
38.1.3 Interruption mode HASH extended processing functions . . . 694
38.1.4 DMA mode HASH extended processing functions . . . 694
38.1.5 Polling mode HMAC extended processing functions . . . 695
38.1.6 Interrupt mode HMAC extended processing functions . . . 695
38.1.7 DMA mode HMAC extended processing functions . . . 695
38.1.8 Multi-buffer DMA mode HMAC extended processing functions . . . 695
38.1.9 Detailed description of functions . . . 696
39 HAL HCD Generic Driver . . . 713
39.1 HCD Firmware driver registers structures . . . 713
39.1.1 __HCD_HandleTypeDef . . . 713
39.2 HCD Firmware driver API description . . . 714
39.2.1 How to use this driver . . . 714
39.2.2 Initialization and de-initialization functions . . . 714
39.2.3 IO operation functions . . . 714
39.2.4 Peripheral Control functions . . . 715
39.2.5 Peripheral State functions . . . 715
39.2.6 Detailed description of functions . . . 715

39.3 HCD Firmware driver defines . . . 723
39.3.1 HCD . . . 723
40 HAL HRTIM Generic Driver . . . 725
40.1 HRTIM Firmware driver registers structures . . . 725
40.1.1 HRTIM_InitTypeDef . . . 725
40.1.2 HRTIM_TimerParamTypeDef . . . 725
40.1.3 __HRTIM_HandleTypeDef . . . 726
40.1.4 HRTIM_TimeBaseCfgTypeDef . . . 728
40.1.5 HRTIM_SimpleOCChannelCfgTypeDef . . . 729
40.1.6 HRTIM_SimplePWMChannelCfgTypeDef . . . 729
40.1.7 HRTIM_SimpleCaptureChannelCfgTypeDef . . . 730
40.1.8 HRTIM_SimpleOnePulseChannelCfgTypeDef . . . 730
40.1.9 HRTIM_TimerCfgTypeDef . . . 731
40.1.10 HRTIM_CompareCfgTypeDef . . . 732
40.1.11 HRTIM_CaptureCfgTypeDef . . . 733
40.1.12 HRTIM_OutputCfgTypeDef . . . 733
40.1.13 HRTIM_TimerEventFilteringCfgTypeDef . . . 734
40.1.14 HRTIM_DeadTimeCfgTypeDef . . . 734
40.1.15 HRTIM_ChopperModeCfgTypeDef . . . 735
40.1.16 HRTIM_EventCfgTypeDef . . . 735
40.1.17 HRTIM_FaultCfgTypeDef . . . 736
40.1.18 HRTIM_BurstModeCfgTypeDef . . . 736
40.1.19 HRTIM_ADCTriggerCfgTypeDef . . . 737
40.2 HRTIM Firmware driver API description . . . 737
40.2.1 Simple mode versus waveform mode . . . 737
40.2.2 How to use this driver . . . 737
40.2.3 Initialization and Time Base Configuration functions . . . 741
40.2.4 Simple time base mode functions . . . 741
40.2.5 Simple output compare functions . . . 742
40.2.6 Simple PWM output functions . . . 742
40.2.7 Simple input capture functions . . . 743
40.2.8 Simple one pulse functions . . . 743

40.2.9 HRTIM configuration functions . . . 743
40.2.10 HRTIM timer configuration and control functions . . . 744
40.2.11 Peripheral State functions . . . 745
40.2.12 Detailed description of functions . . . 745
40.3 HRTIM Firmware driver defines . . . 801
40.3.1 HRTIM . . . 801
41 HAL HSEM Generic Driver . . . 849
41.1 HSEM Firmware driver API description . . . 849
41.1.1 How to use this driver . . . 849
41.1.2 HSEM Take and Release functions . . . 850
41.1.3 HSEM Set and Get Key functions . . . 850
41.1.4 HSEM IRQ handler management and Notification functions . . . 850
41.1.5 Detailed description of functions . . . 850
41.2 HSEM Firmware driver defines . . . 853
41.2.1 HSEM . . . 853
42 HAL I2C Generic Driver . . . 855
42.1 I2C Firmware driver registers structures . . . 855
42.1.1 I2C_InitTypeDef . . . 855
42.1.2 __I2C_HandleTypeDef . . . 855
42.2 I2C Firmware driver API description . . . 857
42.2.1 How to use this driver . . . 857
42.2.2 Initialization and de-initialization functions . . . 862
42.2.3 IO operation functions . . . 863
42.2.4 Peripheral State, Mode and Error functions . . . 865
42.2.5 Detailed description of functions . . . 865
42.3 I2C Firmware driver defines . . . 883
42.3.1 I2C . . . 883
43 HAL I2C Extension Driver . . . 890
43.1 I2CEx Firmware driver API description . . . 890
43.1.1 I2C peripheral Extended features . . . 890
43.1.2 How to use this driver . . . 890
43.1.3 Filter Mode Functions . . . 890

43.1.4 WakeUp Mode Functions . . . 890
43.1.5 Fast Mode Plus Functions . . . 890
43.1.6 Detailed description of functions . . . 891
43.2 I2CEx Firmware driver defines . . . 893
43.2.1 I2CEx . . . 893
44 HAL I2S Generic Driver . . . 895
44.1 I2S Firmware driver registers structures . . . 895
44.1.1 I2S_InitTypeDef . . . 895
44.1.2 __I2S_HandleTypeDef . . . 895
44.2 I2S Firmware driver API description . . . 897
44.2.1 How to use this driver . . . 897
44.2.2 Initialization and de-initialization functions . . . 900
44.2.3 IO operation functions . . . 900
44.2.4 Peripheral State and Errors functions . . . 901
44.2.5 Detailed description of functions . . . 901
44.3 I2S Firmware driver defines . . . 910
44.3.1 I2S . . . 910
45 HAL IRDA Generic Driver . . . 916
45.1 IRDA Firmware driver registers structures . . . 916
45.1.1 IRDA_InitTypeDef . . . 916
45.1.2 __IRDA_HandleTypeDef . . . 916
45.2 IRDA Firmware driver API description . . . 918
45.2.1 How to use this driver . . . 918
45.2.2 Callback registration . . . 920
45.2.3 Initialization and Configuration functions . . . 921
45.2.4 IO operation functions . . . 921
45.2.5 Peripheral State and Error functions . . . 923
45.2.6 Detailed description of functions . . . 923
45.3 IRDA Firmware driver defines . . . 934
45.3.1 IRDA . . . 934
46 HAL IRDA Extension Driver . . . 944
46.1 IRDAEx Firmware driver defines . . . 944

46.1.1 IRDAEx . . . 944
47 HAL IWDG Generic Driver . . . 945
47.1 IWDG Firmware driver registers structures . . . 945
47.1.1 IWDG_InitTypeDef . . . 945
47.1.2 IWDG_HandleTypeDef . . . 945
47.2 IWDG Firmware driver API description . . . 945
47.2.1 IWDG Generic features . . . 945
47.2.2 How to use this driver . . . 946
47.2.3 Initialization and Start functions . . . 946
47.2.4 IO operation functions . . . 946
47.2.5 Detailed description of functions . . . 947
47.3 IWDG Firmware driver defines . . . 947
47.3.1 IWDG . . . 947
48 HAL JPEG Generic Driver . . . 949
48.1 JPEG Firmware driver registers structures . . . 949
48.1.1 JPEG_ConfTypeDef . . . 949
48.1.2 __JPEG_HandleTypeDef . . . 949
48.2 JPEG Firmware driver API description . . . 951
48.2.1 How to use this driver . . . 951
48.2.2 Initialization and de-initialization functions . . . 953
48.2.3 Configuration functions . . . 953
48.2.4 JPEG processing functions . . . 953
48.2.5 JPEG Decode and Encode callback functions . . . 954
48.2.6 JPEG IRQ handler management . . . 954
48.2.7 Peripheral State and Error functions . . . 954
48.2.8 Detailed description of functions . . . 955
48.3 JPEG Firmware driver defines . . . 966
48.3.1 JPEG . . . 966
49 HAL LPTIM Generic Driver . . . 971
49.1 LPTIM Firmware driver registers structures . . . 971
49.1.1 LPTIM_ClockConfigTypeDef . . . 971
49.1.2 LPTIM_ULPClockConfigTypeDef . . . 971

49.1.3 LPTIM_TriggerConfigTypeDef . . . 971
49.1.4 LPTIM_InitTypeDef . . . 972
49.1.5 __LPTIM_HandleTypeDef . . . 972
49.2 LPTIM Firmware driver API description . . . 973
49.2.1 How to use this driver . . . 973
49.2.2 Initialization and de-initialization functions . . . 975
49.2.3 LPTIM Start Stop operation functions . . . 975
49.2.4 LPTIM Read operation functions . . . 976
49.2.5 Peripheral State functions . . . 976
49.2.6 Detailed description of functions . . . 976
49.3 LPTIM Firmware driver defines . . . 988
49.3.1 LPTIM . . . 988
50 HAL LTDC Generic Driver . . . 996
50.1 LTDC Firmware driver registers structures . . . 996
50.1.1 LTDC_ColorTypeDef . . . 996
50.1.2 LTDC_InitTypeDef . . . 996
50.1.3 LTDC_LayerCfgTypeDef . . . 997
50.1.4 __LTDC_HandleTypeDef . . . 998
50.2 LTDC Firmware driver API description . . . 999
50.2.1 How to use this driver . . . 999
50.2.2 Initialization and Configuration functions . . . 1001
50.2.3 IO operation functions . . . 1001
50.2.4 Peripheral Control functions . . . 1001
50.2.5 Peripheral State and Errors functions . . . 1002
50.2.6 Detailed description of functions . . . 1002
50.3 LTDC Firmware driver defines . . . 1015
50.3.1 LTDC . . . 1015
51 HAL MDIOS Generic Driver . . . 1022
51.1 MDIOS Firmware driver registers structures . . . 1022
51.1.1 MDIOS_InitTypeDef . . . 1022
51.1.2 __MDIOS_HandleTypeDef . . . 1022
51.2 MDIOS Firmware driver API description . . . 1023

51.2.1 How to use this driver . . . 1023
51.2.2 Initialization and Configuration functions . . . 1023
51.2.3 IO operation functions . . . 1023
51.2.4 Peripheral Control functions . . . 1024
51.2.5 Detailed description of functions . . . 1024
51.3 MDIOS Firmware driver defines . . . 1029
51.3.1 MDIOS . . . 1029
52 HAL MDMA Generic Driver . . . 1038
52.1 MDMA Firmware driver registers structures . . . 1038
52.1.1 MDMA_InitTypeDef . . . 1038
52.1.2 MDMA_LinkNodeTypeDef . . . 1039
52.1.3 MDMA_LinkNodeConfTypeDef . . . 1040
52.1.4 __MDMA_HandleTypeDef . . . 1040
52.2 MDMA Firmware driver API description . . . 1041
52.2.1 How to use this driver . . . 1041
52.2.2 Initialization and de-initialization functions . . . 1043
52.2.3 Linked list operation functions . . . 1044
52.2.4 IO operation functions . . . 1044
52.2.5 State and Errors functions . . . 1044
52.2.6 Detailed description of functions . . . 1045
52.3 MDMA Firmware driver defines . . . 1050
52.3.1 MDMA . . . 1050
53 HAL MMC Generic Driver . . . 1060
53.1 MMC Firmware driver registers structures . . . 1060
53.1.1 HAL_MMC_CardInfoTypeDef . . . 1060
53.1.2 __MMC_HandleTypeDef . . . 1060
53.1.3 HAL_MMC_CardCSDTypeDef . . . 1062
53.1.4 HAL_MMC_CardCIDTypeDef . . . 1064
53.2 MMC Firmware driver API description . . . 1065
53.2.1 How to use this driver . . . 1065
53.2.2 Initialization and de-initialization functions . . . 1068
53.2.3 IO operation functions . . . 1068

53.2.4 Peripheral Control functions . . . 1068
53.2.5 Detailed description of functions . . . 1069
53.3 MMC Firmware driver defines . . . 1081
53.3.1 MMC . . . 1081
54 HAL MMC Extension Driver . . . 1092
54.1 MMCEx Firmware driver API description . . . 1092
54.1.1 How to use this driver . . . 1092
54.1.2 Multibuffer functions . . . 1092
54.1.3 Detailed description of functions . . . 1092
55 HAL NAND Generic Driver . . . 1095
55.1 NAND Firmware driver registers structures . . . 1095
55.1.1 NAND_IDTypeDef . . . 1095
55.1.2 NAND_AddressTypeDef . . . 1095
55.1.3 NAND_DeviceConfigTypeDef . . . 1095
55.1.4 __NAND_HandleTypeDef . . . 1096
55.2 NAND Firmware driver API description . . . 1096
55.2.1 How to use this driver . . . 1096
55.2.2 NAND Initialization and de-initialization functions . . . 1097
55.2.3 NAND Input and Output functions . . . 1098
55.2.4 NAND Control functions . . . 1098
55.2.5 NAND State functions . . . 1098
55.2.6 Detailed description of functions . . . 1098
55.3 NAND Firmware driver defines . . . 1106
55.3.1 NAND . . . 1106
56 HAL NOR Generic Driver . . . 1107
56.1 NOR Firmware driver registers structures . . . 1107
56.1.1 NOR_IDTypeDef . . . 1107
56.1.2 NOR_CFITypeDef . . . 1107
56.1.3 __NOR_HandleTypeDef . . . 1107
56.2 NOR Firmware driver API description . . . 1108
56.2.1 How to use this driver . . . 1108
56.2.2 NOR Initialization and de_initialization functions . . . 1109

56.2.3 NOR Input and Output functions . . . 1109
56.2.4 NOR Control functions . . . 1109
56.2.5 NOR State functions . . . 1109
56.2.6 Detailed description of functions . . . 1110
56.3 NOR Firmware driver defines . . . 1116
56.3.1 NOR . . . 1116
57 HAL OPAMP Generic Driver . . . 1117
57.1 OPAMP Firmware driver registers structures . . . 1117
57.1.1 OPAMP_InitTypeDef . . . 1117
57.1.2 __OPAMP_HandleTypeDef . . . 1118
57.2 OPAMP Firmware driver API description . . . 1118
57.2.1 OPAMP Peripheral Features . . . 1118
57.2.2 How to use this driver . . . 1119
57.2.3 Initialization and de-initialization functions . . . 1120
57.2.4 IO operation functions . . . 1120
57.2.5 Peripheral Control functions . . . 1121
57.2.6 Peripheral State functions . . . 1121
57.2.7 Detailed description of functions . . . 1121
57.3 OPAMP Firmware driver defines . . . 1125
57.3.1 OPAMP . . . 1125
58 HAL OPAMP Extension Driver . . . 1127
58.1 OPAMPEx Firmware driver API description . . . 1127
58.1.1 Extended IO operation functions . . . 1127
58.1.2 Peripheral Control functions . . . 1127
58.1.3 Detailed description of functions . . . 1127
59 HAL OSPI Generic Driver . . . 1129
59.1 OSPI Firmware driver registers structures . . . 1129
59.1.1 OSPI_InitTypeDef . . . 1129
59.1.2 __OSPI_HandleTypeDef . . . 1130
59.1.3 OSPI_RegularCmdTypeDef . . . 1131
59.1.4 OSPI_HyperbusCfgTypeDef . . . 1133
59.1.5 OSPI_HyperbusCmdTypeDef . . . 1133

59.1.6 OSPI_AutoPollingTypeDef . . . 1133
59.1.7 OSPI_MemoryMappedTypeDef . . . 1134
59.1.8 OSPIM_CfgTypeDef . . . 1134
59.2 OSPI Firmware driver API description . . . 1135
59.2.1 How to use this driver . . . 1135
59.2.2 Initialization and Configuration functions . . . 1138
59.2.3 IO operation functions . . . 1138
59.2.4 Peripheral Control and State functions . . . 1139
59.2.5 IO Manager configuration function . . . 1139
59.2.6 Detailed description of functions . . . 1140
59.3 OSPI Firmware driver defines . . . 1151
59.3.1 OSPI . . . 1151
60 HAL OTFDEC Generic Driver . . . 1161
60.1 OTFDEC Firmware driver registers structures . . . 1161
60.1.1 OTFDEC_RegionConfigTypeDef . . . 1161
60.1.2 __OTFDEC_HandleTypeDef . . . 1161
60.2 OTFDEC Firmware driver API description . . . 1162
60.2.1 How to use this driver . . . 1162
60.2.2 Initialization and de-initialization functions . . . 1163
60.2.3 OTFDEC IRQ handler management . . . 1163
60.2.4 Peripheral Control functions . . . 1163
60.2.5 Peripheral State functions . . . 1163
60.2.6 Detailed description of functions . . . 1163
60.3 OTFDEC Firmware driver defines . . . 1169
60.3.1 OTFDEC . . . 1169
61 HAL PCD Generic Driver . . . 1173
61.1 PCD Firmware driver registers structures . . . 1173
61.1.1 __PCD_HandleTypeDef . . . 1173
61.2 PCD Firmware driver API description . . . 1175
61.2.1 How to use this driver . . . 1175
61.2.2 Initialization and de-initialization functions . . . 1175
61.2.3 IO operation functions . . . 1176

61.2.4 Peripheral Control functions . . . 1176
61.2.5 Peripheral State functions . . . 1176
61.2.6 Detailed description of functions . . . 1176
61.3 PCD Firmware driver defines . . . 1190
61.3.1 PCD . . . 1190
62 HAL PCD Extension Driver . . . 1191
62.1 PCDEx Firmware driver API description . . . 1191
62.1.1 Extended features functions . . . 1191
62.1.2 Detailed description of functions . . . 1191
63 HAL PSSI Generic Driver . . . 1194
63.1 PSSI Firmware driver registers structures . . . 1194
63.1.1 PSSI_InitTypeDef . . . 1194
63.1.2 __PSSI_HandleTypeDef . . . 1194
63.2 PSSI Firmware driver API description . . . 1195
63.2.1 How to use this driver . . . 1195
63.2.2 Initialization and de-initialization functions . . . 1197
63.2.3 IO operation functions . . . 1197
63.2.4 Peripheral State, Mode and Error functions . . . 1197
63.2.5 Detailed description of functions . . . 1198
63.3 PSSI Firmware driver defines . . . 1203
63.3.1 PSSI . . . 1203
64 HAL PWR Generic Driver . . . 1209
64.1 PWR Firmware driver registers structures . . . 1209
64.1.1 PWR_PVDTypeDef . . . 1209
64.2 PWR Firmware driver API description . . . 1209
64.2.1 PWR peripheral overview . . . 1209
64.2.2 How to use this driver . . . 1210
64.2.3 Initialization and De-Initialization Functions . . . 1211
64.2.4 Peripheral Control Functions . . . 1212
64.2.5 Interrupt Handling Functions . . . 1213
64.2.6 Detailed description of functions . . . 1213
64.3 PWR Firmware driver defines . . . 1219

64.3.1 PWR . . . 1219
65 HAL PWR Extension Driver . . . 1229
65.1 PWREx Firmware driver registers structures . . . 1229
65.1.1 PWREx_AVDTypeDef . . . 1229
65.1.2 PWREx_WakeupPinTypeDef . . . 1229
65.2 PWREx Firmware driver API description . . . 1229
65.2.1 How to use this driver . . . 1229
65.2.2 Power supply control functions . . . 1232
65.2.3 Low power control functions . . . 1233
65.2.4 Peripherals control functions . . . 1236
65.2.5 Power Monitoring functions . . . 1237
65.2.6 Detailed description of functions . . . 1237
65.3 PWREx Firmware driver defines . . . 1250
65.3.1 PWREx . . . 1250
66 HAL QSPI Generic Driver . . . 1258
66.1 QSPI Firmware driver registers structures . . . 1258
66.1.1 QSPI_InitTypeDef . . . 1258
66.1.2 __QSPI_HandleTypeDef . . . 1258
66.1.3 QSPI_CommandTypeDef . . . 1259
66.1.4 QSPI_AutoPollingTypeDef . . . 1260
66.1.5 QSPI_MemoryMappedTypeDef . . . 1260
66.2 QSPI Firmware driver API description . . . 1261
66.2.1 How to use this driver . . . 1261
66.2.2 Initialization and Configuration functions . . . 1264
66.2.3 IO operation functions . . . 1264
66.2.4 Peripheral Control and State functions . . . 1265
66.2.5 Detailed description of functions . . . 1265
66.3 QSPI Firmware driver defines . . . 1275
66.3.1 QSPI . . . 1275
67 HAL RAMECC Generic Driver . . . 1283
67.1 RAMECC Firmware driver registers structures . . . 1283
67.1.1 __RAMECC_HandleTypeDef . . . 1283

67.2 RAMECC Firmware driver API description 1283
67.2.1 How to use this driver 1283
67.2.2 Initialization and de-initialization functions 1284
67.2.3 Monitoring operation functions 1284
67.2.4 Error information functions 1284
67.2.5 State and Error Functions 1285
67.2.6 Detailed description of functions 1285
67.3 RAMECC Firmware driver defines 1289
67.3.1 RAMECC 1289
68 HAL RCC Generic Driver 1294
68.1 RCC Firmware driver registers structures 1294
68.1.1 RCC_PLLInitTypeDef 1294
68.1.2 RCC_OscInitTypeDef 1294
68.1.3 RCC_ClkInitTypeDef 1295
68.2 RCC Firmware driver API description 1296
68.2.1 RCC specific features 1296
68.2.2 RCC Limitations 1296
68.2.3 Initialization and de-initialization functions 1297
68.2.4 Peripheral Control functions 1297
68.2.5 Detailed description of functions 1298
68.3 RCC Firmware driver defines 1303
68.3.1 RCC 1303
69 HAL RCC Extension Driver 1398
69.1 RCCEx Firmware driver registers structures 1398
69.1.1 RCC_PLL2InitTypeDef 1398
69.1.2 RCC_PLL3InitTypeDef 1398
69.1.3 PLL1_ClocksTypeDef 1399
69.1.4 PLL2_ClocksTypeDef 1399
69.1.5 PLL3_ClocksTypeDef 1400
69.1.6 RCC_PeriphCLKInitTypeDef 1400
69.1.7 RCC_CRSInitTypeDef 1402
69.1.8 RCC_CRSSynchroInfoTypeDef 1403

69.2 RCCEx Firmware driver API description . . . 1403
69.2.1 Extended Peripheral Control functions . . . 1403
69.2.2 Extended Clock Recovery System Control functions . . . 1403
69.2.3 Detailed description of functions . . . 1405
69.3 RCCEx Firmware driver defines . . . 1413
69.3.1 RCCEx . . . 1413
70 HAL RNG Generic Driver . . . 1466
70.1 RNG Firmware driver registers structures . . . 1466
70.1.1 RNG_InitTypeDef . . . 1466
70.1.2 __RNG_HandleTypeDef . . . 1466
70.2 RNG Firmware driver API description . . . 1467
70.2.1 How to use this driver . . . 1467
70.2.2 Callback registration . . . 1467
70.2.3 Initialization and configuration functions . . . 1467
70.2.4 Peripheral Control functions . . . 1468
70.2.5 Peripheral State functions . . . 1468
70.2.6 Detailed description of functions . . . 1468
70.3 RNG Firmware driver defines . . . 1473
70.3.1 RNG . . . 1473
71 HAL RNG Extension Driver . . . 1477
71.1 RNGEx Firmware driver registers structures . . . 1477
71.1.1 RNG_ConfigTypeDef . . . 1477
71.2 RNGEx Firmware driver defines . . . 1477
71.2.1 RNGEx . . . 1477
72 HAL RTC Generic Driver . . . 1478
72.1 RTC Firmware driver registers structures . . . 1478
72.1.1 RTC_InitTypeDef . . . 1478
72.1.2 RTC_TimeTypeDef . . . 1478
72.1.3 RTC_DateTypeDef . . . 1479
72.1.4 RTC_AlarmTypeDef . . . 1479
72.1.5 __RTC_HandleTypeDef . . . 1480
72.2 RTC Firmware driver API description . . . 1481

72.2.1 RTC Operating Condition . . . 1481
72.2.2 Backup Domain Reset . . . 1481
72.2.3 Backup Domain Access . . . 1481
72.2.4 How to use RTC Driver . . . 1481
72.2.5 RTC and low power modes . . . 1482
72.2.6 Initialization and de-initialization functions . . . 1482
72.2.7 RTC Time and Date functions . . . 1482
72.2.8 RTC Alarm functions . . . 1483
72.2.9 Peripheral Control functions . . . 1483
72.2.10 Peripheral State functions . . . 1483
72.2.11 Detailed description of functions . . . 1483
72.3 RTC Firmware driver defines . . . 1491
72.3.1 RTC . . . 1491
73 HAL RTC Extension Driver . . . 1503
73.1 RTCEx Firmware driver registers structures . . . 1503
73.1.1 RTC_TamperTypeDef . . . 1503
73.2 RTCEx Firmware driver API description . . . 1503
73.2.1 How to use this driver . . . 1504
73.2.2 RTC TimeStamp and Tamper functions . . . 1504
73.2.3 Tamper functions . . . 1505
73.2.4 RTC Wake-up functions . . . 1505
73.2.5 Extended RTC Backup register functions . . . 1505
73.2.6 Extended Peripheral Control functions . . . 1506
73.2.7 Extended features functions . . . 1506
73.2.8 Detailed description of functions . . . 1506
73.3 RTCEx Firmware driver defines . . . 1518
73.3.1 RTCEx . . . 1518
74 HAL SAI Generic Driver . . . 1541
74.1 SAI Firmware driver registers structures . . . 1541
74.1.1 SAI_PdmInitTypeDef . . . 1541
74.1.2 SAI_InitTypeDef . . . 1541
74.1.3 SAI_FrameInitTypeDef . . . 1543

74.1.4 SAI_SlotInitTypeDef . . . 1543
74.1.5 __SAI_HandleTypeDef . . . 1544
74.2 SAI Firmware driver API description . . . 1545
74.2.1 How to use this driver . . . 1545
74.2.2 Initialization and de-initialization functions . . . 1548
74.2.3 IO operation functions . . . 1548
74.2.4 Peripheral State and Errors functions . . . 1549
74.2.5 Detailed description of functions . . . 1549
74.3 SAI Firmware driver defines . . . 1557
74.3.1 SAI . . . 1557
75 HAL SAI Extension Driver . . . 1566
75.1 SAIEx Firmware driver registers structures . . . 1566
75.1.1 SAIEx_PdmMicDelayParamTypeDef . . . 1566
75.2 SAIEx Firmware driver API description . . . 1566
75.2.1 Extended features functions . . . 1566
75.2.2 Detailed description of functions . . . 1566
76 HAL SDRAM Generic Driver . . . 1567
76.1 SDRAM Firmware driver registers structures . . . 1567
76.1.1 __SDRAM_HandleTypeDef . . . 1567
76.2 SDRAM Firmware driver API description . . . 1567
76.2.1 How to use this driver . . . 1567
76.2.2 SDRAM Initialization and de_initialization functions . . . 1568
76.2.3 SDRAM Input and Output functions . . . 1569
76.2.4 SDRAM Control functions . . . 1569
76.2.5 SDRAM State functions . . . 1569
76.2.6 Detailed description of functions . . . 1569
76.3 SDRAM Firmware driver defines . . . 1577
76.3.1 SDRAM . . . 1577
77 HAL SD Generic Driver . . . 1579
77.1 SD Firmware driver registers structures . . . 1579
77.1.1 HAL_SD_CardInfoTypeDef . . . 1579
77.1.2 __SD_HandleTypeDef . . . 1579

77.1.3 HAL_SD_CardCSDTypeDef . . . 1581
77.1.4 HAL_SD_CardCIDTypeDef . . . 1583
77.1.5 HAL_SD_CardStatusTypeDef . . . 1584
77.2 SD Firmware driver API description . . . 1585
77.2.1 How to use this driver . . . 1585
77.2.2 Initialization and de-initialization functions . . . 1587
77.2.3 IO operation functions . . . 1587
77.2.4 Peripheral Control functions . . . 1588
77.2.5 Detailed description of functions . . . 1588
77.3 SD Firmware driver defines . . . 1598
77.3.1 SD . . . 1598
78 HAL SD Extension Driver . . . 1608
78.1 SDEx Firmware driver API description . . . 1608
78.1.1 How to use this driver . . . 1608
78.1.2 Multibuffer functions . . . 1608
78.1.3 Detailed description of functions . . . 1608
79 HAL SMARTCARD Generic Driver . . . 1611
79.1 SMARTCARD Firmware driver registers structures . . . 1611
79.1.1 SMARTCARD_InitTypeDef . . . 1611
79.1.2 SMARTCARD_AdvFeatureInitTypeDef . . . 1612
79.1.3 __SMARTCARD_HandleTypeDef . . . 1613
79.2 SMARTCARD Firmware driver API description . . . 1615
79.2.1 How to use this driver . . . 1615
79.2.2 Callback registration . . . 1617
79.2.3 Initialization and Configuration functions . . . 1618
79.2.4 IO operation functions . . . 1618
79.2.5 Peripheral State and Errors functions . . . 1620
79.2.6 Detailed description of functions . . . 1620
79.3 SMARTCARD Firmware driver defines . . . 1630
79.3.1 SMARTCARD . . . 1630
80 HAL SMARTCARD Extension Driver . . . 1643
80.1 SMARTCARDEx Firmware driver API description . . . 1643

80.1.1 SMARTCARD peripheral extended features . . . 1643
80.1.2 Peripheral Control functions . . . 1643
80.1.3 IO operation functions . . . 1643
80.1.4 Peripheral FIFO Control functions . . . 1643
80.1.5 Detailed description of functions . . . 1644
80.2 SMARTCARDEx Firmware driver defines . . . 1647
80.2.1 SMARTCARDEx . . . 1647
81 HAL SMBUS Generic Driver . . . 1652
81.1 SMBUS Firmware driver registers structures . . . 1652
81.1.1 SMBUS_InitTypeDef . . . 1652
81.1.2 __SMBUS_HandleTypeDef . . . 1653
81.2 SMBUS Firmware driver API description . . . 1654
81.2.1 How to use this driver . . . 1654
81.2.2 Initialization and de-initialization functions . . . 1656
81.2.3 IO operation functions . . . 1657
81.2.4 Peripheral State and Errors functions . . . 1658
81.2.5 Detailed description of functions . . . 1658
81.3 SMBUS Firmware driver defines . . . 1667
81.3.1 SMBUS . . . 1667
82 HAL SMBUS Extension Driver . . . 1675
82.1 SMBUSEx Firmware driver API description . . . 1675
82.1.1 SMBUS peripheral Extended features . . . 1675
82.1.2 How to use this driver . . . 1675
82.1.3 WakeUp Mode Functions . . . 1675
82.1.4 Fast Mode Plus Functions . . . 1675
82.1.5 Detailed description of functions . . . 1675
82.2 SMBUSEx Firmware driver defines . . . 1677
82.2.1 SMBUSEx . . . 1677
83 HAL SPDIFRX Generic Driver . . . 1678
83.1 SPDIFRX Firmware driver registers structures . . . 1678
83.1.1 SPDIFRX_InitTypeDef . . . 1678
83.1.2 SPDIFRX_SetDataFormatTypeDef . . . 1679

83.1.3 __SPDIFRX_HandleTypeDef . . . 1679
83.2 SPDIFRX Firmware driver API description . . . 1680
83.2.1 How to use this driver . . . 1680
83.2.2 Initialization and de-initialization functions . . . 1682
83.2.3 IO operation functions . . . 1682
83.2.4 Peripheral State and Errors functions . . . 1683
83.2.5 Detailed description of functions . . . 1683
83.3 SPDIFRX Firmware driver defines . . . 1690
83.3.1 SPDIFRX . . . 1690
84 HAL SPI Generic Driver . . . 1695
84.1 SPI Firmware driver registers structures . . . 1695
84.1.1 SPI_InitTypeDef . . . 1695
84.1.2 __SPI_HandleTypeDef . . . 1696
84.2 SPI Firmware driver API description . . . 1698
84.2.1 How to use this driver . . . 1698
84.2.2 Initialization and de-initialization functions . . . 1700
84.2.3 IO operation functions . . . 1701
84.2.4 Peripheral State and Errors functions . . . 1702
84.2.5 Detailed description of functions . . . 1702
84.3 SPI Firmware driver defines . . . 1711
84.3.1 SPI . . . 1711
85 HAL SPI Extension Driver . . . 1724
85.1 SPIEx Firmware driver API description . . . 1724
85.1.1 IO operation functions . . . 1724
85.1.2 Detailed description of functions . . . 1724
86 HAL SRAM Generic Driver . . . 1726
86.1 SRAM Firmware driver registers structures . . . 1726
86.1.1 __SRAM_HandleTypeDef . . . 1726
86.2 SRAM Firmware driver API description . . . 1726
86.2.1 How to use this driver . . . 1726
86.2.2 SRAM Initialization and de_initialization functions . . . 1727
86.2.3 SRAM Input and Output functions . . . 1728

86.2.4 SRAM Control functions . . . 1728
86.2.5 SRAM State functions . . . 1728
86.2.6 Detailed description of functions . . . 1728
86.3 SRAM Firmware driver defines . . . 1735
86.3.1 SRAM . . . 1735
87 HAL SWPMI Generic Driver . . . 1736
87.1 SWPMI Firmware driver registers structures . . . 1736
87.1.1 SWPMI_InitTypeDef . . . 1736
87.1.2 __SWPMI_HandleTypeDef . . . 1736
87.2 SWPMI Firmware driver API description . . . 1737
87.2.1 How to use this driver . . . 1737
87.2.2 Initialization and Configuration functions . . . 1739
87.2.3 IO operation methods . . . 1740
87.2.4 SWPMI IRQ handler and callbacks . . . 1741
87.2.5 Peripheral Control methods . . . 1741
87.2.6 Detailed description of functions . . . 1741
87.3 SWPMI Firmware driver defines . . . 1748
87.3.1 SWPMI . . . 1748
88 HAL TIM Generic Driver . . . 1754
88.1 TIM Firmware driver registers structures . . . 1754
88.1.1 TIM_Base_InitTypeDef . . . 1754
88.1.2 TIM_OC_InitTypeDef . . . 1754
88.1.3 TIM_OnePulse_InitTypeDef . . . 1755
88.1.4 TIM_IC_InitTypeDef . . . 1756
88.1.5 TIM_Encoder_InitTypeDef . . . 1756
88.1.6 TIM_ClockConfigTypeDef . . . 1757
88.1.7 TIM_ClearInputConfigTypeDef . . . 1757
88.1.8 TIM_MasterConfigTypeDef . . . 1758
88.1.9 TIM_SlaveConfigTypeDef . . . 1758
88.1.10 TIM_BreakDeadTimeConfigTypeDef . . . 1759
88.1.11 __TIM_HandleTypeDef . . . 1759
88.2 TIM Firmware driver API description . . . 1762

88.2.1 TIMER Generic features . . . 1762
88.2.2 How to use this driver . . . 1762
88.2.3 Time Base functions . . . 1764
88.2.4 TIM Output Compare functions . . . 1764
88.2.5 TIM PWM functions . . . 1765
88.2.6 TIM Input Capture functions . . . 1765
88.2.7 TIM One Pulse functions . . . 1766
88.2.8 TIM Encoder functions . . . 1766
88.2.9 TIM Callbacks functions . . . 1767
88.2.10 Detailed description of functions . . . 1767
88.3 TIM Firmware driver defines . . . 1806
88.3.1 TIM . . . 1806
89 HAL TIM Extension Driver . . . 1834
89.1 TIMEx Firmware driver registers structures . . . 1834
89.1.1 TIM_HallSensor_InitTypeDef . . . 1834
89.1.2 TIMEx_BreakInputConfigTypeDef . . . 1834
89.2 TIMEx Firmware driver API description . . . 1834
89.2.1 TIMER Extended features . . . 1834
89.2.2 How to use this driver . . . 1835
89.2.3 Timer Hall Sensor functions . . . 1835
89.2.4 Timer Complementary Output Compare functions . . . 1836
89.2.5 Timer Complementary PWM functions . . . 1836
89.2.6 Timer Complementary One Pulse functions . . . 1837
89.2.7 Peripheral Control functions . . . 1837
89.2.8 Extended Callbacks functions . . . 1837
89.2.9 Extended Peripheral State functions . . . 1837
89.2.10 Detailed description of functions . . . 1838
89.3 TIMEx Firmware driver defines . . . 1857
89.3.1 TIMEx . . . 1857
90 HAL UART Generic Driver . . . 1863
90.1 UART Firmware driver registers structures . . . 1863
90.1.1 UART_InitTypeDef . . . 1863

90.1.2 UART_AdvFeatureInitTypeDef . . . 1864
90.1.3 __UART_HandleTypeDef . . . 1864
90.2 UART Firmware driver API description . . . 1867
90.2.1 How to use this driver . . . 1867
90.2.2 Callback registration . . . 1868
90.2.3 Initialization and Configuration functions . . . 1869
90.2.4 IO operation functions . . . 1869
90.2.5 Peripheral Control functions . . . 1870
90.2.6 Peripheral State and Error functions . . . 1870
90.2.7 Detailed description of functions . . . 1871
90.3 UART Firmware driver defines . . . 1888
90.3.1 UART . . . 1888
91 HAL UART Extension Driver . . . 1908
91.1 UARTEx Firmware driver registers structures . . . 1908
91.1.1 UART_WakeUpTypeDef . . . 1908
91.2 UARTEx Firmware driver API description . . . 1908
91.2.1 UART peripheral extended features . . . 1908
91.2.2 Initialization and Configuration functions . . . 1908
91.2.3 IO operation functions . . . 1909
91.2.4 Peripheral Control functions . . . 1909
91.2.5 Detailed description of functions . . . 1910
91.3 UARTEx Firmware driver defines . . . 1916
91.3.1 UARTEx . . . 1916
92 HAL USART Generic Driver . . . 1918
92.1 USART Firmware driver registers structures . . . 1918
92.1.1 USART_InitTypeDef . . . 1918
92.1.2 __USART_HandleTypeDef . . . 1919
92.2 USART Firmware driver API description . . . 1921
92.2.1 How to use this driver . . . 1921
92.2.2 Callback registration . . . 1921
92.2.3 Initialization and Configuration functions . . . 1922
92.2.4 IO operation functions . . . 1923

92.2.5 Peripheral State and Error functions . . . 1924
92.2.6 Detailed description of functions . . . 1924
92.3 USART Firmware driver defines . . . 1934
92.3.1 USART . . . 1934
93 HAL USART Extension Driver . . . 1947
93.1 USARTEx Firmware driver API description . . . 1947
93.1.1 USART peripheral extended features . . . 1947
93.1.2 IO operation functions . . . 1947
93.1.3 Peripheral Control functions . . . 1947
93.1.4 Detailed description of functions . . . 1947
93.2 USARTEx Firmware driver defines . . . 1950
93.2.1 USARTEx . . . 1950
94 HAL WWDG Generic Driver . . . 1952
94.1 WWDG Firmware driver registers structures . . . 1952
94.1.1 WWDG_InitTypeDef . . . 1952
94.1.2 __WWDG_HandleTypeDef . . . 1952
94.2 WWDG Firmware driver API description . . . 1952
94.2.1 WWDG Specific features . . . 1952
94.2.2 How to use this driver . . . 1953
94.2.3 Initialization and Configuration functions . . . 1954
94.2.4 IO operation functions . . . 1954
94.2.5 Detailed description of functions . . . 1954
94.3 WWDG Firmware driver defines . . . 1957
94.3.1 WWDG . . . 1957
95 LL ADC Generic Driver . . . 1960
95.1 ADC Firmware driver registers structures . . . 1960
95.1.1 LL_ADC_CommonInitTypeDef . . . 1960
95.1.2 LL_ADC_InitTypeDef . . . 1960
95.1.3 LL_ADC_REG_InitTypeDef . . . 1961
95.1.4 LL_ADC_INJ_InitTypeDef . . . 1961
95.2 ADC Firmware driver API description . . . 1962
95.2.1 Detailed description of functions . . . 1962

95.3 ADC Firmware driver defines . . . 2074
95.3.1 ADC . . . 2074
96 LL BDMA Generic Driver . . . 2115
96.1 BDMA Firmware driver registers structures . . . 2115
96.1.1 LL_BDMA_InitTypeDef . . . 2115
96.2 BDMA Firmware driver API description . . . 2116
96.2.1 Detailed description of functions . . . 2116
96.3 BDMA Firmware driver defines . . . 2164
96.3.1 BDMA . . . 2164
97 LL BUS Generic Driver . . . 2171
97.1 BUS Firmware driver API description . . . 2171
97.1.1 Detailed description of functions . . . 2171
97.2 BUS Firmware driver defines . . . 2320
97.2.1 BUS . . . 2320
98 LL COMP Generic Driver . . . 2326
98.1 COMP Firmware driver registers structures . . . 2326
98.1.1 LL_COMP_InitTypeDef . . . 2326
98.2 COMP Firmware driver API description . . . 2326
98.2.1 Detailed description of functions . . . 2326
98.3 COMP Firmware driver defines . . . 2338
98.3.1 COMP . . . 2338
99 LL CORDIC Generic Driver . . . 2342
99.1 CORDIC Firmware driver API description . . . 2342
99.1.1 Detailed description of functions . . . 2342
99.2 CORDIC Firmware driver defines . . . 2355
99.2.1 CORDIC . . . 2355
100 LL CORTEX Generic Driver . . . 2358
100.1 CORTEX Firmware driver API description . . . 2358
100.1.1 Detailed description of functions . . . 2358
100.2 CORTEX Firmware driver defines . . . 2363
100.2.1 CORTEX . . . 2363

101 LL CRC Generic Driver . . . 2364
101.1 CRC Firmware driver API description . . . 2364
101.1.1 Detailed description of functions . . . 2364
101.2 CRC Firmware driver defines . . . 2371
101.2.1 CRC . . . 2371
102 LL CRS Generic Driver . . . 2374
102.1 CRS Firmware driver API description . . . 2374
102.1.1 Detailed description of functions . . . 2374
102.2 CRS Firmware driver defines . . . 2387
102.2.1 CRS . . . 2387
103 LL DAC Generic Driver . . . 2390
103.1 DAC Firmware driver registers structures . . . 2390
103.1.1 LL_DAC_InitTypeDef . . . 2390
103.2 DAC Firmware driver API description . . . 2390
103.2.1 Detailed description of functions . . . 2391
103.3 DAC Firmware driver defines . . . 2418
103.3.1 DAC . . . 2418
104 LL DELAYBLOCK Generic Driver . . . 2425
104.1 DELAYBLOCK Firmware driver API description . . . 2425
104.1.1 DelayBlock peripheral features . . . 2425
104.1.2 How to use this driver . . . 2425
104.1.3 Initialization and de-initialization functions . . . 2425
104.1.4 Detailed description of functions . . . 2425
104.2 DELAYBLOCK Firmware driver defines . . . 2426
104.2.1 DELAYBLOCK . . . 2426
105 LL DMA2D Generic Driver . . . 2427
105.1 DMA2D Firmware driver registers structures . . . 2427
105.1.1 LL_DMA2D_InitTypeDef . . . 2427
105.1.2 LL_DMA2D_LayerCfgTypeDef . . . 2429
105.1.3 LL_DMA2D_ColorTypeDef . . . 2431
105.2 DMA2D Firmware driver API description . . . 2433

105.2.1 Detailed description of functions . . . 2433
105.3 DMA2D Firmware driver defines . . . 2479
105.3.1 DMA2D . . . 2479
106 LL DMAMUX Generic Driver . . . 2483
106.1 DMAMUX Firmware driver API description . . . 2483
106.1.1 Detailed description of functions . . . 2483
106.2 DMAMUX Firmware driver defines . . . 2526
106.2.1 DMAMUX . . . 2526
107 LL DMA Generic Driver . . . 2535
107.1 DMA Firmware driver registers structures . . . 2535
107.1.1 LL_DMA_InitTypeDef . . . 2535
107.2 DMA Firmware driver API description . . . 2536
107.2.1 Detailed description of functions . . . 2537
107.3 DMA Firmware driver defines . . . 2604
107.3.1 DMA . . . 2604
108 LL EXTI Generic Driver . . . 2609
108.1 EXTI Firmware driver registers structures . . . 2609
108.1.1 LL_EXTI_InitTypeDef . . . 2609
108.2 EXTI Firmware driver API description . . . 2609
108.2.1 Detailed description of functions . . . 2609
108.3 EXTI Firmware driver defines . . . 2681
108.3.1 EXTI . . . 2681
109 LL FMAC Generic Driver . . . 2688
109.1 FMAC Firmware driver API description . . . 2688
109.1.1 Detailed description of functions . . . 2688
109.2 FMAC Firmware driver defines . . . 2710
109.2.1 FMAC . . . 2710
110 LL GPIO Generic Driver . . . 2712
110.1 GPIO Firmware driver registers structures . . . 2712
110.1.1 LL_GPIO_InitTypeDef . . . 2712
110.2 GPIO Firmware driver API description . . . 2712

110.2.1 Detailed description of functions . . . 2712
110.3 GPIO Firmware driver defines . . . 2732
110.3.1 GPIO . . . 2732
111 LL HRTIM Generic Driver . . . 2736
111.1 HRTIM Firmware driver API description . . . 2736
111.1.1 Detailed description of functions . . . 2736
111.2 HRTIM Firmware driver defines . . . 2939
111.2.1 HRTIM . . . 2939
112 LL HSEM Generic Driver . . . 2971
112.1 HSEM Firmware driver API description . . . 2971
112.1.1 Detailed description of functions . . . 2971
112.2 HSEM Firmware driver defines . . . 2988
112.2.1 HSEM . . . 2989
113 LL I2C Generic Driver . . . 2991
113.1 I2C Firmware driver registers structures . . . 2991
113.1.1 LL_I2C_InitTypeDef . . . 2991
113.2 I2C Firmware driver API description . . . 2991
113.2.1 Detailed description of functions . . . 2992
113.3 I2C Firmware driver defines . . . 3040
113.3.1 I2C . . . 3040
114 LL IWDG Generic Driver . . . 3046
114.1 IWDG Firmware driver API description . . . 3046
114.1.1 Detailed description of functions . . . 3046
114.2 IWDG Firmware driver defines . . . 3050
114.2.1 IWDG . . . 3050
115 LL LPTIM Generic Driver . . . 3052
115.1 LPTIM Firmware driver registers structures . . . 3052
115.1.1 LL_LPTIM_InitTypeDef . . . 3052
115.2 LPTIM Firmware driver API description . . . 3052
115.2.1 Detailed description of functions . . . 3052
115.3 LPTIM Firmware driver defines . . . 3082

115.3.1 LPTIM . . . 3083
116 LL LPUART Generic Driver . . . 3088
116.1 LPUART Firmware driver registers structures . . . 3088
116.1.1 LL_LPUART_InitTypeDef . . . 3088
116.2 LPUART Firmware driver API description . . . 3088
116.2.1 Detailed description of functions . . . 3089
116.3 LPUART Firmware driver defines . . . 3143
116.3.1 LPUART . . . 3143
117 LL MDMA Generic Driver . . . 3151
117.1 MDMA Firmware driver registers structures . . . 3151
117.1.1 LL_MDMA_InitTypeDef . . . 3151
117.1.2 LL_MDMA_LinkNodeTypeDef . . . 3154
117.2 MDMA Firmware driver API description . . . 3154
117.2.1 Detailed description of functions . . . 3155
117.3 MDMA Firmware driver defines . . . 3270
117.3.1 MDMA . . . 3270
118 LL OPAMP Generic Driver . . . 3279
118.1 OPAMP Firmware driver registers structures . . . 3279
118.1.1 LL_OPAMP_InitTypeDef . . . 3279
118.2 OPAMP Firmware driver API description . . . 3279
118.2.1 Detailed description of functions . . . 3279
118.3 OPAMP Firmware driver defines . . . 3290
118.3.1 OPAMP . . . 3290
119 LL PWR Generic Driver . . . 3294
119.1 PWR Firmware driver API description . . . 3294
119.1.1 Detailed description of functions . . . 3294
119.2 PWR Firmware driver defines . . . 3328
119.2.1 PWR . . . 3328
120 LL RCC Generic Driver . . . 3334
120.1 RCC Firmware driver registers structures . . . 3334
120.1.1 LL_RCC_ClocksTypeDef . . . 3334

120.1.2 LL_PLL_ClocksTypeDef . . . 3334
120.2 RCC Firmware driver API description . . . 3334
120.2.1 Detailed description of functions . . . 3334
120.3 RCC Firmware driver defines . . . 3443
120.3.1 RCC . . . 3443
121 LL RNG Generic Driver . . . 3460
121.1 RNG Firmware driver registers structures . . . 3460
121.1.1 LL_RNG_InitTypeDef . . . 3460
121.2 RNG Firmware driver API description . . . 3460
121.2.1 Detailed description of functions . . . 3460
121.3 RNG Firmware driver defines . . . 3466
121.3.1 RNG . . . 3466
122 LL RTC Generic Driver . . . 3468
122.1 RTC Firmware driver registers structures . . . 3468
122.1.1 LL_RTC_InitTypeDef . . . 3468
122.1.2 LL_RTC_TimeTypeDef . . . 3468
122.1.3 LL_RTC_DateTypeDef . . . 3468
122.1.4 LL_RTC_AlarmTypeDef . . . 3469
122.2 RTC Firmware driver API description . . . 3469
122.2.1 Detailed description of functions . . . 3470
122.3 RTC Firmware driver defines . . . 3549
122.3.1 RTC . . . 3549
123 LL SPI Generic Driver . . . 3561
123.1 SPI Firmware driver registers structures . . . 3561
123.1.1 LL_SPI_InitTypeDef . . . 3561
123.1.2 LL_I2S_InitTypeDef . . . 3562
123.2 SPI Firmware driver API description . . . 3562
123.2.1 Detailed description of functions . . . 3563
123.3 SPI Firmware driver defines . . . 3642
123.3.1 SPI . . . 3642
124 LL SWPMI Generic Driver . . . 3650

124.1 SWPMI Firmware driver registers structures . . . 3650
124.1.1 LL_SWPMI_InitTypeDef . . . 3650
124.2 SWPMI Firmware driver API description . . . 3650
124.2.1 Detailed description of functions . . . 3650
124.3 SWPMI Firmware driver defines . . . 3676
124.3.1 SWPMI . . . 3676
125 LL SYSTEM Generic Driver . . . 3680
125.1 SYSTEM Firmware driver API description . . . 3680
125.1.1 Detailed description of functions . . . 3680
125.2 SYSTEM Firmware driver defines . . . 3719
125.2.1 SYSTEM . . . 3719
126 LL TIM Generic Driver . . . 3727
126.1 TIM Firmware driver registers structures . . . 3727
126.1.1 LL_TIM_InitTypeDef . . . 3727
126.1.2 LL_TIM_OC_InitTypeDef . . . 3727
126.1.3 LL_TIM_IC_InitTypeDef . . . 3728
126.1.4 LL_TIM_ENCODER_InitTypeDef . . . 3729
126.1.5 LL_TIM_HALLSENSOR_InitTypeDef . . . 3729
126.1.6 LL_TIM_BDTR_InitTypeDef . . . 3730
126.2 TIM Firmware driver API description . . . 3731
126.2.1 Detailed description of functions . . . 3732
126.3 TIM Firmware driver defines . . . 3826
126.3.1 TIM . . . 3826
127 LL USART Generic Driver . . . 3849
127.1 USART Firmware driver registers structures . . . 3849
127.1.1 LL_USART_InitTypeDef . . . 3849
127.1.2 LL_USART_ClockInitTypeDef . . . 3849
127.2 USART Firmware driver API description . . . 3850
127.2.1 Detailed description of functions . . . 3850
127.3 USART Firmware driver defines . . . 3946
127.3.1 USART . . . 3946

128 LL UTILS Generic Driver 3956
128.1 UTILS Firmware driver registers structures 3956
128.1.1 LL_UTILS_PLLInitTypeDef 3956
128.1.2 LL_UTILS_ClkInitTypeDef 3956
128.2 UTILS Firmware driver API description 3957
128.2.1 System Configuration functions 3957
128.2.2 Detailed description of functions 3957
128.3 UTILS Firmware driver defines 3962
128.3.1 UTILS 3962
129 LL WWDG Generic Driver 3963
129.1 WWDG Firmware driver API description 3963
129.1.1 Detailed description of functions 3963
129.2 WWDG Firmware driver defines 3967
129.2.1 WWDG 3967
130 FAQs 3969
Revision history 3972
Contents 3973
List of tables 4018
List of figures 4019

List of tables

Table 1. Acronyms and definitions . . . 4
Table 2. HAL driver files . . . 11
Table 3. User-application files . . . 11
Table 4. API classification . . . 15
Table 5. List of devices supported by HAL drivers . . . 17
Table 6. HAL API naming rules . . . 21
Table 7. Macros handling interrupts and specific clock configurations . . . 22
Table 8. Callback functions . . . 23
Table 9. HAL generic APIs . . . 24
Table 10. HAL extension APIs . . . 25
Table 11. Define statements used for HAL configuration . . . 28
Table 12. Description of GPIO_InitTypeDef structure . . . 30
Table 13. Description of EXTI configuration macros . . . 31
Table 14. MSP functions . . . 37
Table 15. Timeout values . . . 41
Table 16. LL driver files . . . 45
Table 17. Common peripheral initialization functions . . . 48
Table 18. Optional peripheral initialization functions . . . 48
Table 19. Specific Interrupt, DMA request and status flags management . . . 49
Table 20. Available function formats . . . 49
Table 21. Peripheral clock activation/deactivation management . . . 50
Table 22. Peripheral activation/deactivation management . . . 50
Table 23. Peripheral configuration management . . . 50
Table 24. Peripheral register management . . . 50
Table 25. Document revision history . . . 3972

List of figures

Figure 1. Example of code portion used for dual-core architectures . . . 7
Figure 2. Dual-core project structure . . . 8
Figure 3. Dual-core example structure . . . 9
Figure 4. Example of project template . . . 12
Figure 5. Adding family-specific functions . . . 25
Figure 6. Adding new peripherals . . . 25
Figure 7. Updating existing APIs . . . 26
Figure 8. File inclusion model . . . 27
Figure 9. HAL driver model . . . 34
Figure 10. Low-layer driver folders . . . 46
Figure 11. Low-layer driver CMSIS files . . . 47

IMPORTANT NOTICE – READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgment.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2022 STMicroelectronics – All rights reserved