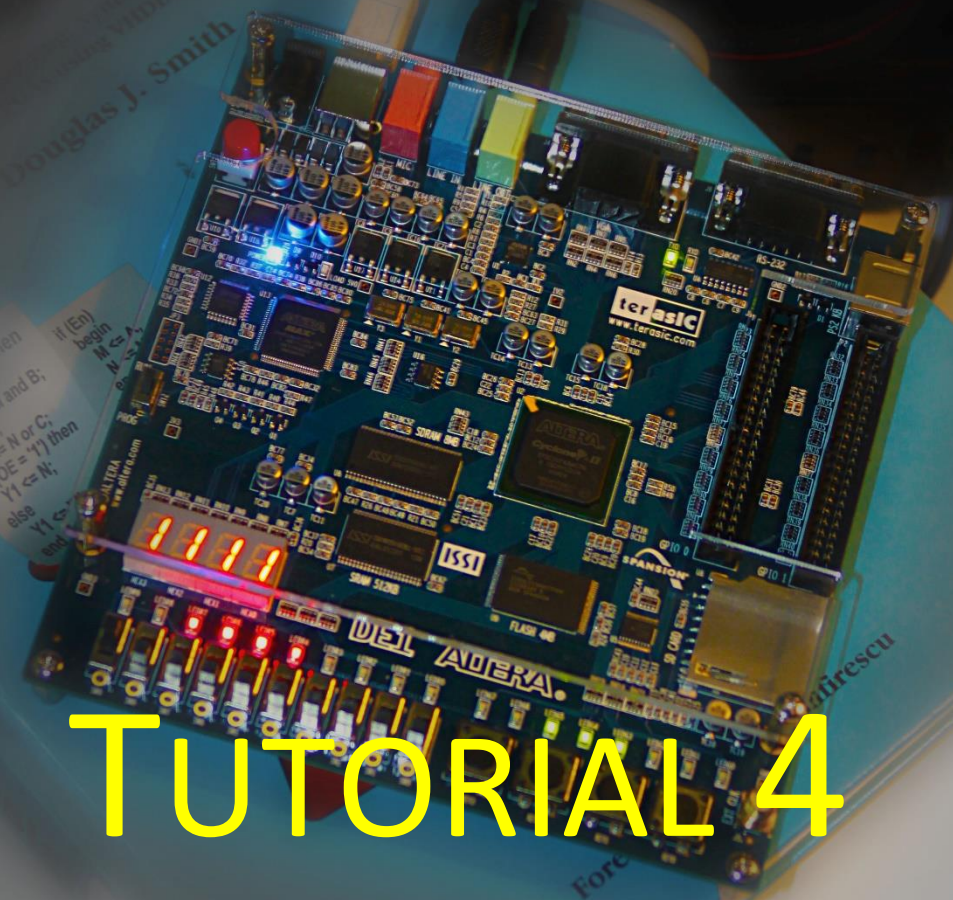
FPGA Tutorial on DE1 Board



Tutorial 4

Using Quartus II V13.0

Marsi 2014 - University of TRIESTE

Tutorial 4

Realizzazione di processore NIOS-2 e di alcune interfacce di I/O

Descrizione: L'FPGA montata sulla DE1 ha risorse sufficienti per poter realizzare al suo interno un microprocessore (soft processore) senza che il progettista progetti ex-novo un processore, esistono una serie di blocchi già sviluppati (simili alle MegaFunction) che consentono di realizzare in pochi passi un processore Nios-II corredato di tutte le periferiche che possono servire per un progetto specifico[1]. In questo tutorial verranno analizzati i rudimenti per realizzare, configurare e programmare opportunamente un sistema completo basato su detto processore.

Scopo: Realizzazione di una semplice architettura di un microprocessore, e di alcune interfacce dedicate di I/O e programmare il medesimo.

Apprendimenti previsti:

- Utilizzo del tool "Qsys" per la configurazione dell'architettura del processore
- Utilizzo del tool "Altera Monitor Program" per la programmazione ed il debugging del software
- Impiego di memorie esterne all'FPGA e problematiche di timing

Procedimento:

Si inizi un nuovo progetto per `Ciclone II - EP2C20F484C7N`

Definizione dell'architettura del Processore.

Si apra il tool Qsys [2]

`Tool > Qsys`

Di default appare già presente un blocco per la gestione di clock e reset. Tale blocco in realtà serve solamente a trasportare i segnali di clock e di reset dall'esterno verso il sistema di interfacciamento interno al sistema, o a seconda della configurazione a sincronizzare opportunamente il segnale di reset sul clock.

Si aggiunga a questo i blocchi che compongono l'architettura (si consiglia di guardare quali sono i segnali che afferiscono a ciascun blocco)

1. Una CPU:
   - `Embedded Processor > Nios II Processor` – **ADD**
   - Si configuri nella versione Nios II/e
   - `Finish` (altri parametri saranno configurati in un secondo tempo)
2. Una memoria interna all'FPGA:
   - `Memories … > On Chip > On Chip Memories (RAM or ROM)`
   - Si configuri come `RAM` da `16384` Bytes ed un "Data Witdh" da `32` bits
3. Un' Interfaccia UART - JTAG:
   - `Interface Protocol > Serial > JTAG UART`
   - Configurazione come da default

4. Un' Interfaccia Parallela di I/O come ingresso:
   - `Peripherals > Microcontroller Peripherals > PIO`
   - Si configuri come `Ingresso` a `10` bits

5. Un' Interfaccia Parallela di I/O come Uscita:
   - `Peripherals > Microcontroller Peripherals > PIO`
   - Si configuri come `Uscita` a `10` bits

6. Un' identificativo del sistema:
   - `Peripherals > Debug and Performance > System ID Peripheral`
   - Questa periferica non richiede alcuna configurazione ma eventualmente si potrebbe fissare un valore identificativo per il sistema stesso. Questo valore viene utilizzato da alcuni strumenti software di compilazione e debugging per riconoscere univocamente il sistema.

A questo punto si possono modificare i nomi delle varie periferiche, è consigliabile dare un nome mnemonico che evidenzi la loro futura funzione (ed esempio la PIO in ingresso si potrebbe nominare SWITCHES e quella in uscita LED). Per cambiare nome si evidenzi la periferica e si digiti `ctl-r`. Analogamente si può modificare il nome del clock. Sebbene in questa esercitazione non venga usata, per il futuro è consigliabile denominare la periferica identificativa del sistema "`sysid`".

Si realizzino tutti i collegamenti tra i bus ed i segnali. In particolare

- Il `cloclk` ed il `reset` in uscita dal blocco preposto alla gestione di questi segnali arrivino a tutti i blocchi
- Il segnale `JTAG_debug_module_reset` in uscita dalla CPU venga utilizzato anche esso come reset dei vari blocchi
- Il bus `data_master` in uscita dalla CPU raggiunga tutti i blocchi che ne fanno uso nella porta `s1`
- Il bus `instruction master` raggiunga `memorie` e modulo `jtag_uart`

Assegnare automaticamente gli indirizzi delle varie periferiche:

```
System > Assign Base Addresses
```

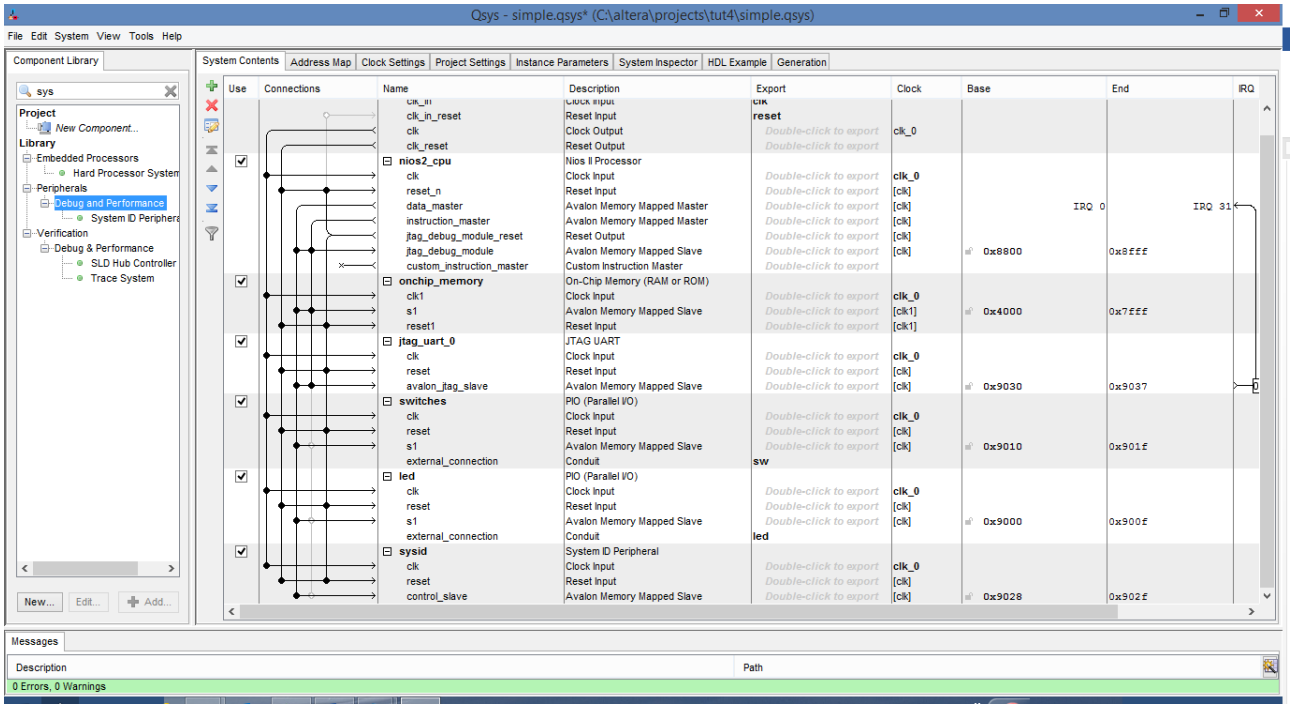
Ove richiesto si possono anche assegnare indirizzi specifici pur di prestare attenzione che non vi siano sovrapposizioni

Riaprire la CPU (doppio click) e modificare la configurazione sia per quanto riguarda "reset vector" che "exception vector" e scegliere come memoria alla quale fare riferimento la memoria realizzata al punto 2.

Effettuare un click sull'identificativo dell'IRQ del blocco JTAG_UART per collegarlo con la CPU ed assegnargli una priorità (si può lasciare in questa fase il valore di default).

Creare inoltre un nome specifico per i segnali che vengono esportati verso l'esterno dai blocchi di IO destinati a input e output (switches e leds)

Da un punto di vista globale l'architettura del processore dovrebbe essere più o meno simile a quanto riportato in figura



Si salvi la configurazione

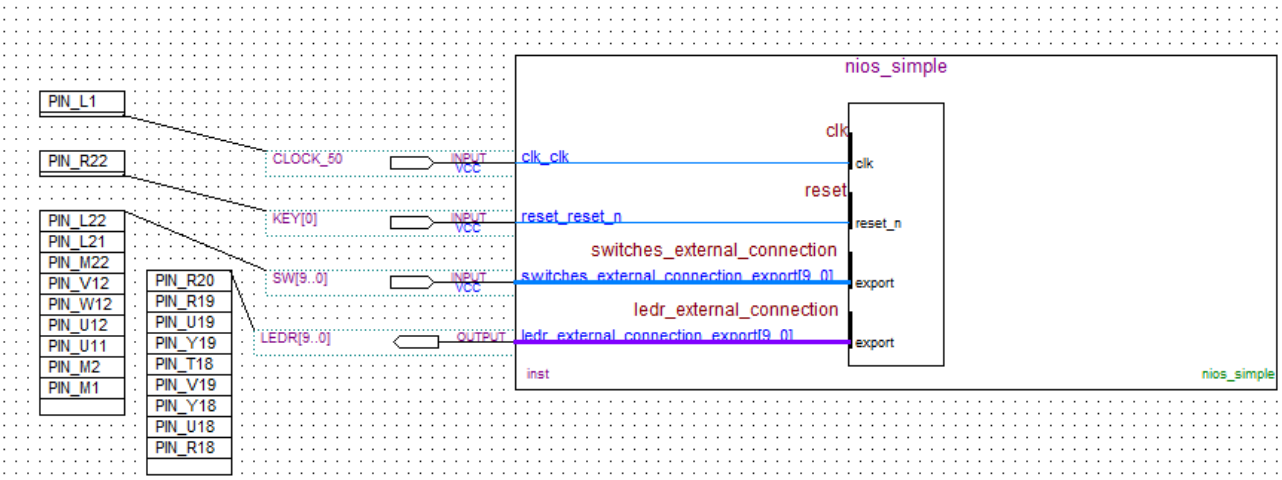
```
File > Save
```

- Si prenda nota degli indirizzi ai quali sono state mappate le varie periferiche (Address Map);
- Sella scheda Project Setting ci si accerti che la FPGA utilizzata come target sia corretta
- Nella scheda HDL Example ci si accerti di utilizzare il linguaggio verilog
- Accedendo alla scheda Generation ci si assicuri che sia attivata l'opzione "Create Block Symbol File" e si clicchi su "Generate".
- A processo ultimato si può chiudere la finestra del Qsys

Realizzazione del sistema completo

Il processore completato con le sue periferiche, così realizzato, può trovare posto in un sistema completo ove può essere integrato con eventuali ulteriori blocchi hardware.

- Si crei un nuovo schematico
- Si importi il Sistema appena generato
- Si colleghino tutti gli ingressi ed uscite ad opportuni PIN
- Si assegnino dei nomi coerenti col file di vincoli che si verrà successivamente ad assegnare al progetto
- Si importi un opportuno file di vincoli



Eventualmente lo schematico potrebbe venir sostituito da un opportuno file strutturale scritto in Verilog che potrebbe forse risultare più agevole. In tal caso il file potrebbe essere il seguente:

```
module Simple_CPU (
        // Inputs
        CLOCK_50,
        KEY,
        SW,

        // Outputs
        LEDR
);


/*****************************************************************************
 *                             Port Declarations                             *
 *****************************************************************************/
// Inputs
input                   CLOCK_50;
input                   [3:0]   KEY;
input                   [9:0]   SW;

// Outputs
output                  [9:0]   LEDR;


/*****************************************************************************
 *                               Internal Modules                            *
 *****************************************************************************/

    simple inst (
        .clk_clk       (CLOCK_50),    //   clk.clk
        .reset_reset_n (KEY[0]),      // reset.reset_n
        .sw_export     (SW),          //    sw.export
        .led_export    (LEDR)         //   led.export
    );

endmodule
```

Un esempio su come istanziare il blocco inerente il processore è presente all'interno del tool Qsys nella scheda HDL Example.

Si noti che eventualmente un sistema comodo per generare un file Vertilog HDL di tipo strutturale, partendo da uno schematico è quello di utilizzare il comando:

```
File > Create/Update > Create HDL Design File from current File …
```

Ove scegliere il linguaggio Verilog (o eventualmente VHDL)

- Ci si assicuri che Il file appena realizzato (schematico oppure Verilog) sia configurato come "TOP Level ENTITY"
- Si aggiunga tra i files di progetto il file `nome.qip` creato in fase di generazione del processore e risiedente tipicamente nel direttorio
  `<nome>/synthesis/<nome>.qip`
- Si compili quindi l'intero progetto
- Si esegua il download su FPGA

Si noti che tra il vari files generati in fase di realizzazione del processore vi è in particolare un file di documentazione `<nome>.html` che contiene molte informazioni utili sulla struttura dei vari blocchi, sui loro collegamenti e sui loro indirizzi.

Sviluppo Software

A questo punto il sistema è pronto per funzionare, ma vi si deve ancora caricare l'opportuno software. Per fare questo esistono vari sistemi, in questo tutorial utilizzeremo "**ALTERA monitor Program**" [3][4]: un sistema utile per la compilazione del sorgente, il download del codice nella memoria del processore ed il debugging.

Si supponga che si voglia far girare sul processore il seguente semplice programma:

```
#define Switches (volatile int *) 0x0009010
#define LEDR (int *) 0x0009000

   int main()
   {
   int a;
   while (1)
          {a = *Switches;
          *LEDR = a;}
   }
```

Questo programma legge nella locazione assegnata agli switches il doto associato alla disposizione di questi, e scrivendo tale dato in corrispondenza all'indirizzo sul quale sono mappati i led rossi, porta all'accensione o allo spegnimento dei medesimi.

Ovviamente gli indirizzi rispettivamente di "Switches" e "LEDR" devono corrispondere a quelli del sistema hardware sviluppato al passo precedente. Utilizzando un normale text editor si editi pertanto questo testo e lo si salvi all'interno di un opportuno direttorio. (E' consigliabile adottare un opportuno direttorio dedicato al software all'interno della cartella in cui si è sviluppato il progetto hardware)

Si lanci il programma "Altera Monitor Program" (se non sono presenti collegamenti sul desktop un probabile percorso è il seguente)

Start > Programmi > Altera > University Program > Altera Monitor Program > Altera Monitor Program

Si apra un nuovo progetto

```
File > New Project
```

Si scelga il direttorio in cui far risiedere il progetto (è consigliabile usare lo stesso direttorio nel quale è stato salvato il file scritto in C) ed il nome del progetto stesso. (NEXT)

Si selezioni il sistema su cui far risiedere il progetto, si scelga il sistema

```
<Custom system>
```

Si noti che nell'elenco sono presenti diversi altri processori[5][6] realizzati appositamente per le schede DE1, DE2, … con diverse potenzialità e risorse che pertanto potrebbero essere impiegati laddove non vi fosse la necessità di utilizzare un processore "custom" e se non si intendono realizzare risorse hardware dedicate.

Si indichi al sistema il file `.qsys` (che contiene le informazioni sulla struttura e sull'architettura del sistema realizzato in HW) che risiede nel direttorio in cui si è sviluppato il progetto HW.
Si indichi eventualmente il file `.sof` con cui configurare l'FPGA. (NEXT)

Si selezioni la tipologia di linguaggio che si desidera adottare

```
C Program
(Next)
```

E si includa il file C che è stato salvato nei passi precedenti. (NEXT)

Se vi fossero più CPU disponibili o più periferiche da usarsi come STDIO si potrebbero indicare in questa pagina, altrimenti (NEXT)

Se vi fossero più memorie a disposizione si potrebbe scegliere in quale far risiedere il testo ed i dati (NEXT)

Il sistema chiede se si vuole riconfigurare l'FPGA col file `.sof` indicato in precedenza (questo è un metodo comodo per configurare l'FPGA direttamente dall'interno di "Altera Monitor Program" senza dover utilizzare il programma quartus. Da sottolineare che tale possibilità sussiste solamente se il sistema non impiega blocchi sviluppati da terze parti protetti da copyright e con licenza a tempo, nel qual caso la programmazione dell'FPGA DEVE essere fatta mediante il "programming tool" di Quartus).

A questo punto non rimane che da compilare il sorgente

```
Action > Compile (ctrl-Shift-C)
```

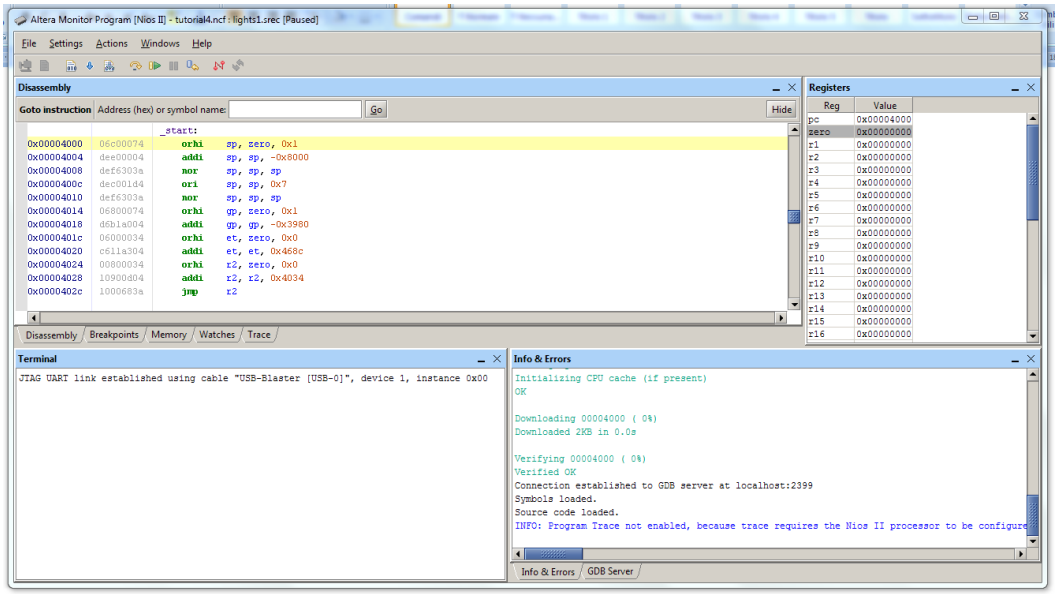
E fare il download del codice nella memoria del processore

```
Action > Load (ctrl-shift-L)
```

Alternativamente le due azioni possono essere svolte in rapida sequenza una di seguito all'altra con

`Action > Compile & Load (F5)`

Se tutto ha funzionato correttamente si dovrebbe ottenere una schermata simile



A questo punto il sistema è pronto e programmato, per avviarlo

`Action > Continue (F3)` – oppure si usino gli opportuni tasti

Nota 1: Si provi ad usare il tasto di "pause" o in alternativa il tasto di reset del processore (nell'esempio usato è KEY[0])

Nota 2: Si provi ora a definire i puntatori non a "int" bensì a "char". Cosa cambia nel funzionamento ?

Nota 3: si noti che sebbene il ciclo while si sarebbe potuto scrivere più semplicemente come

```
while (1)
   {*LEDR = *Switches;}
```

Questo può comportare talvolta problemi di interpretazione al compilatore, ovvero ad ogni passo del ciclo si deve aggiornare la posizione di memoria occupata dai LED, ma questo deve essere fatto in base all'attuale dato presente negli switches oppure in base al precedente?

Nota 4: Si provi a modificare la tipologia di puntatore alla locazione degli Switches da (`volatile int*`) a `(int*)` **:** cosa cambia nel funzionamento [7] ?

Nota 5: provi ad aprire la scheda dedicata la memoria (0x9000) e si acceda alle locazioni di memoria dedicate a LED e SWITCHES. Si fermi il processore, si rinfreschi la memoria e si legga quanto riportato nella locazione degli SWITCHES. Successivamente si provi a scrivere qualche valore alla locazione destinata ai LED .

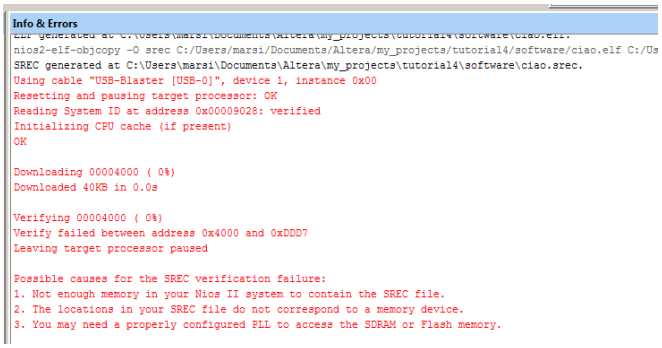
Nota 6: si provi ora qualche altro programma, ad esempio:

```
for (a=1;a<34235;a++) *LEDR = a;
```

Nota 7 : si provi ora con:

```
#include <stdio.h>
int main()
   {printf("CIAO/n");}
```

Sebbene sintatticamente corretto, in fase di download su dispositivo si ottiene il seguente errore:



Si può infatti notare che la memoria interna all'FPGA da dedicare al processore risulta abbastanza limitata 16384 bits sono in pratica 512 parole da 32 bits, e pertanto essa non riesce a contenere le risorse utili a gestire le funzioni di input/output su STDIO.

Si deve pertanto far ricorso alle risorse esterne all'FPGA ovvero alla SRAM ed alla SDRAM montate sulla DE1.

Utilizzo di memorie esterne [8].

SRAM

La DE1 monta una memoria SRAM IS61LV25616 da 256K x 16 bits.

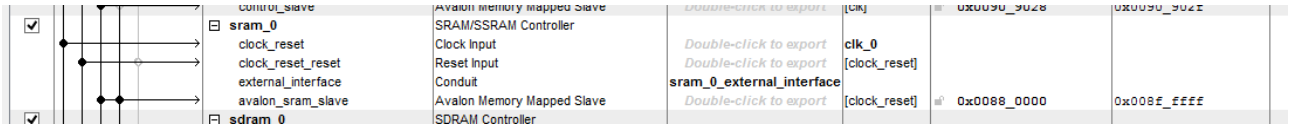
Per includere questa tra le periferiche del sistema da sviluppare si può utilizzare il controllore già sviluppato in ambito dell'"**University Program**" che mette gratuitamente a disposizione degli utenti una serie di interfacce dedicate appunto ai componenti montati sulle schede DE0, DE1, DE2, DE2-x e DE3.

All'interno dell'SOPC Builder selezionare:

```
University Program > Memory > SRAM/SSRAM Controller
```

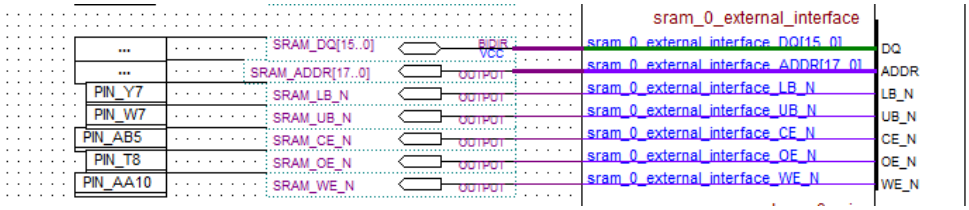
Come configurazione si scelga la scheda su cui si sta lavorando (DE1) ed eventualmente se questa memoria volesse essere impiegata come Frame Buffer in un sistema di elaborazione video (in questo caso no).

Vi si aggiungano gli opportuni collegamenti ed un nome per i segnali da esportare



Dopo aver rilanciato "GENERATE" il sistema complessivo comprenderà anche le linee per il collegamento con la memoria esterna. Si dovrà pertanto collegare queste ad opportuni PIN, e dare a questi ultimi un nome uguale a quello assegnato all'interno del file di vincoli per garantire il corretto collegamento.

NOTA BENE: mentre tutte le linee sono configurate come uscite, la linea dati deve essere configurata come bidirezionale (infatti normalmente in una memoria i dati si possono sia scrivere che leggere)



SDRAM

La DE1 monta una memoria SDRAM IS42S16400 da 1Mword x 16 bit x 4 Banks (64 Mbits) ove gli indirizzi sono organizzati in matrici di 12 righe x 8 colonne. La gestione dei segnali di una SDRAM, se fatta direttamente a livello hardware risulta abbastanza problematica (si ricordi ad esempio che necessita di opportuni cicli di refresh). Però per l'uso che se ne fa in questo tutorial torna comodo l'impiego di un controller già opportunamente sviluppato da terze parti che rende praticamente trasparente tutta la gestione dei segnali.

All'interno dell'SOPC Builder selezionare:

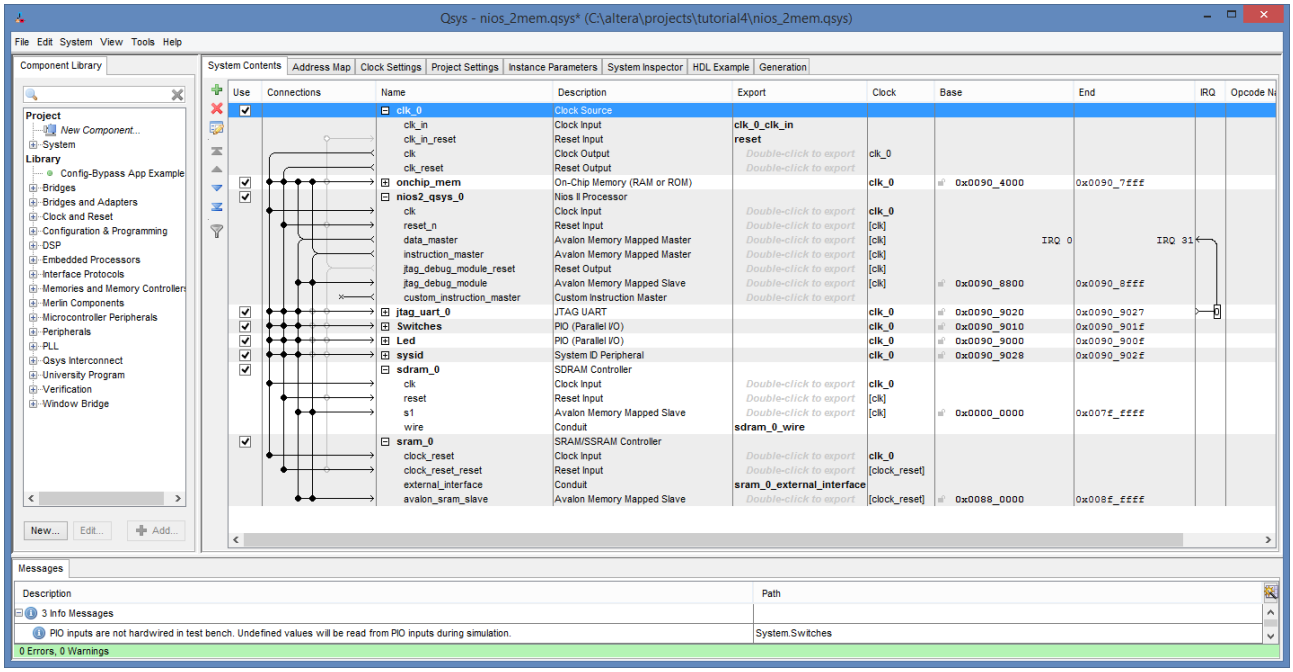
```
Memories & Memory Controller > SDRAM > SDRAM Controller
```

- Successivamente si configuri il blocco come
  - Preset: Custom
  - Bits : 16
  - Architecture: 1 chip select, 4 banks
  - Address width: row 12, column 8

Si lascino inalterati tutti gli altri parametri di configurazione.

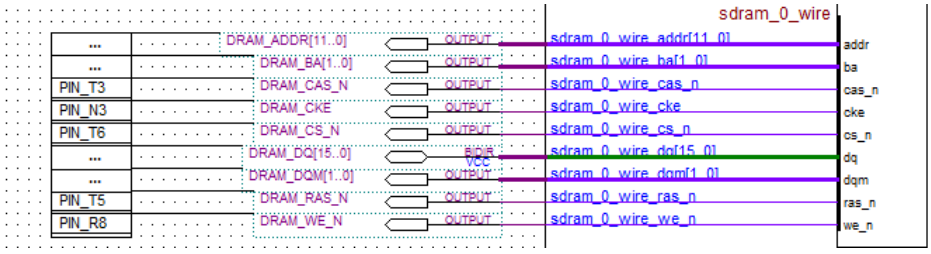
Si assegnino i collegamenti e si dia un nome ai segnali da esportare

Si riassegnino automaticamente tutti gli indirizzi. Il sistema completo dei controllori per le memorie SDRAM e SRAM dovrebbe più o meno coincidere con quanto sotto riportato.



Dopo aver rilanciato “GENERATE” il blocco complessivo comprenderà, analogamente a quanto accaduto per la SRAM anche le linee per il collegamento con la memoria esterna. Si dovrà pertanto collegare queste ad opportuni PIN, e dare a questi ultimi un nome uguale a quello assegnato all’interno del file di vincoli per garantire il corretto collegamento.

NOTA BENE: mentre tutte le linee sono configurate come uscite, la linea dati deve essere configurata come bidirezionale (infatti normalmente in una memoria i dati si possono sia scrivere che leggere) questo è posto in evidenza anche dal colore verde che assume il corrispondente collegamento verso il processore all’interno del simbolo.



SDRAM-PLL

A differenza della SRAM che non è controllata da alcun clock, la SDRAM necessita un clock opportuno e la sincronizzazione di questo è particolarmente critica, si deve infatti riuscire a compensare i ritardi introdotti a livello di scheda dalle linee di collegamento tra l’FPGA e la SDRAM. Per fare questo, torna utile l’impiego di un elemento specifico per la ri-sincronizzazione dei segnali, una PLL (Phase Lock Loop) [9].

All’interno di Quartus:

```
Tools > Mega Wizzard Plugin Manager
```

(Create a New Custom Megafunction)

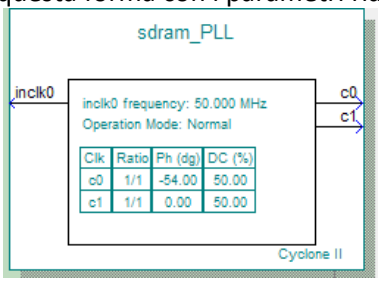
```
Installed Plugins > I/O > ALTPLL
```

Si dia un nome opportuno (ad esempio SDRAM_PLL)

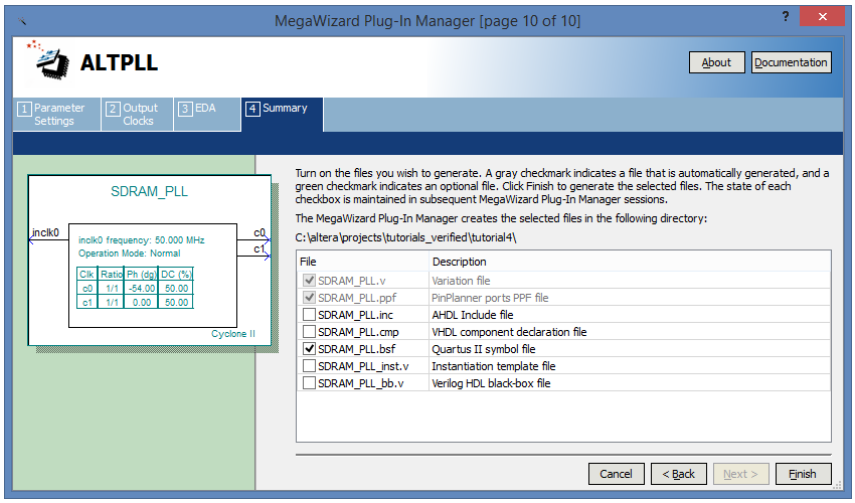
Si scelga il linguaggio di descrizione (Verilog HDL) e la famiglia di FPGA (Ciclone II).
Si configuri opportunamente il blocco

Parameter Settings|General/Modes – Si imposti il clock in ingress a 50 MHz
Parameter Settings|Inputs/Locks – Si disattivi "*create pllena*" e "*create locked output*"
Output clocks |clk0 – si attivi "use this clock" e si fissi *clock phase shift = -3 ns*
Output clocks |clk1 – si attivi "use this clock"

Alla fine il blocco dovrebbe assumere questa forma con i parametri riassunti nella tabella:



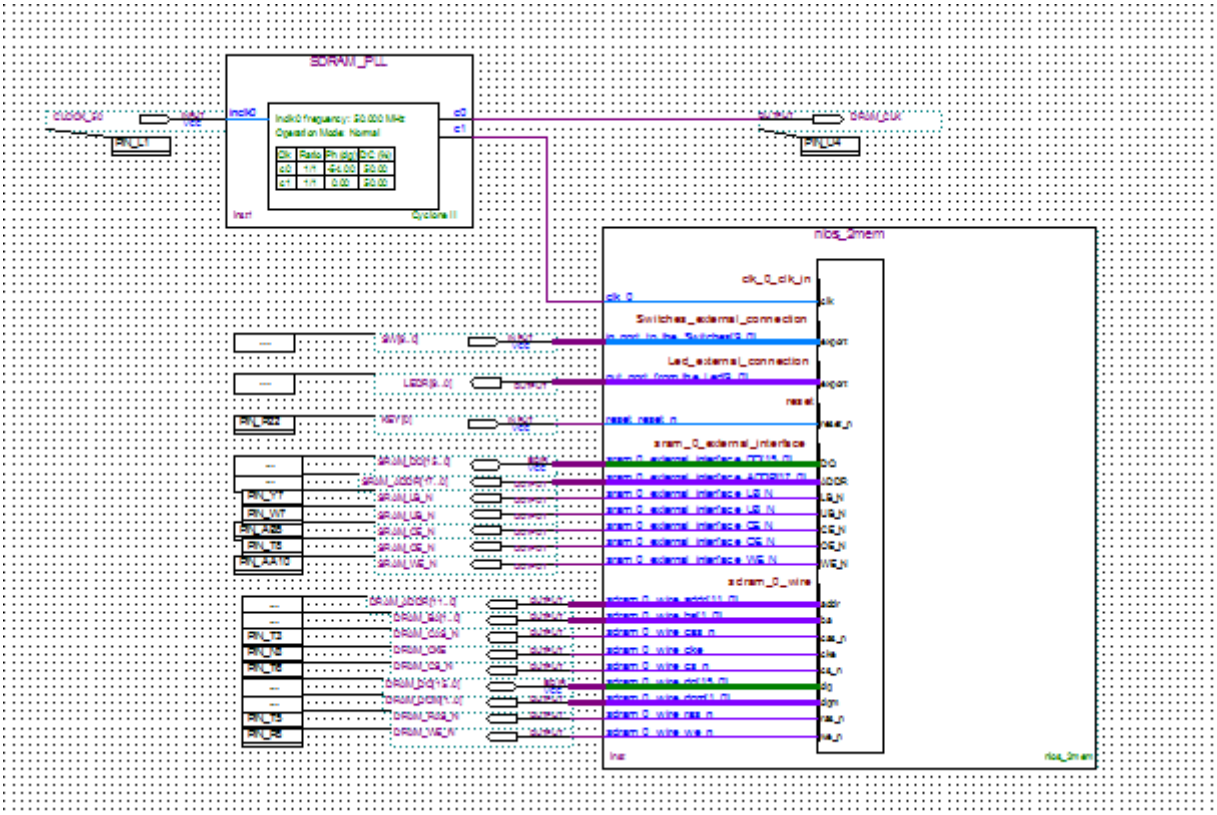
A questo punto si può generare il blocco (scegliendo quali file generare) e lo si farà entrare a far parte del progetto in essere.




```
>  Finish
```

SCHEMA finale

La PLL così generata fornirà i clock tanto per il processore quanto per la SDRAM. Lo schema finale da realizzare, comprensivo di tutti i pin sarà il seguente:



Si compili il progetto complessivo e si effettui il download sulla DE1

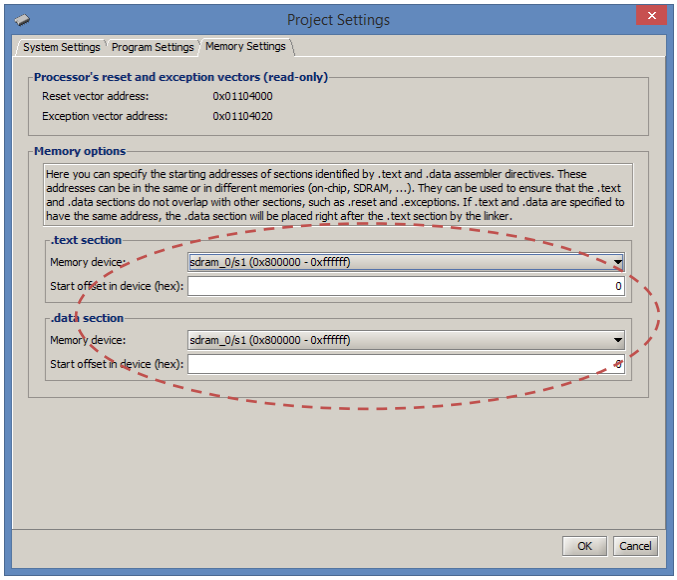
Si Noti che tra i blocchi messi a disposizione dalla libreria “University Program” ve ne è, per inciso, uno specifico (`Clock Signals for DE-Series Board Peripheral`) [10] realizzato espressamente per la generazione del clock da collegare alla SDRAM montata sulla DE1 (che peraltro è realizzato nel medesimo modo), esso ovviamente è molto più immediato da impiegare rispetto il PLL appena visto che è stato introdotto a scopo essenzialmente didattico. Inoltre il suddetto blocco può essere impiegato anche per generare in parallelo anche i clocks per pilotare sia il decoder audio che la VGA.

Sviluppo software

Con questo nuovo sistema, la memoria a disposizione è di molto aumentata rispetto il sistema precedente, si può pertanto provare a far girare, utilizzando “Altera Monitor Program” anche programmi che richiedano maggior risorse in termini di memoria, come ad esempio

```
#include <stdio.h>
int main()
   {printf("Hello World\n");}
```

Ricordandosi peraltro di modificare gli indirizzi di memoria delle varie periferiche e di assicurarsi, in fase di progetto software di fare sì che i codici eseguibili risiedano nelle memorie più ampie rispetto la memoria disponibile on-chip.



L'uscita testuale, indirizzata di default STDIO ed avendo provveduto a reindirizzare quest'ultimo su UART-JTAG (in fase di configurazione del progetto all'interno di Altera Monitor Program), viene visualizzata sulla finestra "TERMINAL" del tool predetto.

Sviluppo software

Per tutti i blocchi impiegati a livello Hardware ed in particolare per quelli presenti all'interno della libreria "University Program" sono disponibili delle funzioni ad alto livello di astrazione le cosiddette HAL (Hardware Abstraction Level) esse si compongono di una serie di primitive talvolta anche abbastanza evolute che l'utente può utilizzare per dialogare col dispositivo senza dover scendere a livello di registro o di locazione di memoria. In tal modo la stesura del codice C risulta meno legata all'architettura del sistema e può in un certo senso essere resa più indipendente dall'architettura migliorandone così la portabilità.

Per lavorare in tal senso quando si realizza il software per il sistema utilizzando "Altera Monitor Program", si deve selezionare come tipologia linguaggio che si desidera adottare (Program Type)

```
Program with Device Driver Support
```

Così facendo in fase di compilazione il tool provvederà a generare tutte le librerie gli header da utilizzare all'interno della nostra programmazione. La fase di compilazione ovviamente si allunga ed anche il file compilato così generato richiederà maggior spazio in memoria.

Si consiglia di analizzare il file

```
BSP/system.h
```

Che contiene tutte le definizioni inerenti ai blocchi Hardware componenti il processore. Inoltre per conoscere nel dettaglio le funzioni disponibili per ciascun blocco impiegato si consiglia di fare riferimento alla documentazione reperibile all'interno del tool Qsys attraverso il tasto "Documentation" disponibile per ciascun blocco utilizzato.

Per fare un esempio il nostro programmino di lettura e scrittura da switches e led potrebbe essere redatto, sfruttando le HAL, nella seguente forma:

```
#include "system.h"
#include "altera_avalon_pio_regs.h"

int main()
{
int a;
while (1)
      {
      a=IORD_ALTERA_AVALON_PIO_DATA(SWITCHES_BASE);
      IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE,a);
      }
}
```

Si tenga conto che nella fattispecie si sono impiegate delle generiche porte di I/O per switches e leds, ma eventualmente all'interno della libreria "University Program" vi sono anche delle porte di I/O [12] espressamente realizzate per pilotare esattamente i dispositivi presenti sulla scheda DE1 o DE2 e che si possono trovare seguendo il percorso:

```
University Program > Generic IO > Parallel Port
```

Utilizzando questi ultimi per interfacciarsi ai led ed agli switches un codice potrebbe essere il seguente:

```
#include <stdio.h>
#include "system.h"
#include "alt_types.h"
#include "system.h"
#include "altera_up_avalon_parallel_port.h"
#include "altera_up_avalon_parallel_port_regs.h"

int main()
{

alt_up_parallel_port_dev *SW;
alt_up_parallel_port_dev *LED;

printf("Hello from Nios II!\n");

// SWITHES

SW = alt_up_parallel_port_open_dev("/dev/switches");
if(SW == NULL)
         printf("ERROR:Error opening Swithes\n");
      else
         printf ("Opened Switches device \n");

printf("SW: %04x\n", alt_up_parallel_port_read_data(SW));
```

```
// LED

LED = alt_up_parallel_port_open_dev("/dev/led");
if(LED == NULL)
          printf("ERROR:Error opening LEDS\n");
      else
          printf ("Opened LED device \n");


while(1) {
alt_up_parallel_port_write_data(LED,alt_up_parallel_port_read_data(SW));}
}
```

Bibliografia

[1] Altera Corporation: Introduction to the Altera Nios II Soft Processor
ftp://ftp.altera.com/up/pub/Altera_Material/13.0/Tutorials/Nios2_introduction.pdf

[2] Altera Corporation : Introduction to the Altera Qsys System Integration Tool
ftp://ftp.altera.com/up/pub/Altera_Material/13.0/Tutorials/Introduction_to_the_Altera_Qsys_Tool.pdf

[3] Altera Corporation: Altera Monitor Program web page
http://www.altera.com/education/univ/software/monitor/unv-monitor.html

[4] Altera Corporation: Altera Monitor Program Tutorial
ftp://ftp.altera.com/up/pub/Altera_Material/13.0/Tutorials/Altera_Monitor_Program.pdf

[5] Basic Computer System for the Altera DE1 Board
ftp://ftp.altera.com/up/pub/Altera_Material/13.0/Examples/DE1/NiosII_Computer_Systems/DE1_Basic_Computer.pdf

[6] Altera Corporation: Media Computer System for the Altera DE1 Board
ftp://ftp.altera.com/up/pub/Altera_Material/13.0/Examples/DE1/NiosII_Computer_Systems/DE1_Media_Computer.pdf

[7] Eric Eide John Regehr, Volatiles Are Miscompiled, and What to Do about It
http://www.cs.utah.edu/~regehr/papers/emsoft08-preprint.pdf

[8] Altera Corporation: Using the SDRAM on Altera's DE1 Board with Verilog Designs
ftp://ftp.altera.com/up/pub/Altera_Material/13.0/Tutorials/Verilog/DE1/Using_the_SDRAM.pdf

[9] Altera Corporation: Phase-Locked Loop (ALTPLL) Megafunction
http://www.altera.com/literature/ug/ug_altpll.pdf

[10] Altera Corporation: Clock Signals for Altera DE-Series Boards
ftp://ftp.altera.com/up/pub/Altera_Material/13.0/University_Program_IP_Cores/Altera_UP_Clocks.pdf

[11] Altera Corporation: Using HAL Device Drivers with the Altera Monitor Program
ftp://ftp.altera.com/up/pub/Altera_Material/13.0/Tutorials/HAL_tutorial.pdf

[12]Altera Corporation: Parallel Port for Altera DE-Series Boards
ftp://ftp.altera.com/up/pub/Altera_Material/13.0/University_Program_IP_Cores/Input_Output/Parallel_Port.pdf