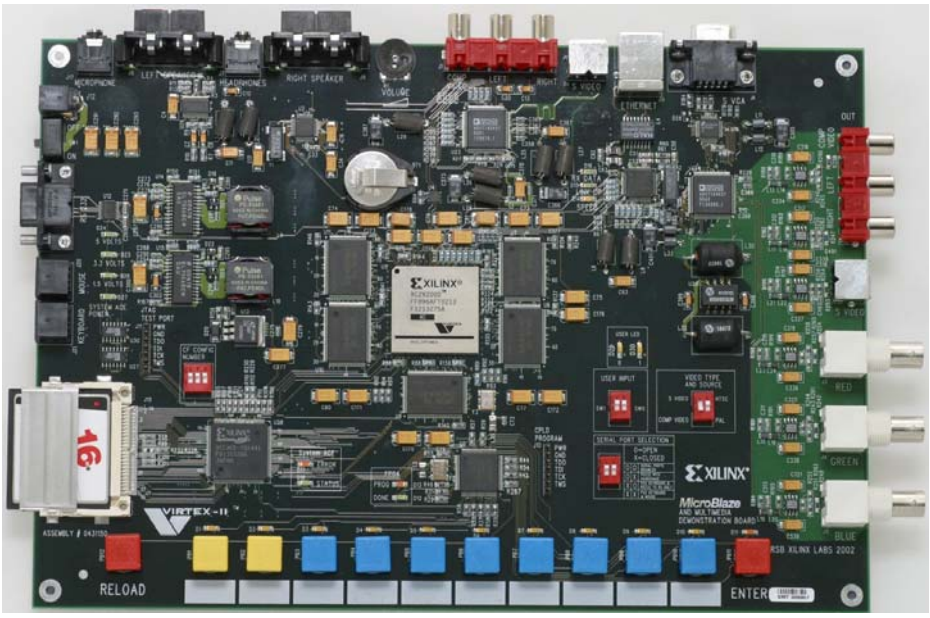
Martino Jerian 83101542
Università degli Studi di Trieste
Corso di Elettronica - Prof. Stefano Marsi
Anno Accademico 2003-2004

REALIZZAZIONE DI UN'INFRASTRUTTURA PER ELABORAZIONE VIDEO IN TEMPO REALE

su Xilinx MultiMedia Board



con un tutorial introduttivo all'utilizzo dello Xilinx Embedded Development Kit

INDICE

1. INTRODUZIONE

La *MicroBlaze and Multimedia Demo Board* é una piattaforma orientata allo sviluppo di applicazioni multimediali prodotta dalla *Xilinx*. Il suo nucleo é costituito da un FPGA Virtex2 interfacciabile con un gran numero di periferiche integrate direttamente sulla board quali ingressi e uscite video PAL o NTSC, ingressi e uscite audio, porta ethernet, true color VGA, interfacce PS2 e RS232, cinque blocchi di RAM esterna di tipo ZBT da 512 kilobytes l'uno, due led. Sono presenti inoltre degli switch e dei tasti che possono essere configurati per interagire con il sistema. Il controller SystemACE permette infine di caricare direttamente il sistema da una memoria di tipo CompactFlash.

Il software utilizzato per lavorare con la MultiMedia Board é l'*Embedded Development Kit 6.1i*; per funzionare correttamente esso necessita della presenza dell'*Integrated System Environment 6.1i*, di cui utilizza le funzioni di sintetizzazione e implementazione dell'hardware. Entrambi i programmi necessitano dei rispettivi Service Packs per operare correttamente.

L'Embedded Development Kit é un ambiente di sviluppo prodotto dalla Xilinx che permette di integrare il design di hardware e software nello stesso progetto. Esso é orientato alla generazione di sistemi che incoroporano al loro interno un soft processor chiamato *MicroBlaze*, che può essere poi programmato in assembler, in C o in C++, utilizzando eventualmente le librerie fornite dal produttore.
L'EDK permette di implementare su FPGA progetti anche molto complessi dandoci la possibilità di utilizzare e interfacciare fra loro dei cores già pronti o importarne di nuovi.

I passi fondamentali della realizzazione di un progetto sono:

- scelta dei cores necessari (fra cui il MicroBlaze) e configurazione dei loro parametri;
- definizione dei segnali che permettono alle varie periferiche interne all'FPGA di comunicare tra loro;
- definizione dei segnali e delle porte che permettono all'FPGA di comunicare con l'esterno;
- sintetizzazione dell'hardware (creazione della netlist e del bitstream);
- scrittura del codice da far eseguire al MicroBlaze;
- generazione delle librerie software necessarie e compilazione del codice (con eventuale debugging)
- aggiornamento del bitstream con l'eseguibile per il Microblaze;
- download del bitstream finale su FPGA.

Nel presente tutorial affronteremo tutti i passi della realizzazione di un progetto per la Multimedia Board con l'EDK.
Il primo esempio consisterà nella realizzazione di un sistema che scriva sulla porta RS232, connessa al nostro computer, il classico "Hello World!"; vedremo tutti gli stadi della realizzazione dell'hardware e del software e un piccolo approfondimento sulle diverse librerie utilizzabili per programmare il MicroBlaze.
Il secondo esempio sarà un sistema poco più complesso del precedente, a cui connetteremo in più degli switch e dei led.
Il terzo esempio sarà più impegnativo, dovremo far girare l'"Hello world!" dalla ram esterna; coglieremo l'occasione per vedere come importare un core scritto in VHDL o Verilog nell'EDK e come effettuare il debugging del software.
Del quarto progetto, la realizzazione di un'infrastruttura per l'elaborazione video in tempo reale, vedremo soltanto una descrizione sul suo funzionamento e non la spiegazione di tutta la sua creazione, data la sua complessità.

2. IL PRIMO PROGETTO: HELLO WORLD!

Il primo progetto consiste nel programmare l'FPGA in modo da inviare dei caratteri ASCII sulla porta seriale del nostro computer, sul quale avremo abilitato un'interfaccia di tipo terminale (ad esempio *HyperTerminal* di Windows); si tratta in pratica del classico "Hello world!" che si prende spesso come primo esempio di programmazione per imparare un nuovo linguaggio.
Nel nostro caso dovremo però creare anche l'hardware necessario a far girare il nostro codice, che sarà poi scritto in linguaggio C.
Sebbene sia sempre difficile effettuare una distinzione netta fra hardware e software, e nel nostro caso é forse anche improprio, esso può facilitare la comprensione della metodologia di lavoro con l'EDK: intenderemo quindi hardware quando si tratta di connettere i cores e sintetizzare i rispettivi codici VHDL; chiameremo invece software il codice C da far eseguire al MicroBlaze.
Nello stadio finale del nostro progetto, però, non dovremo più considerare questa distinzione, in quanto ciò che scaricheremo sull'FPGA sarà un semplice bitstream ricavato dall'unione delle due parti.
Nell'archivio *helloworld.zip* é contenuto il progetto completo e funzionante.

2.1 PROGETTARE L'HARDWARE

Il sistema che vogliamo creare é costituito dai seguenti componenti:

- un soft processor MicroBlaze;
- una memoria di tipo block RAM in cui possano risiedere dati e codice eseguibile;
- un controller per la parte dati della block RAM;
- un controller per la parte istruzioni della block RAM;
- un'interfaccia di tipo JTAG tramite cui scaricare il bitstream su FPGA e eventualmente eseguire il debugging del codice eseguibile;
- un'interfaccia di tipo UART che invierà i dati al nostro computer tramite la porta RS232.

I dispositivi comunicano con il MicroBlaze tramite:

- due local bus, uno per la parte dati e uno per la parte istruzioni della RAM;
- un OPB (on-chip peripheral bus) per gli altri dispositivi.

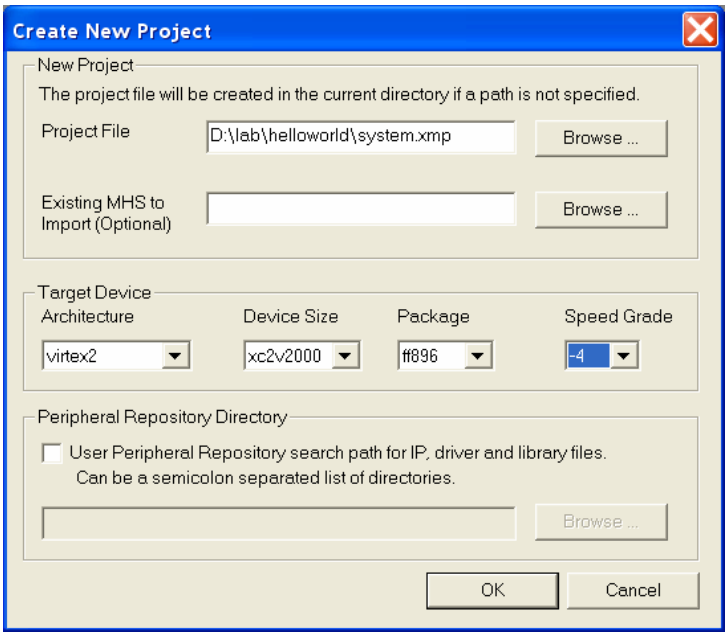
Dal menù dell'EDK 6.1 caricare il programma *Xilinx Platform Studio*, l'interfaccia tramite cui seguire i vari passi della progettazione e dell'implementazione, fino alla conversione in bitstream e al download su FPGA.

2.1.1 CREARE UN NUOVO PROGETTO

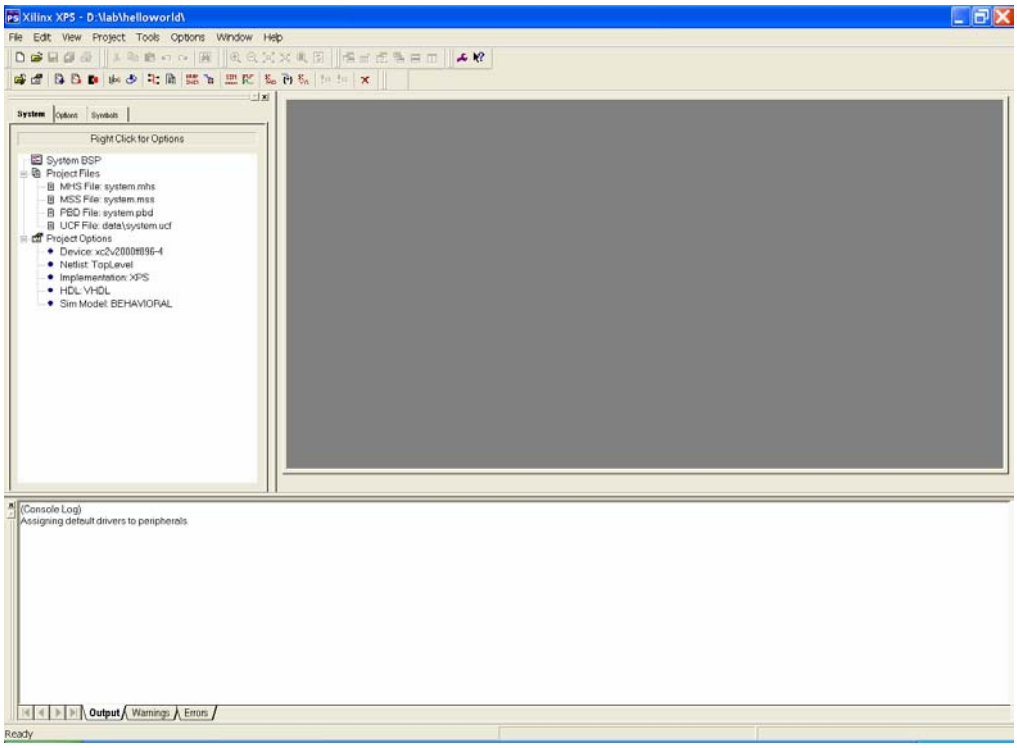
Cliccare su *File -> New Project -> Platform Studio...*

L'opzione *Base System Builder* avrebbe permesso di generare un sistema in maniera automatica, potendo configurare facilmente i componenti più comuni secondo le nostre esigenze, ma allo stato attuale la MultiMedia Board non é supportata.

Definire la directory di lavoro desiderata e selezionare il tipo di FPGA su cui sarà implementato il design, i campi corretti sono indicati in figura.
E' importante ricordare sempre che il percorso della cartella in cui é situato il nostro progetto non può contenere spazi o caratteri particolari (come lettere accentate) altrimenti i files non potranno essere trovati dagli script che vengono fatti eseguire tramite il Platform Studio.



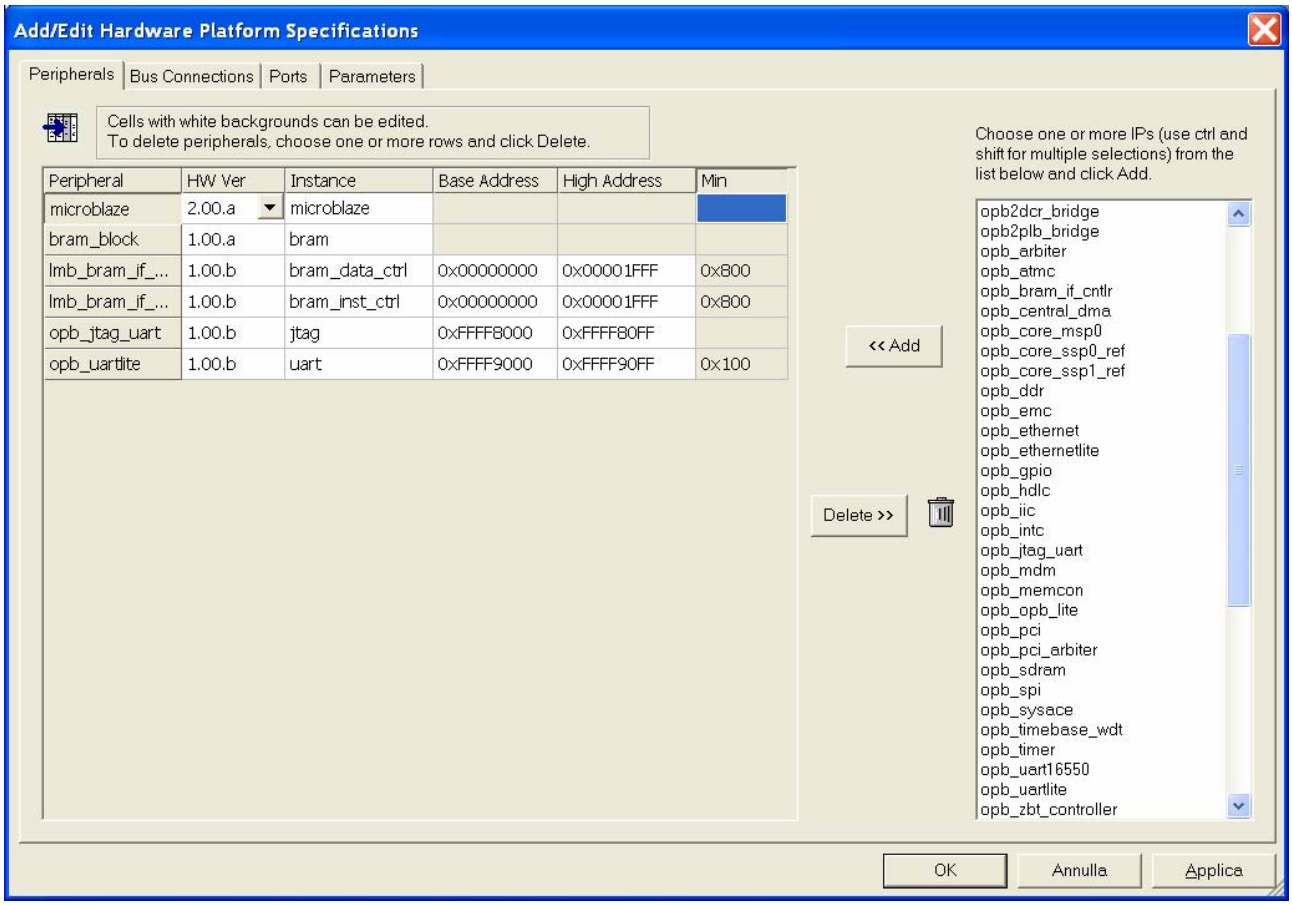
Dopo un paio di avvertimenti la schermata si presenta così:



Prima di tutto dobbiamo scegliere i cores che compongono l'hardware del nostro sistema. Cliccare su *MHS File*: *system.mhs* con il tasto destro del mouse e scegliere *Add/Edit Cores*. Apparirà un'interfaccia che ci permette di scegliere e configurare i dispositivi.

2.1.2 IL MENU' PERIPHERALS

Il primo passo é definire i componenti che costituiscono il sistema e i rispettivi indirizzi di memoria. Configurare come in figura.



Le indicazioni sugli indirizzi dei controller della BRAM indicano quanto sarà grande la memoria; essi sono specificati in esadecimale quindi il nostro sistema avrà 8 kilobytes di BRAM; é fondamentale che la sua dimensione sia adeguata alla quantità di dati e istruzioni che deve contenere.
Scrivendo all'indirizzo specificato per la UART, invece, invieremo dati tramite la porta RS232.

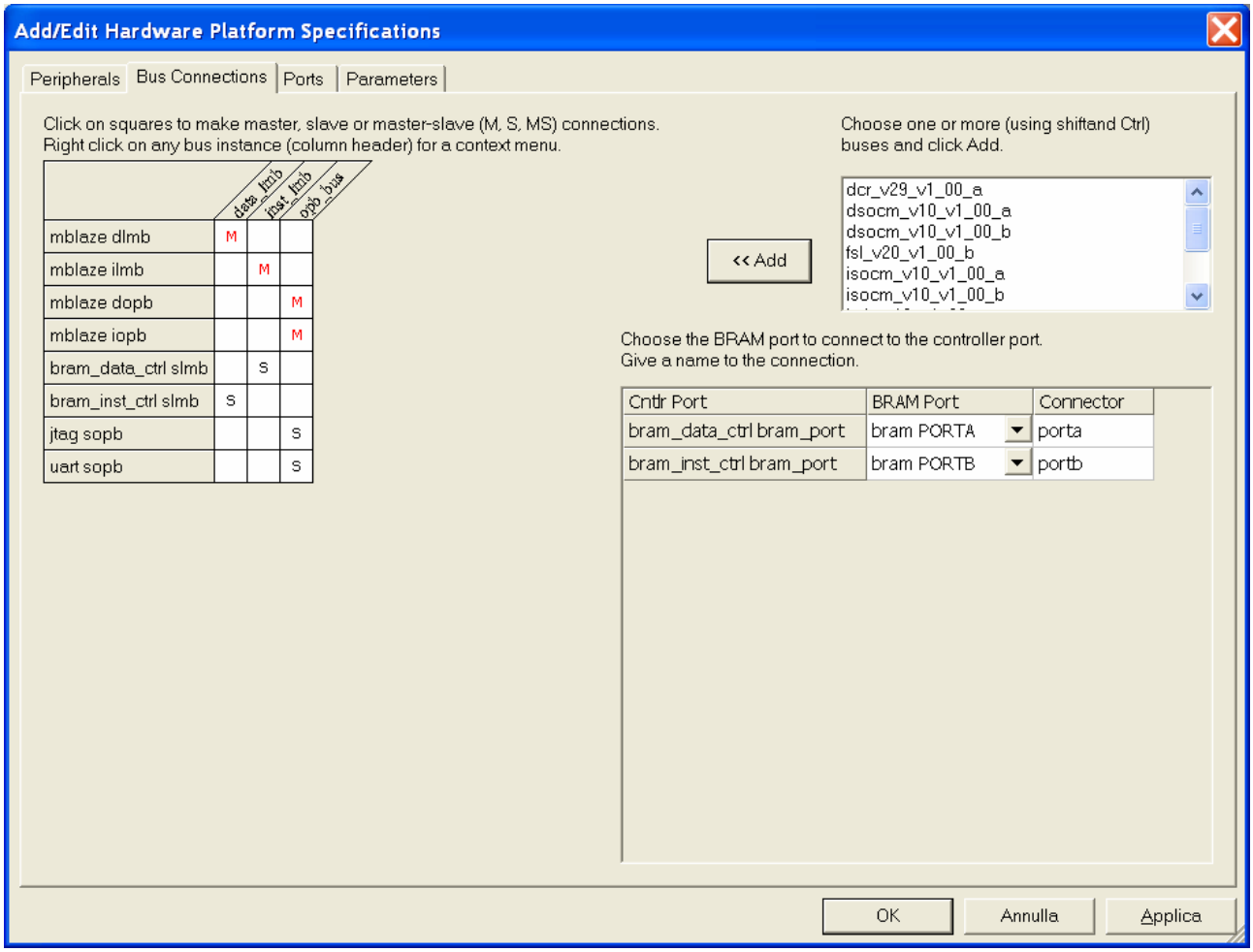
Cliccando su *Applica* viene modificato il file MHS (Microprocessor Hardware Specification) in cui sono indicati i nomi dei componenti, i loro collegamenti e la configurazione dei loro parametri [1].

2.1.3 IL MENU'BUS CONNECTIONS

Cliccando su *Bus Connections* accediamo all'interfaccia per gestire i collegamenti fra le varie periferiche che comunicano con il MicroBlaze tramite bus. Come si può vedere in figura il MicroBlaze fungerà da Master per:

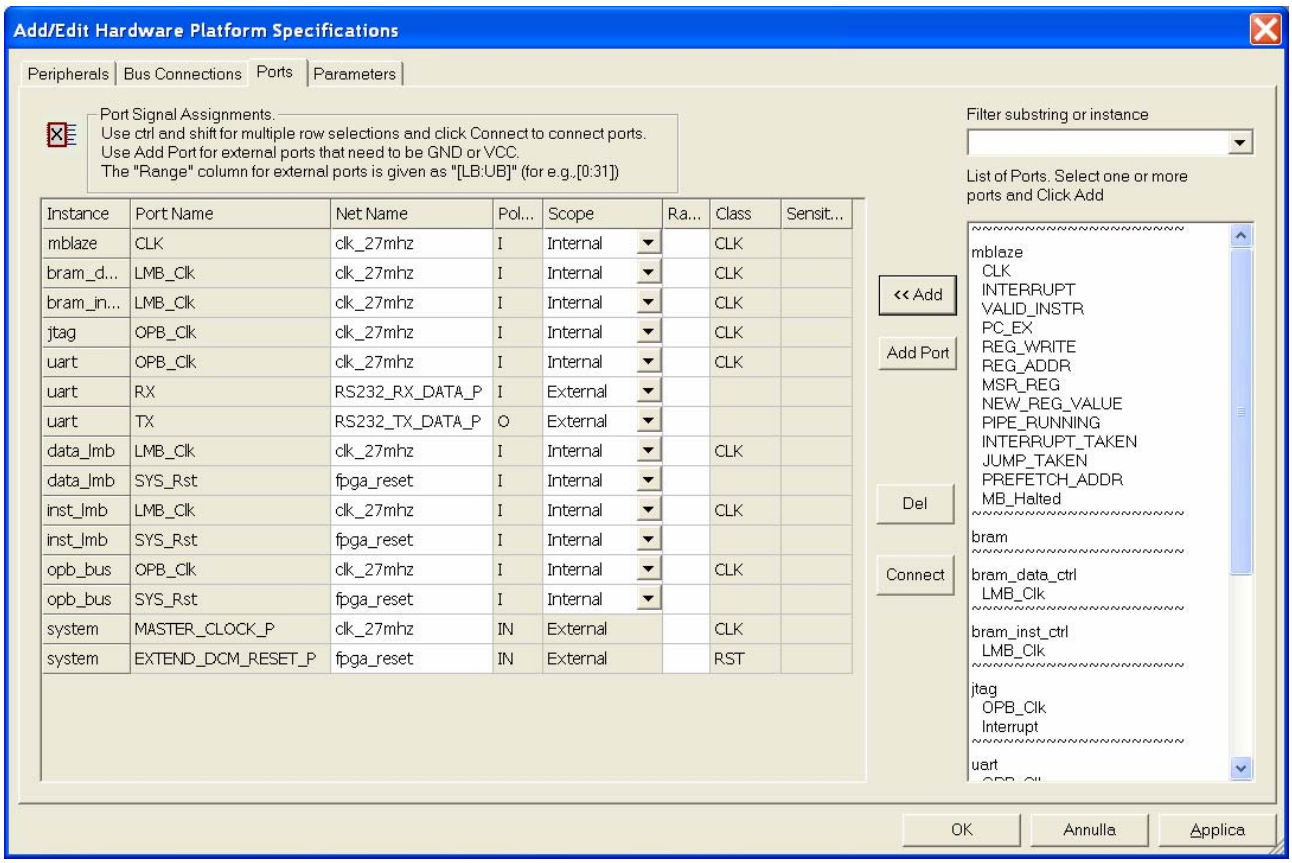
- il data local memory bus, a cui é collegata la BRAM;
- l'instructions local memory bus, anch'esso collegato alla BRAM;
- l'OPB bus, a cui verranno connesse la UART e la JTAG.

Sempre nella stessa schermata vanno collegate le due porte della BRAM ai rispettivi controllers.



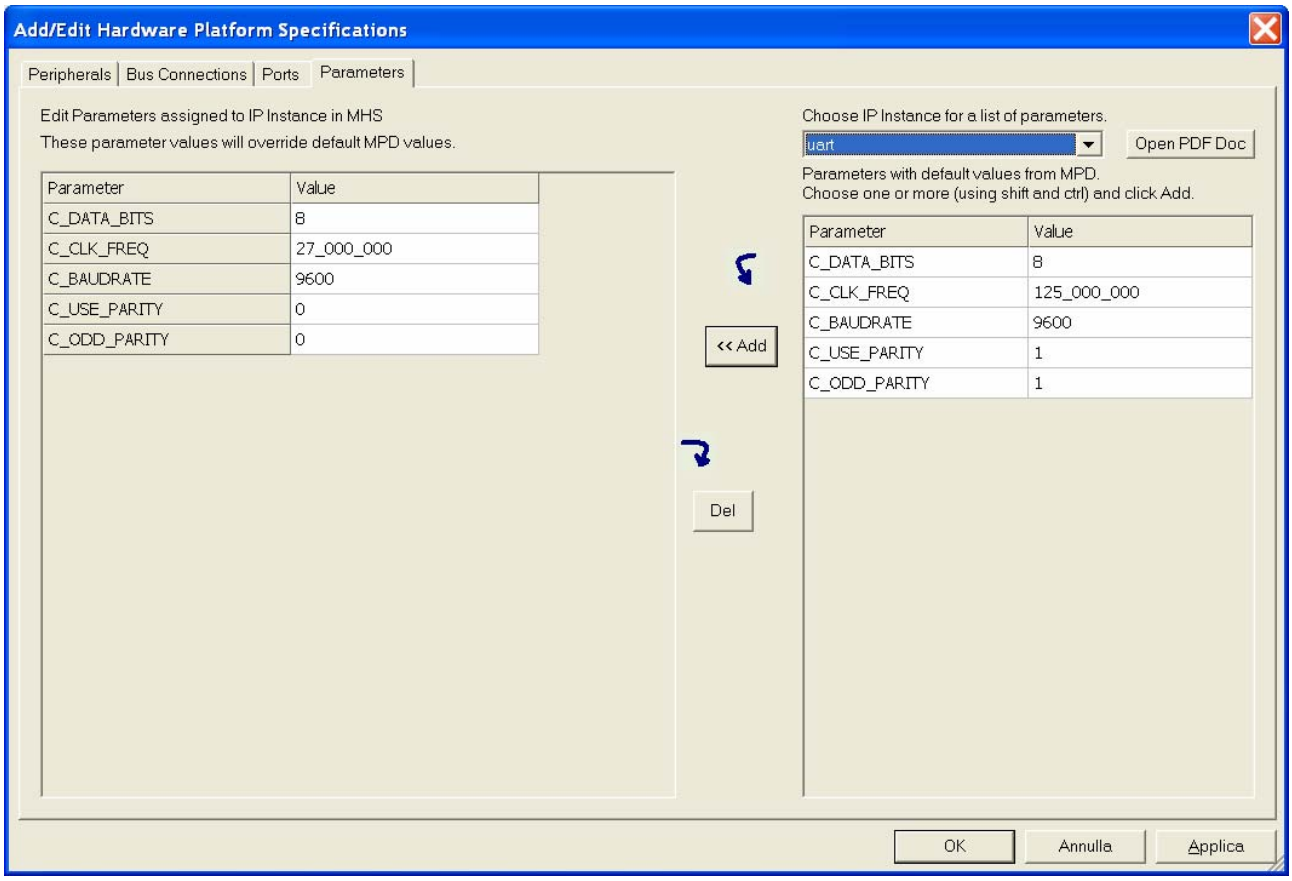
2.1.4 IL MENU'PORTS

Il passo successivo consiste nel definire i segnali tramite cui comunicheranno i vari IP che verranno implementati nell'FPGA e le porte tramite cui l'FPGA comunicherà con l'esterno.
Il clock entra tramite la porta *MASTER_CLOCK_P* e viene quindi distribuito fra i vari dispositivi sulla rete *clk_27mhz*.
Lo stesso discorso vale per l'*EXTEND_DCM_RESET_P* che viene distribuito internamente su *fpga_reset*.
Le porte di comunicazione RX e TX della UART vengono portate direttamente all'esterno senza passaggi intermedi in quanto interessano un solo dispositivo (tuttavia potevano essere create delle porte esterne anche per esse senza comportare alcuna differenza).
In realtà avremmo potuto collegare direttamente anche tutti i segnali di clock e di reset all'esterno senza creare le due porte aggiuntive, ma il nostro progetto risulta in questo modo più pulito.
Se i segnali esterni fossero stati dei vettori avremmo dovuto indicare il loro numero o intervallo nel campo *Range*.



2.1.5 IL MENU'PARAMETERS

L'ultimo menù a tendina ci permette di configurare i parametri dei vari IP, in caso avessimo bisogno di modificare i valori di default.
Nel nostro caso dobbiamo portare a 0 il valore di *C_EXT_RESET_HIGH* per le istanze *data_lmb*, *inst_lmb* e *opb_bus*, in quanto il segnale di reset proveniente dalla board é attivo con il valore logico "0".
Impostare inoltre come in figura i parametri per la UART. Essi devono corrispondere ai valori che immetteremo nel configurare il terminale di Windows.



2.1.6 GENERATE NETLIST

Cliccando su *Tools -> Generate Netlist* vengono generati i files VHDL che interconnettono tutti i cores del sistema secondo i parametri che abbiamo fornito loro; una volta che questi sono stati compilati e sintetizzati singolarmente viene fatto lo stesso per il top-level del progetto; abbiamo così un'unica netlist globale.

2.1.7 DEFINIRE LE CONSTRAINTS

Il passo successivo é creare un file con le constraints, ossia definire i pin dell'FPGA che verranno collegati alle porte esterne che abbiamo definito nel sistema e eventuali caratteristiche di tali porte. Questo file si deve chiamare obbligatoriamente *system.ucf* e risiedere nella sottodirectory */data* del progetto.

Andando nuovamente su *Add/Edit Cores* e scegliendo *Ports* dobbiamo trovare tutte le reti che hanno scritto *External* nel campo *Scope* e definire i pin di collegamento facendo coincidere il loro nome con quello scritto nella colonna *Net Name* (*RS232_TX_DATA_P, RS232_RX_DATA_P*), a meno che abbiamo creato una porta esterna (ad esempio per *MASTER_CLOCK_P, EXTEND_DCM_RESET_P*) nel cui caso il nome sarà quello definito nella colonna *Port Name* e il nome dell'instance sarà sempre *system*.

Dai datasheets o dai files forniti dalla Xilinx possiamo trovare i pin ed eventuali altri parametri da definire per ciascun collegamento.
Nel nostro caso il file *system.ucf* [2] risulta:

```
# SYSTEM CLOCKS
NET "MASTER_CLOCK_P" LOC = "AH15";

# DCM RESET EXTENSION
NET "EXTEND_DCM_RESET_P" LOC = "AH7";

# RS232 COMMUNICATION PORT
NET "RS232_TX_DATA_P" LOC = "C9";
NET "RS232_TX_DATA_P" SLOW;
NET "RS232_TX_DATA_P" IOSTANDARD = LVTTL;
NET "RS232_TX_DATA_P" DRIVE = 12;
NET "RS232_RX_DATA_P" LOC = "C8"
```

2.1.7 GENERATE BITSTREAM

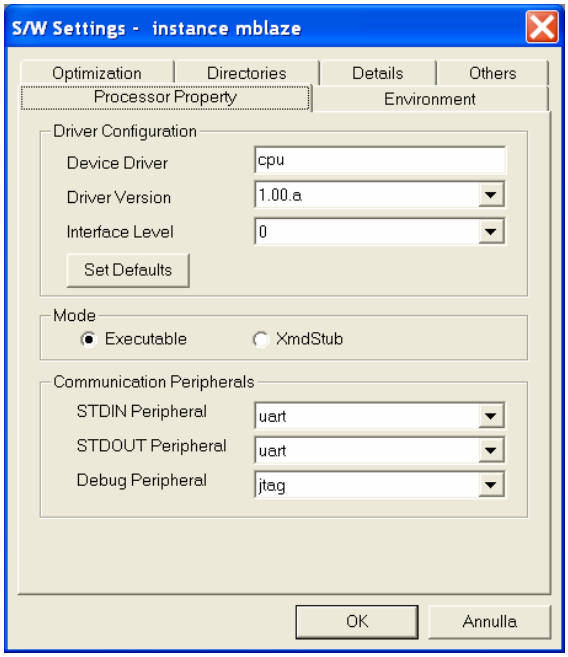
Cliccando su *Tools -> Generate Bitstream* la Netlist precedentemente ottenuta viene implementata per la piattaforma specificata in fase di creazione del progetto e i segnali provenienti dall'esterno vengono collegati come definito nel file *system.ucf*.

2.2 IL SOFTWARE

Prima di scrivere il codice C da far eseguire al MicroBlaze dobbiamo compilare le librerie che potranno venire utilizzate dal nostro hardware.

2.2.1 LIBRARY GENERATOR

Clicchiamo su *mblaze* con il tasto destro e quindi *S\W Settings*: definiremo qui lo standard in, standard out e standard debug del nostro progetto. Settiamoli tutti e tre anche se per il momento utilizzeremo soltanto lo standard out, il controller UART che invierà i dati alla porta seriale del nostro computer.



Esistono un gran numero di librerie fornite dalla Xilinx per agevolare la programmazione dei dispositivi con cui comunica il MicroBlaze; generalmente per ogni device ci sono due gruppi di librerie, uno a più basso livello, che occupa meno memoria e svolge funzioni più semplici, e uno a più alto livello che é però più pesante [3]. Sta alle nostre esigenze per ogni specifica situazione scegliere che tipo di librerie utilizzare. Per ora conserviamo la configurazione di default, vedremo in seguito come scegliere le librerie da compilare.

Cliccare quindi *Tools -> Generate Libraries.*

2.2.2 IL PROGRAMMA C

Come ultimo passo dobbiamo scrivere il programma C da far eseguire al MicroBlaze.
Creare un file con il nome desiderato (per esempio *helloworld.c* nella sottodirectory */code*, ma non vi sono restrizioni in merito, come in altri casi) con il seguente codice:

```
#include "stdio.h"

main() {
    while( 1 )
    {
       print("Hello world!\n\r");
       print("I am your new MicroBlaze\n\r");
    }
}
```

Nel menù della finestra a sinistra, sotto *mblaze*, cliccare su *Sources* con il tasto destro del mouse, quindi *Add Files* e infine selezionare il file creato.

2.2.3 COMPILE PROGRAM SOURCES

Compilare il file sorgente con *Tools -> Compile Program Sources.*

Dalla finestra di output vediamo che la dimensione del nostro programma compilato é di 1499 bytes.

2.2.4 UPDATE BISTREAM

Aggiorniamo con il codice compilato il bitstream da scaricare sull'FPGA con il comando *Tools -> Update Bitstream*.

2.3 CONFIGURAZIONE DEL SISTEMA E DOWNLOAD

Colleghiamo opportunamente i cavi di alimentazione e di programmazione dell'FPGA e inoltre un cavo seriale per connettere le due porte RS232 di computer e board.
Per scaricare il bitstream su FPGA potremmo utilizzare *iMPACT*, accessorio dell'ISE 6.1, ma é molto più comodo richiamare le sue funzioni direttamente dall'EDK tramite uno script.
Creiamo un file che si deve chiamare *download.cmd* [4] e deve risiedere nella sottodirectory */etc* del nostro progetto.

Scriviamo al suo interno:

```
setMode -bscan
setCable -p lpt1
addDevice -p 1 -file etc/systemace.bsd
assignfile -p 1 -file  etc/systemace.bsd
addDevice -p 2 -file implementation/download.bit
program -p 2
quit
```

Il tool di download del bitstream rileva due periferiche collegate in serie; a noi interesserebbe soltanto scaricare il *download.bit* sull'FPGA (identificata come Device 2); tuttavia é preferibile programmare sempre anche il SystemAce con l'opportuno *systemace.bsd* [5]: i file di tipo *BSDL* servono per gestire separatamente la programmazione di periferiche connesse in cascata.
Cliccare su *Tools -> Download*.

Se tutto é stato effettuato a dovere il MicroBlaze ci saluterà.

2.4 LE LIBRERIE XILINX

Per il primo esempio di programma abbiamo usato la libreria *stdio.h*, una libreria standard del C, e la sua funzione *print* che scrive sullo *STDOUT* che abbiamo specificato nel pannello *S\W Settings*; utilizziamo ora delle librerie specifiche della Xilinx per ottenere lo stesso risultato.

Per utilizzare le librerie di basso livello della porta UART non serve modificare alcun parametro nel *S\W Settings* del controller perché sono impostate di default; le funzioni da richiamare sono definite dal file *xuartlite_l.h*.
Per capire come utilizzarle possiamo consultare la documentazione di tutti i drivers forniti tramite *Help -> EDK Help*, quindi cliccando sul link *Documents* e infine *Xilinx Drivers*.

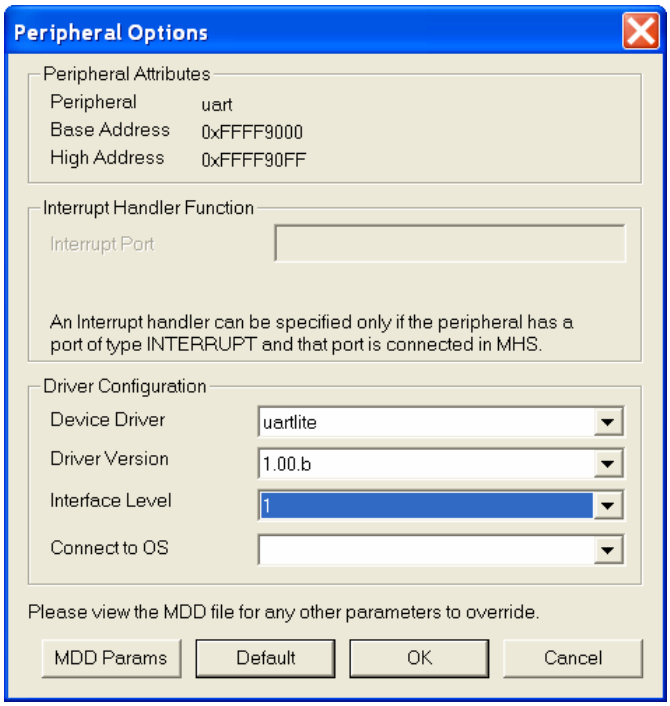
Utilizziamo la funzione *XuartLite_SendByte* [6], che invia un byte di dati all'indirizzo della porta UART; possiamo indicare direttamente l'indirizzo di base della UART (in questo caso 0xFFFF9000) ma é più comodo includere il file *xparameters.h*, generato automaticamente dall'EDK, e indicare l'indirizzo come *XPAR_UART_BASEADDR*.
Cliccando su *Generated Header: mblaze/include/xparameters.h* (nella schermata principale dell'XPS) possiamo leggere le definizioni di tutti gli indirizzi generati, fra cui *#define XPAR_UART_BASEADDR 0xFFFF9000*.

Notare che non sarebbe stato necessario indicare lo *STDOUT* come prima perché comunque la funzione indirizza il byte da spedire all'indirizzo specificato e non allo standard output.

```
// low_level_lib_example.c

#include "xparameters.h"
#include "xuartlite_l.h"

main()
{
  Xuint8 text[] = "using low level library - ";
  int count = 0;
  while (1)
  {
    count = 0;
    while (count < sizeof(text))
    {
      XUartLite_SendByte(XPAR_UART_BASEADDR, text[count]);
      count++;
    }
  }
}
```

Compilando il file notiamo che con questa libreria la dimensione dell'eseguibile é di 1643 bytes, cioé leggermente superiore a quello precedente.

Proviamo ora ad utilizzare le librerie a più alto livello, cliccando su UART(con il tasto destro) e quindi su *S\W Settings*, per accedere al *Peripheral Options*.



Settiamo *Interface Level* a 1, che corrisponde alla libreria di livello più alto *uartlite.h* [7].
Prima non avevamo dovuto modificare alcun parametro per utilizzare la libreria *uartlite_l.h* [8], essa é accessibile in ogni caso, poiché anche se indichiamo *uartlite.h* essa stessa la include a sua volta per richiamare le funzioni più basilari. in pratica il sistema é costruito a livelli.

Dobbiamo impostare lo stesso livello anche per il driver della JATG; anche se in realtà non lo utilizziamo, il *Library Generator* lo considera comunque e, essendo lo stesso driver utilizzato dalla UART, deve essere impostato sullo stesso livello; livelli differenti dello stesso driver non sono supportati e ciò genererebbe quindi un errore.

```
// hi_level_lib_example.c

#include "parameters.h"
#include "xuartlite.h"

main()
{
  XUartLite uart;
  XUartLite_Initialize(&uart, XPAR_UART_DEVICE_ID);
  while(1)
  {
    xil_printf("using high level libraries - ");
  }
}
```

La funzione *xil_printf* che utilizziamo qui é quindi molto più immediata, basta indicare la stringa che si vuole ottenere in output. L'unico accorgimento necessario é che bisogna inizializzare la porta con il comando *XuartLite_Initialize* [9].

Compilando il sorgente notiamo che l'eseguibile generato é di 4433 bytes, più del triplo di quello precedente, a causa del diverso spessore dei driver.

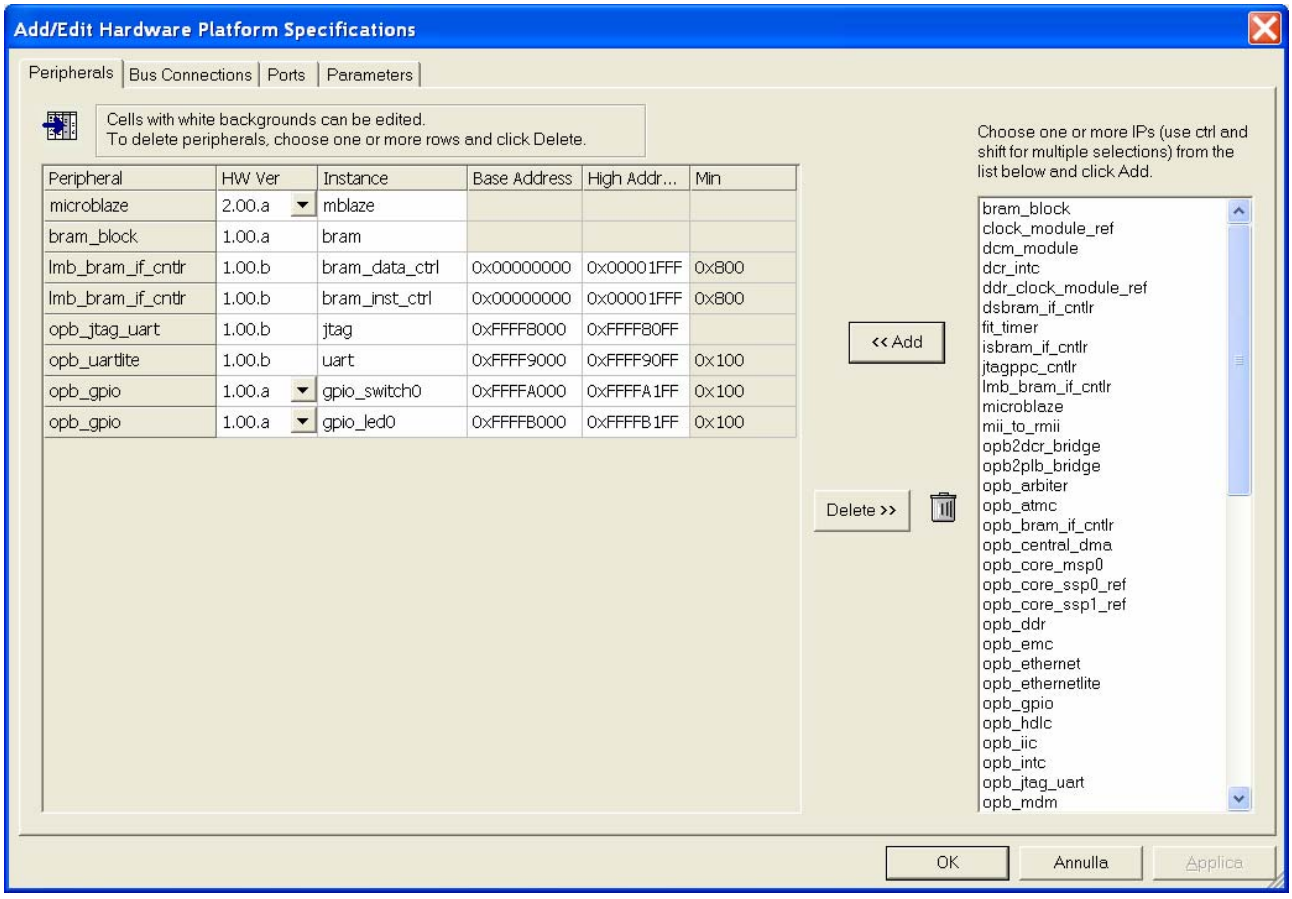
Sta quindi a noi decidere in ogni occasione qual é il driver più opportuno da utilizzare.

3. INTERFACCIARE SWITCH E LED

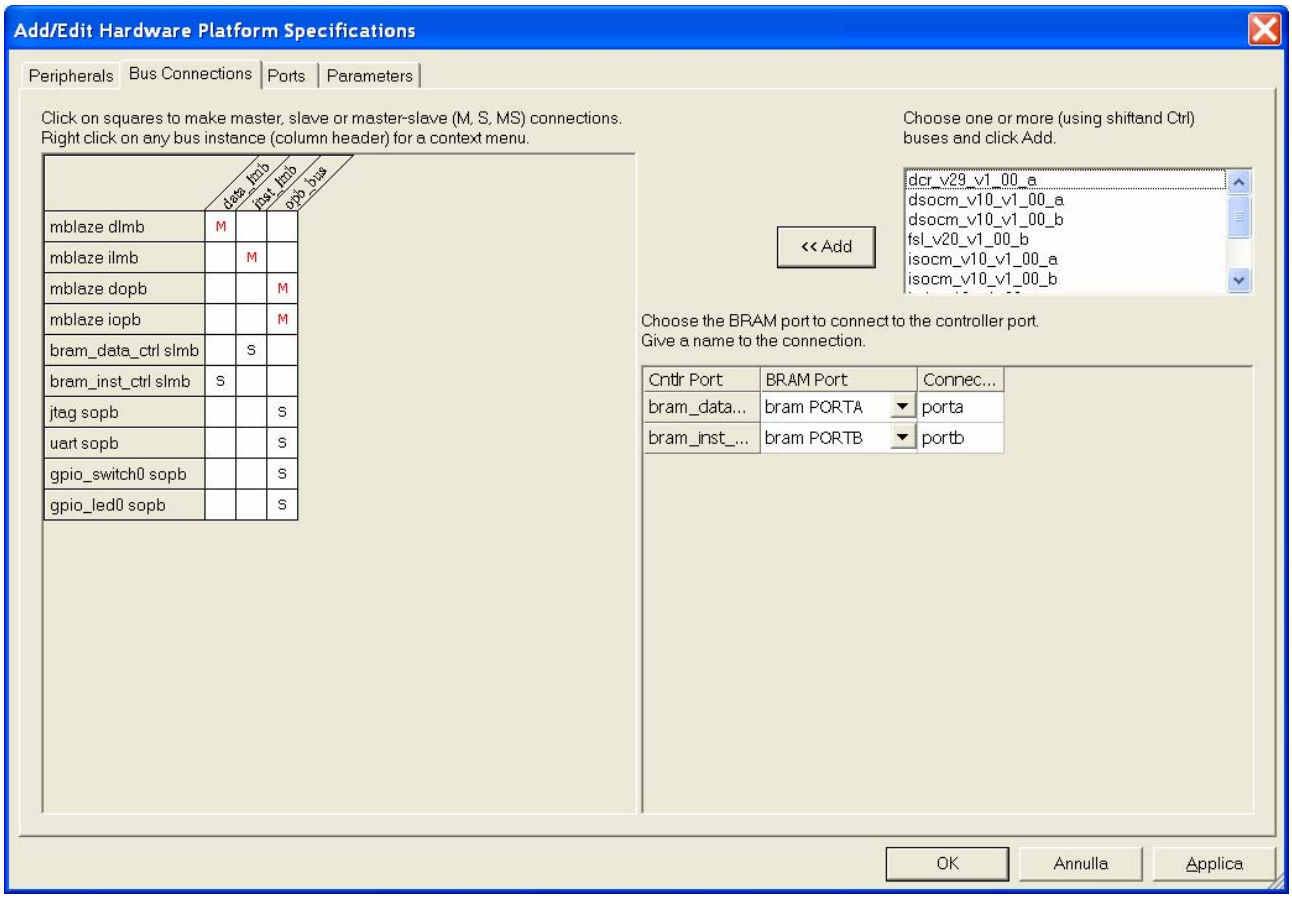
Il secondo progetto riprende gran parte dell'hardware del primo. La variazione consiste nell'aggiunta del controllo dei due led della MutiMedia Board tramite due switch.
Lo *switch0* é collegato a un GPIO (*general purpose input output*) che comunica con il MicroBlaze sul bus OPB. A quest'ultimo é connesso un altro GPIO che comunica con il *led0*. I valori di questi due dispositivi possono essere quindi letti o scritti da un programma eseguito dal MicroBlaze.
Lo *switch1*, invece, é connesso direttamente con il *led1*, lo stato del secondo corrisponde sempre e comunque a quello del primo e il MicroBlaze non può né leggere né scrivere su alcuno dei due, poiché fra loro esiste un collegamento puramente hardware.
Nell'archivio *switchled.zip* é contenuto il progetto completo e funzionante.

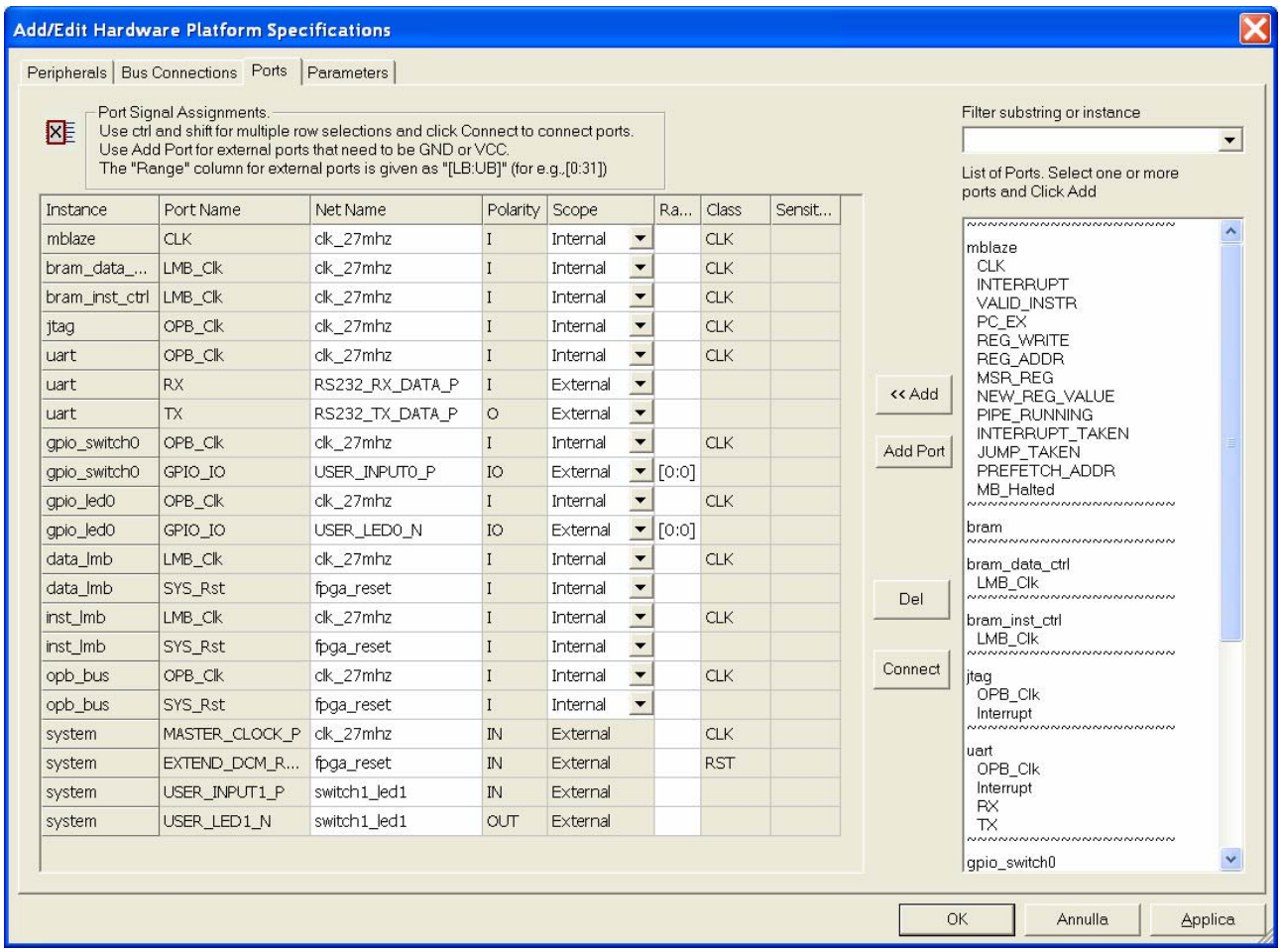
3.1 L'HARDWARE

Configuriamo il sistema come riportato nelle figure successive; aggiungiamo due istanze dell'IP *opb_gpio 1.00.a*; sono presenti pure le versioni 2.00 e 3.00, ma la 1.00 é molto più semplice (quindi occupa meno spazio sull'FPGA) e sufficiente per il nostro utilizzo.



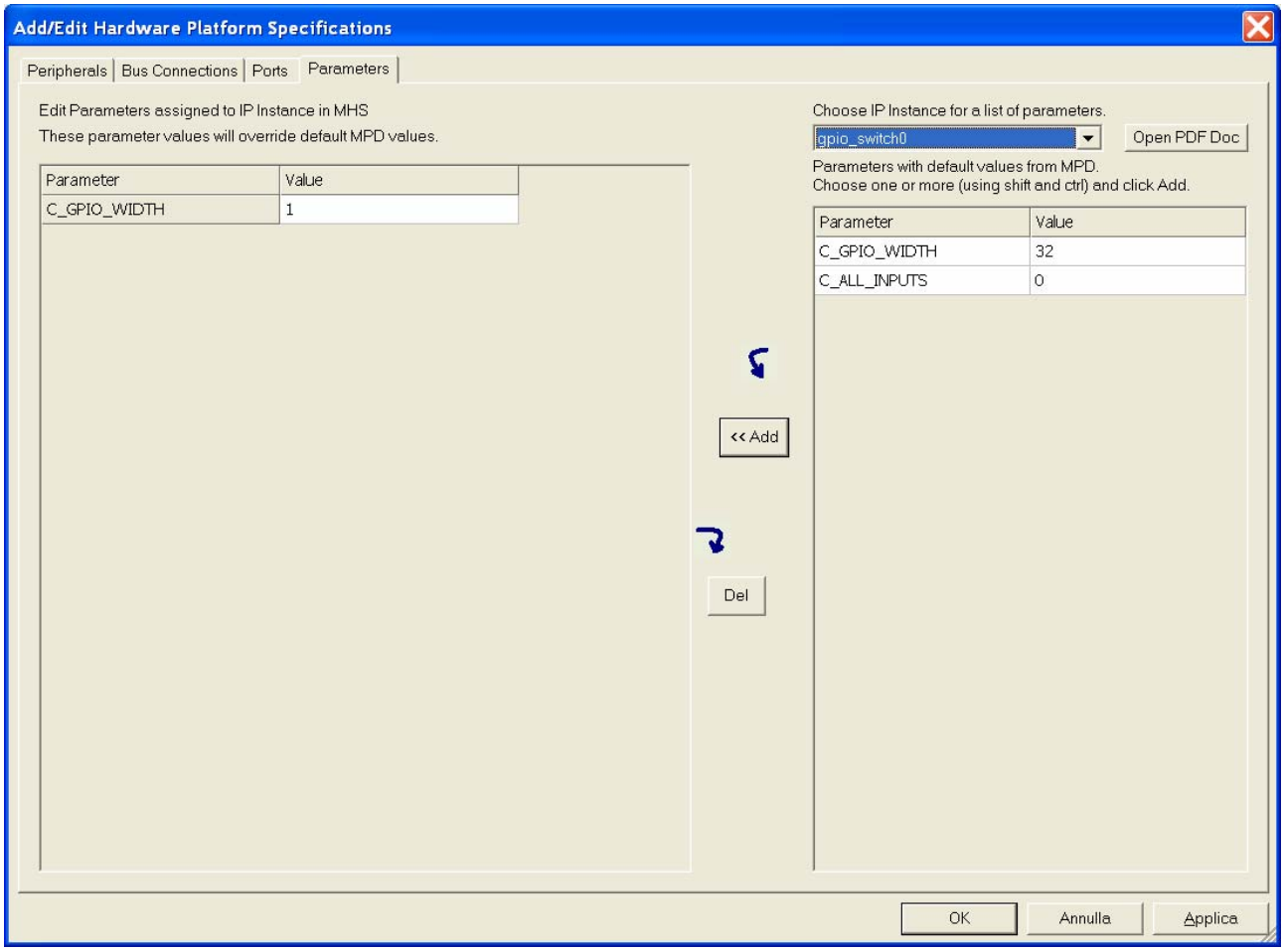
Connettiamo i due GPIO all'*opb_bus*.


Add/Edit Hardware Platform Specifications
Peripherals
Bus Connections
Ports
Parameters
Click on squares to make master, slave or master-slave (M, S, MS) connections.
Right click on any bus instance (column header) for a context menu.
data_lmb
inst_lmb
opb_bus
mblaze dlmb
mblaze ilmb
mblaze dopb
mblaze iopb
bram_data_ctrl slmb
bram_inst_ctrl slmb
jtag sopb
uart sopb
gpio_switch0 sopb
gpio_led0 sopb
<< Add
Choose one or more (using shiftand Ctrl) buses and click Add.
dcr_v29_v1_00_a
dsocm_v10_v1_00_a
dsocm_v10_v1_00_b
fsl_v20_v1_00_b
isocm_v10_v1_00_a
isocm_v10_v1_00_b
Choose the BRAM port to connect to the controller port.
Give a name to the connection.
Cntlr Port
BRAM Port
Connec...
bram_data...
bram PORTA
porta
bram_inst_...
bram PORTB
portb
OK
Annulla
Applica


Add/Edit Hardware Platform Specifications
Peripherals
Bus Connections
Ports
Parameters
Port Signal Assignments.
Use ctrl and shift for multiple row selections and click Connect to connect ports.
Use Add Port for external ports that need to be GND or VCC.
The "Range" column for external ports is given as "[LB:UB]" (for e.g.,[0:31])
Instance
Port Name
Net Name
Polarity
Scope
Ra...
Class
Sensit...
mblaze CLK clk_27mhz I Internal CLK
bram_data_... LMB_Clk clk_27mhz I Internal CLK
bram_inst_ctrl LMB_Clk clk_27mhz I Internal CLK
jtag OPB_Clk clk_27mhz I Internal CLK
uart OPB_Clk clk_27mhz I Internal CLK
uart RX RS232_RX_DATA_P I External
uart TX RS232_TX_DATA_P O External
gpio_switch0 OPB_Clk clk_27mhz I Internal CLK
gpio_switch0 GPIO_IO USER_INPUT0_P IO External [0:0]
gpio_led0 OPB_Clk clk_27mhz I Internal CLK
gpio_led0 GPIO_IO USER_LED0_N IO External [0:0]
data_lmb LMB_Clk clk_27mhz I Internal CLK
data_lmb SYS_Rst fpga_reset I Internal
inst_lmb LMB_Clk clk_27mhz I Internal CLK
inst_lmb SYS_Rst fpga_reset I Internal
opb_bus OPB_Clk clk_27mhz I Internal CLK
opb_bus SYS_Rst fpga_reset I Internal
system MASTER_CLOCK_P clk_27mhz IN External CLK
system EXTEND_DCM_R... fpga_reset IN External RST
system USER_INPUT1_P switch1_led1 IN External
system USER_LED1_N switch1_led1 OUT External
<< Add
Add Port
Del
Connect
Filter substring or instance
List of Ports. Select one or more ports and Click Add
mblaze
CLK
INTERRUPT
VALID_INSTR
PC_EX
REG_WRITE
REG_ADDR
MSR_REG
NEW_REG_VALUE
PIPE_RUNNING
INTERRUPT_TAKEN
JUMP_TAKEN
PREFETCH_ADDR
MB_Halted
bram
bram_data_ctrl
LMB_Clk
bram_inst_ctrl
LMB_Clk
jtag
OPB_Clk
Interrupt
uart
OPB_Clk
Interrupt
RX
TX
gpio_switch0
OK
Annulla
Applica

Connettiamo quindi le porte necessarie dei due GPIO e creiamo un collegamento diretto fra *USER_INPUT1_P* e *USER_LED1_N*.

Impostiamo il parametro *C_GPIO_WIDTH* a 1 per entrambi i GPIO (infatti necessitiamo di un solo bit per i dati).



Il file *system.ucf* sarà in questo caso [2]:

```
# SYSTEM CLOCKS

NET "MASTER_CLOCK_P" LOC = "AH15";


# DCM RESET EXTENSION

NET "EXTEND_DCM_RESET_P" LOC = "AH7";


# RS232 COMMUNICATION PORT

NET "RS232_TX_DATA_P" LOC = "C9";
NET "RS232_TX_DATA_P" SLOW;
NET "RS232_TX_DATA_P" IOSTANDARD = LVTTL;
NET "RS232_TX_DATA_P" DRIVE = 12;

NET "RS232_RX_DATA_P" LOC = "C8";


# USER INPUTS AND OUTPUT

NET "USER_INPUT0_P<0>"  LOC = "D10";
NET "USER_INPUT1_P"  LOC = "F14";

NET "USER_LED0_N<0>"  LOC = "B27";
NET "USER_LED0_N<0>" SLOW;
NET "USER_LED0_N<0>" IOSTANDARD = LVTTL;
NET "USER_LED0_N<0>" DRIVE = 12;

NET "USER_LED1_N"  LOC = "B22";
NET "USER_LED1_N" SLOW;
NET "USER_LED1_N" IOSTANDARD = LVTTL;
NET "USER_LED1_N" DRIVE = 12;
```

Generare la Netlist e quindi il Bitstream.

3.2 IL SOFTWARE

Creiamo ora un programma che legga lo stato dello switch e a seconda della sua posizione accenda o spenga il led. Inoltre scriva sulla UART una “U” quando é su e “D” quando é giù.

```
#include "xparameters.h"
#include "xgpio_l.h"
#include "xuartlite_l.h"

main()
{
  Xuint8 button[]= "UD";
  int OnOff;

  XGpio_mSetDataDirection(XPAR_GPIO_SWITCH0_BASEADDR, 1);
  XGpio_mSetDataDirection(XPAR_GPIO_LED0_BASEADDR, 0);
  XGpio_mSetDataReg(XPAR_GPIO_LED0_BASEADDR, 0);

  while(1)
  {
    OnOff = XGpio_mGetDataReg(XPAR_GPIO_SWITCH0_BASEADDR);
    XGpio_mSetDataReg(XPAR_GPIO_LED0_BASEADDR, OnOff);
    XUartLite_SendByte(XPAR_UART_BASEADDR, button[OnOff]);
  }

}
```

Abbiamo utilizzato delle librerie di basso livello sia per il GPIO che per la UART.
La funzione *Xgpio_mSetDataDirection* [10] stabilisce la direzione dei dati per la periferica indicata: 0 corrisponde a input (per lo switch) e 1 corrisponde a output (per il led).
Settiamo poi il valore iniziale del led a spento con la funzione *Xgpio_mSetDataReg* [11].
I comandi del ciclo infine memorizzano in una variabile di nome *OnOff* lo stato dello switch e lo inviano al led e al terminale.

4. UTILIZZO DELLA ZBT RAM

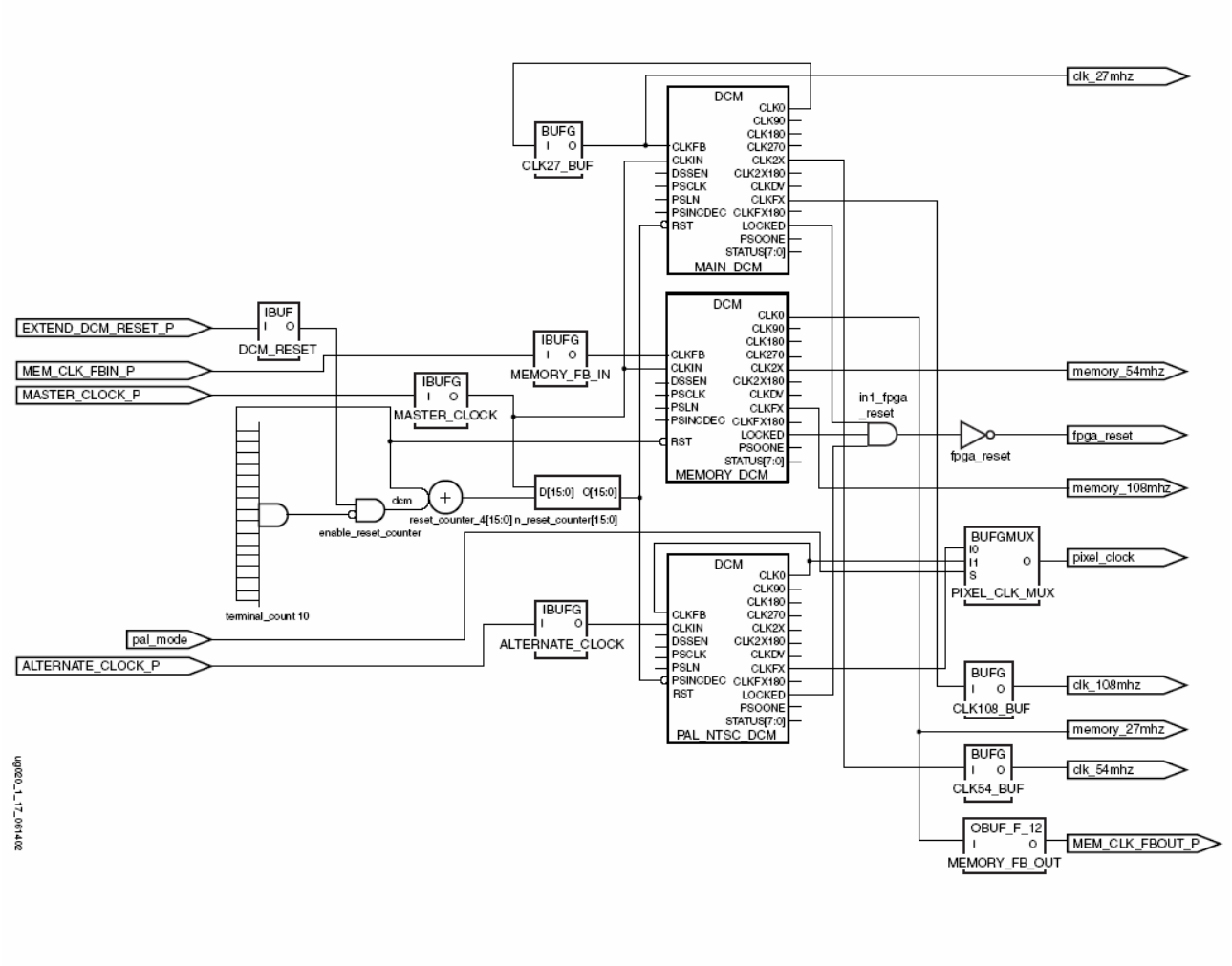
La MultiMedia Board é dotata di 5 banchi di ZBT RAM da 512 kilobytes ciascuno. In questo esempio vedremo come creare un sistema in cui far eseguire un programma dalla RAM esterna e memorizzare dati su di essa. Impareremo anche a importare un core scritto in VHDL o Verilog per essere utilizzato nel *Platform Studio* e a eseguire il debugging del software.

Il sistema da creare é molto simile al primo progetto (l'*helloworld*), ma dobbiamo aggiungere due cores: un modulo per la distribuzione del clock e il controller per la RAM esterna.
Nell'archivio *zbtram.zip* é contenuto il progetto completo e funzionante.

4.1 IL CLOCK GENERATOR

Quando abbiamo a che fare con un progetto che non sia estremamente semplice abbiamo bisogno di un dispositivo che distribuisca il clock all'interno del sistema evitando ritardi e ove necessario moltiplicando la frequenza del segnale per far sì che diversi dispositivi sulla stessa macchina possano lavorare a frequenze diverse.

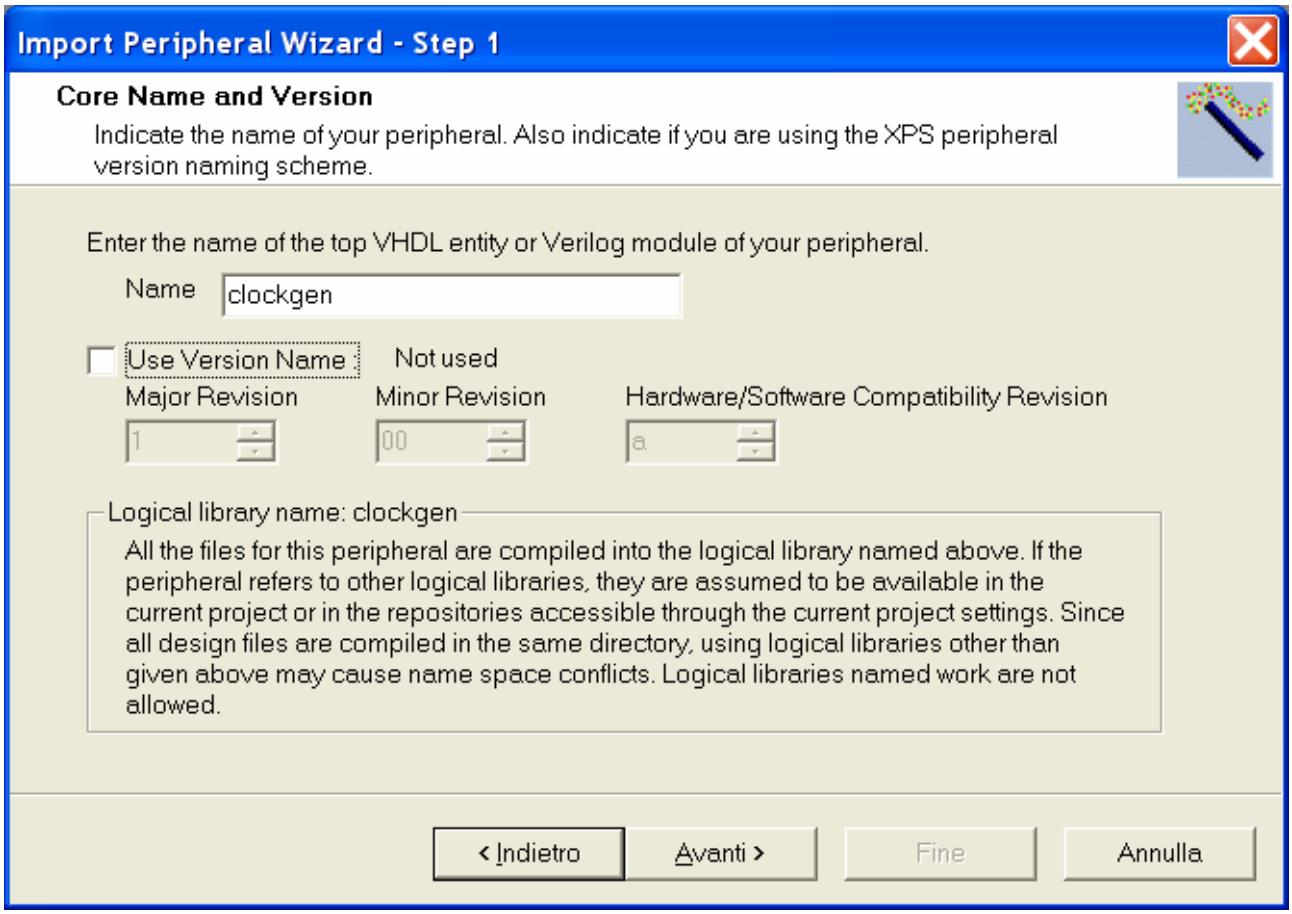
La struttura del componente é riportata in figura [12].

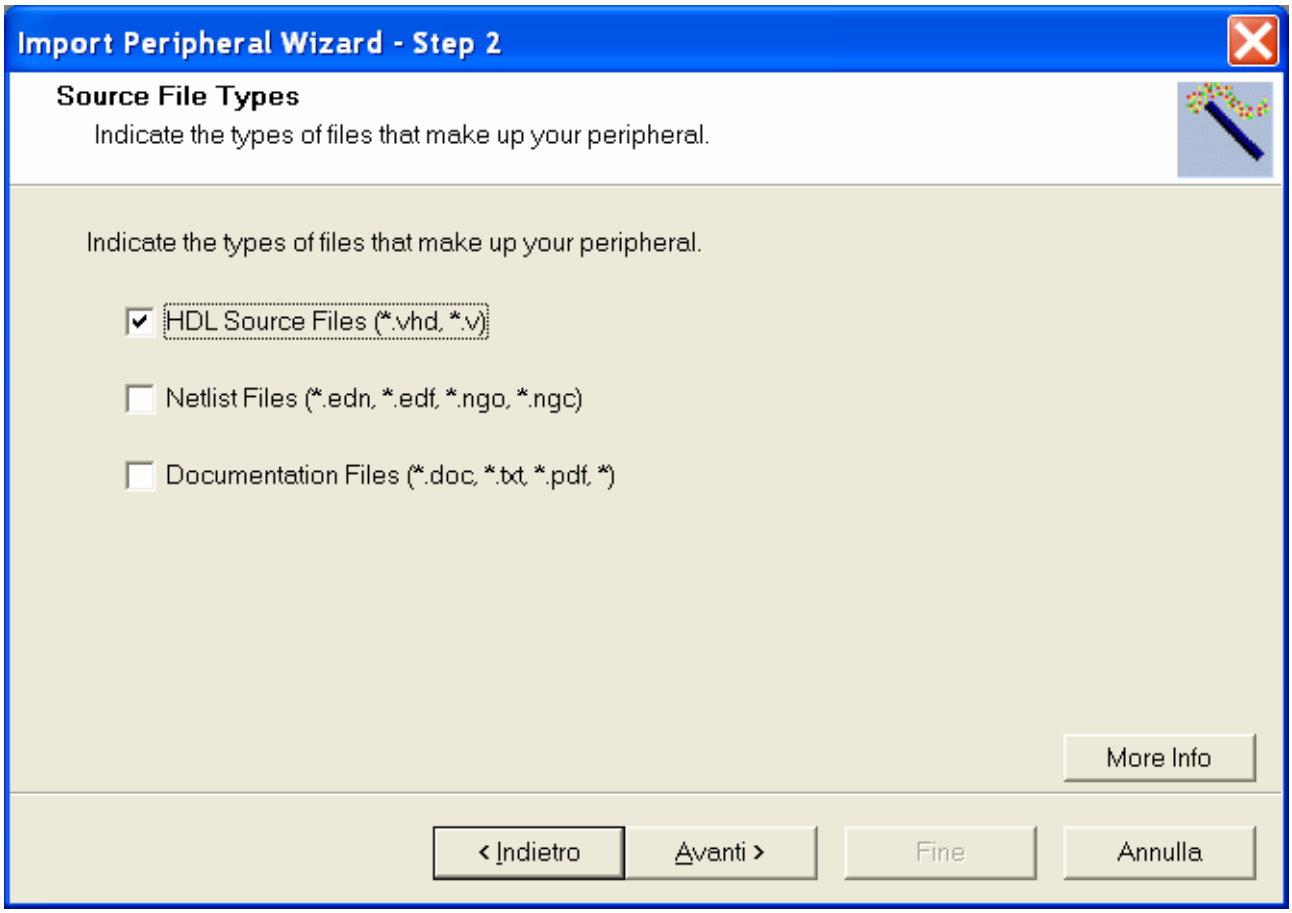


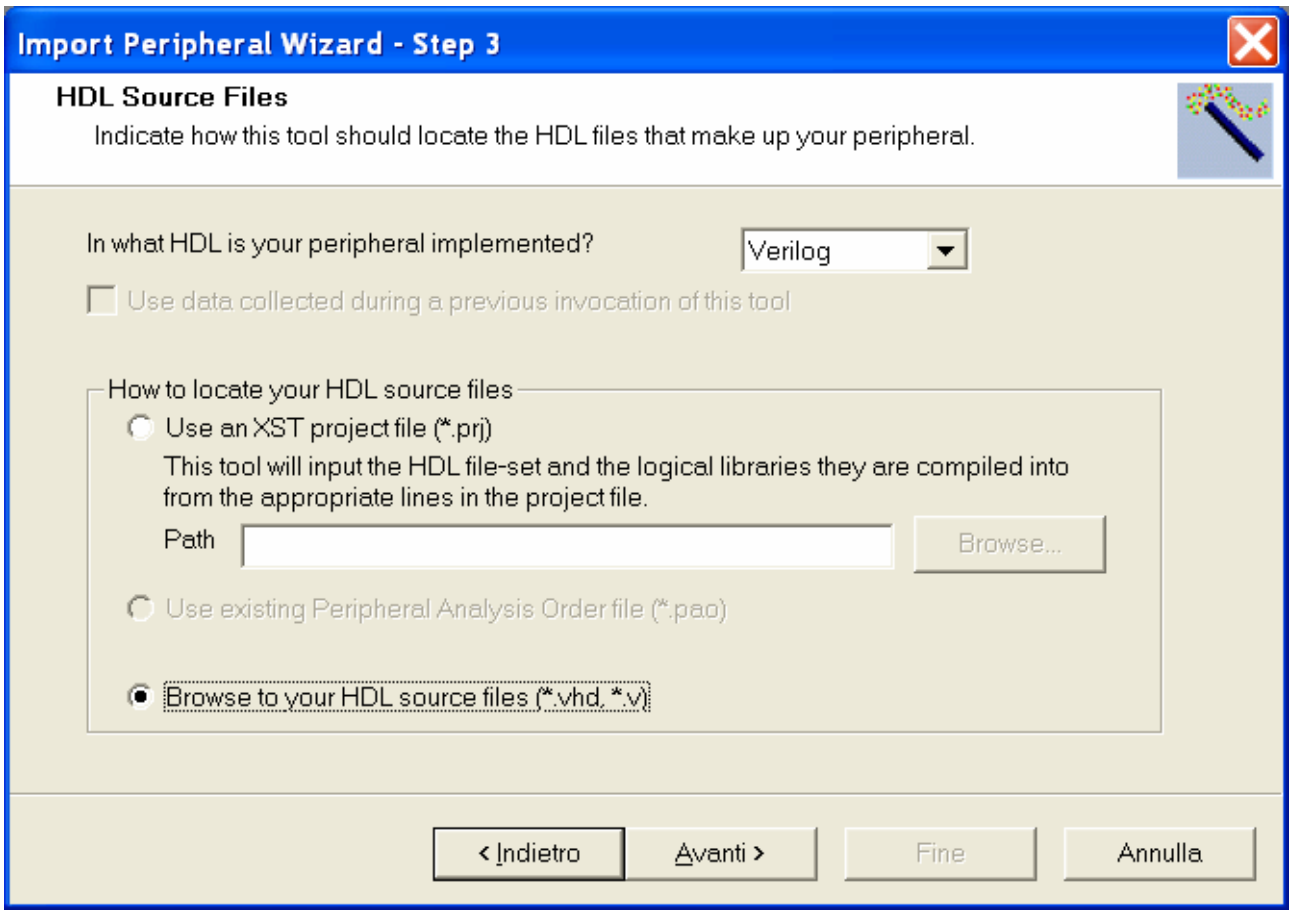
Questo componente non é presente fra i cores di base che possiamo scegliere per creare il nostro sistema; tuttavia la Xilinx fornisce il sorgente in Verilog [13], ed esso può essere importato tramite l'*Import Peripheral Wizard* per essere utilizzato nell'EDK. Conviene utilizzare la versione leggermente modificata di questo file che é inclusa nella directory */pcores* del progetto finito, poiché l'originale dà degli errori in fase di importazione.

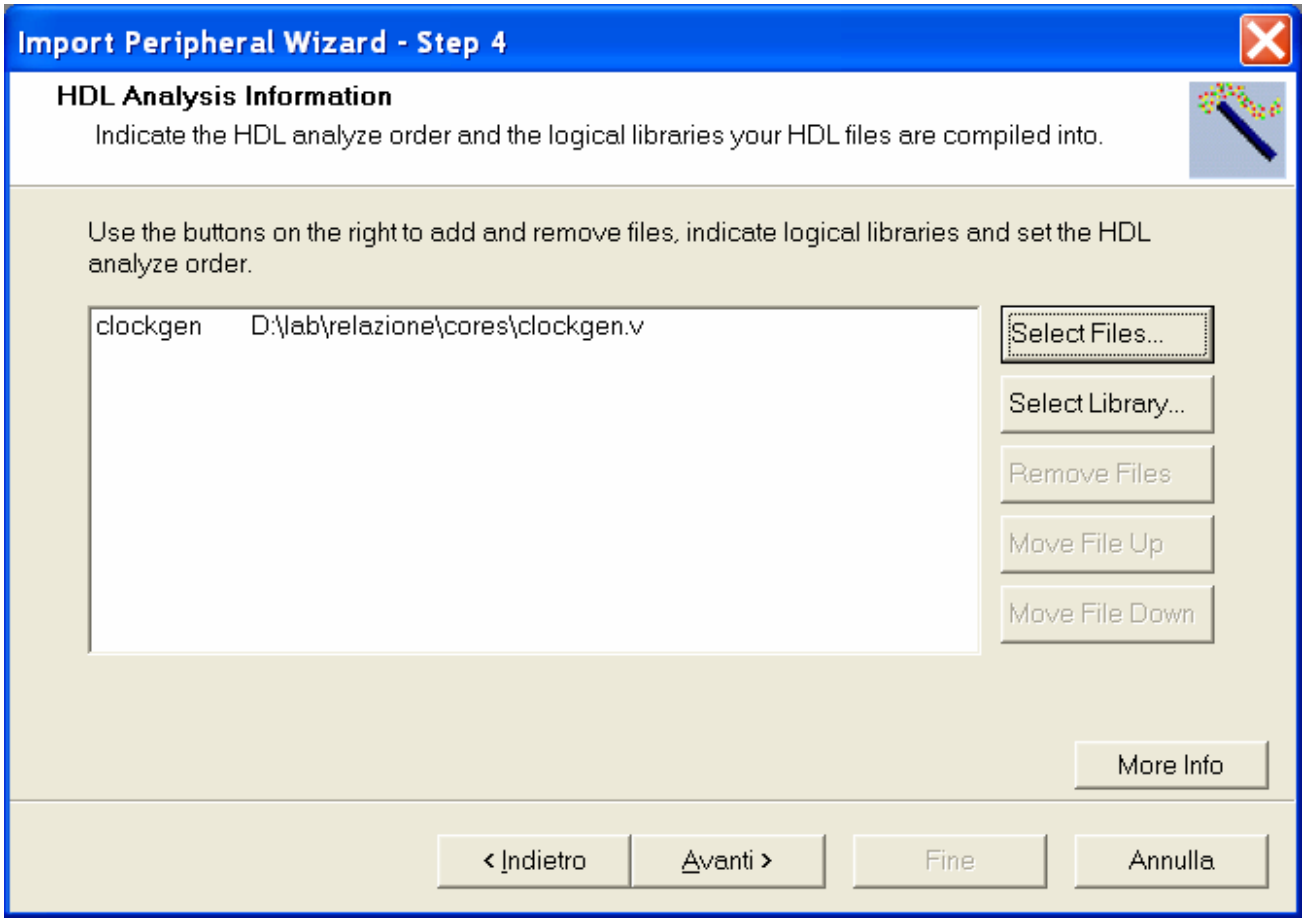
4.2 IMPORT PERIPHERAL WIZARD

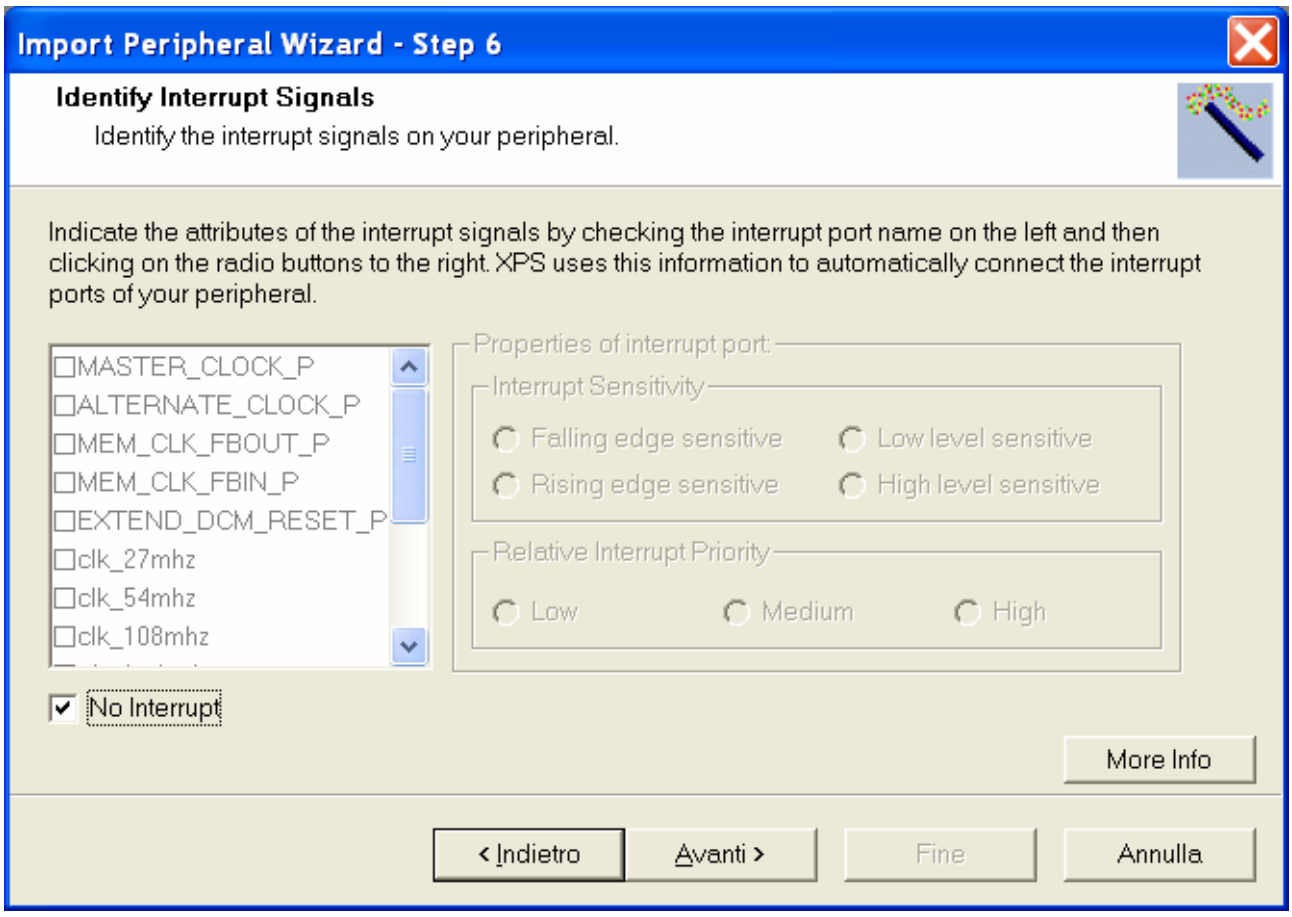
Cliccare su *Tools -> Import Peripheral Wizard ...* e seguire la procedura riportata nelle figure.

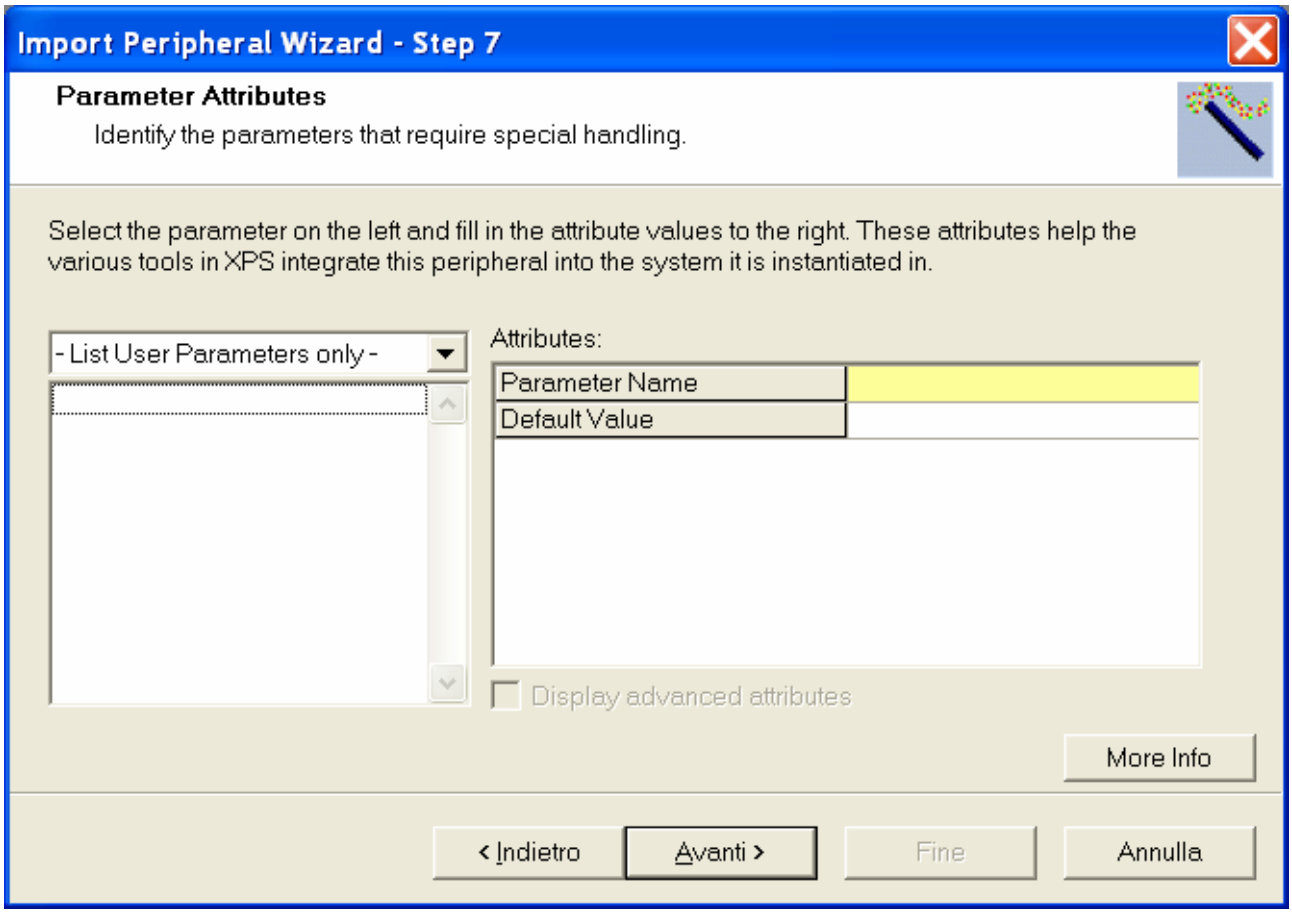


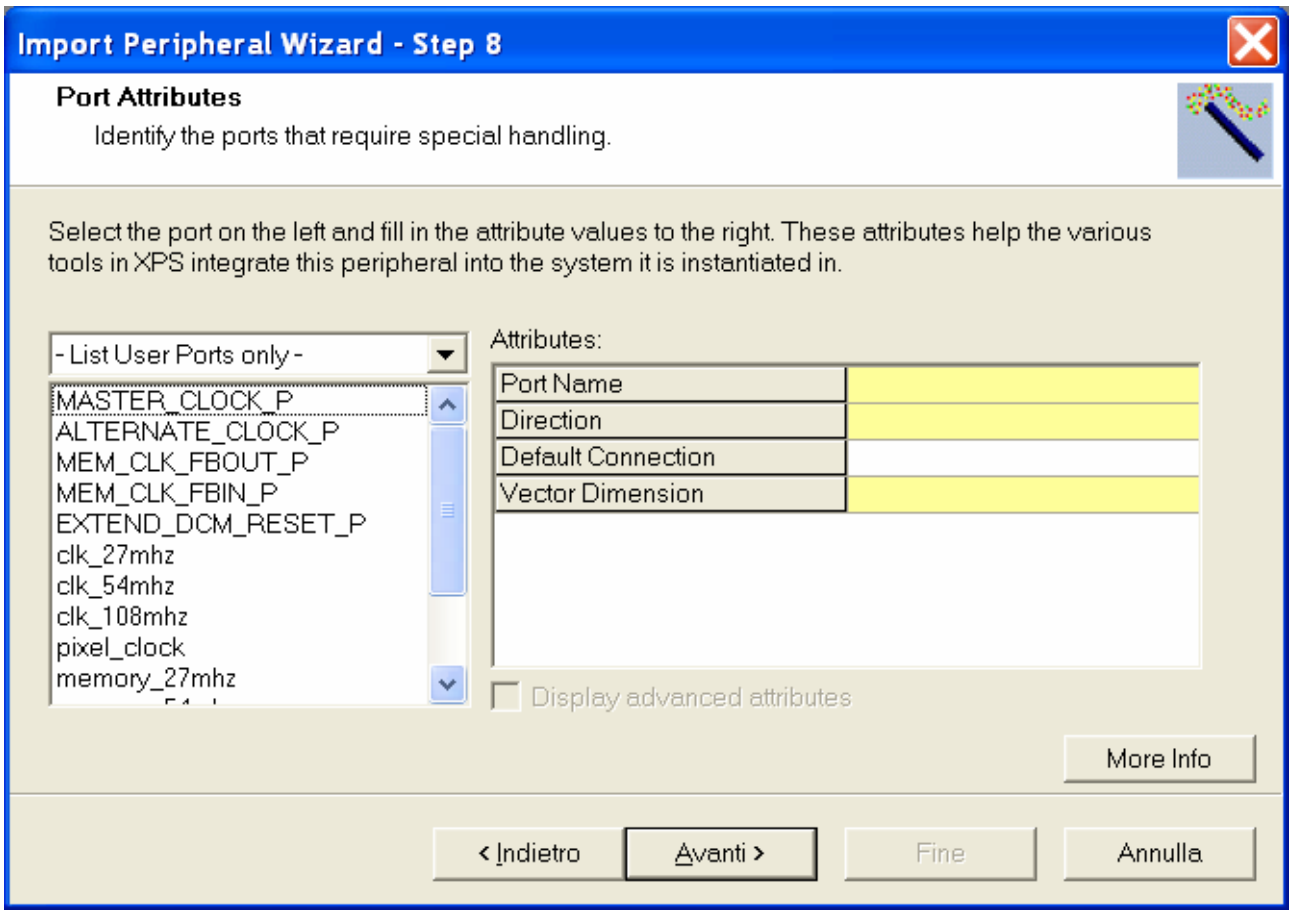

Import Peripheral Wizard - Step 2
Source File Types
Indicate the types of files that make up your peripheral.
Indicate the types of files that make up your peripheral.
HDL Source Files (*.vhd, *.v)
Netlist Files (*.edn, *.edf, *.ngo, *.ngc)
Documentation Files (*.doc, *.txt, *.pdf, *)
More Info
< Indietro
Avanti >
Fine
Annulla


Import Peripheral Wizard - Step 3
HDL Source Files
Indicate how this tool should locate the HDL files that make up your peripheral.
In what HDL is your peripheral implemented?
Verilog
Use data collected during a previous invocation of this tool
How to locate your HDL source files
Use an XST project file (*.prj)
This tool will input the HDL file-set and the logical libraries they are compiled into from the appropriate lines in the project file.
Path
Browse...
Use existing Peripheral Analysis Order file (*.pao)
Browse to your HDL source files (*.vhd, *.v)
< Indietro
Avanti >
Fine
Annulla


Import Peripheral Wizard - Step 4
HDL Analysis Information
Indicate the HDL analyze order and the logical libraries your HDL files are compiled into.
Use the buttons on the right to add and remove files, indicate logical libraries and set the HDL analyze order.
clockgen D:\lab\relazione\cores\clockgen.v
Select Files...
Select Library...
Remove Files
Move File Up
Move File Down
More Info
< Indietro
Avanti >
Fine
Annulla


Import Peripheral Wizard - Step 6
Identify Interrupt Signals
Identify the interrupt signals on your peripheral.
Indicate the attributes of the interrupt signals by checking the interrupt port name on the left and then clicking on the radio buttons to the right. XPS uses this information to automatically connect the interrupt ports of your peripheral.
MASTER_CLOCK_P
ALTERNATE_CLOCK_P
MEM_CLK_FBOUT_P
MEM_CLK_FBIN_P
EXTEND_DCM_RESET_P
clk_27mhz
clk_54mhz
clk_108mhz
Properties of interrupt port:
Interrupt Sensitivity
Falling edge sensitive
Low level sensitive
Rising edge sensitive
High level sensitive
Relative Interrupt Priority
Low
Medium
High
No Interrupt
More Info
< Indietro
Avanti >
Fine
Annulla


Import Peripheral Wizard - Step 7
Parameter Attributes
Identify the parameters that require special handling.
Select the parameter on the left and fill in the attribute values to the right. These attributes help the various tools in XPS integrate this peripheral into the system it is instantiated in.
- List User Parameters only -
Attributes:
Parameter Name
Default Value
Display advanced attributes
More Info
< Indietro
Avanti >
Fine
Annulla


Import Peripheral Wizard - Step 8
Port Attributes
Identify the ports that require special handling.
Select the port on the left and fill in the attribute values to the right. These attributes help the various tools in XPS integrate this peripheral into the system it is instantiated in.
- List User Ports only -
MASTER_CLOCK_P
ALTERNATE_CLOCK_P
MEM_CLK_FBOUT_P
MEM_CLK_FBIN_P
EXTEND_DCM_RESET_P
clk_27mhz
clk_54mhz
clk_108mhz
pixel_clock
memory_27mhz
Attributes:
Port Name
Direction
Default Connection
Vector Dimension
Display advanced attributes
More Info
< Indietro
Avanti >
Fine
Annulla

Conclusa la procedura é necessario chiudere e riaprire il progetto per poter accedere all'IP importato.
Se andiamo poi su *Add Cores*, vediamo che nella lista dei componenti é comparso *clock_gen*.
Aggiungerlo e cliccare su *OK*.
Nella finestra a sinistra cliccare sull'istanza del *clock_gen* da noi creato con il tasto destro del mouse e quindi su *View MPD* (Microprocessor Peripheral Description) [14].
Questo é un file che é stato creato automaticamente analizzando il codice HDL per trovare le connessioni possibili con l'esterno. Purtroppo però esso dovrà essere modificato a mano poiché alcune delle impostazioni sono errate e darebbero origine a errori.

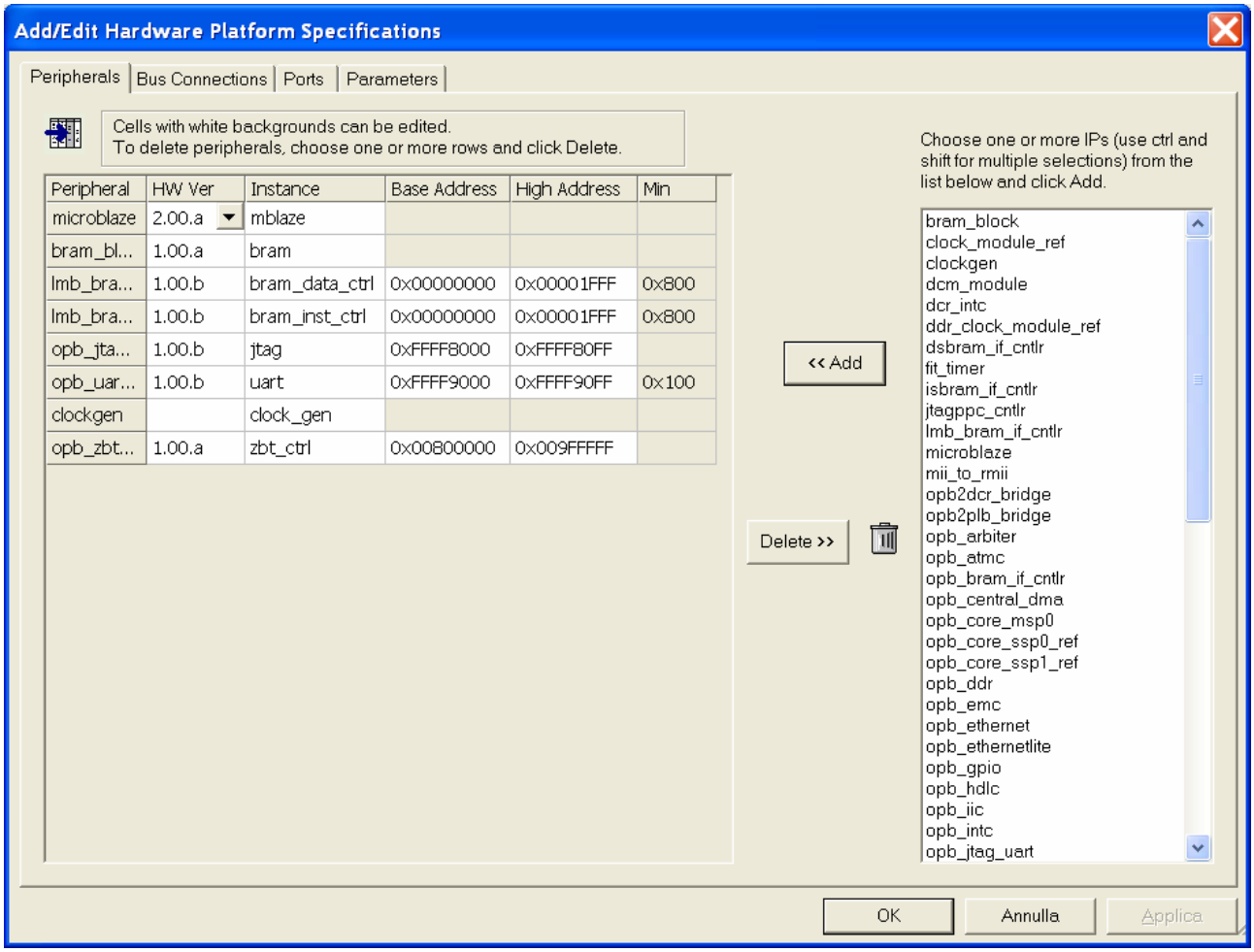
Nella sezione *Peripheral Options* sostituiamo *OPTION IPTYPE = PERIPHERAL* con *OPTION IPTYPE = IP*; possiamo leggere sulla documentazione del formato MPD che l'opzione *PERIPHERAL* va impostata quando il core é connesso con il MicroBlaze tramite bus; il nostro core non ha invece a che fare con il MicroBlaze ma é connesso a un livello puramente hardware con il resto del sistema.
Un'altra impostazione che può risultare spesso utile é *OPTION CORE_STATE = DEVELOPMENT*. Aggiungendo quest'opzione il codice HDL per la periferica a cui si riferisce il file MPD verrà ricompilato ogni volta che verrà eseguito *Generate Netlist*; utile quindi per provare dei cambiamenti "al volo".

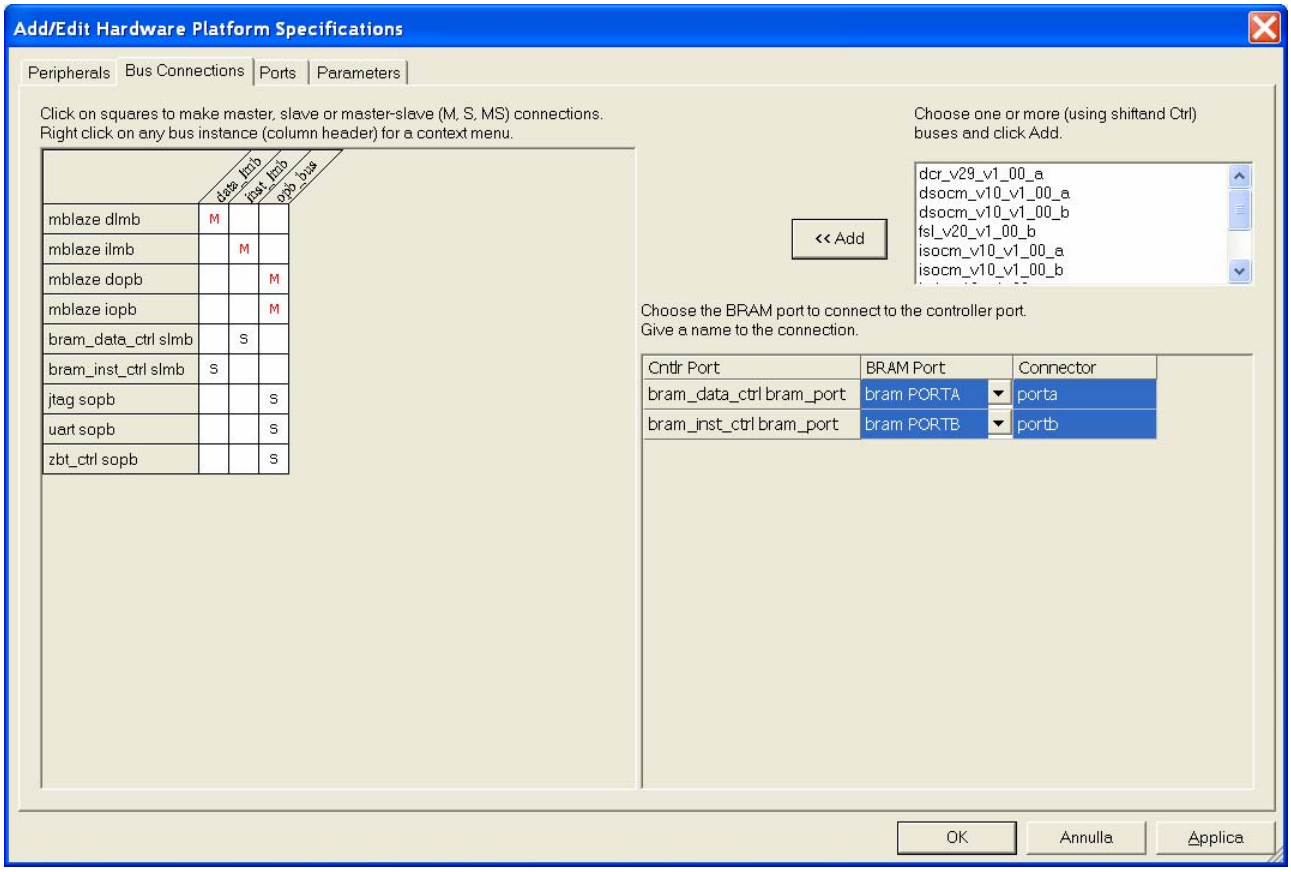
```
##############################################################################
##
## File     : D:\lab\relazione\zbtram\pcores\clockgen\data\clockgen_v2_1_0.mpd
## Desc     : Microprocessor Peripheral Description
##          : Automatically generated by PsfUtility
##
## Created  : Fri Mar 05 15:08:03 2004
##
##############################################################################

BEGIN clockgen

## Peripheral Options
OPTION IPTYPE = IP
OPTION IMP_NETLIST = TRUE
OPTION HDL = VERILOG


## Bus Interfaces

## Generics for VHDL or Parameters for Verilog

## Ports
PORT MASTER_CLOCK_P = "", DIR = I , IOB_STATE = BUF
PORT ALTERNATE_CLOCK_P = "", DIR = I , IOB_STATE = BUF
PORT MEM_CLK_FBOUT_P = "", DIR = O , IOB_STATE = BUF
PORT MEM_CLK_FBIN_P = "", DIR = I , IOB_STATE = BUF
PORT EXTEND_DCM_RESET_P = "", DIR = I , IOB_STATE = BUF
PORT clk_27mhz = "", DIR = O
PORT clk_54mhz = "", DIR = O
PORT clk_108mhz = "", DIR = O
PORT pixel_clock = "", DIR = O
PORT memory_27mhz = "", DIR = O
PORT memory_54mhz = "", DIR = O
PORT memory_108mhz = "", DIR = O
PORT pal_mode = "", DIR = I
PORT fpga_reset = "", DIR = O

END
```

I parametri più importanti da modificare sono nella sezione *Ports*. Dobbiamo infatti aggiungere alle porte *MASTER_CLOCK_P, ALTERNATE_CLOCK_P, MEM_CLK_FBOUT_P, MEM_CLK_FBIN_P* e *EXTEND_DCM_RESET_P* il parametro *IOB_STATE = BUF*. Esso indica di aggiungere a tutte queste porte un buffer interno, poiché sono connesse direttamente ad un segnale esterno; senza questa aggiunta il core darebbe errore in fase di implementazione.

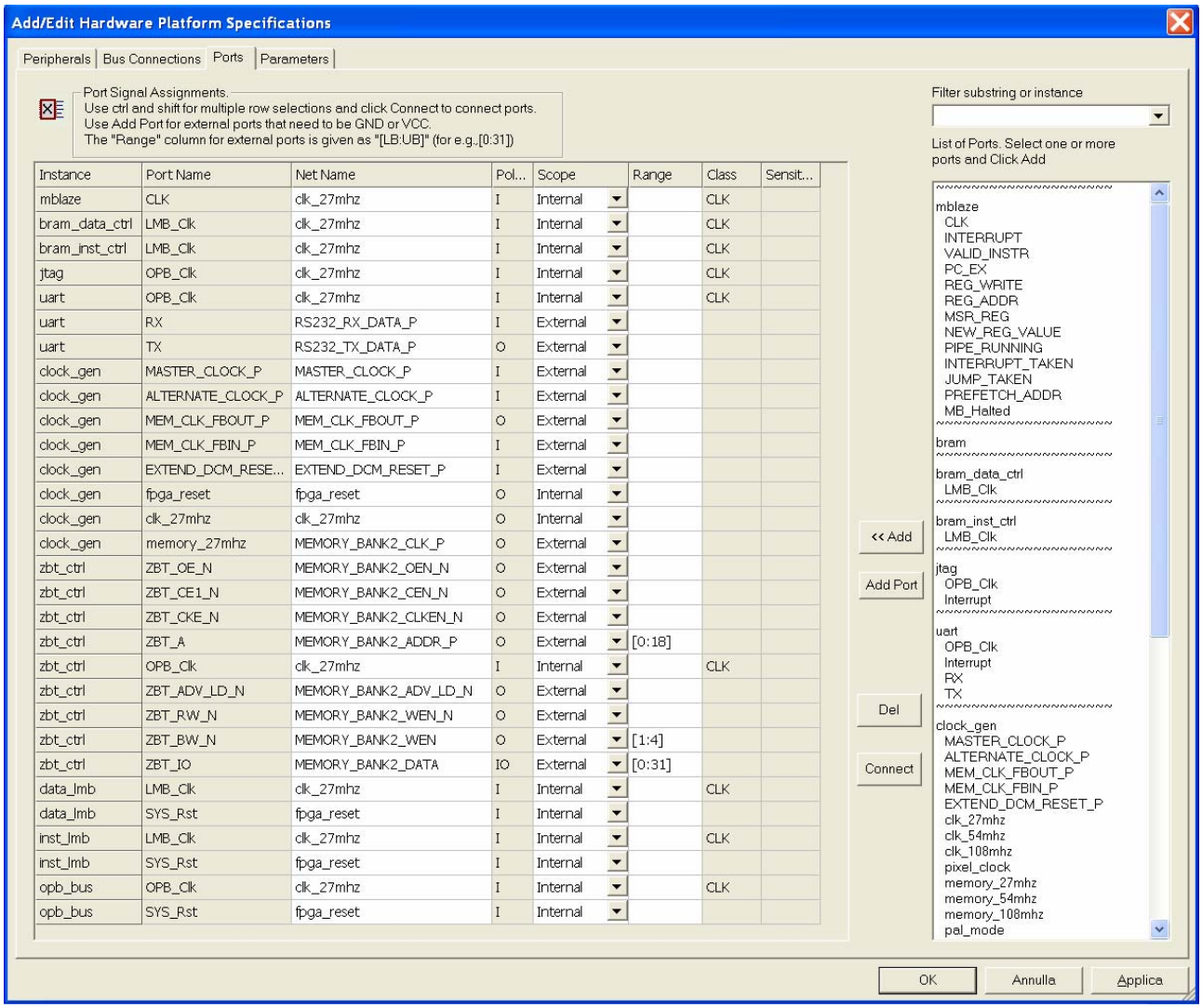
4.3 IL CONTROLLER PER LA ZBT RAM E LA CONFIGURAZIONE DEL SISTEMA

Dobbiamo ora aggiungere il controller per la RAM esterna *opb_zbt_controller*; quando cerchiamo di cambiare sezione il programma ci avvertirà che questo core é *deprecated*; il suo utilizzo é sconsigliato perché non é più supportato, ma visto che non ce ne sono altri che fanno al caso nostro e che questo comunque funziona non possiamo fare altro che ignorare l'avvertimento.





Colleghiamo tutte le porte come indicato, la RAM necessita di un gran numero di segnali per funzionare.



Ora il reset é gestito dal *clock_gen*, che ne trasforma il suo stato attivo da alto a basso; quindi se eravamo partiti dai progetti precedenti rimuovere il parametro *C_EXT_RESET_HIGH* a 0 che avevamo impostato prima e lasciamo così il valore di default che é 1.
Dobbiamo modificare inoltre due parametri dell'*opb_zbt_controller*: poniamo *C_ZBT_ADDR_SIZE* a 19 e *C_EXTERNAL_DLL* a 1.
Il primo parametro indica il numero di indirizzi della memoria, facendo un rapido calcolo vediamo che 2 alla diciannovesima potenza corrisponde proprio alla dimensione della memoria che é di 512 KB. Il secondo indica al controller della memoria di non implementare un DLL interno per il riallineamento del clock in quanto questa funzione é già svolta dal clock generator.
Per avere informazioni più precise sui parametri del controller o di una qualsiasi altra periferica basta consultare i rispettivi data sheets.

Non ci resta che modificare il file *system.ucf* [2] con le constraints relative al clock generator e al controller della RAM:

```
# SYSTEM CLOCKS
NET "MASTER_CLOCK_P" LOC = "AH15";
NET "ALTERNATE_CLOCK_P" LOC = "AD16";

# MEMORY CLOCK FEEDBACK LOOP
NET "MEM_CLK_FBIN_P" LOC = "AE15";
NET "MEM_CLK_FBOUT_P" LOC = "AH14";
NET "MEM_CLK_FBOUT_P" FAST;
NET "MEM_CLK_FBOUT_P" IOSTANDARD = LVTTL;
NET "MEM_CLK_FBOUT_P" DRIVE = 12;

# DCM RESET EXTENSION
NET "EXTEND_DCM_RESET_P" LOC = "AH7";

# RS232 COMMUNICATION PORT
NET "RS232_TX_DATA_P" LOC = "C9";
NET "RS232_TX_DATA_P" SLOW;
NET "RS232_TX_DATA_P" IOSTANDARD = LVTTL;
NET "RS232_TX_DATA_P" DRIVE = 12;
NET "RS232_RX_DATA_P" LOC = "C8";

# define the DCMs
# fout = 27,54,108 MHz FOR INTERNAL LOGIC
INST  "clock_gen/clock_gen/MAIN_DCM" DLL_FREQUENCY_MODE = LOW;
INST  "clock_gen/clock_gen/MAIN_DCM" CLK_FEEDBACK = 1X;
INST  "clock_gen/clock_gen/MAIN_DCM" DUTY_CYCLE_CORRECTION = TRUE;
INST  "clock_gen/clock_gen/MAIN_DCM" STARTUP_WAIT = FALSE;
INST  "clock_gen/clock_gen/MAIN_DCM" DFS_FREQUENCY_MODE = LOW;
INST  "clock_gen/clock_gen/MAIN_DCM" CLKFX_DIVIDE = 1;
INST  "clock_gen/clock_gen/MAIN_DCM" CLKFX_MULTIPLY = 4;
INST  "clock_gen/clock_gen/MAIN_DCM" CLKOUT_PHASE_SHIFT = NONE;
INST  "clock_gen/clock_gen/MAIN_DCM" PHASE_SHIFT = 0;

# fout = 27,54,108 MHz FOR MEMORIES (uses external feedback loop)
INST  "clock_gen/clock_gen/MEMORY_DCM" DLL_FREQUENCY_MODE = LOW;
INST  "clock_gen/clock_gen/MEMORY_DCM" CLK_FEEDBACK = 1X;
INST  "clock_gen/clock_gen/MEMORY_DCM" DUTY_CYCLE_CORRECTION = TRUE;
INST  "clock_gen/clock_gen/MEMORY_DCM" STARTUP_WAIT = FALSE;
INST  "clock_gen/clock_gen/MEMORY_DCM" DFS_FREQUENCY_MODE = LOW;
INST  "clock_gen/clock_gen/MEMORY_DCM" CLKFX_DIVIDE = 1;
INST  "clock_gen/clock_gen/MEMORY_DCM" CLKFX_MULTIPLY = 4;
INST  "clock_gen/clock_gen/MEMORY_DCM" CLKOUT_PHASE_SHIFT = NONE;
INST  "clock_gen/clock_gen/MEMORY_DCM" PHASE_SHIFT = 0;


# fout = 40.00MHz for 60Hz vertical refresh NTSC
# fout = 50.00MHz for 75Hz vertical refresh PAL
INST  "clock_gen/clock_gen/PAL_NTSC_DCM" DLL_FREQUENCY_MODE = LOW;
INST  "clock_gen/clock_gen/PAL_NTSC_DCM" CLK_FEEDBACK = 1X;
INST  "clock_gen/clock_gen/PAL_NTSC_DCM" DUTY_CYCLE_CORRECTION = TRUE;
INST  "clock_gen/clock_gen/PAL_NTSC_DCM" STARTUP_WAIT = FALSE;
INST  "clock_gen/clock_gen/PAL_NTSC_DCM" DFS_FREQUENCY_MODE = LOW;
INST  "clock_gen/clock_gen/PAL_NTSC_DCM" CLKFX_DIVIDE = 5;
INST  "clock_gen/clock_gen/PAL_NTSC_DCM" CLKFX_MULTIPLY = 4;
INST  "clock_gen/clock_gen/PAL_NTSC_DCM" CLKOUT_PHASE_SHIFT = NONE;
INST  "clock_gen/clock_gen/PAL_NTSC_DCM" PHASE_SHIFT = 0;



# PINOUT CONSTRAINTS FOR THE MICROBLAZE and MULTIMEDIA DEMO BOARD
# ZBT RAM MEMORY BANK2
NET "MEMORY_BANK2_CLK_P" LOC = "AJ22";
NET "MEMORY_BANK2_CLK_P" FAST;
NET "MEMORY_BANK2_CLK_P" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_CLK_P" DRIVE = 12;
NET "MEMORY_BANK2_CLKEN_N" LOC = "AE20";
NET "MEMORY_BANK2_CLKEN_N" FAST;
NET "MEMORY_BANK2_CLKEN_N" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_CLKEN_N" DRIVE = 12;
NET "MEMORY_BANK2_WEN_N" LOC = "AJ23";
NET "MEMORY_BANK2_WEN_N" FAST;
NET "MEMORY_BANK2_WEN_N" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_WEN_N" DRIVE = 12;
NET "MEMORY_BANK2_WEN<1>" LOC = "AF24";
NET "MEMORY_BANK2_WEN<1>" FAST;
NET "MEMORY_BANK2_WEN<1>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_WEN<1>" DRIVE = 12;
```

```
NET "MEMORY_BANK2_WEN<2>" LOC = "AG23";
NET "MEMORY_BANK2_WEN<2>" FAST;
NET "MEMORY_BANK2_WEN<2>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_WEN<2>" DRIVE = 12;
NET "MEMORY_BANK2_WEN<3>" LOC = "AD20";
NET "MEMORY_BANK2_WEN<3>" FAST;
NET "MEMORY_BANK2_WEN<3>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_WEN<3>" DRIVE = 12;
NET "MEMORY_BANK2_WEN<4>" LOC = "AD21";
NET "MEMORY_BANK2_WEN<4>" FAST;
NET "MEMORY_BANK2_WEN<4>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_WEN<4>" DRIVE = 12;
NET "MEMORY_BANK2_ADV_LD_N" LOC = "AF23";
NET "MEMORY_BANK2_ADV_LD_N" FAST;
NET "MEMORY_BANK2_ADV_LD_N" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_ADV_LD_N" DRIVE = 12;
NET "MEMORY_BANK2_OEN_N" LOC = "AE19";
NET "MEMORY_BANK2_OEN_N" FAST;
NET "MEMORY_BANK2_OEN_N" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_OEN_N" DRIVE = 12;
NET "MEMORY_BANK2_CEN_N" LOC = "AK25";
NET "MEMORY_BANK2_CEN_N" FAST;
NET "MEMORY_BANK2_CEN_N" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_CEN_N" DRIVE = 12;
NET "MEMORY_BANK2_ADDR_P<18>" LOC = "AF22";
NET "MEMORY_BANK2_ADDR_P<18>" FAST;
NET "MEMORY_BANK2_ADDR_P<18>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_ADDR_P<18>" DRIVE = 12;
NET "MEMORY_BANK2_ADDR_P<17>" LOC = "AG21";
NET "MEMORY_BANK2_ADDR_P<17>" FAST;
NET "MEMORY_BANK2_ADDR_P<17>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_ADDR_P<17>" DRIVE = 12;
NET "MEMORY_BANK2_ADDR_P<16>" LOC = "AE11";
NET "MEMORY_BANK2_ADDR_P<16>" FAST;
NET "MEMORY_BANK2_ADDR_P<16>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_ADDR_P<16>" DRIVE = 12;
NET "MEMORY_BANK2_ADDR_P<15>" LOC = "AJ7";
NET "MEMORY_BANK2_ADDR_P<15>" FAST;
NET "MEMORY_BANK2_ADDR_P<15>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_ADDR_P<15>" DRIVE = 12;
NET "MEMORY_BANK2_ADDR_P<14>" LOC = "AJ8";
NET "MEMORY_BANK2_ADDR_P<14>" FAST;
NET "MEMORY_BANK2_ADDR_P<14>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_ADDR_P<14>" DRIVE = 12;
NET "MEMORY_BANK2_ADDR_P<13>" LOC = "AG7";
NET "MEMORY_BANK2_ADDR_P<13>" FAST;
NET "MEMORY_BANK2_ADDR_P<13>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_ADDR_P<13>" DRIVE = 12;
NET "MEMORY_BANK2_ADDR_P<12>" LOC = "AG8";
NET "MEMORY_BANK2_ADDR_P<12>" FAST;
NET "MEMORY_BANK2_ADDR_P<12>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_ADDR_P<12>" DRIVE = 12;
NET "MEMORY_BANK2_ADDR_P<11>" LOC = "AD11";
NET "MEMORY_BANK2_ADDR_P<11>" FAST;
NET "MEMORY_BANK2_ADDR_P<11>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_ADDR_P<11>" DRIVE = 12;
NET "MEMORY_BANK2_ADDR_P<10>" LOC = "AD10";
NET "MEMORY_BANK2_ADDR_P<10>" FAST;
NET "MEMORY_BANK2_ADDR_P<10>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_ADDR_P<10>" DRIVE = 12;
NET "MEMORY_BANK2_ADDR_P<9>" LOC = "AH23";
NET "MEMORY_BANK2_ADDR_P<9>" FAST;
NET "MEMORY_BANK2_ADDR_P<9>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_ADDR_P<9>" DRIVE = 12;
NET "MEMORY_BANK2_ADDR_P<8>" LOC = "AG22";
NET "MEMORY_BANK2_ADDR_P<8>" FAST;
NET "MEMORY_BANK2_ADDR_P<8>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_ADDR_P<8>" DRIVE = 12;
NET "MEMORY_BANK2_ADDR_P<7>" LOC = "AK24";
NET "MEMORY_BANK2_ADDR_P<7>" FAST;
NET "MEMORY_BANK2_ADDR_P<7>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_ADDR_P<7>" DRIVE = 12;
NET "MEMORY_BANK2_ADDR_P<6>" LOC = "AH24";
NET "MEMORY_BANK2_ADDR_P<6>" FAST;
NET "MEMORY_BANK2_ADDR_P<6>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_ADDR_P<6>" DRIVE = 12;
NET "MEMORY_BANK2_ADDR_P<5>" LOC = "AE9";
```

```
NET "MEMORY_BANK2_ADDR_P<5>" FAST;
NET "MEMORY_BANK2_ADDR_P<5>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_ADDR_P<5>" DRIVE = 12;
NET "MEMORY_BANK2_ADDR_P<4>" LOC = "AE10";
NET "MEMORY_BANK2_ADDR_P<4>" FAST;
NET "MEMORY_BANK2_ADDR_P<4>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_ADDR_P<4>" DRIVE = 12;
NET "MEMORY_BANK2_ADDR_P<3>" LOC = "AF7";
NET "MEMORY_BANK2_ADDR_P<3>" FAST;
NET "MEMORY_BANK2_ADDR_P<3>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_ADDR_P<3>" DRIVE = 12;
NET "MEMORY_BANK2_ADDR_P<2>" LOC = "AF8";
NET "MEMORY_BANK2_ADDR_P<2>" FAST;
NET "MEMORY_BANK2_ADDR_P<2>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_ADDR_P<2>" DRIVE = 12;
NET "MEMORY_BANK2_ADDR_P<1>" LOC = "AK7";
NET "MEMORY_BANK2_ADDR_P<1>" FAST;
NET "MEMORY_BANK2_ADDR_P<1>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_ADDR_P<1>" DRIVE = 12;
NET "MEMORY_BANK2_ADDR_P<0>" LOC = "AJ6";
NET "MEMORY_BANK2_ADDR_P<0>" FAST;
NET "MEMORY_BANK2_ADDR_P<0>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_ADDR_P<0>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<7>" LOC = "AK14";
NET "MEMORY_BANK2_DATA<7>" FAST;
NET "MEMORY_BANK2_DATA<7>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<7>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<6>" LOC = "AG15";
NET "MEMORY_BANK2_DATA<6>" FAST;
NET "MEMORY_BANK2_DATA<6>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<6>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<5>" LOC = "AC14";
NET "MEMORY_BANK2_DATA<5>" FAST;
NET "MEMORY_BANK2_DATA<5>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<5>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<4>" LOC = "AJ14";
NET "MEMORY_BANK2_DATA<4>" FAST;
NET "MEMORY_BANK2_DATA<4>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<4>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<3>" LOC = "AF11";
NET "MEMORY_BANK2_DATA<3>" FAST;
NET "MEMORY_BANK2_DATA<3>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<3>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<2>" LOC = "AH8";
NET "MEMORY_BANK2_DATA<2>" FAST;
NET "MEMORY_BANK2_DATA<2>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<2>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<1>" LOC = "AK9";
NET "MEMORY_BANK2_DATA<1>" FAST;
NET "MEMORY_BANK2_DATA<1>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<1>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<0>" LOC = "AG9";
NET "MEMORY_BANK2_DATA<0>" FAST;
NET "MEMORY_BANK2_DATA<0>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<0>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<15>" LOC = "AH22";
NET "MEMORY_BANK2_DATA<15>" FAST;
NET "MEMORY_BANK2_DATA<15>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<15>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<14>" LOC = "AF21";
NET "MEMORY_BANK2_DATA<14>" FAST;
NET "MEMORY_BANK2_DATA<14>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<14>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<13>" LOC = "AK17";
NET "MEMORY_BANK2_DATA<13>" FAST;
NET "MEMORY_BANK2_DATA<13>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<13>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<12>" LOC = "AC16";
NET "MEMORY_BANK2_DATA<12>" FAST;
NET "MEMORY_BANK2_DATA<12>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<12>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<11>" LOC = "AG17";
NET "MEMORY_BANK2_DATA<11>" FAST;
NET "MEMORY_BANK2_DATA<11>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<11>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<10>" LOC = "AJ16";
NET "MEMORY_BANK2_DATA<10>" FAST;
```

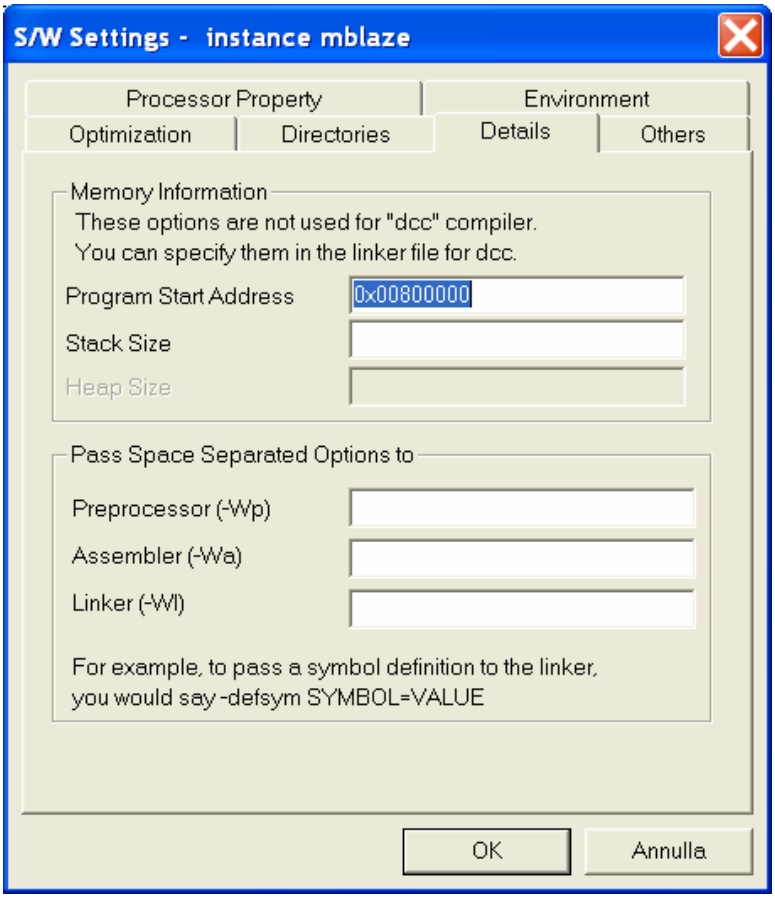
```
NET "MEMORY_BANK2_DATA<10>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<10>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<9>" LOC = "AH17";
NET "MEMORY_BANK2_DATA<9>" FAST;
NET "MEMORY_BANK2_DATA<9>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<9>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<8>" LOC = "AD15";
NET "MEMORY_BANK2_DATA<8>" FAST;
NET "MEMORY_BANK2_DATA<8>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<8>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<23>" LOC = "AK15";
NET "MEMORY_BANK2_DATA<23>" FAST;
NET "MEMORY_BANK2_DATA<23>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<23>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<22>" LOC = "AH16";
NET "MEMORY_BANK2_DATA<22>" FAST;
NET "MEMORY_BANK2_DATA<22>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<22>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<21>" LOC = "AE16";
NET "MEMORY_BANK2_DATA<21>" FAST;
NET "MEMORY_BANK2_DATA<21>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<21>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<20>" LOC = "AJ17";
NET "MEMORY_BANK2_DATA<20>" FAST;
NET "MEMORY_BANK2_DATA<20>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<20>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<19>" LOC = "AG16";
NET "MEMORY_BANK2_DATA<19>" FAST;
NET "MEMORY_BANK2_DATA<19>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<19>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<18>" LOC = "AC17";
NET "MEMORY_BANK2_DATA<18>" FAST;
NET "MEMORY_BANK2_DATA<18>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<18>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<17>" LOC = "AK18";
NET "MEMORY_BANK2_DATA<17>" FAST;
NET "MEMORY_BANK2_DATA<17>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<17>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<16>" LOC = "AF20";
NET "MEMORY_BANK2_DATA<16>" FAST;
NET "MEMORY_BANK2_DATA<16>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<16>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<31>" LOC = "AE12";
NET "MEMORY_BANK2_DATA<31>" FAST;
NET "MEMORY_BANK2_DATA<31>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<31>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<30>" LOC = "AG10";
NET "MEMORY_BANK2_DATA<30>" FAST;
NET "MEMORY_BANK2_DATA<30>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<30>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<29>" LOC = "AJ9";
NET "MEMORY_BANK2_DATA<29>" FAST;
NET "MEMORY_BANK2_DATA<29>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<29>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<28>" LOC = "AH9";
NET "MEMORY_BANK2_DATA<28>" FAST;
NET "MEMORY_BANK2_DATA<28>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<28>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<27>" LOC = "AF10";
NET "MEMORY_BANK2_DATA<27>" FAST;
NET "MEMORY_BANK2_DATA<27>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<27>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<26>" LOC = "AJ15";
NET "MEMORY_BANK2_DATA<26>" FAST;
NET "MEMORY_BANK2_DATA<26>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<26>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<25>" LOC = "AC15";
NET "MEMORY_BANK2_DATA<25>" FAST;
NET "MEMORY_BANK2_DATA<25>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<25>" DRIVE = 12;
NET "MEMORY_BANK2_DATA<24>" LOC = "AG14";
NET "MEMORY_BANK2_DATA<24>" FAST;
NET "MEMORY_BANK2_DATA<24>" IOSTANDARD = LVTTL;
NET "MEMORY_BANK2_DATA<24>" DRIVE = 12;
```

4.4 IL SOFTWARE

Utilizziamo il codice *helloworld.c* ma aggiungiamo un paio di righe che ci permettano di scrivere direttamente sull'area di memoria desiderata:

```
#include "stdio.h"
#define MEMADDR ((volatile unsigned long *) 0x00900000)

main()
{
   *MEMADDR=0xABCD1234;
   print("Hello World!\n\r");
   print("I am your new Microblaze\n\r");
}
```

Prima di compilare il file dobbiamo indicare a che indirizzo andrà scaricato l'eseguibile; indichiamo il *base address* del controller della RAM, come in figura.
Il valore di default é posto a 0, indirizzo a cui si trova la BRAM, quindi negli esempi precedenti non avevamo avuto bisogno di cambiare questo parametro.

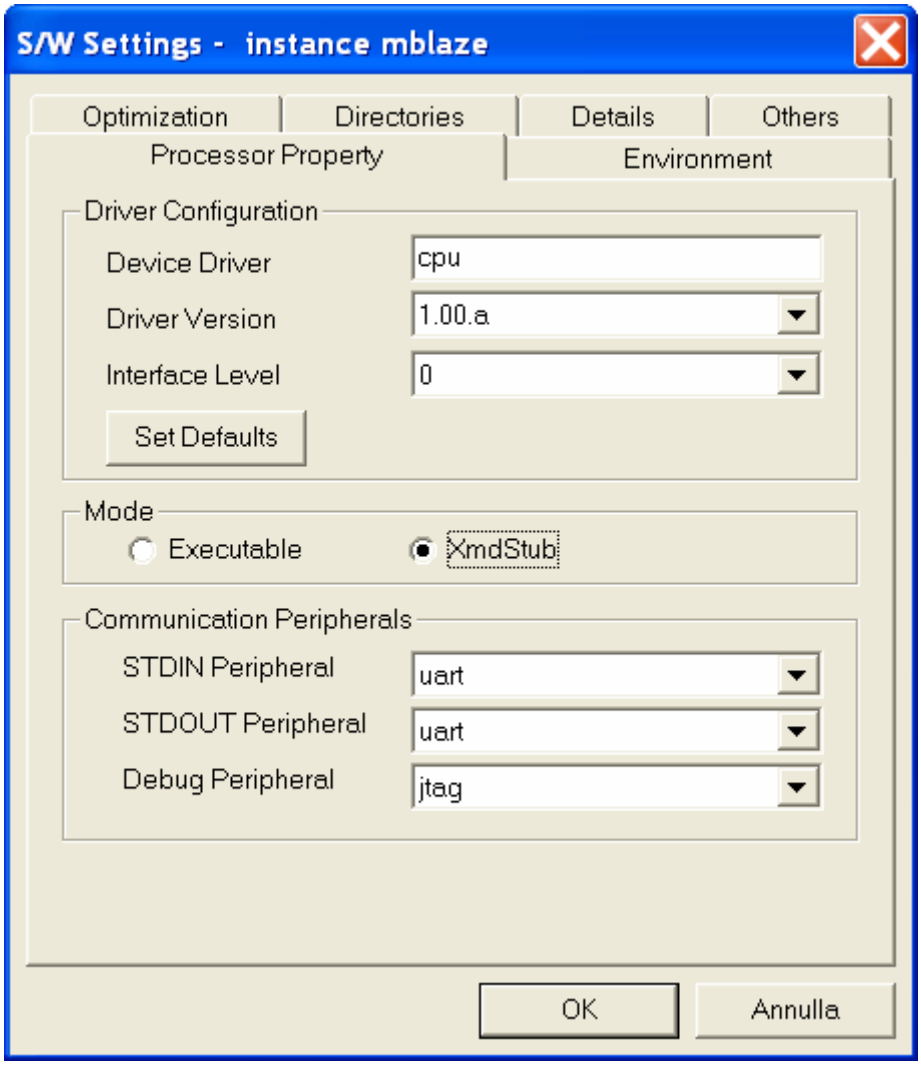


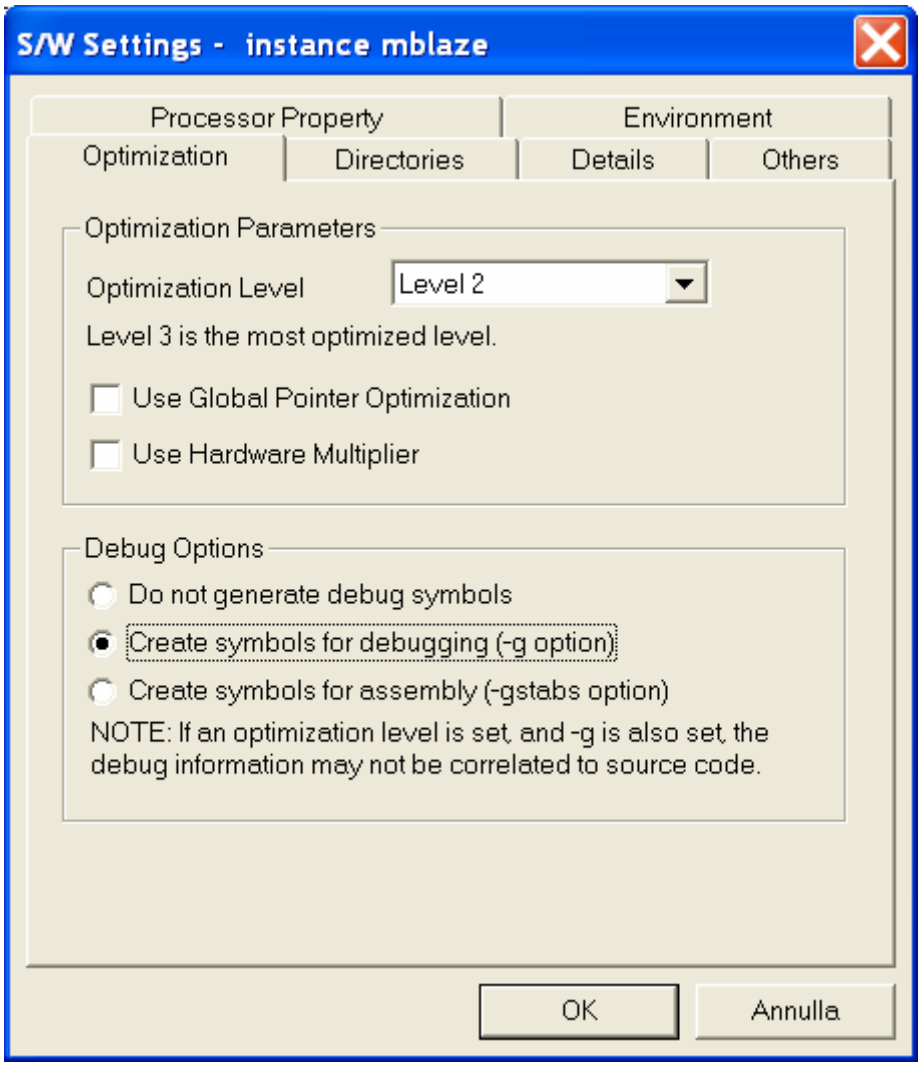
Infine compiliamo, aggiorniamo il bitstream e scarichiamolo sull'FPGA.
Vedremo nuovamente l'"Hello World!" sul nostro terminale.

4.5 IL DEBUGGER

Per vedere che effetto hanno avuto quelle poche righe che abbiamo aggiunto al programma originale vediamo l'utilizzo del debugger.

Innanzitutto modifichiamo il formato in cui verrà compilato il codice sorgente. Esso non sarà più direttamente eseguibile ma potrà essere avviato solo tramite debugger.
Gli indirizzi fino a 0x400 non saranno modificabili perché conterranno il codice di interfacciamento con il debugger.





Creando dei simboli per il debugging potremo seguire in parallelo lo svolgersi delle istruzioni C e il codice assembler a loro correlato.

E' preferibile fare *Clean Netlist* prima di eseguire le prossime operazioni.
Ricompilare il codice, aggiornare il bitstream e scaricare il bitstream sull'FPGA. Ora non apparirà niente sul nostro terminale perché il programma non va direttamente in esecuzione ma attende di essere caricato nel debugger.

Creiamo nella root del progetto un file *xmd.ini* [15] che funga da script per far sì che il debugger si connetta automaticamente al MicroBlaze al suo avvio:

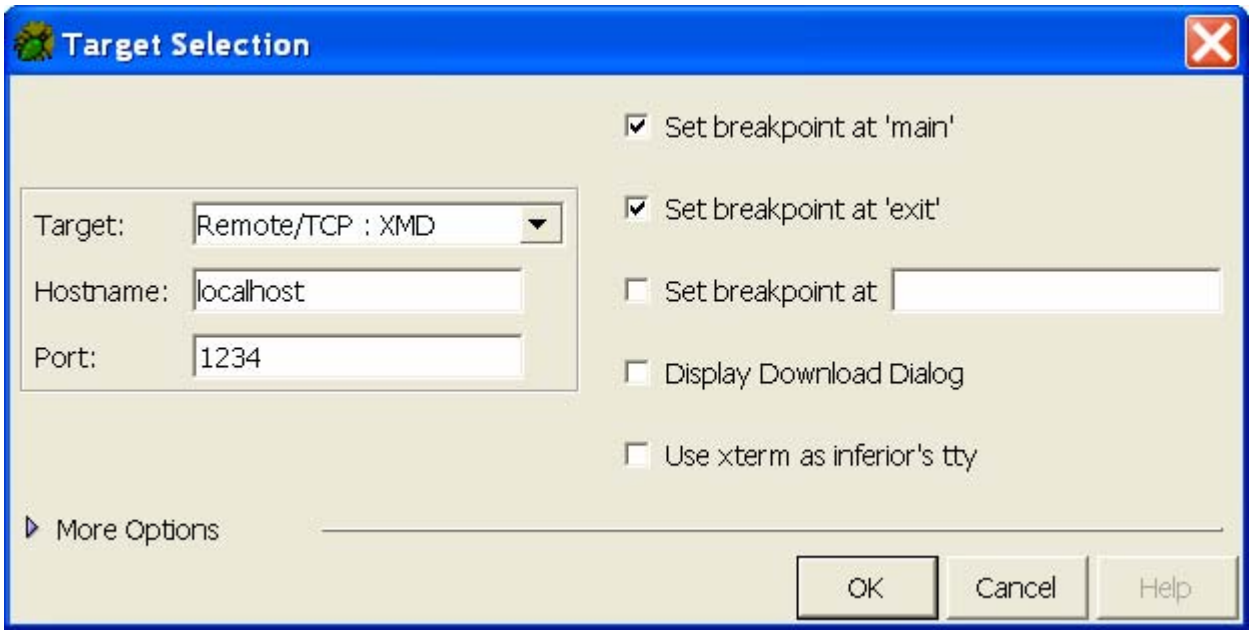
```
# Initialization Tcl script which is sourced by xmd on startup
# display the command help on startup
 mbconnect stub -comm jtag -debugdevice devicenr 2
```

Caricare il debugger con *Tools -> XMD*

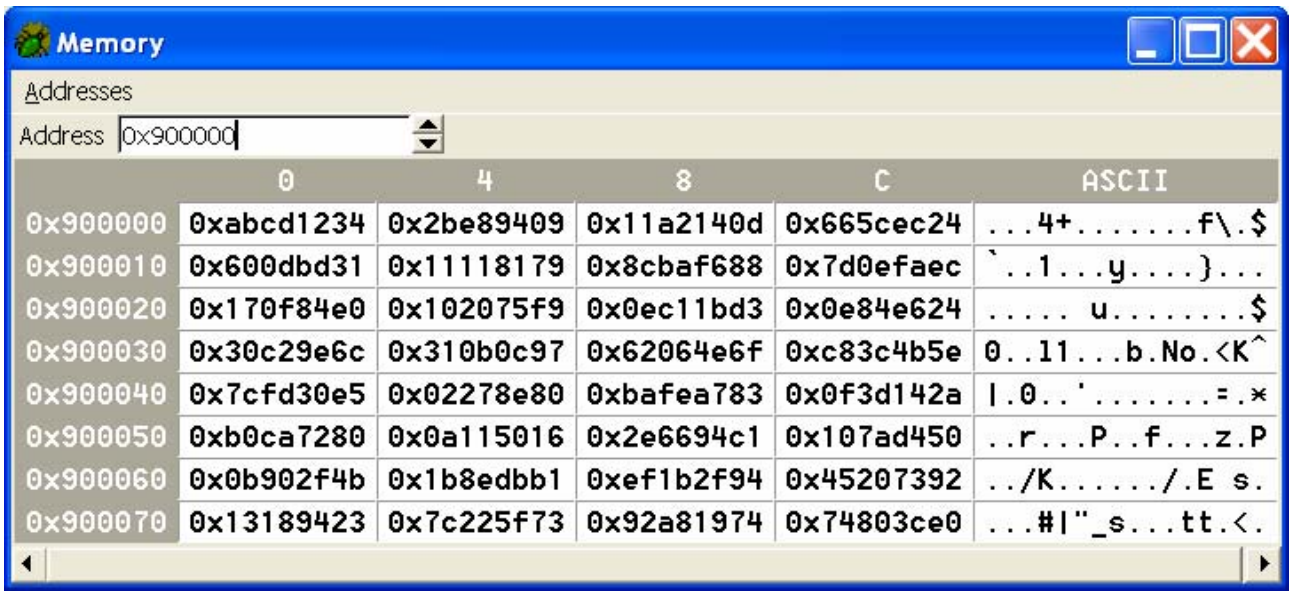
Grazie allo script esso si collega direttamente al MicroBlaze, tramite interfaccia testuale si può accedere alle varie locazioni di memoria e caricare il programma in memoria, ma é molto più comodo lavorare con un'interfaccia grafica che gestisca al meglio le sue funzioni.

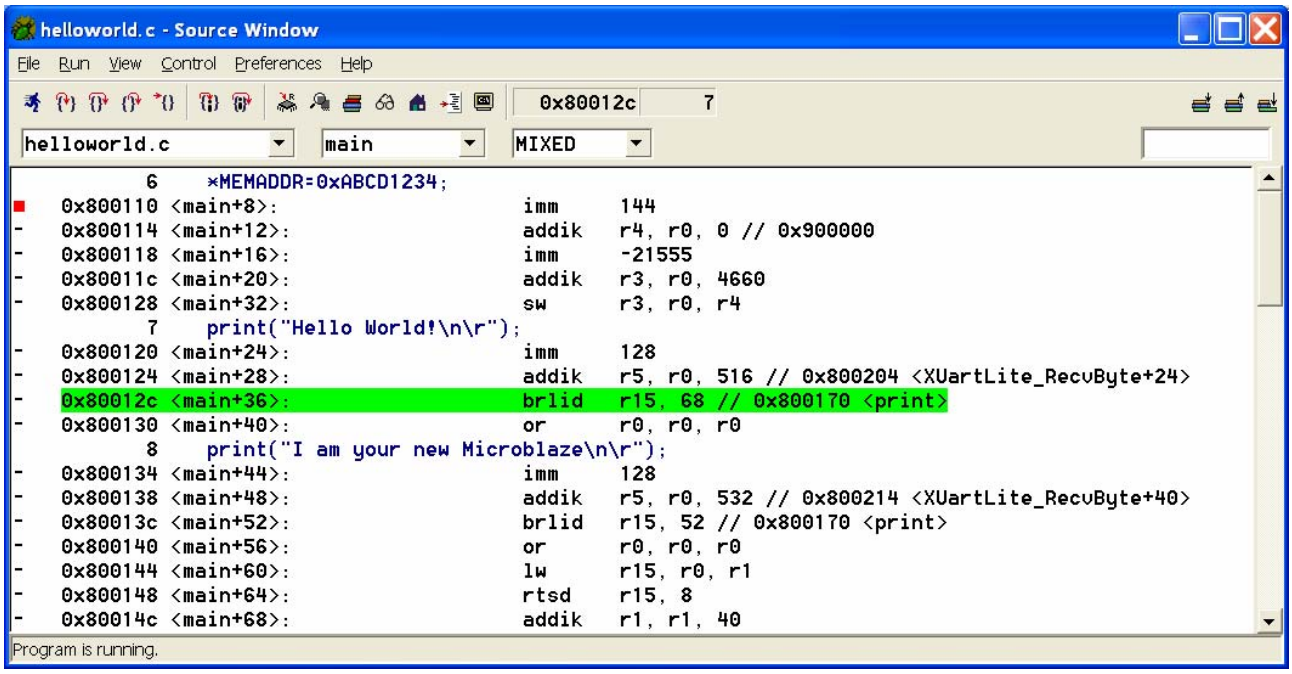
Cliccare quindi su *Tools -> Software Debugger.*

Una volta caricato il programma scegliere *Run -> Connect* to target e settare le impostazioni come in figura. Il Software Debugger si interfaccia così al MicroBlaze tramite il tool XMD.



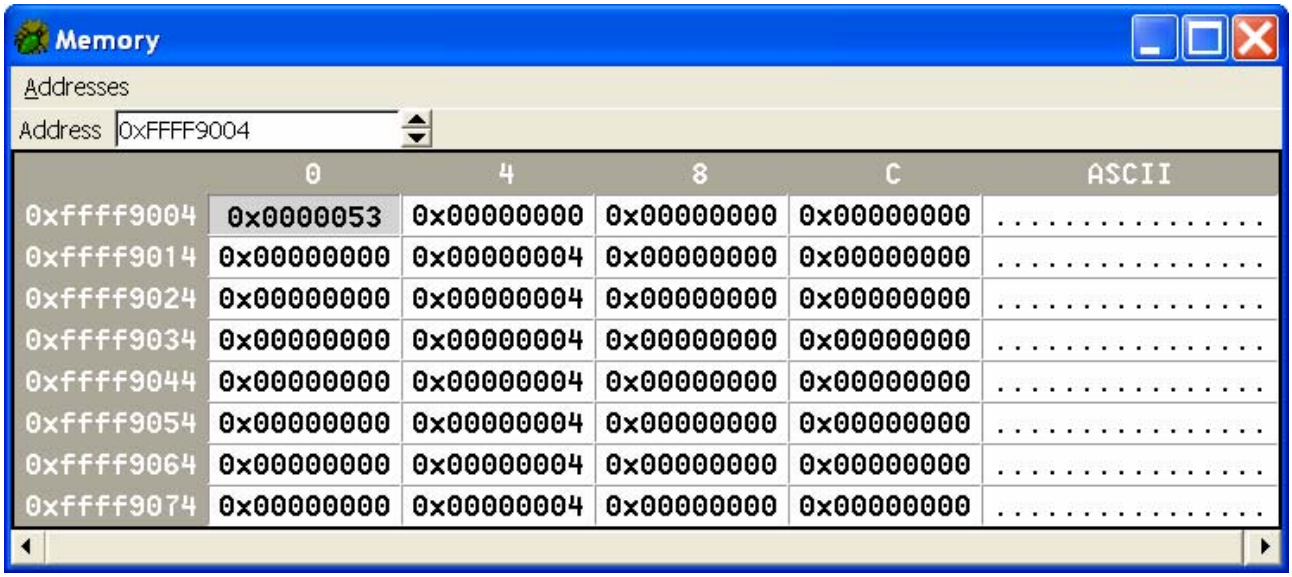
Con *Run -> Run* possiamo eseguire il programma; si può vedere cosa succede eseguendo una sola istruzione per volta, un intero ciclo e così via.
Il menù *View* ci permette di indagare sullo stato del MicroBlaze, analizzare il contenuto dei registri, della memoria....
Clicchiamo *View -> Memory* e scegliamo di visualizzare i valori relativi all'indirizzo 0x00900000
Facendo scorrere il programma vediamo che a questa locazione viene scritto il valore 0xabcd1234 come da noi indicato nel codice C.





Abbiamo varie opzioni per visualizzare il codice che viene eseguito passo dopo passo, in questo caso possiamo vedere per ogni istruzione assembler l'istruzione C associata e i relativi indirizzi di memoria in cui é memorizzato il codice.

Scrivendo negli opportuni indirizzi di memoria possiamo testare le periferiche (ad esempio accendere o spegnere un led, che però qui non é presente) o spedire dati attraverso la UART. Proviamo per esempio a scrivere 53 all'indirizzo 0xFFFF9004 (il byte 4 é destinato alla trasmissione), sul terminale di Windows apparirà una S, corrispondente al carattere ASCII 0x53.



5. ELABORAZIONE VIDEO IN TEMPO REALE

L'ultimo progetto, molto più complesso e articolato dei precedenti, consiste nella realizzazione di un'infrastruttura per editing video in tempo reale su MultiMedia Board; gli effetti che verranno applicati al video sono limitati ma può essere un buon punto di partenza per elaborazioni più complesse. Il progetto completo é contenuto all'interno del file *videoelab.zip*.
A differenza dei progetti precedenti, vedremo solo la struttura e il funzionamento del sistema in quanto oramai l'utilizzo dell'EDK dovrebbe essere stato acquisito; inoltre una spiegazione passo dopo passo sarebbe in questo caso troppo lunga e noiosa, data la complessità del sistema.

5.1 Introduzione

Il sistema finito verrà realizzato connettendo all'ingresso video composito della MultiMedia Board un dispositivo che generi un segnale secondo lo standard PAL (ad esempio una telecamera); tale segnale analogico verrà convertito in digitale dal decoder integrato sulla board.
L'output di tale dispositivo é costituito da un flusso di 10 bit in cui sono multiplexati i dati video utili e i codici di controllo per la sincronizzazione che indicano la fine di ogni frame: i vari canali del segnale dovranno essere quindi separati opportunamente; separate le componenti potremo elaborarle individualmente tramite una periferica hardware, le cui impostazioni saranno fornite dal MicroBlaze, interfacciato agli otto tasti di colore giallo presenti sulla board.
A questo punto le componenti, eventualmente modificate, saranno nuovamente multiplexate e verranno reinserite nel flusso di bit le informazioni per la sincronizzazione.
Il segnale complessivo verrà inviato quindi all'encoder video che lo convertirà da digitale in analogico inviandolo all'uscita video composito a cui avremo connesso un televisore.
Il decoder e l'encoder dovranno essere configurati secondo il protocollo I^2C (Inter-Integrated Circuit) in maniera opportuna da poter trattare un segnale video di tipo PAL.

Analizzeremo separatamente le 5 parti in cui può essere scomposto il sistema:

- il nucleo, composto dal MicroBlaze e le periferiche più generiche;
- l'utilizzo e la configurazione dei tasti;
- la programmazione dei dispositivi secondo l'I^2C;
- il trattamento dei segnali video;
- l'interfacciamento con il software.

5.2 IL NUCLEO DEL SISTEMA

Il nucleo del sistema é molto simile a quello del progetto precedente; esso é composto da:

- un MicroBlaze che comunica con le memorie e le periferiche tramite due local memory bus e un OPB bus;
- 32 kilobytes di RAM interna all'FPGA;
- 512 kilobytes di ZBT RAM esterna, essa non viene utilizzata ma é stata integrata qualora, volendo considerare questo progetto come punto di partenza per un sistema più complesso, si necessiti di maggiore memoria per il programma o i dati;
- un distributore e allineatore di clock, più semplice di quello utilizzato precedentemente ma con alcune funzioni aggiuntive;
- una JTAG per la programmazione e il debugging;
- una UART RS232 per visualizzare su terminale le impostazioni del sistema.

Alcune periferiche e l'impostazione generale del sistema sono state prese dall'esempio *MicroBlaze_Media_Brd_IIC_vidtest_edk3_2_sp1* presente sul sito della Xilinx all'indirizzo http://www.xilinx.com/products/support/multimedia/examples/; tuttavia, essendo stato concepito

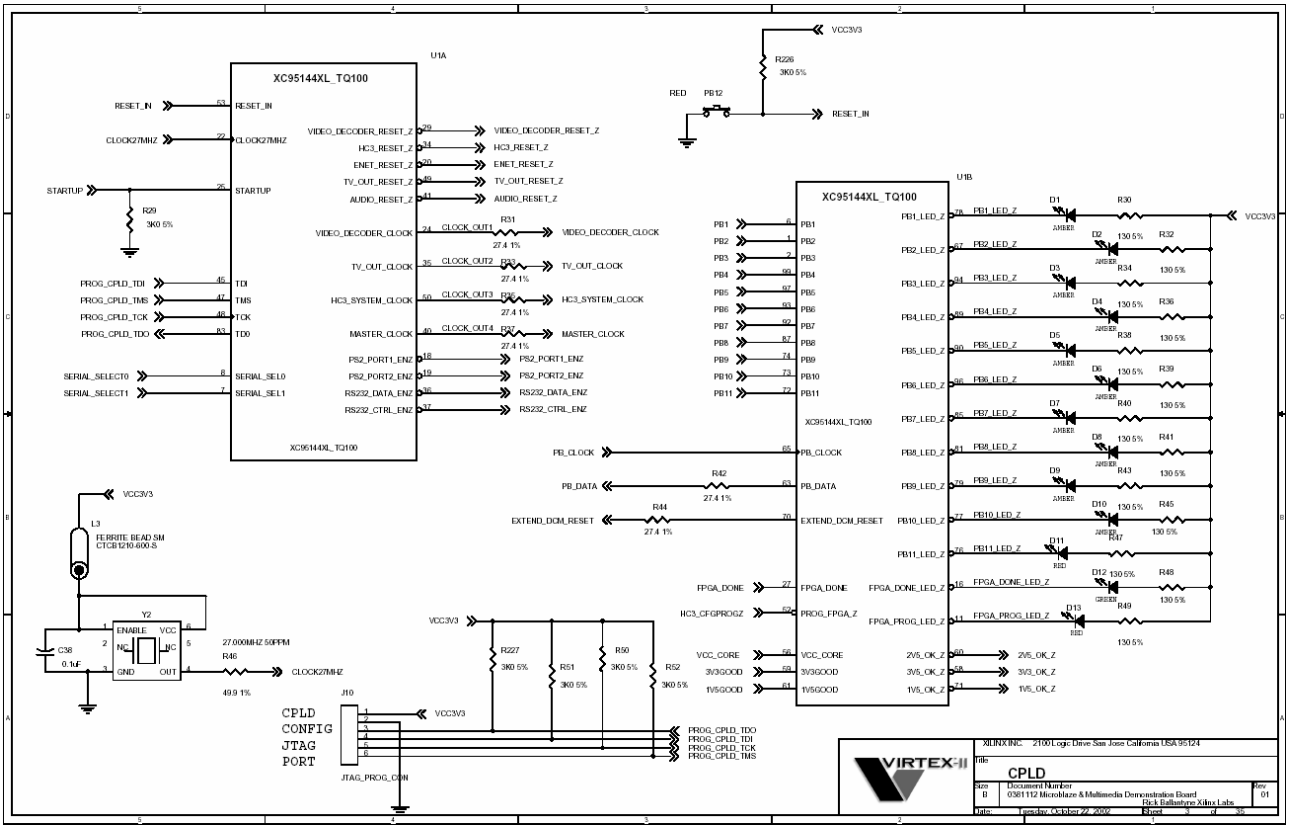
per la versione 3.2 dell'EDK, esso ha necessitato di un po' di modifiche per poter funzionare correttamente sulla versione 6.1i.

Il core *MicroBlaze_Brd_ZBT_ClkGen*, preso dall' esempio citato, é costituito da due soli DCM (uno per il sistema in generale e uno per la memoria) e non da tre come quello fornito dalla Xilinx (il *clockgen.v* che abbiamo visto in precedenza). Esso inoltre integra alcune utili funzioni aggiuntive, come il segnale di *Startup* alla CPLD, che vedremo in maggior dettaglio nel prossimo paragrafo.

Il sistema presenta un difetto a causa del fatto che il *MicroBlaze_Brd_ZBT_ClkGen* contiene un errore di programmazione: la porta a cui dovrebbe essere inviato il segnale di clock riallineato, pronto per essere inviato alla RAM esterna, non é collegata in realtà a nessun segnale interno. A causa di ciò non é possibile far girare correttamente il programma dalla ZBT RAM, anche se in teoria il sistema sarebbe predisposto per farlo. Per ovviare a questo problema sarebbe necessario studiare a fondo il codice VHDL del componente e correggerlo, magari confrontandolo con il core *clockgen.v* (che però é in verilog) [13] per individuare l'errore e collegare la porta al segnale opportuno.

5.3 I PUSHBUTTONS E LA CPLD

Sulla MultiMedia Board é integrata anche una CPLD [16]. Come si può vedere nei suoi schematici [17], sotto riportati, essa provvede a un gran numero di compiti diversi, quali la distribuzione dei clock, del reset e la gestione dei 10 pushbuttons e i led a loro associati.
I tasti vengono gestiti dalla CPLD anziché dall'FPGA perché altrimenti stato necessario utilizzare 10 pin per interfacciarli tutti; con questo sistema essi possono essere configurati a piacere, premendo il tasto *ENTER* la loro configurazione viene rilevata dalla CPLD che trasmette a sua volta il loro stato in maniera seriale all'FPGA utilizzando quindi un solo pin (in realtà sono tre perché si utilizza anche uno per il clock e uno per il reset).



L'utilizzo dei tasti può essere impostato in due modi differenti, a seconda di come viene programmata la CPLD. La Xilinx fornisce due configurazioni di base da trasferire tramite JTAG sulla CPLD: i files *normal.jed* e *video.jed* [18]. Noi utilizzeremo il primo, in quanto esso permette di impostare una combinazione qualsiasi dei 10 tasti e poi selezionarla tramite il tasto *ENTER*. Se avessimo deciso di optare per l'altra configurazione sarebbe stato possibile utilizzare solo due tasti contemporaneamente, cioé uno dei due gialli (che sul manuale suggeriscono di utilizzare per la scelta della sorgente video) e uno degli otto blu (che dovrebbero essere dedicati alla scelta dell'elaborazione da applicare al video).
Bisogna quindi ricordarsi di programmare la CPLD prima di programmare l'FPGA altrimenti potrebbe darsi che essa non si trovi nella modalità corretta.
A seconda della configurazione della CPLD bisogna importare un core differente fornito dalla Xilinx: il file *PB_SCAN_DATA_IN.v* (relativo al *normal.jed*) oppure *VIDEO_PB_SCAN_DATA_IN.v* (relativo al *video.jed*); bisogna apportare della lievi modifiche a questi file altrimenti danno degli errori in compilazione, pertanto suggerisco di usare il file incluso in questo progetto, che é già stato corretto; tuttavia se si preferisce partire con i file originali si riesce ad individuare velocemente il problema con l'*Answer Database* del sito della Xilinx.

Vedremo successivamente come il *pb_scan_data_in* é connesso con il resto del sistema per la sua configurazione.

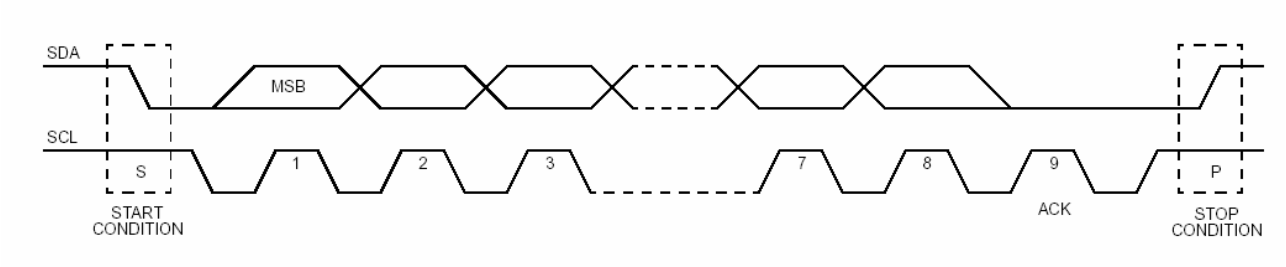
Per il suo corretto funzionamento la CPLD necessita di ricevere un segnale di *STARTUP* dal resto del sistema. Nel nostro caso questo segnale viene fornito all'avvio del sistema dal *MicroBlaze_Brd_ZBT_ClkGen* all'avvio.

5.4 LA CONFIGURAZIONE DELL'ENCODER E DEL DECODER

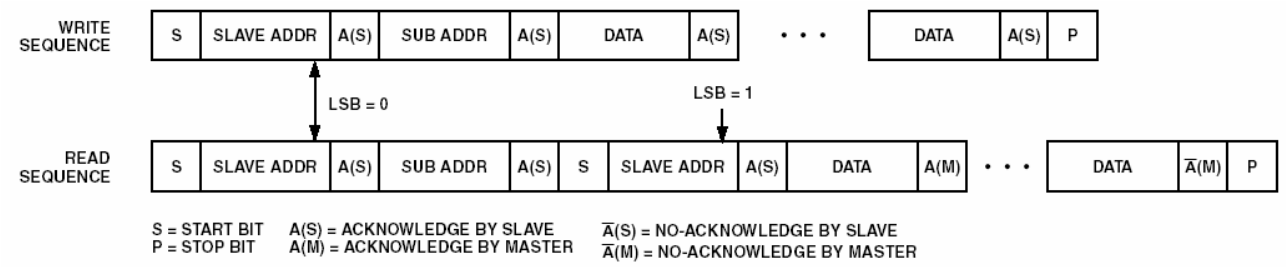
I convertitori A/D e D/A integrati sulla MultiMedia Board [19] possono essere configurati in maniera molto ampia in quanto sono predisposti per supportare un gran numero di standard diversi e permettono di intervenire su molti parametri per elaborare direttamente il segnale oltre a convertirlo. Per i nostri scopi sarà sufficiente configurare i registri dei dispositivi in modo da convertire un segnale video composito di tipo PAL in un segnale digitale in formato *YCrCb 4:2:2* e viceversa; la Xilinx fornisce dei valori consigliati da memorizzare nei registri [20], che devono essere programmati secondo lo standard I^2C. Riportiamo qui una breve spiegazione del protocollo I^2C; qualora fossero necessarie informazioni più dettagliate ci si riferisca alla pagina web http://www.semiconductors.philips.com/buses/i2c/.

Il bus I^2C é composto da due fili, serial data (SDA) e serial clock (SCL), che trasportano informazioni fra i dispositivi connessi al bus. Le linee, entrambe bidirezionali, sono connesse a un generatore di tensione positiva tramite una resistenza di pull-up. In questa maniera quando il bus é libero entrambe le linee sono alte.
Ogni dispositivo connesso al bus ha un unico indirizzo e può operare sia come trasmittente o ricevente; inoltre ognuno può essere configurato come Master o Slave.
Il Master é il dispositivo che inizia il trasferimento di dati sul bus e genera i segnali di clock per permettere questo trasferimento. Tutti gli altri dispositivi collegati sono considerati Slave.
La trasmissione di dati sul bus I^2C ha inizio quando si é in condizione di START e ha termine quando si é in condizione di STOP.
Lo stato alto o basso della linea SDA può cambiare solo quando SCL é basso.
La condizione di START é un caso unico definito come una transizione da alto a basso su SDA mentre SCL é alto.
Allo stesso modo la condizione di STOP si ha solo nel caso in cui SDA passi da basso ad alto mentre SCL é alto.
Queste condizioni esclusive assicurano che START e STOP non possano mai essere confusi con i dati validi.

Ogni pacchetto di dati consiste di otto bit seguiti da un bit di acknowledge, in modo che la trasmissione completa consista di nove giri di clock. I dati sono trasmessi cominciando dal bit più significativo (MSB).
Il dispositivo che trasmette rilascia la SDA durante il bit di acknowledge e il ricevente deve abbassare SDA per indicare l'avvenuta ricezione dei dati.



La comunicazione sul bus fra il Master e uno Slave é composta da quattro parti: START, indirizzo dello Slave, trasmissione dati, STOP.
Il protocollo I^2C definisce un formato per la trasmissione dei dati sia per indirizzi di 7 che di 10 bit.
Un indirizzo di 7 bit é inizializzato nella seguente maniera: dopo una condizione di START, viene inviato l'indirizzo dello Slave. Questo indirizzo é composto da 7 bit seguiti da un ottavo bit che indica se il dispositivo deve essere inizializzato in lettura ("1") o in scrittura ("0"). Solo lo Slave con l'indirizzo corrispondente a quello trasmesso dal Master risponde trasmettendo a sua volta un acknowledge (abbassando la linea SDA al nono giro di clock).
Una volta che viene realizzato con successo l'indirizzamento dello Slave il trasferimento di dati può procedere byte per byte.



Per quanto riguarda l'implementazione della programmazione dell'I^2C mi sono basato sul sopracitato esempio tratto dal sito Xilinx. I registri vengono programmati in maniera quasi totalmente software. Il MicroBlaze viene interfacciato ai dispositivi tramite una sorta di GPIO personalizzato programmato dall'autore, tale che permette di trattare più segnali indipendenti (a differenza di quello originale che lavora solo con vettori); esso é connesso a un'interfaccia molto semplice che si collega direttamente ai due segnali SCL e SDA del bus I^2C.
I rispettivi cores sono *gpio_mod.vhd* e *gpio2iic.vhd*.
Tutta la gestione pratica dei segnali di START, STOP, trasmissione dell'indirizzo e trasmissione dati é realizzata via software, la parte hardware si limita a fare passare i segnali generati dal codice C e inviarli in output ai pin specificati.

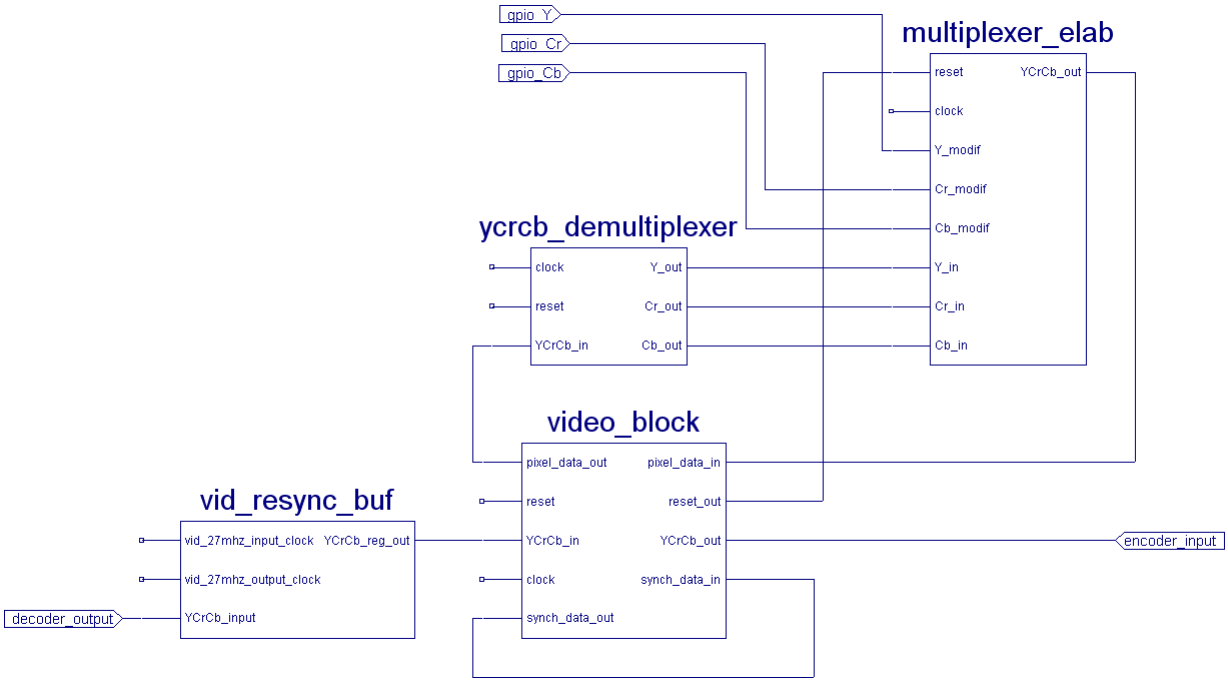
Si presti attenzione al fatto che gli switch *S VIDEO/COMP VIDEO* e *NTSC/PAL* presenti sulla board sono connessi direttamente al decoder e all'encoder. Per un corretto utilizzo nel nostro sistema dovremo settarli a *comp video* e *pal* rispettivamente.

5.5 L'ELABORAZIONE VIDEO

L'elaborazione del segnale video all'interno del sistema viene svolta dai seguenti componenti:

- *vid_resync_buf*: uniforma il clock del flusso di dati video alla frequenza del resto del sistema;
- *video_block*: separa i dati video utili dalle informazioni di controllo relative alla sincronizzazione e poi le ricompone in un unico flusso di dati dopo un'eventuale elaborazione;
- *YCrCb_demultiplexer*: separa le tre componenti (una luminanza e due crominanze) dei dati video utili;
- *multiplexer_elab*: ricompone in un unico flusso di bit le tre componenti dell'immagine aggiungendo ad ognuna un eventuale valore configurato tramite i tasti, interfacciati con il MicroBlaze.

Ho riportato qui sotto uno schema indicativo della parte di sistema che si occupa dell'elaborazione video. Gli ingressi del clock e del reset non sono stati collegati per non appesantire la figura (eccetto il reset di *multiplexer_elab*, poiché é interfacciato direttamente con il *video_block* e non con il reset del resto del sistema).



5.5.1 vid_resync_buf

```
BEGIN vid_resync_buf
 PARAMETER INSTANCE = vid_resync_buf_in
 PARAMETER HW_VER = 1.00.a
 PORT reset = fpga_reset
 PORT YCrCb_reg_out = YCrCb_in
 PORT vid_27mhz_input_clock = line_locked_clock
 PORT vid_27mhz_output_clock = clk_27mhz
 PORT YCrCb_input = YCRCB
END
```

YCrCb_reg_input: dati in ingresso nel buffer provenienti dal decoder PAL
Y_CrCb_reg_out: dati in uscita dal buffer
vid_27mhz_input_clock: clock dei dati in ingresso
vid_27mhz_output_clock: clock dei dati in uscita

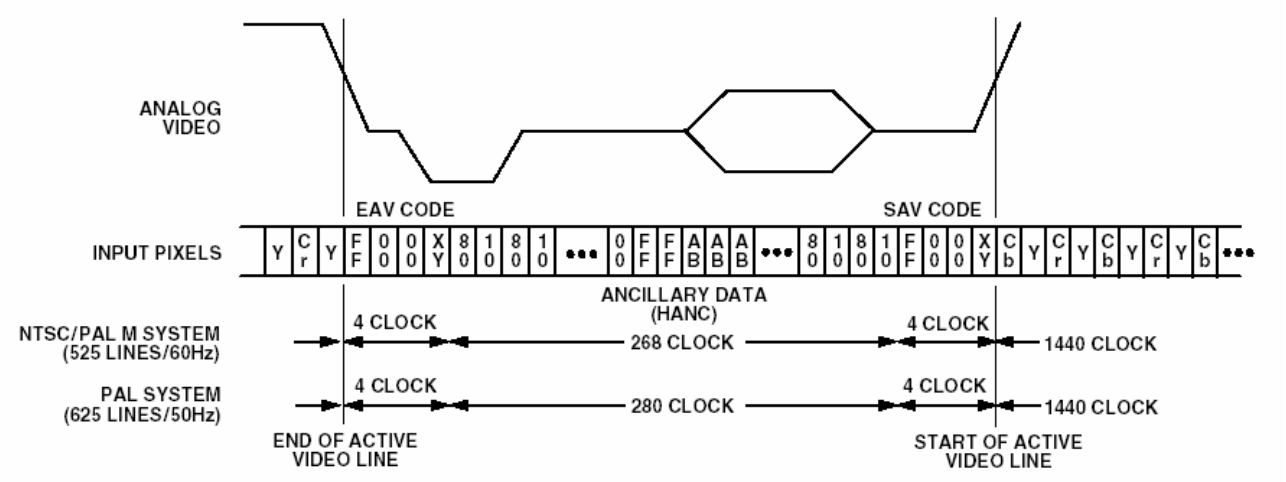
Il decoder PAL fa uscire i 10 bit del segnale video con una frequenza che é diversa da quella del resto del sistema.
Il *vid_resync_buf*, pertanto, prende in ingresso i 10 bit che giungono con la frequenza a cui il decoder li trasmette e li memorizza in un buffer; i dati vengono poi letti dal buffer a una velocità che é allineata a quella del resto del sistema.
Le frequenze dei due segnali di clock a cui i bit vengono prima ricevuti e poi inviati sono leggermente diverse, entrambe a circa 27 mHz; per questo motivo noteremo sullo schermo una perdita di sincronia momentanea con una frequenza di circa 1 Hz. Questo avviene quando vengono scritti nel buffer i dati relativi a una zona di schermo che non é ancora stata letta.

5.5.2 video_block

```
BEGIN video_block
 PARAMETER INSTANCE = video_block_inst
 PORT clock = clk_27mhz
 PORT reset = fpga_reset
 PORT synch_data_in = synch_data
 PORT synch_data_out = synch_data
 PORT pixel_data_out = pixel_data_out
 PORT pixel_data_in = pixel_data_in
 PORT YCrCb_in = YCrCb_in
 PORT YCrCb_out = YCRCB_P
 PORT reset_out = reset_out
END
```

YCrCb_in: dati provenienti dal vid_resync_buf
synch_data_out/pixel_data_out: uscite separate per codici di controllo e dati video ottenute dal demultiplexing del segnale proveniente da YCrCb_in
synch_data_in/pixel_data_in: ingressi separati per codici di controllo e dati video
YCrCb_out: porta da cui escono multiplexati i dati provenienti da synch_data_in e pixel_data_in, diretti all'encoder PAL
reset_out: porta collegata a YCrCb_demultiplexer, per indicare quando si tratta di dati video (segnale basso) o codici di controllo (segnale alto)

Il componente *video_block* accetta in ingresso il flusso di dati a 10 bit proveniente dal *vid_resync_buf* e ne separa le parti relative alla sincronizzazione (cioé i codici di controllo che indicano la fine di un frame) e i dati video utili.
Nella figura sottostante é indicato il formato dei codici di controllo:



Al loro interno possono essere indicate varie configurazioni relative alle impostazioni desiderate, ma andare a fondo in ciò esula dallo scopo del progetto.
Come vediamo in figura, nello standard PAL a 50 Hz i codici di controllo durano per 4 + 280 + 4 = 288 giri di clock.
Questi codici sono del tipo indicato e inizieranno sempre e comunque con una situazione in cui tutti i 10 bit sono a 1 (in figura é indicato “FF” perché é relativa a un formato a 8 bit).
Le configurazioni in cui tutti i bit sono a “0” o tutti i bit sono a “1” sono riservate, non indicano dati validi ma sicuramente sono informazioni di controllo.
Tutto ciò che fa il video_elab é rilevare l’arrivo di 10 bit tutti a 1 dalla porta *YCrCb_in*: in questo caso invia tali bit e i successivi 287 gruppi di 10 bit all’uscita *synch_data_out* e all’uscita diretta all’encoder PAL (*YCrCb_out*) associa il segnale proveniente dal *synch_data_in*.
Per tutto il resto del tempo invia i bit provenienti da *YCrCb_in* all’uscita *pixel_data_out* e associa a *YCrCb_out* il segnale proveniente dall’ingresso *pixel_data_in*.

In questo modo possiamo intervenire solamente sul flusso di dati video utili (che escono quindi da *pixel_data_out* e rientrano da *pixel_data_in*) senza modificare i codici di controllo; ovviamente applicando anche ad essi le elaborazioni verrebbero perse le informazioni relative alla sincronia.
L’uscita dei codici di controllo *synch_data_out* é stata collegata direttamente al rispettivo ingresso *synch_data_in* poiché non ci interessa intervenire su tali segnali, ma tuttavia é chiaramente possibile operare anche su di essi, qualora se ne presentasse la necessità.
Fra *pixel_data_out* e *pixel_data_in* sono invece inseriti gli altri componenti dedicati all’elaborazione vera e propria del segnale video, che vedremo più in dettaglio nelle sezioni successive. In questa posizione bisogna stare attenti a non inserire logica troppo complessa, altrimenti é possibile che i dati video utili elaborati arrivino in ritardo rispetto ai codici di controllo, causando una perdita di sincronia. Ad esempio, se i dati video arrivassero con il ritardo di un periodo clock, avremmo che luminanza e crominanze verrebbero scambiate.

5.5.3 YCrCb_demultiplexer

```
BEGIN YCrCb_demultiplexer
 PARAMETER INSTANCE = demultiplexer_inst
 PORT clock = clk_27mhz
 PORT reset = reset_out
 PORT Y_out = Y_dem_out
 PORT Cr_out = Cr_dem_out
 PORT Cb_out = Cb_dem_out
 PORT YCrCb_in = pixel_data_out
END
```

reset: il componente viene mantenuto in stato di reset dal video_block finché non comincia la trasmissione di dati video
YCrCb_in: dati video in ingresso
Y_out, Cr_out, Cb_out: canali dei dati video in uscita

Il *YCrCb_demultiplexer* elabora il flusso di dati video utili ottenuto dal *video_block* e invia su 3 canali distinti la luminanza e le due crominanze. I dati video sono codificati in formato *YCrCb 4:2:2* a 10 bit.
Secondo questo standard ogni pixel ha un valore per la luminanza e due valori per le crominanze, questi ultimi però sono in comune con il pixel adiacente. In questo caso i canali multiplexati, così come provengono dal decoder, sono del tipo:

Cb dei pixel 1 e 2
Y del pixel 1
Cr dei pixel 1 e 2
Y del pixel 2
Cb dei pixel 3 e 4
Y del pixel 3
Cr dei pixel 3 e 4
Y del pixel 4
e così via...

Il *YCrCb_demultiplexer* viene mantenuto in stato di reset dal *video_block* finché non comincia la trasmissione di dati video utili; senza questo accorgimento succede spesso che i tre segnali Y, Cr e Cb demultiplexati non vengano posti sulla propria uscita ma su quella di uno degli altri canali. Mantenendo il componente in stato di reset finché non vengono ricevuti dati fa sì che al primo turno abbiamo sempre la Cb, al secondo la Y, al terzo la Cr, al quarto nuovamente la Y, e via discorrendo.

5.5.4 multiplexer_elab

```
BEGIN multiplexer_elab
 PARAMETER INSTANCE = multiplexer_elab_inst
 PORT clock = clk_27mhz
 PORT reset = fpga_reset
 PORT Y_in = Y_dem_out
 PORT Cr_in = Cr_dem_out
 PORT Cb_in = Cb_dem_out
 PORT Y_modif = Y_modif
 PORT Cr_modif = Cr_modif
 PORT Cb_modif = Cb_modif
 PORT YCrCb_out = pixel_data_in
END
```

Y_in, Cr_in, Cb_in: dati provenienti dal multiplexer
Y_modif, Cr_modif, Cb_modif: quantità che si vuole aggiungere a ogni componente
YCrCb_out: uscita dati video elaborati e multiplexati da inviare all'ingresso del video_block

Il *multiplexer_elab* multiplexa i canali provenienti dal *YCrCb_demultiplexer* aggiungendo a ogni componente una costante indicata tramite il MicroBlaze e i tasti della board. Le tre costanti da aggiungere ai tre canali sono indipendenti.
Qualora il valore ottenuto dalla somma di un valore con la costante addizionale superi il valore 1023 (massimo numero rappresentabile con 10 bit) il bit più significativo viene trascurato (in pratica si somma il valore in esubero ripartendo da 0).

5.6 L'INTERFACCIA SOFTWARE

5.6.1 Le periferiche di input/output

Come già visto i 10 tasti presenti sulla MultiMedia Board sono interfacciati con il resto del sistema tramite il *pb_scan_data_in*; i segnali corrispondenti agli otto bottoni blu (da *pb3* a *pb10*) vengono quindi collegati a *scal2vect8bit*. Questo semplice componente converte gli 8 segnali indipendenti in un unico vettore a 8 componenti, in modo da poter essere interfacciato con un GPIO (*gpio_button_in*) tramite cui leggere dal MicroBlaze lo stato degli 8 tasti.
Si é deciso di lasciare scollegati i primi 2 tasti qualora potessero servire per interfacciarsi a qualche altra periferica indipendente dal MicroBlaze.

La configurazione dei tasti viene letta dal MicroBlaze e in base al programma caricato verrrano inviati al *multiplexer_elab* il valori opportuni da aggiungere alle componenti Y, Cr e Cb dei dati. I rispettivi dati saranno inviati dal MicroBlaze alla periferica tramite tre diversi GPIO (*gpio_Y*, *gpio_Cr*, *gpio_Cb*).

Bisogna fare attenzione ad utilizzare la versione del GPIO più consona ad ogni situazione. Sono state riscontrate parecchie difficoltà nel collegare segnali interni al sistema tramite GPIO. Infatti sembra che la porta *GPIO_IO* possa essere utilizzata solo per collegare pin esterni dell'FPGA, ed é peraltro l'unica porta disponibile sulla versione 1.00 del GPIO. Si é dovuto ricorrere a porte unidirezionali (presenti nella versione 3.00) quali le *GPIO_in* e *GPIO_d_out*, a seconda che il GPIO sia utilizzato in lettura o in scrittura dal MicroBlaze.

5.6.2 Il codice C

gpio.c e gpio.h

Il codice dei files *gpio.c* e *gpio.h* si occupa della configurazione dell'encoder e del decoder secondo il protocollo I^2C. Sono stati presi entrambi dall'esempio presente sul sito della Xilinx e sono piuttosto complessi perciò non ci soffermeremo su di essi.
Le uniche parti che sono state modificate rispetto all'originale sono alcuni valori che vanno scritti nei registri dell'encoder e del decoder. Le impostazioni di entrambi i dispositivi sono ora settate per funzionare con ingresso e uscita di tipo PAL video composito. Modificando i valori dei registri i dispositivi possono essere configurati in maniera molto ampia, ma si presti attenzione che se si volesse lavorare con video NTSC sarebbe necessario cambiare anche altri parametri del sistema, modificando ad esempio il numero di cicli di clock relativi ai codici di controllo nel componente *video_block*.

videoelab.c

Caricando nella memoria del sistema l'eseguibile creato da *videoelab.c* l'hardware viene configurato nel seguente modo:

- con la funzione *program_video()* descritta nei files *gpio.h* e *gpio.c* vengono configurati i registri del decoder e dell'encoder.

- tramite i tasti blu viene impostato il valore da aggiungere a ogni canale, in particolare: nello stato iniziale a tutti le componenti viene aggiunto un valore nullo; ad ogni passo successivo, in base alla configurazione dei pulsanti al momento della pressione del tasto *ENTER* viene modificata ogni singola componente nella seguente maniera:

pb3: il valore da aggiungere alla luminosità Y viene diminuito di 10;
pb4: il valore da aggiungere alla luminosità Y viene incrementato di 10;
pb5: il valore da aggiungere alla crominanza Cr viene diminuito di 10;
pb6: il valore da aggiungere alla crominanza Cr viene incrementato di 10;
pb7: il valore da aggiungere alla crominanza Cb viene diminuito di 10;
pb8: il valore da aggiungere alla crominanza Cb viene incrementato di 10;

- i valori devono essere compresi fra 0 e 1023, le modifiche non possono superare questi limiti;
- possono essere premuti anche più tasti allo stesso tempo, il loro effetto verrà applicato contemporaneamente (ciò significa ad esempio che se premiamo contemporaneamente pb3 e pb4, non verrà apportata alcuna modifica alla luminosità);
- il programma rileva solo il cambiamento di stato dei pulsanti: ciò significa che fra due pressioni consecutive del tasto ENTER la configurazione dei pulsanti deve essere diversa altrimenti la seconda non sortirà alcun effetto.

- I valori che vengono aggiunti volta per volta possono essere visualizzati tramite un terminale eventualmente connesso alla porta RS232, ad esempio, con l'*HyperTerminal* di Windows.

- I valori settati tramite il programma vengono inviati ai tre GPIO destinati a configurare il *multiplexer_elab* con i tre valori da aggiungere in ogni momento.

Per la comunicazione del software con i GPIO sono state utilizzate delle funzioni prese dalle librerie Xilinx di livello più alto, includendo il file *xgpio.h*.

Segue una breve descrizione delle funzioni utilizzate [21]:

*Xstatus XGpio_Initialize(XGpio * InstancePtr, Xuint16 DeviceId)* inizializza l'istanza di tipo XGpio indicate dal DeviceId.
InstancePtr é un puntatore ad un'istanza di tipo XGpio. La memoria a cui si riferisce il puntatore dev'essere preallocata dal programma che richiama la funzione.
DeviceId é un identificatore che definisce univocamente il dispositivo controllato dal componente di tipo XGpio. Le specifiche con i vari identificatori e i relativi indirizzi de memoria per ogni componente sono indicate dal file xparameters.h.

*void XGpio_SetDataDirection(XGpio * InstancePtr, Xuint32 DirectionMask)* é utilizzato per settare la direzione dei dati in transito sul GPIO.
DirectionMask é una maschera che specifica per ogni bit se esso é utilizzato in scrittura ("1") o in lettura ("0").

Le funzioni void XGpio_DiscreteWrite(XGpio * InstancePtr, Xuint32 Data)
e Xuint32 XGpio_DiscreteRead(XGpio * InstancePtr) sono utilizzate per accedere rispettivamente in scrittura e in lettura ai registri del GPIO.

Poiché lo stato dei pulsanti viene visto dal MicroBlaze solo come un numero, il programma deve estrarre da tale valore lo stato di ogni singolo tasto e memorizzarlo in una variabile. In questa maniera é possibile per il software rilevare in ogni momento quali tasti sono settati e quali no e configurare così il sistema secondo la procedura vista sopra.

demo.c

Il codice del programma *demo.c* é molto simile a quello di *videoelab.c*; l'unica differenza é nella maniera in cui viene configurato il sistema.
Se pb3 é attivo ad ogni ciclo del programma verrà incrementato di 8 il valore da aggiungere alla luminosità; nello stato iniziale esso é 0, quando arriva a 1024 si ricomincia da 0.
Se pb4 é attivo viene incrementato in maniera analoga il valore da aggiungere alla crominanza Cr.
Se pb5 é attivo viene incrementato in maniera analoga il valore da aggiungere alla crominanza Cb.

5.7 Boot da CompactFlash

Una caratteristica molto interessante offerta dalla MultiMedia Board é la possibilità di caricare la configurazione del sistema direttamente da CompactFlash [21]. E' possibile memorizzare fino a 8 file diversi; la scelta della configurazione da caricare su FPGA viene fatta tramite i tre switch *CF CONFIG NUMBER* presenti sulla board.
Il tool *iMPACT*, disponibile fra gli Accessories dell'ISE, permette di preparare la configurazione della CompactFlash in maniera molto veloce.
Una volta avviato il programma basta scegliere *Prepare Configuration Files*, *System ACE File*, *System ACE CF* e modalità *Novice.* Da qui in poi tutta la procedura é molto intuitiva. Dovremo semplicemente indicare i file di design *.bit* (nella sottodirectory */implementation* dei nostri progetti) da associare a ciascuno degli 8 slot di memoria e copiare tutti i file e le directory create dal tool nella root della Compact Flash tramite uno degli appositi card writer disponibili in commercio.
Ho allegato fra i miei files l'archivio *cfboot.zip*: copiando sulla compact flash il suo contenuto é possibile caricare i files di configurazione ottenuti tramite il codice *demo.c* (configurazione degli switch "111") e *videoelab.c* (configurazione "110").
All'accensione della board il SystemACE legge lo stato degli switch e scarica il file di configurazione nello slot di memoria associato. Qualora si volesse utilizzare un'altro file di configurazione non occorre spegnere la board ma é sufficiente cambiare lo stato degli switch e premere il tasto *RELOAD* per aggiornare il sistema.

6.0 RIFERIMENTI

Il materiale disponibile in rete per la MultiMedia Board é piuttosto scarso; si trovano on-line alcuni tutorial per l'EDK, ma generalmente sono creati per l'utilizzo con altre board.
Sebbene tali tutorial possano essere molto utili, essi obbligano il novizio a fare molti esperimenti prima di riuscire a far funzionare gli esempi su MultiMedia Board, specialmente quando si tratta di utilizzare periferiche non troppo generiche come la ZBT RAM.

Alcuni tutorial si possono comunque trovare su:
http://support.xilinx.com/support/techsup/tutorials/index.htm
http://www.eece.unm.edu/xup

Gli unici esempi sviluppati specificatamente per MultiMedia board che sono riuscito a trovare, sono reperibili all'indirizzo:
http://www.xilinx.com/products/boards/multimedia/examples.htm
Tali files sono testati però su una vecchia versione dell'EDK e danno non pochi problemi. I cambiamenti fra una versione e l'altra dell'EDK sono infatti sostanziali e generalmente ogni progetto va riadattato di versione in versione.

Tutti i riferimenti tecnici sulla MultiMedia Board, l'EDK e i tools dell'ISE sono stati tratti dagli Help e dalla documentazione fornita a corredo dei prodotti; essi sono comunque reperibili sul sito http://www.xilinx.com; tale sito si rivela utile anche grazie al suo forum e un ricco *Answer Database* in cui sono risolti i problemi tecnici più frequenti.

Per quanto riguarda la documentazione relativa ai singoli cores, ricordo che essa può essere visualizzata dalla schermata principale del Platform Studio, cliccando con il tasto destro del mouse sulla periferica desiderata e scegliendo *View PDF Doc*.

Seguono i riferimenti a cui si rimanda in vari punti del presente documento per approfondimento e consultazione tecnica.

[1] Per la documentazione relativa al formato MHS ci si può riferire al file *est_guide.pdf* (*Embedded System Tools Guide*, pag.197 e seguenti) richiamabile dall'*Help* dell'EDK.

[2] Per quanto riguarda la connessione dei pin dell'FPGA con la board si vedano le pagg.19-34 del file *UG020.pdf* presente nella directory *MicroBlaze & Multimedia Demo Board User Guide* del cd contenente la documentazione fornito dalla Xilinx con la board. Sempre nello stesso cd, nella directory *FPGA IO Constraint Files*, sono presenti i file relativi alle varie periferiche (raggruppate per tipo) in formato *UCF*, pronti per l'uso.
Per quanto riguarda la sintassi del formato UCF si veda il documento:
http://toolbox.xilinx.com/docsan/xilinx5/pdf/docs/cgd/cgd.pdf

[3] Per la documentazione relativa alle librerie Xilinx ci si può riferire al file *xilinx_drivers.pdf* (*Xilinx Device Drivers Documentation*) richiamabile dall'*Help* dell'EDK.

[4] Si veda *iMPACT* on-line help: *Command Line and Batch Mode*.

[5] Il file *systemace.bsd* è stato preso dal progetto
MicroBlaze_Media_Brd_IIC_vidtest_edk3_2_sp1.
Per informazioni sui file *BSDL* si veda l'*Answer Record # 15346* presente sul sito della Xilinx.

[6] Si veda *xilinx_drivers.pdf* (pag.1063).

[7] Si veda *xilinx_drivers.pdf* (pag. 1041 e seguenti).

[8] Si veda *xilinx_drivers.pdf* (pag. 1059 e seguenti).

[9] Si veda *xilinx_drivers.pdf* (pag. 1045).

[10] Si veda *xilinx_drivers.pdf* (pag. 528).

[11] Si veda *xilinx_drivers.pdf* (pag. 532).

[12] Si veda *UG020.pdf* (pagg.17-18).

[13] Si veda Documentation CD: *Design Examples\CLOCK_GENERATION\CLOCKGEN.v.*

[14] Si veda *est_guide.pdf* (pag.206 e seguenti).

[15] Si veda *est_guide.pdf* (pag.165 e seguenti).

[16] Si veda *UG020.pdf* (pagg.15-17).

[17] Si veda Documentation CD: *Board Schematics, 0381112r01.pdf*.

[18] Si veda Documentation CD: directory *CPLD Documentation Files*

[19] Si veda *UG020.pdf* (pagg.6-8).
Maggiori dettagli nel Documentation CD: directory *Component Data Sheets*, files *ADV7185_video_decoder.pdf* e *ADV7194_video_encoder.pdf*.

[20] Si veda Documentation CD: directory *I2C Register Values*

[21] Si veda *xilinx_drivers.pdf* (pag. 524 e seguenti).

[22] Si veda *UG020.pdf* (pagg.13-15)
Maggiori dettagli nel Documentation CD:
Component Data Sheets\XCCACE_SystemACE_Controller.pdf