Debugging and tracing

Debugging

- In debugging mode,
  - CPU can be halted/restarted
  - programs can be run step-by-step
    - entering / not entering in subroutines
  - breakpoints can be inserted
  - registers and memory content can be visualized / modified
- Some specialized hardware is needed
  - inside the MCU
  - between the MCU and the IDE

JTAG

- JTAG port is often used to interface a chip for debugging one or more cores
- JTAG (Joint Test Action Group) is the name used for the IEEE 1149.1 standard entitled *Standard Test Access Port and Boundary-Scan Architecture* for test access ports (TAP) used for testing printed circuit boards (PCB) using boundary scan
- Functionalities:
  - *Debug Access:* to access the internals of a chip making its resources and functionality available and modifiable
    - e.g. registers, memories and system state
  - *Boundary Scan:* to test the physical connection of a device, e.g. on a PCB

JTAG

- Although TAP (Test Access Port) access itself is generic for all architectures, the functionality implemented behind JTAG is different for each device

- It uses a 4-pin interface (plus GND) to access the internal registers of the tested chip
- an in-circuit emulator (or, more correctly, a "JTAG adapter") uses JTAG as transport mechanism to access on-chip debug modules inside the target CPU

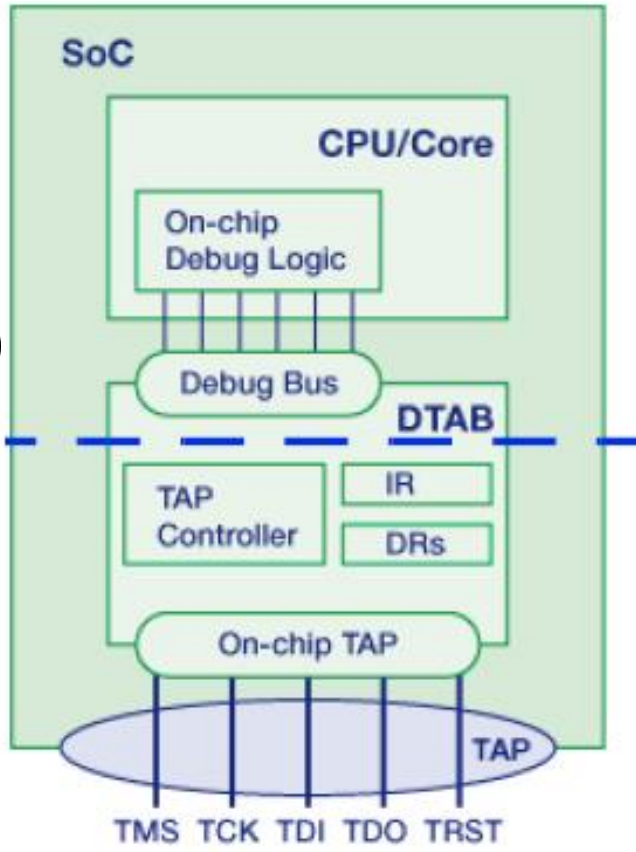
- JTAG also allows device programmer hardware to transfer data into internal non-volatile device memory

JTAG

Example of JTAG adapter

JTAG

- JTAG is defined as a serial communication protocol and a state machine accessible via a TAP
- The DTAB (Debug and Test Access Block) is implemented on the target chip as a “passive” device that never sends data without request
- The DTAB mainly consists of:
  - TAP (Test Access Port) with its physical connections (signals) to the external world
  - TAP Controller (a 16-state state machine)
  - one IR (Instruction Register) and several DRs (Data Registers)
  - The Debug Bus for communication with the on-chip debug logic



JTAG standard

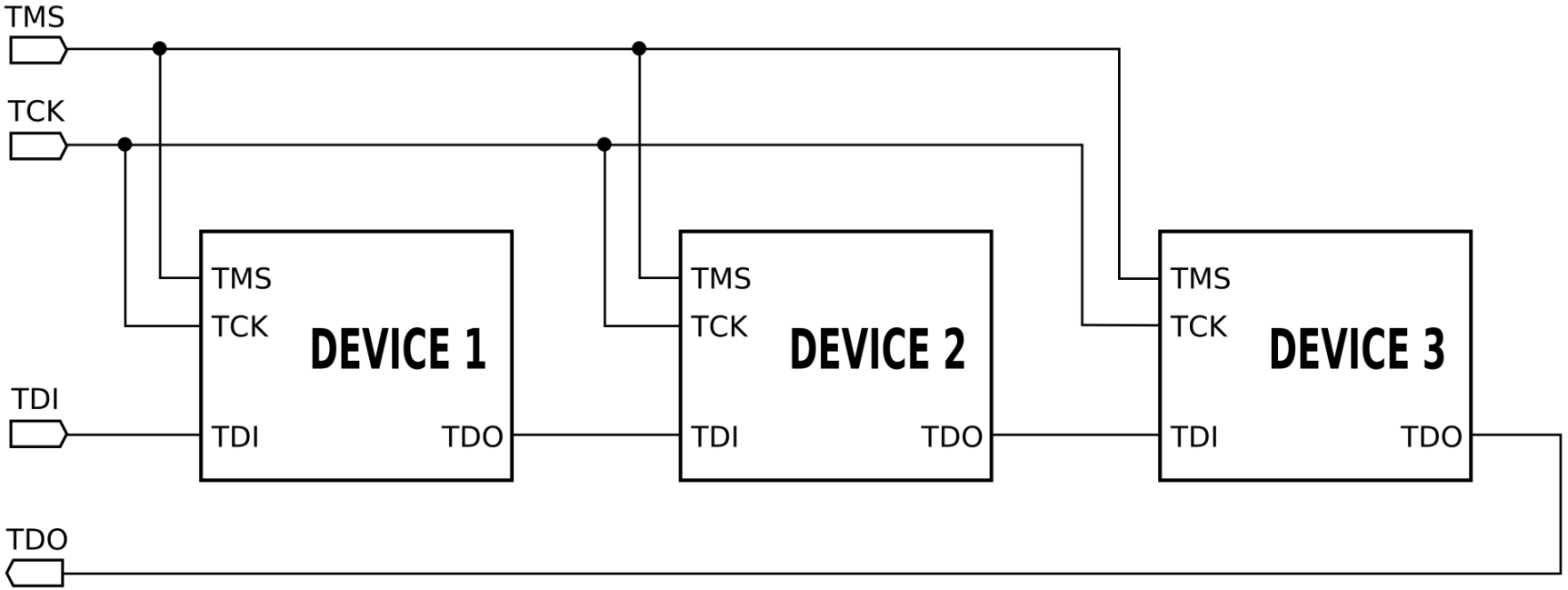
JTAG

- Basically, JTAG links all internal registers of the chip to a chain, which can be serially read from pin **TDO** (Test Data Out).
  - so, it is possible to read the contents of all registers
- the chain can also be shifted into the controller through pin **TDI** (Test Data In)
  - so modifications of registers are possible as well
- the interface is synchronous, so it requires a clock line, **TCK**
- the fourth pin of the interface is the test mode select pin, **TMS**
  - can be used to select different test modes
- optional fifth pin: test reset, **TRST**

- JTAG connectors can have different forms ☹

JTAG

- Several devices can be daisy-chained

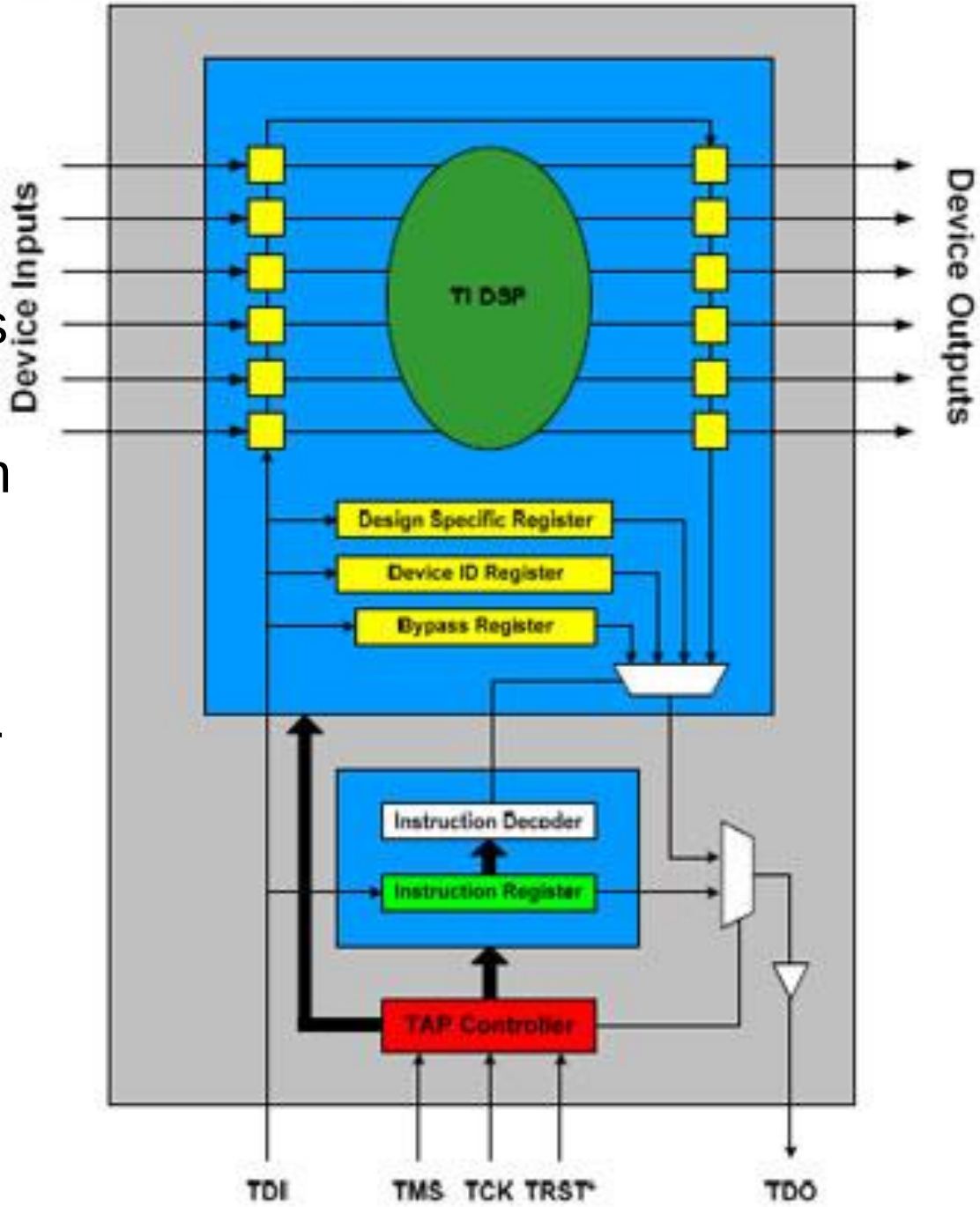


JTAG

- The protocol is serial
  - the clock input is at the TCK pin
  - one bit of data is transferred in from TDI out to TDO per TCK rising clock edge
- TCK frequency varies depending on the chips in the chain (lowest speed must be used); typically 10-100 MHz
  - some ARMs have a 6th pin, RTCK, for adaptive clocking

JTAG: boundary scan

- *Boundary scan:* used to test signal lines on PCBs, ICs or sub-blocks inside ICs
- this involves the addition of *cells* that are connected to the pins of the device and that can selectively override their functionality
- these cells are connected together to form the external boundary scan shift register, BSR



SWD (Serial Wire Debug)

- Alternative 2-pin (**SWD_DIO**, **SWD_CLK**, plus **GND**) electrical interface, uses the same protocol
- it is an ARM standard bi-directional wire protocol
- data rate is up to 4 MB/s at 50 MHz
- on JTAG devices with SWD capability, the TMS and TCK are used as SWD_DIO and SWD_CLK signals

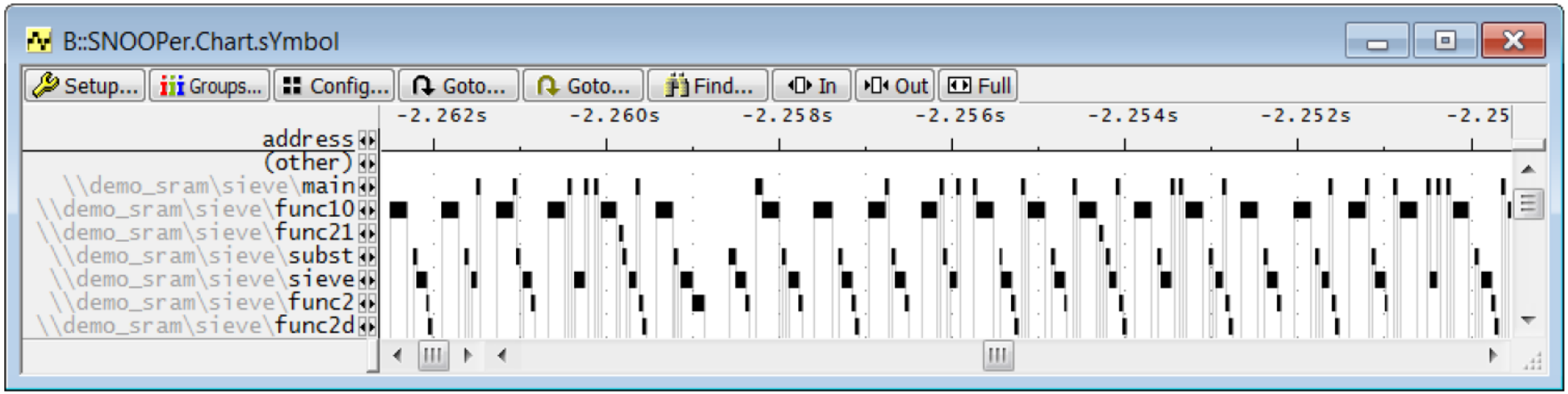
Other solutions

An external JTAG or SWD adapter is not strictly necessary for debugging:

- *in our boards, the job is done by ST-LINK*
- some chips support MRI (Monitor for Remote Inspection)
  - MRI is a debug monitor which allows GDB to debug Cortex-M3/M4 processors
    - the GNU Debugger (GDB) is a portable debugger that runs on many Unix-like systems
    - full featured source level debugger with no extra hardware other than a serial connection

Snooping

- JTAG can be used for *snooping*
  - the program counter is sampled periodically, so that the user has an idea on the routines that have been called
  - may be intrusive (i.e. the program is periodically stopped for a while) or not, depending on the h/w
  - the same approach can be used to sample a part of the memory, or some variables…



Tracing

- Tracing is used to accurately (*and non-invasively*) follow the behavior of the program moving from one routine to another
- trace data are generated for the instruction execution sequence and the task/process switches
  - basically, for each jump the new value of the program counter (PC) is saved in a *trace record*
- some systems also generate trace data for load/store operations (*data trace*)
- Tracing is used for
  - trace-based debugging
  - s/w optimization
    - also, energy profiling (using also an analog probe)…
  - s/w test and qualification
    - temporal requirements, code coverage…

Debugging vs tracing

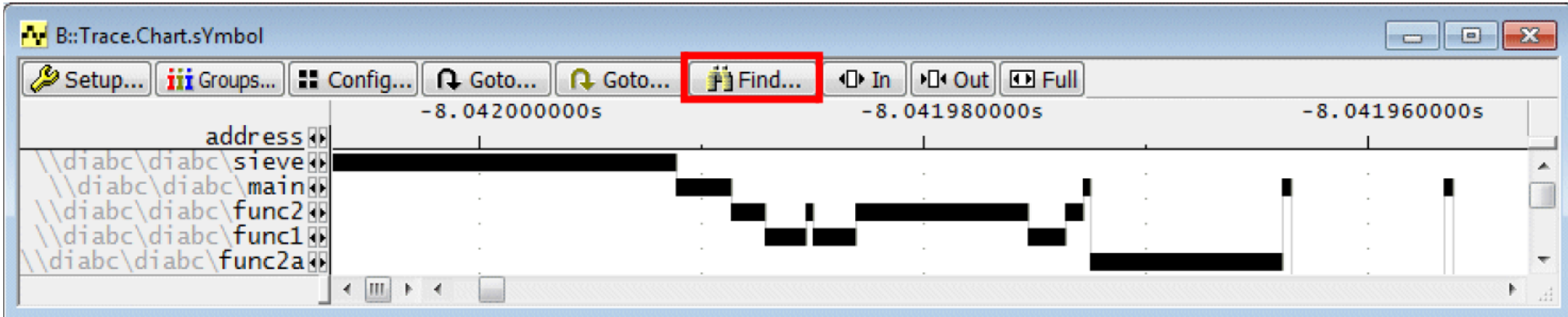
Debugging

Take a snapshot…

Real-Time Tracing

… take a video

Example of trace output:



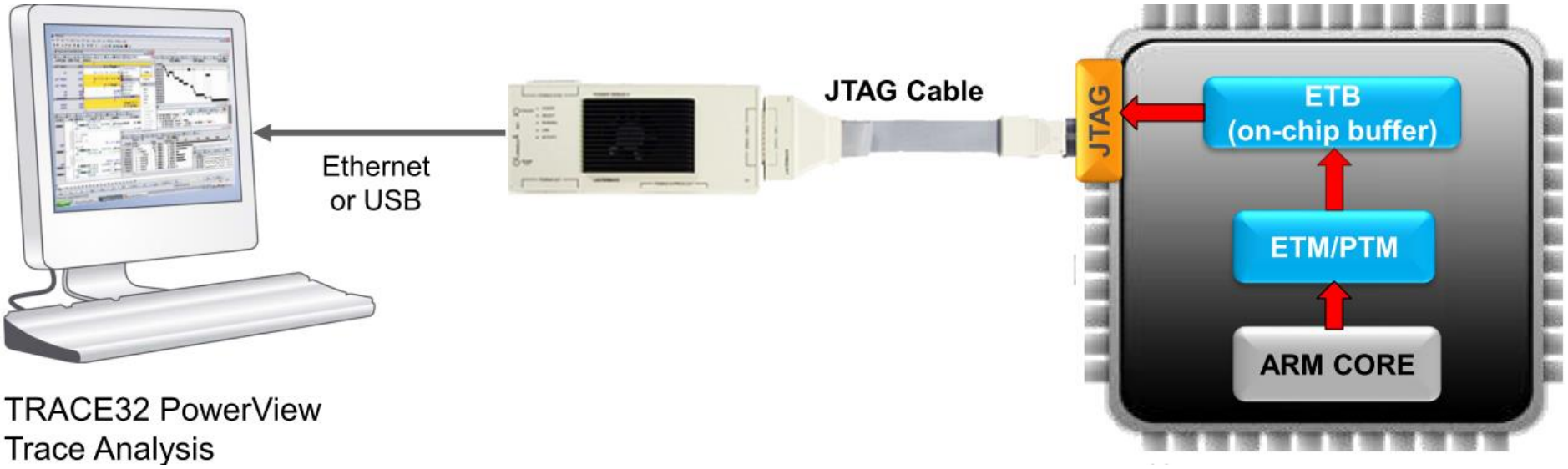
Tracing: ARM CoreSight

- In ARM/Cortex chips debug and trace h/w are built in the *ARM CoreSight* Design Kit
- main components:
  - [source] ***ETM*** *(Embedded Trace Macrocell):* provides instruction and data tracing
    - in Cortex-A9 and higher it is called ***PTM*** *(Program Trace Macrocell)*
  - [sink] ***ETB*** *(Embedded Trace Buffer)*: buffer for on-chip trace
  - [sink] ***TPIU*** *(Trace Port Interface Unit)*: port for off-chip trace

- More documentation available here:
  - *https://developer.arm.com/documentation/ddi0314/h?lang=en*
  - https://developer.arm.com/documentation/ddi0314/h/Introduction/About-the-CoreSight-components/Structure-of-the-CoreSight-Design-Kit?lang=en

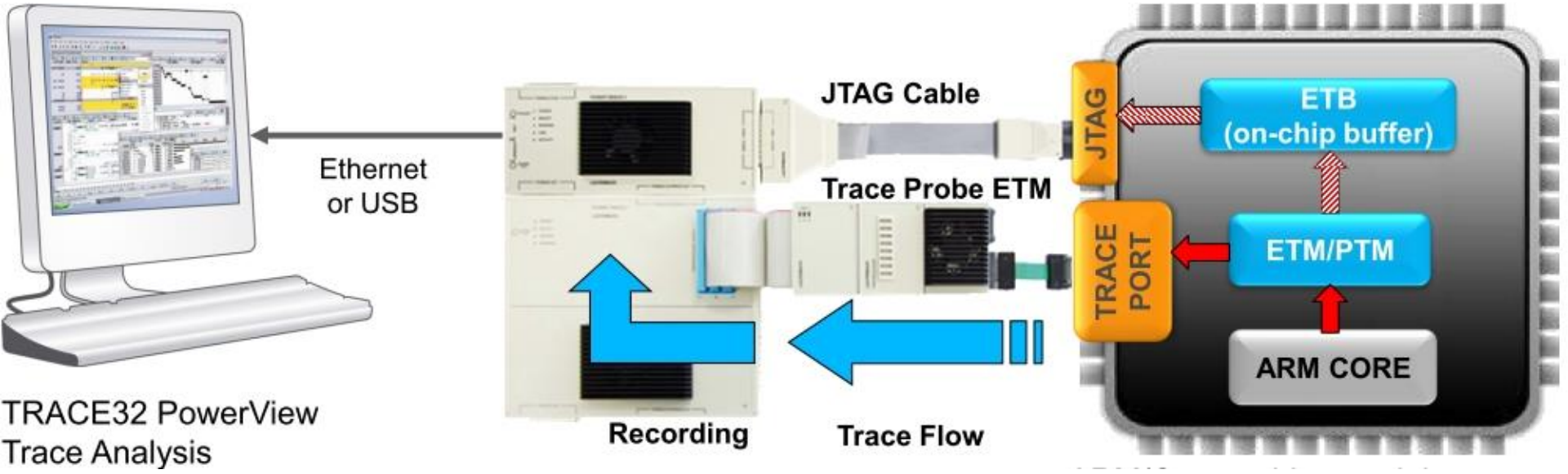
On-chip tracing

- Some processors have specific h/w to do this
  - (but size of on-chip memory for trace is limited; e.g. 4k)
- at break time, all trace records are passed to the debugger via debug port (e.g. JTAG)



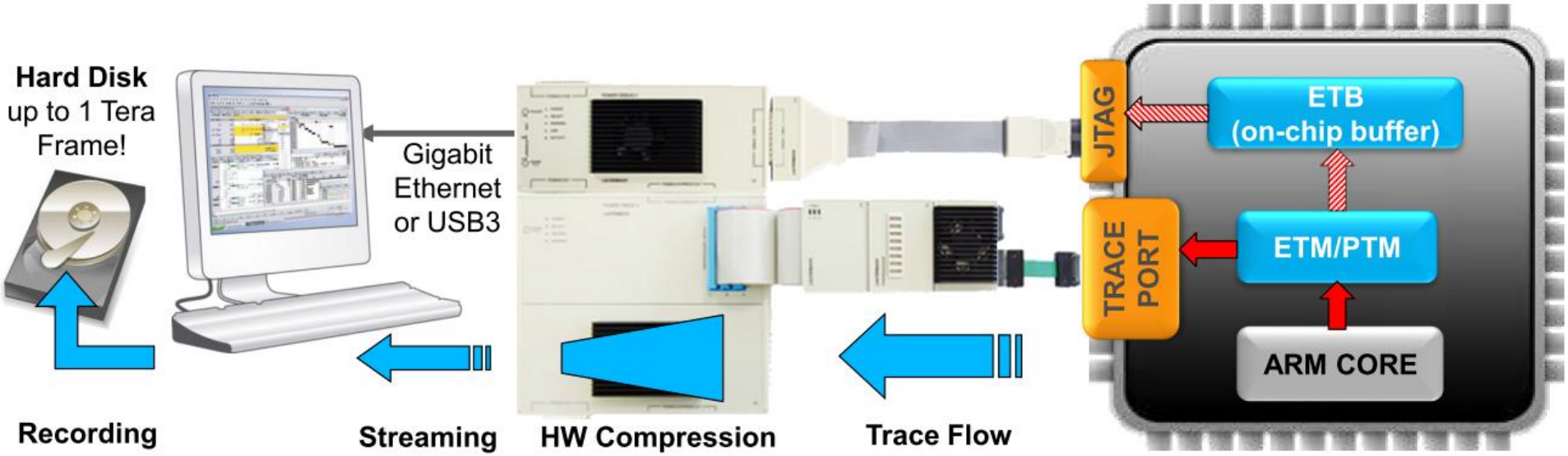
Off-chip tracing

- If longer tracing is needed:
  - a *tracing port* is need on the target system
  - an extension has to be added to the debug system
  - tracing length is now limited by the memory of the trace module (e.g. 4 GB)



Off-chip tracing: streaming mode

- If even longer tracing is needed:
  - the trace-flow is compressed by the debugger h/w, sent to the PC (via gigabit ethernet or USB3) and saved on HDD



Mixed signal probe

- A *mixed signal probe* may also be needed, which allows digital and analog signals to be recorded
  - recorded digital/analog signals are correlated with program flow

Tracing: ARM Serial Wire Viewer (SWV)

- Other components of *CoreSight* are
  - [source] ***ITM*** *(Instrumentation Trace Macrocell*)
  - [sink] ***SWO*** *(Serial Wire Output)*
    - supports a limited subset of the full TPIU functionality
    - minimizes gate count for a simple debug solution
- ITM and SWO can be used to form a ***SWV*** *(Serial Wire Viewer)*
- SWV provides program information in real-time
  - outputs ITM trace through a single pin
  - displays data reads, writes, exceptions (interrupts…)
  - displays executed instructions
    - SWV only samples them
    - -> use ETM to capture all instructions executed

STM32CubeIDE: debug

- With the default MX configuration, load and build *debug.c*
- Only the first time
  - Right button on the project name… Debug As… 1 STM32…
  - otherwise, Run… Debug
- IDE switches to debug mode
- Then we can
  - Restart, Resume, Suspend
  - Add breakpoint
  - Step Into, Step Over, Step Return
    - (also in Instruction Stepping Mode)
  - …
- Terminate exits debug mode

Example with DMA

- In MX:
  - System Core… DMA… DMA1, DMA2… Add
  - Select… MEMTOMEM
  - DMA Request Settings: leave the default values
- In IDE: test *DMA1-M2M* with both 10 and 100000 bytes
  - Run…Debug
  - Window… Show View… Expressions… add *Buffer_Src* and *Buffer_Dest* and open them to see memory content
  - use Step Over (F6) to proceed
  - Restart to start again
- In MODE2 code crashes (red LED on)

STM32CubeIDE: SWV

It allows you to see e.g. variable values in time

- Prepare example *SWV*
- Run… Debug
- Run… Debug Configurations… <project>Debug… Debugger… Serial Wire Viewer: Enable
- Apply… Debug
- Window... Show View... SWV... SWV Data Trace Timeline Graph
- (Window icon) Configure trace: Comparator 0 Enable, Var: LedState
- (Window icon) Start trace
- Resume

- *This might actually be snooping, not tracing*