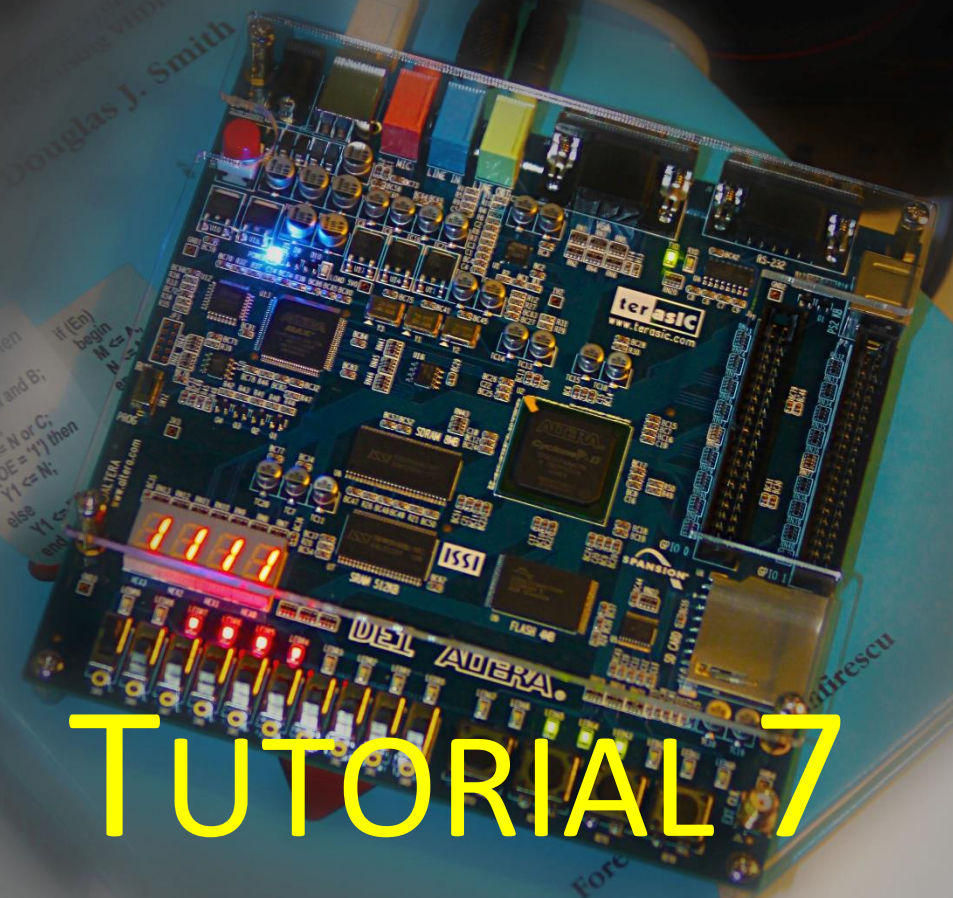
FPGA Tutorial on DE1 Board



TUTORIAL 7

Using Quartus II V13.0

Marsi 2014 - University of TRIESTE

Tutorial FPGA n.7

Realizzazione hardware di interfacce Avalon.

Descrizione: Si andrà a realizzare di un insieme di interfacce dedicate, da integrare col processore NIOS-II all'interno del tool QSyS: Verrà realizzata prima una semplicissima ROM ai cui valori vi si possa accedere direttamente via software, successivamente si vedrà come integrare il filtro realizzato nel tutorial precedente ed altri eventuali blocchi di elaborazione audio con altre interfacce a disposizione nelle librerie all'interno di Qsys

Scopo: Apprendere i fondamenti per la realizzazione di interfacce "custom" da integrare col processore Nios-II da utilizzarsi per la realizzazione di un sistema di elaborazione congiunta HW-SW

Apprendimenti previsti: Approfondimento dei diversi modi di interfacciamento al processore, Realizzazione di interfacce.

1. Introduzione:

Nel tutorial precedente si è visto che, per far funzionare correttamente un sistema, il progettista deve decidere quali siano le operazioni che devono essere svolte via software e quali invece conviene vengano eseguite da un hardware dedicato. Tali blocchi hardware che finora hanno trovato posto in uno schema "a fianco" del processore possono essere facilmente integrati in quella ricca libreria di interfacce realizzate da terze parti alla quale abbiamo abbondantemente attinto nei precedenti tre tutorial, consentendo pertanto un loro riutilizzo più agevole e soprattutto demandando a QSyS il compito di adattare i vari bus e eventualmente di gestire l'arbitrato per l'accesso alle risorse. Per fare questo però bisognerà approfondire come il processore fornisce e riceve i dati attraverso il suo sistema di comunicazione.

In particolare va sottolineato che vi sono a disposizione due tipologie diverse di reti di interconnessione [1]:

Avalon MM (Memory Mapped): è una rete di interconnessione bidirezionale che consente operazioni di lettura e scrittura in forma Master-Slave (ad esempio tra il Nios e una memoria) e che realizza una serie di collegamenti utili appunto per interfacciarsi verso vari moduli (memorie, dispositivi di I/O, ecc.). Questo è un sistema di comunicazione piuttosto lento ma utile per un accesso seriale a dati o istruzioni.

Avalon ST (Streaming): è una rete di comunicazione ad alta velocità che consente di trasferire grandi moli di dati direttamente da un master e verso uno slave in modo **monodirezionale**. Sono possibili diverse metodologie di comunicazione.

Ritornando all'esempio del filtro svolto nel tutorial precedente, Il flusso di elaborazione dei dati audio realizzato dal filtro IIR potrebbe essere gestito attraverso la rete Avalon-ST, mentre il sistema di controllo svolto dal processore per modificare i parametri del filtro stesso potrebbe essere realizzato tramite il protocollo Avalon MM (e questo sarà per l'appunto l'obiettivo del presente tutorial)

2. Semplice interfaccia Avalon MM

Si inizi un progetto finalizzato ad essere implementato su una FPGA `EP02C20F484C7`

Si descriva in Verilog HDL un semplice circuito che realizzi una ROM a 8 indirizzi di 32 bit. I segnali fondamentali di interfaccia saranno: l'indirizzo (3 bit) il dato (32 bit) oltre ad un segnale di clock ed uno di reset. Una descrizione del blocco potrebbe essere la seguente:

```
module simple_rom(
                // inputs:
                 address,
                 clock,
                 reset_n,

                // outputs:
                 readdata
              );

  output  [31:0] readdata;
  input   [2:0]    address;
  input            clock;
  input            reset_n;

  reg    [31:0] readdata;
  reg    [2:0]  addr_reg;

always @( posedge clock)
       addr_reg=address;

always @ (addr_reg)
       case (addr_reg)
              3'h0: readdata = 32'h0001;
              3'h1: readdata = 32'h0002;
              3'h2: readdata = 32'h0003;
              3'h3: readdata = 32'h0004;
              3'h4: readdata = 32'h0011;
              3'h5: readdata = 32'h0012;
              3'h6: readdata = 32'h0013;
              3'h7: readdata = 32'h0014;
       endcase
endmodule
```

Eventualmente si compili il sorgente per verificare non vi siano errori e, alla bisogna, si effettuino le debite simulazioni per verificarne il corretto funzionamento.

Si apra il tool QSyS e si realizzi un sistema composto (nella sua forma minima) dai seguenti dispositivi:

- Clock Source (default)
- Nios II – e
- JTAG Uart
- SysId
- On-chip Memory

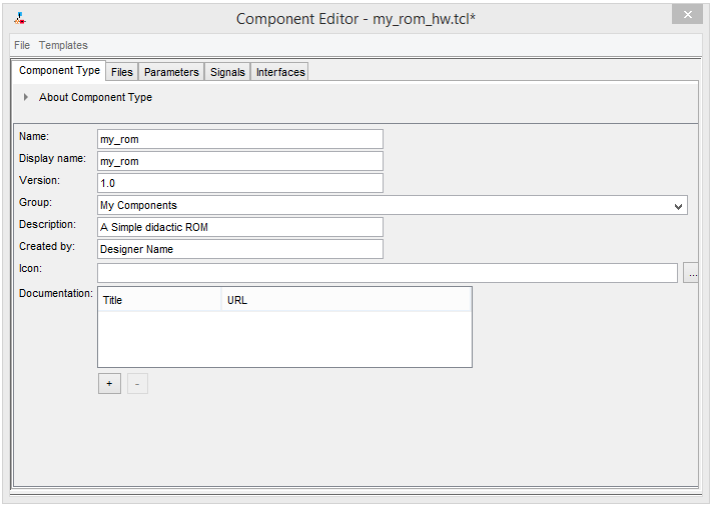
Eventualmente si può integrare l'architettura con altre interfacce per la gestione di memorie o di dispositivi di I/O, quali LED o Swithes.

Si colleghino opportunamente i segnali di interfaccia, si dia un nome ai segnali verso l'esterno (in particolare il segnale di clcok e di Reset), si assegni un indirizzo alle varie periferiche e agli eventuali interrupt.

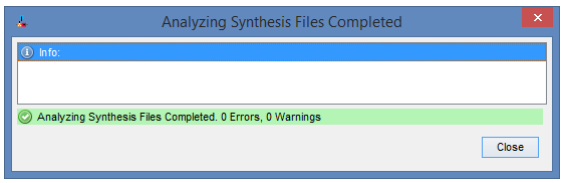
Per importare adesso il nuovo blocco all'interno di QSyS [2][3]

```
File > New Component
```

Nella scheda "Components Type" si forniscano il nome ed eventualmente le info caratteristiche del nuovo componente



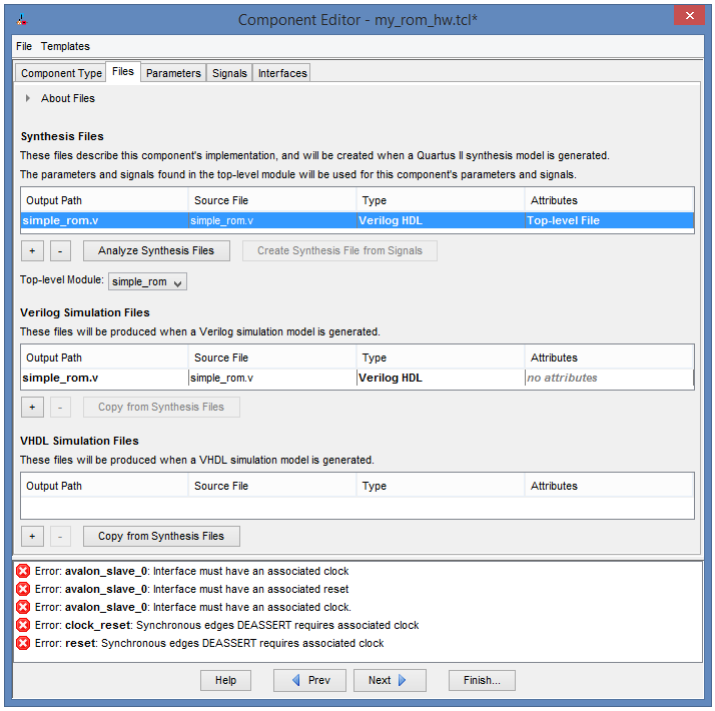
Nella scheda "Files" si importi col tasto "+" il file Verilog che descrive il componente (realizzato precedentemente) e quindi si clicchi su "`Analyze Synthesis Files`". Se il componente non presenta errori di sintassi appare la seguente finestra



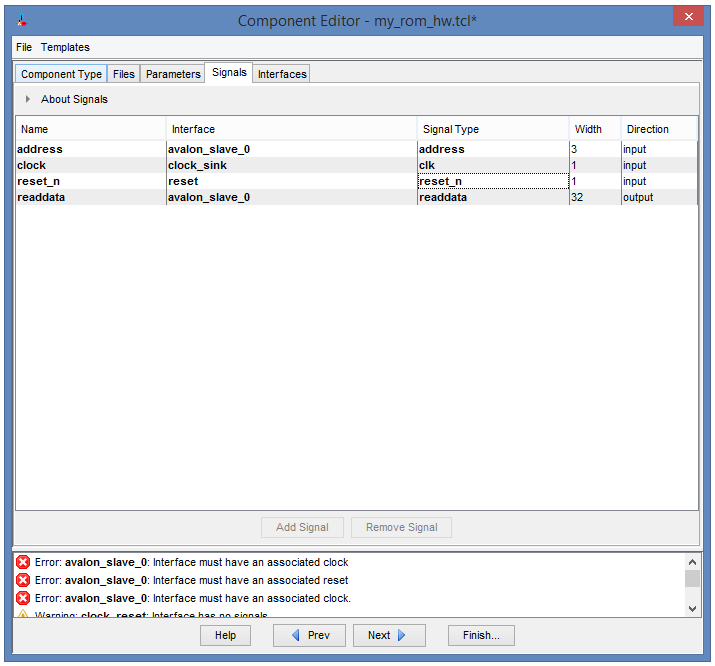

```
> Close
```

Al momento non si presti troppa attenzione agli errori che viceversa compaiono nella finestra principale e che verranno via via corretti ed eliminati con i successivi passi:

Eventualmente si possono importare i files per eventuali simulazioni (che però non sono previste in questo tutorial) ad esempio col tasto "`Copy from Synthesis Files`"



Nella scheda **Signals** definire per il segnale clock nella colonna ***Interface***: "`New Clock Input`" e nella colonna ***Signal Type*** scegliere "`clk`". Gli altri segnali probabilmente sono stati automaticamente individuati con la corretta interfaccia. Se così non fosse modificarli per renderli aderenti allo schema della figura seguente



Nella scheda **Interfaces**: associare il corretto segnale di clock e di reset a tutte le varie interfacce. Così facendo dovrebbero scomparire tutti gli errori dalla console. Se permane ancora un "Warning" del tipo: "`Warning: clock_reset: Interface has no signals`". Si clicchi sul tasto Remove Interfaces With No Signals. Ora il nostro blocco dovrebbe avere tre interfacce:

- Un' interfaccia di `clock`
- Un' interfaccia di `reset` (attivo allo stato basso)
- Un' interfaccia Avalon_slave limitata ai segnali `address` e `readdata`

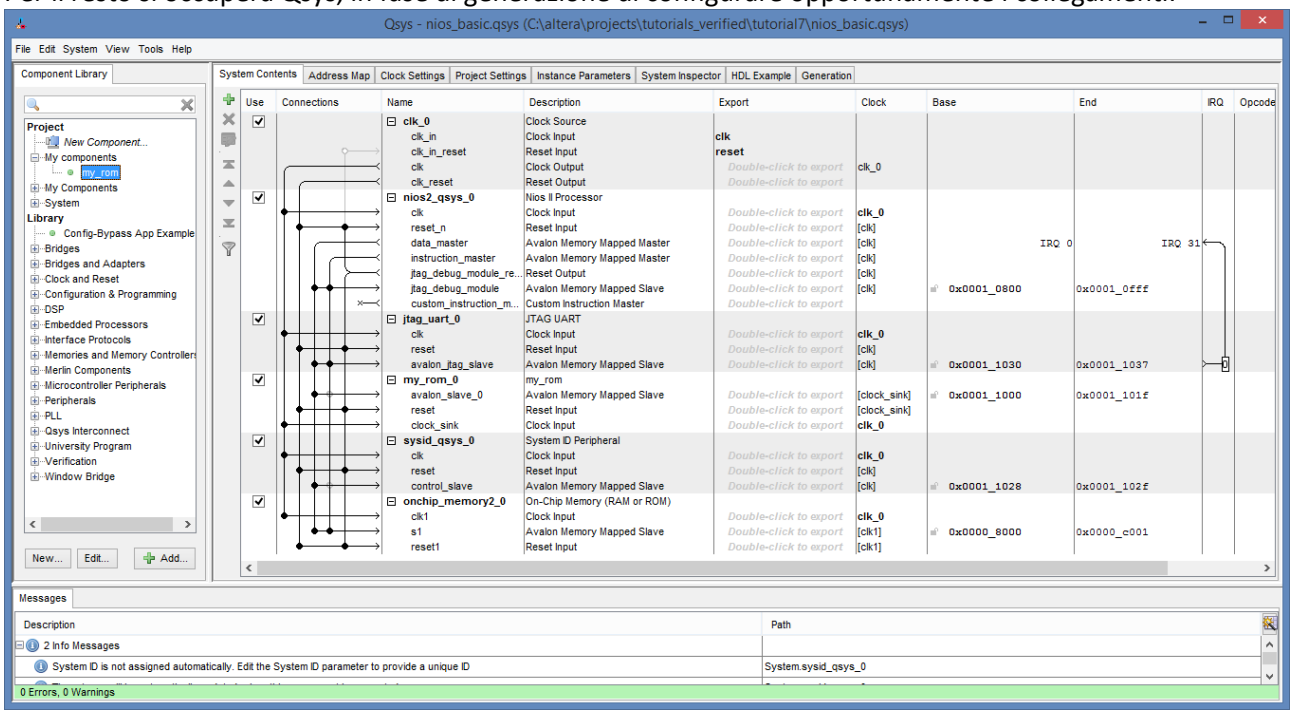
```
> Finish
> Save
```

Si noti che tutta questa procedura si limita a generare un file .tcl di configurazione che eventualmente potrebbe essere realizzato a mano quando si prendesse un po' di pratica col sistema [4].

Ora nella scheda "Component Library" di QSys dovrebbe essere comparso il nuovo componente appena realizzato, che può essere incluso tra gli altri blocchi costituendi il nostro sistema e opportunamente collegato rispettivamente agli opportuni segnali di `clock`, `reset` e `data_master`. Si aggiornino ora gli indirizzi dei vari blocchi

```
System > Assign Base Addresses
```

Per il resto si occuperà Qsys, in fase di generazione di configurare opportunamente i collegamenti.

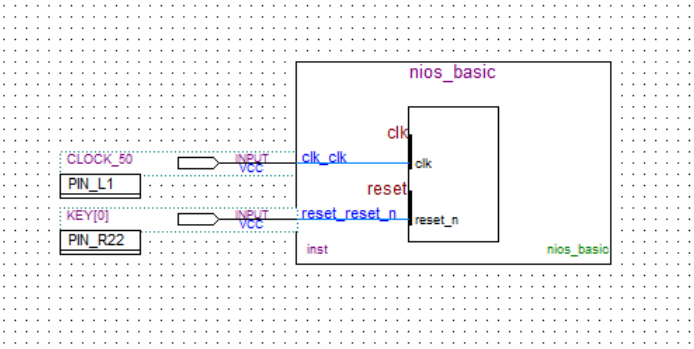


A questo punto si può generare il sistema

Scheda Generation: `Generate`

- Chiudere Qsys

- All'interno di quartus realizzare un sistema (tramite schematico oppure tramite verilog) per collegare i vari ingressi ed uscite del sistema appena generato ai debiti piedini di I/O, che nel caso semplice in esame sono solo il clock ed il reset (se non si sono aggiunte ulteriori interfacce)



- Assegnare a questi ultimi un nome congruo con il file di vincoli.
- Importare il corretto file di vincoli
- Includere tra i files di progetto il file .qip associato al sistema sviluppato con QSys
- compilare il tutto
- eseguire il download su scheda
- prendere nota degli indirizzi dei vari dispositivi (in particolare della ROM)

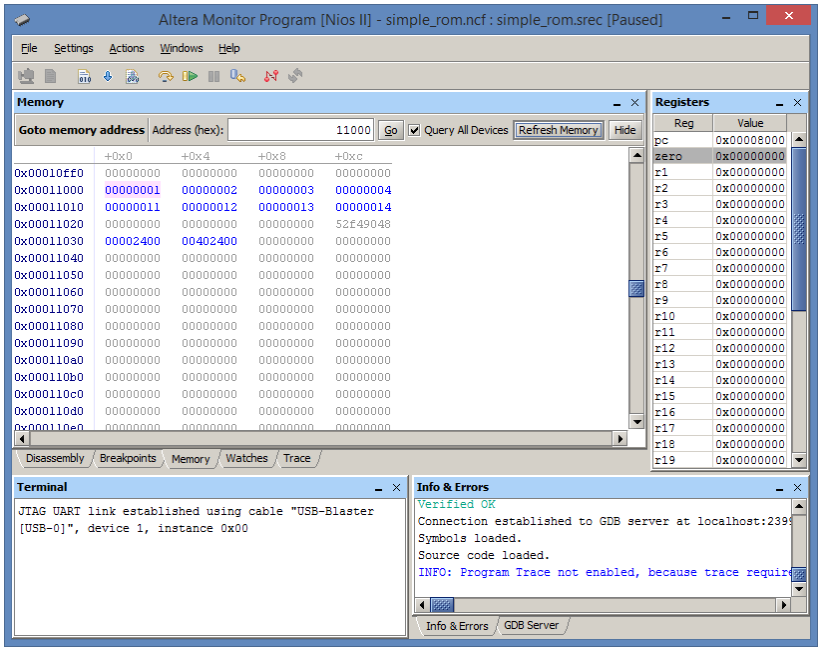
A questo punto si può chiudere Quartus e si attivi "Altera Monitor Program".

- Si realizzi un nuovo progetto per il sistema appena generato
- Gli si associ un qualsiasi file di programmazione, possibilmente semplice. Ricordando che il sistema così generato praticamente non ha quasi modo di comunicare con l'esterno in quanto l'unica forma di comunicazione attraverso la UART necessiterebbe almeno di un po' di memoria in più per contenere le primitive della libreria **stdio.**

Un possibile file davvero elementare è il seguente

```
void main()
{
int a;
a=1;
}
```

- Si compili il file, lo si carichi sul sistema.
- Attraverso Altera Monitor Program si acceda ora alla scheda Memory. Si visualizzi la posizione di memoria in cui è mappata la ROM si attivi l'opzione "**Query all Devices**" e si clicchi sul tasto "**Refresh Memory**"



Si potrà così notare che nelle relative locazioni di memoria appaiono i dati che erano stati assegnati alla ROM e che quindi possono essere disponibili per qualsiasi software sviluppato per il processore in questione.

3. Sistema di interfacciamento congiunto Avalon MM e Avalon ST

In questa seconda parte si andrà a realizzare un sistema congiunto di interfacciamento ad un blocco funzionale attraverso entrambe le interfacce Avalon a disposizione.

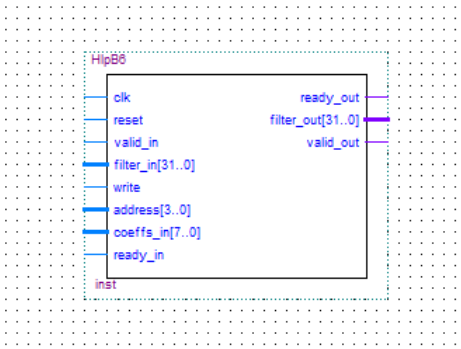
In particolare si andrà a realizzare un filtro IIR (similmente al tutorial precedente), attraversato da un flusso continuo di dati audio che viene gestito con l'interfaccia Avalon ST ove, al contempo sia possibile modificare i suoi coefficienti attraverso l'interfaccia Avalon MM.

Il codice Verilog del filtro da realizzare è disponibile nel sito web del corso [].

Nella fattispecie esso prevede oltre ai canonici segnali di **clock** e **reset** (allo stato basso) i segnali:

- **Valid_in, filter_in, ready_out** (di cui I primi due sono ingressi ed il terzo, è un'uscita) e gestiscono il flusso di dati audio in ingresso dall'interfaccia Avalon_ST.
- **Valid_out, filter_out, ready_in** (di cui I primi due sono uscite ed il terzo, è un ingresso) e gestiscono il flusso di dati audio filtrati in uscita verso'interfaccia Avalon_ST.
- **Address, coeff_in,write** sono i segnali che servono a scrivere i coefficienti del filtro attraverso l'interfaccia Avalon MM

Da notare che sebbene i segnali **ready_xx** potrebbero essere utili per gestire delle eventuali situazioni di "back_pressure" il nostro blocco, essendo sprovvisto di FIFO non riesce ne' a gestire, ne' a creare lui stesso situazioni di overflow di eventuali buffer, pertanto i due segnali sono assolutamente trasparenti l'uno verso l'altro.



Attraverso Quartus e quindi QSys si realizzi un sistema più evoluto del precedente composto dai seguenti blocchi:

- Clock Source (default)
- Clock Signal for DE-series Board Peripherals
    - Configurato per generare I tre seguenti segnali
        - Audio clock da 12.288 MHz
        - SDRAM Clock
        - Sys Clock
- Nios II – e
- JTAG Uart
- SysId
- SWITCHES
- KEYS
- LED (rossi e verdi)
- SDRAM Memory
- Audio e Video Config
    - Auto Initialize Device(s)
    - Audio Path: Line in
    - Enable DAC Output
    - Data Format: Left Justified
    - Bit Length: 32 bit
    - Sampling Rate: 48 KHz
- Audio
    - Interface : Avalon Type : Streaming
    - Enable: Audio In
    - Enable: Audio out
    - Data Width : 32

Secondo la procedura vista sopra si importi tra le periferiche il nuovo componente costituito dal filtro IIR e si configurino i vari segnali come afferenti alle diverse interfacce secondo lo schema sotto riportato:

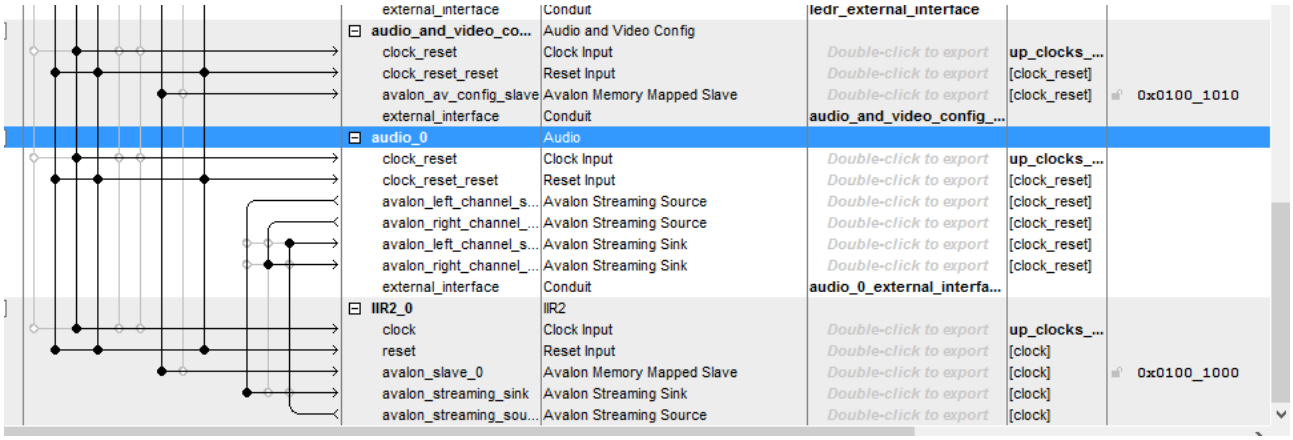
Component Type | Files | Parameters | Signals | Interfaces

About Signals

Name	Interface	Signal Type	Width	Direction
clk	clock	clk	1	input
reset	reset	reset_n	1	input
write	avalon_slave_0	write	1	input
address	avalon_slave_0	address	4	input
coeffs_in	avalon_slave_0	writedata	8	input
valid_in	avalon_streaming_sink	valid	1	input
filter_in	avalon_streaming_sink	data	32	input
ready_out	avalon_streaming_sink	ready	1	output
filter_out	avalon_streaming_source	data	32	output
valid_out	avalon_streaming_source	valid	1	output
ready_in	avalon_streaming_source	ready	1	input

Ci si assicuri che tutte le interfacce ricevano i corretti segnali di clock e di reset.

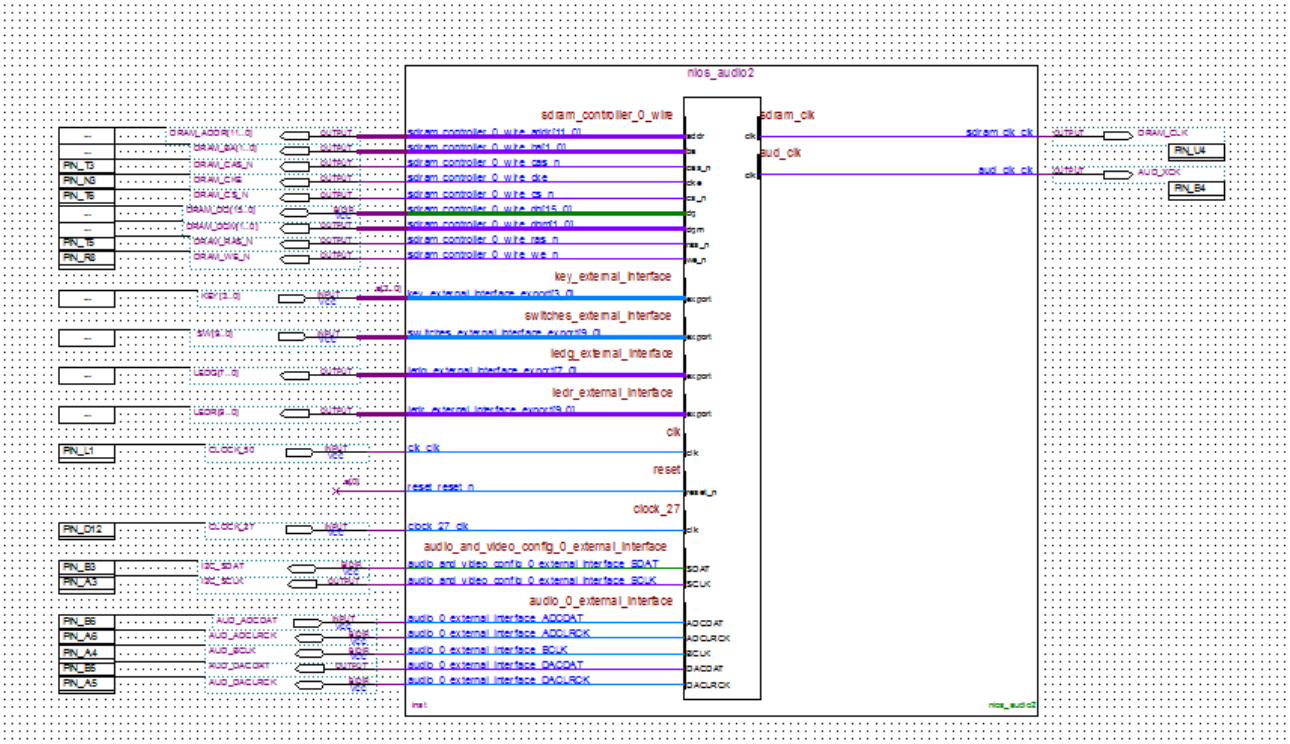
Si importi il nuovo componente tra le periferiche del sistema complessivo e si colleghino opportunamente le varie interfacce.



(Si noti che nella configurazione proposta il filtro opererà solo sul canale sinistro, mentre il destro risulta copiare in uscita i dati presenti in ingresso.

Si assegnino opportuni nomi ai segnali d'uscita, si assegnino indirizzi opportuni alle varie periferiche, si prenda nota di questi ultimi e si generi il sistema completo.

Si realizzi ora uno schematico (o un file Verilog) che assegni gli opportuni pin di ingresso e uscita ai vari segnali del sistema, con nomi compatibili ai files di vincoli, si importi un opportuno file di vincoli, si includa il file .qip che descrive il sistema tra i files di progetto e si compili l'intero sistema.



A questo punto Utilizzando Altera Monitor Program si può provare a realizzare un semplice sistema che configuri opportunamente il filtro.

Nel caso proposto si realizzerà un ciclo infinito che leggerà i dati forniti dagli switches e attraverso essi configurerà i parametri del filtro.

Si ricordi che come dal tutorial precedente che:

- Il filtro realizzato dal blocco hardware presenta un'evoluzione temporale modellizzata dalle seguenti equazioni:

$$x_n = K_1(in)$$
$$y_n = K_2(b_o * x_n + b_1 * x_{n-1} + b_2 * x_{n-2} - a_1 * y_{n-1} - a_2 * y_{n-2})$$

- Gli indirizzi dei registri ai quali fare riferimento per scrivere i vari parametri del filtro sono i seguenti:

Indirizzo	Parametro
000	K1
001	B0
010	B1
011	B2
100	A1
101	A2
110	shift
111	K2

- Esauriti tutti i dati si deve scrivere nel registro di locazione 0x8 per aggiornare contemporaneamente tutti i coefficienti ed i parametri del filtro.

Un codice possibile per realizzare questa procedura è il seguente:

```
#include <stdio.h>

#define Keys (volatile int *) 0x01001050
#define Switches (volatile long int *) 0x01001040
#define LEDG (int*) 0x01001030
#define LEDR (long int*) 0x01001020
#define IIR_COEFF (char*) 0x01001000

void delay(void);

int main()
{
long int a,b,dat,we;
long int ta,sa;

printf("Start \n");

while (1)
{
a = *Switches;
*LEDR = a;
b = *Keys;
*LEDG = b;

ta = (a & 0x0000003F);
sa = (a & 0x000003C0)>>6;

*IIR_COEFF=0x7F;      // input gain
*(IIR_COEFF+1)=0x7F;  // b0
*(IIR_COEFF+2)=0x00;  // b1
*(IIR_COEFF+3)=0x80;  // b2

*(IIR_COEFF+4)=0x84+ta; //a1  (freq)
*(IIR_COEFF+5)=0x3C;    //a2  (selettività)

*(IIR_COEFF+6)=sa;     // shift

*(IIR_COEFF+7)=0x7F;    // output gain

*(IIR_COEFF+8)=0x80;    // done
}
}
```

Si noti in particolare che gli switches `SW[9:6]` regolano il volume del segnale d'uscita, mentre `SW[5:0]` regolano la frequenza centrale del filtro. Inoltre il pulsante `KEY[0]` realizza un reset Hardware.

Si può inoltre notare che mentre nella ROM trattata all'inizio dell'esercitazione la memoria dedicata era di 4 byte per parola (per un totale di 32 bytes), ora gli 8 coefficienti occupano un solo byte a testa, infatti ad essi è riservata una zona di memoria di appena 4 byte (questo in quanto il bus dati per la scrittura dei coefficienti è stato realizzato da 8 bit). Pertanto per accedere alla locazione corretta per ciascun coefficiente il puntatore ad esso deve essere definito come `(char*)`.

Si noti inoltre che anche laddove arrivi un reset al processore ed il ciclo di assegnazione dei coefficienti dovesse fermarsi il flusso dei segnali audio non viene bloccato.

4. Interrupt

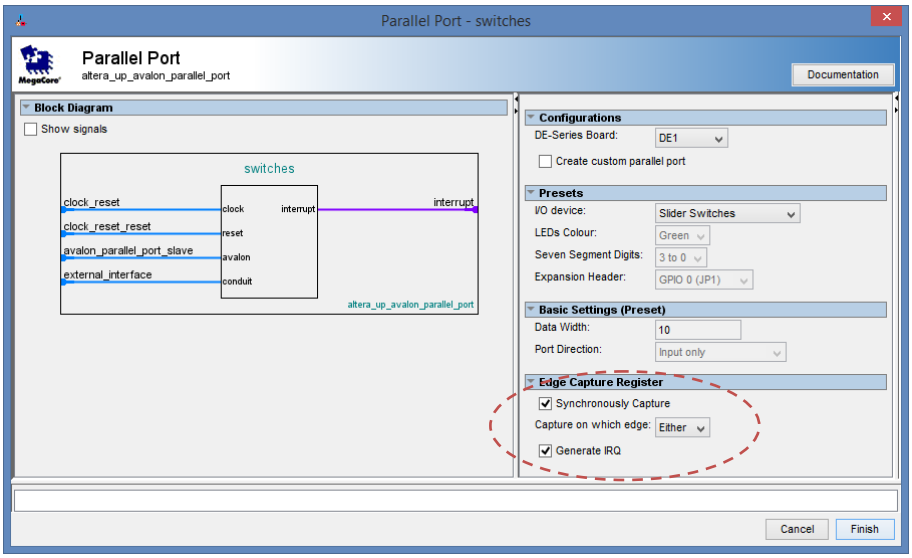
La soluzione fin qui proposta, sebbene funzionale non è di certo la più elegante ed un ciclo infinito di lettura dei dati dagli switches e modifica dei coefficienti sul filtro oltre ad impegnare inutilmente risorse potrebbe anche creare rumore sul canale audio laddove la scrittura dei coefficienti nei registri portasse delle discontinuità sul canale audio.

Per migliorare le prestazioni sarebbe utile aggiornare i coefficienti SOLO quando c'e' stata una reale modifica in questi ultimi. Questo può esser fatto utilizzando i segnali di "interrupt" [5]. Ovvero quando viene modificato uno switch si genera un segnale di interrupt che interrompe momentaneamente il processore per fogli eseguire una nuova routine (che nel nostro casa sarà l'aggiornamento dei coefficienti), ultimata la quale il processore riprende il suo ciclo.

Per ottenere un sistema che funzioni in tal senso si richiedono sia delle modifiche Hardware che Sotfware del nostro sistema

Hardware

Da un punto di vista Hardware andremo a modificare i blocchi relativi a "Switches" e "Keys" in modo che essi generino un opportuno segnale di interrupt, di cui per quanto riguarda gli switch esso venga generato sia sul fronte di salita che di discesa, mentre per le Keys solo sul fronte di discesa.



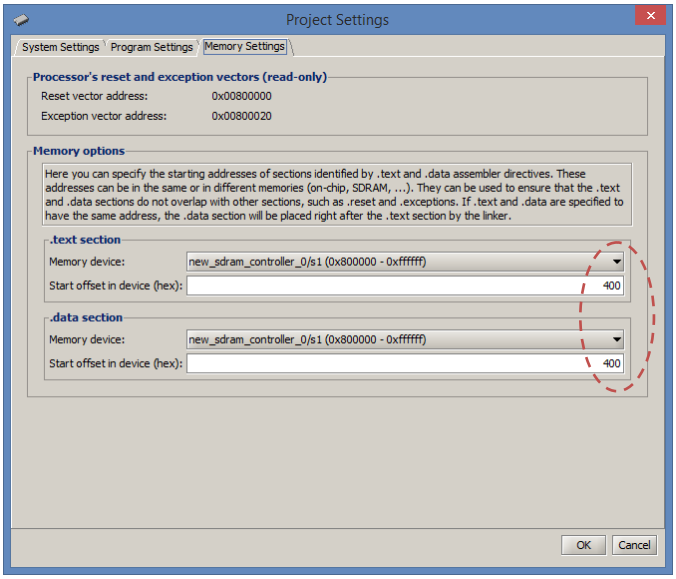
Successivamente nella colonna relativa alla priorità di interrupt vi si assegnino delle priorità mediante un numero compreso tra 0 e 31. Da notarsi che le priorità non sono gestite in alcun modo in hardware (quando arriva un interrupt viene solo forzato un salto alla posizione 0x20), ma consentono al software di riconoscere chi ha generato l'interrupt e eventualmente agire di conseguenza. Ad esempio si assegni priorità 0 agli switches, 1 alle keys e 8 a JTAG UART.

Ora si ricompili il sistema.

Software

Dal punto di vista software si deve:

1- Realizzare una debita routine da inserite alla locazione 0x20 per la gestione ed il riconoscimento dell'interrupt (tale routine è abbastanza standardizzata e può essere facilmente modificata per garantire l'attivazione di diverse subroutines da parte di diversi interrupt)
2- Creare abbastanza spazio a tale routine all'interno della memoria per evitare che vada in conflitto con il programma principale



3- All'interno del programma principale bisogna settare opportunamente i registri che abilitano gli interrupt. Nel nostro caso per attivare l'interrupt 0 e 1 scriveremo 3 (000…00011) se volessimo attivare gli interrupt di livello 5,4 e 0 scriveremmo 49 (000…110001).

```
#include "nios2_ctrl_reg_macros.h"

// set interrupt mask bits for levels 0 and level 1
NIOS2_WRITE_IENABLE( 0x3 );
// enable Nios II interrupts
NIOS2_WRITE_STATUS( 1 );
```

4- Inoltre in base a come è gestito l'interrupt da un punto di vista Hardware [6] dovremo agire (ove previsto) sulle maschere delle diverse periferiche. Con le periferiche impiegate dovremo introdurre le istruzioni:

```
*(Keys + 2) = 0xE;
*(Switches + 2) = 0x3FF;
```

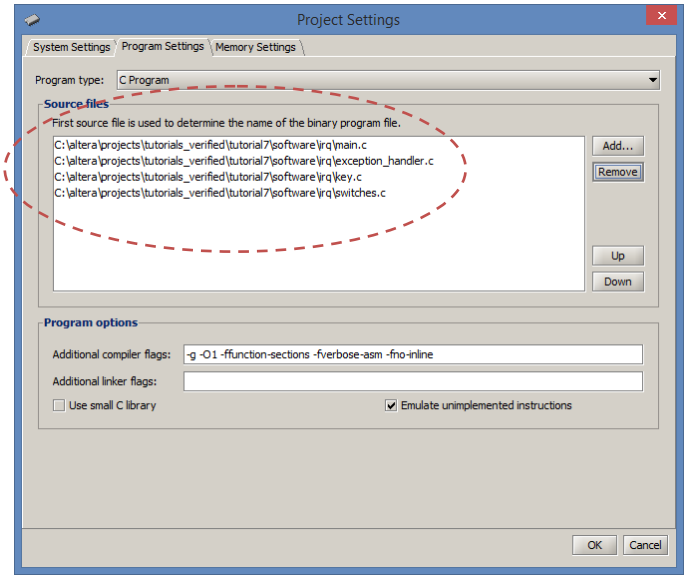
Si rimanda al manuale specifico [] per la descrizione di come siano stati realizzati in HW i sistemi di interfaccia e su come agire su di essi in base ai loro registri specifici, ma brevemente:

a. All'indirizzo 0 vi è lo stato delle line di ingresso
b. All'indirizzo 1 la direzione di queste
c. All'indirizzo 2 la maschera di interrupt
d. All'indirizzo 3 si accumulano le variazioni di stato che possono essere resettata scrivendo un qualsiasi dato in questa locazione

Comunque se si volesse realizzare un'interfaccia custom che generi un segnale di interrupt si dovrebbe prevedere un metodo software che consenta di disattivare lo stato dell'interrupt, infatti …

5- Una volta giunti nella routine richiamata dallo specifico interrupt si deve andare a disattivare l'interrupt stesso che ha generato la chiamata.

In fase di compilazione si includa nel progetto tutti i sorgenti in questione



6- Un esempio di questi sorgenti è riportato nella pagina web del corso.

Bibliografia

[1] Altera Corporation: **Avalon Interface Specification**: *http://www.altera.com/literature/manual/mnl_avalon_spec.pdf*

[2] Altera Corporation: Creating Qsys Components: *http://www.altera.com/literature/hb/qts/qsys_components.pdf*

[3] Altera Corporation: Making QSys Component: *ftp://ftp.altera.com/up/pub/Altera_Material/13.0/Tutorials/making_qsys_components.pdf*

[4] Altera Corporation: Component Interface Tcl Reference: *http://www.altera.com/literature/hb/qts/qsys_tcl.pdf*

[5] Altera Corporation: Exception Handling *http://www.altera.com/literature/hb/nios2/n2sw_nii52006.pdf*

[6] 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*