FPGA Tutorial on DE1 Board

TUTORIAL 6

Using Quartus II V13.0

Marsi 2014 - University of TRIESTE

Tutorial FPGA n.6

Realizzazione hardware di un filtro audio controllato tramite NIOS-2

Descrizione: Realizzazione di un filtro audio completamente in hardware, controllato nel suo funzionamento, ovvero nei valori dei coefficienti, tramite Nios-II.

Scopo: Apprendere i principi della progettazione congiunta Hardware-Software (Hardware-Software co-design)[1]

Apprendimenti previsti:

- Sviluppo di un filtro IIR audio a coefficienti variabili
- Gestione in tempo reale di un flusso audio
- Utilizzo del Nios-II per la configurazione dei coefficienti del filtro

Introduzione:

Nel tutorial precedente si è visto che, quando impiegato per elaborazioni in tempo reale, il Nios II esaurisce presto le sue risorse, d'altro canto è opportuno comprendere, all'interno di un sistema da sviluppare, quali siano le operazioni da far eseguire al processore e quali viceversa siano quelle per le quali merita sviluppare un dispositivo hardware dedicato.

Da un punto di vista pratico operazioni semplici, ripetitive e che devono essere svolte in tempo reale tipicamente vengono processate attraverso un sistema hardware dedicato, mentre viceversa operazioni complesse (tipicamente di controllo) che non richiedano una cadenza di esecuzione particolarmente elevata vengono demandate al processore.

L'esempio che verrà qui realizzato è quello di un filtro digitale audio, simile a quello svolto nel tutorial precedente, dove però il filtraggio dei campioni audio verrà svolto da un HW dedicato, mentre il processore sarà impiegato solo per aggiornare i coefficienti del filtro.

Il sistema completo potrebbe essere impiegato ad esempio per eseguire un filtraggio adattivo, ovvero un sistema, che basandosi su alcune caratteristiche del segnale in ingresso configuri il filtro per ottenere un certo particolare funzionamento. Un esempio banale potrebbe essere quello di controllo automatico del guadagno, ma applicazioni più lungimiranti potrebbero prevedere di utilizzare il sistema per sviluppare ad esempio una equalizzazione adattiva del segnale.

In questo tutorial d'altro canto ci occuperemo esclusivamente del sistema di controllo dei parametri del filtro e non dell'analisi del segnale, cosa che peraltro è stata svolta almeno in parte nel tutorial precedente.

Procedimento:

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

Definizione dell'architettura del Processore.

Utilizzando QSyS realizzare (come da precedente tutorial) un'architettura composta dai seguenti elementi:

- Clock Source (presente di default)
- Nios II, versione (s)
- JTAG UART
- System ID
- SRAM Controller
- SDRAM Controller
- Porta parallela per pilotare i LED verdi
- Porta parallela per pilotare i LED rossi
- Porta parallela di interfaccia verso i gli interruttori a scorrimento
- Porta parallela di interfaccia verso i pulsanti
- External Clocks for DE Board Periherals .
- Blocco di configurazione per il decoder.

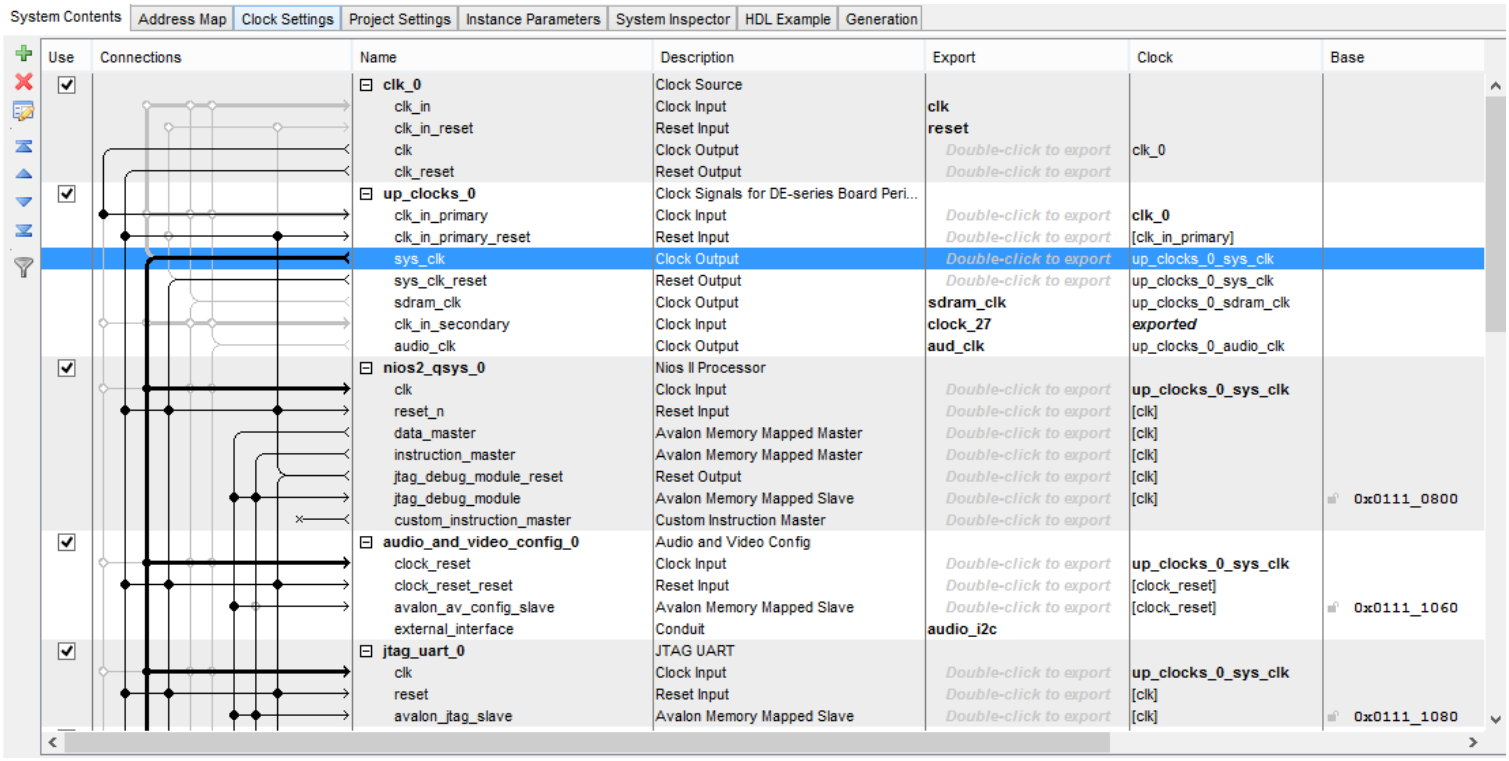
Si aggiungano inoltre due ulteriori porte palallele "custom" in uscita

- Una di 12 bit per fornite congiuntamente i coefficienti al filtro (`addr[3:0]`, e `data[7:0]`)
- Una di 1 bit per gestire la linea (`we`) per la scrittura dei coefficienti

Si forniscano dei nomi mnemonici ai vari blocchi.
Si stabiliscano tutte le opportune connessioni tra gli elementi con alcune attenzioni.

- Nel blocco "Clock Source" entrano i segnali di clock e di reset esterno e le uscite di questo vengono impiegate come segue:
  - Il clock entra nel blocco per "External Clocks for DE Board Periherals"
  - Il reset viene utilizzato come reset per tutti gli altri blocchi
- Nel blocco "External Clocks for DE Board Periherals"
  - Il segnale di clock primario in ingresso viene prelevato da "Clock Source"
  - Il segnale in_primary reset viene alimentato tanto dal reset del blocco "clock Source" che dal Nios sull'uscita "Jtag _debug_module_reset"
  - Il segnale sys_clock alimenterà tutti i blocchi di intefaccia seguenti.
  - Il segnale sys_clock_reset verrà utilizzato per resettare tutti i blocchi di intefaccia seguenti
  - Il segnale sdram_clock viene portato verso l'esterno ed alimenterà la sdram
  - Il segnale di ingresso secondario viene prelevato dall'esterno dall'oscillatore a 27MHz (si consiglia ad esempio di assegnargli il nome "clock_27")
  - Il segnale audio_clock viene portato all'esterno per alimentare il decoder audio
- Nel Blocco Nios
  - Il segnale Data Master deve alimetare tutte le interfaccie con un ingresso Avalo Master oltre alla linea stessa del Jtag_debug_module
  - Il segnale Istruction master alimenta solo i blocchi che fungono da memoria RAM oltre alla linea stessa del Jtag_debug_module
- Per tutti gli altri blocchi accertarsi

- Che ricevano il system clock
- Che ricevano il reset da tre blocchi: “Clock Source”, “External Clocks for DE Board Periherals” e Nios tramite la linea Jtag_debug_module_reset
- Che ricevano i dati dal Nios tramite l’interfaccia Avalon Master.
- Che tutte le interfacce verso l’esterno siano state opportunamente definite.



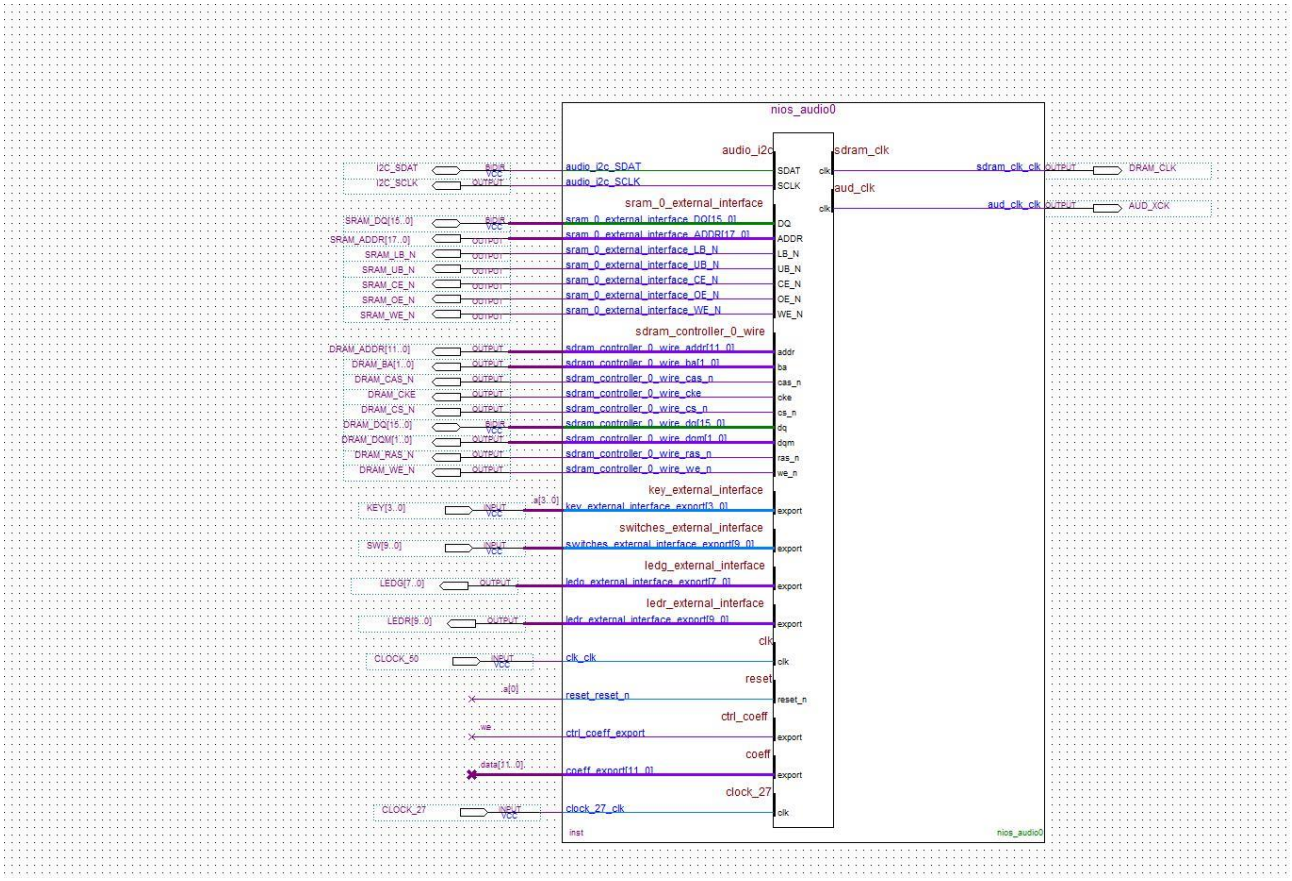
Si assegnino automaticamente gli indirizzi di memoria ai veri blocchi.
Si assegni automaticamente il livello di interrupt ai vari blocchi.

Si prenda nota dei vari indirizzi dei diversi blocchi e si generi (GENERATE) il sistema.

A questo punto si realizzi uno schematico in cui importare l’architettura del processore appena definita, si colleghino i vari pin con opportuni piedini di I/O e si dia a questi dei nomi concordi con quelli adottati nel file di vincoli che verrà accluso al progetto.

Non si colleghino viceversa i segnali in uscita dai due elementi di PIO di cui sopra ad alcun pin di uscita, ma viceversa si assegnino ad essi dei nomi mnemonici che verranno utilizzati successivamente per il collegamento con il filtro HW.

Il sistema completo di tutti i piedini di I/O dovrebbe più o meno assomigliare al seguente:



Definizione dell'architettura sistema di filtraggio.

Per realizzare un filtraggio dei campioni audio, in primo luogo vi è la necessità di realizzare due blocchi che eseguano prima la conversione da seriale a parallelo e successivamente da parallelo a seriale per i segnali da e per il convertitore audio.

Il convertitore seriale-parallelo, avrà in ingresso la linea `dati`, la `bclk` e la `lrck`, mentre in uscita vi saranno due linee a 32 bit sulle quali viaggeranno, alla cadenza del `lrck` rispettivamente i segnali relativi al canale destro e sinistro.
Una possibile realizzazione di questo blocco potrebbe essere la seguente:

```
module s2p
(
      // Input Ports
      input bck,lrck,
      input adc_dat,

      // Output Ports
      output reg [31:0] left,
      output reg [31:0] right
);

reg [31:0] left_reg, right_reg;
reg [5:0] counter; // 64 bits
wire [4:0] counter1;
reg mem,pulse;

assign counter1[4:0]=~(counter[4:0]);

// generazione di un impulso unitario in corrispondenza al fronte di salita di lrck
// solo mentre il reset è attivo per allineare il contatore
always @(posedge bck)  mem=lrck;
always @(posedge bck) begin if (!mem & lrck) pulse=1; else pulse=0; end

// contatore
```

```
always @(posedge bck) begin
      if (pulse) counter=6'b000010;
      else  counter=counter+1;
end

always @(posedge bck) begin
    if (lrck)
                left_reg[counter1]=adc_dat;
      else
                right_reg[counter1]=adc_dat;
end

always @(posedge lrck) right=right_reg;
always @(negedge lrck) left=left_reg;

endmodule
```

Esso si basa sui seguenti elementi:

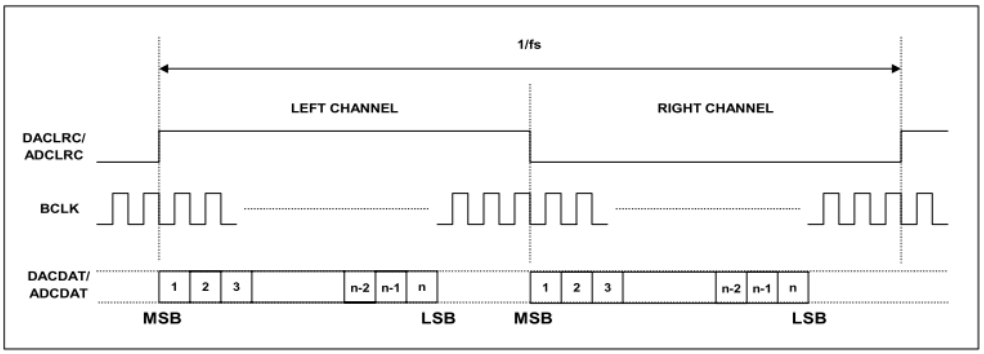
- Generazione di un impulso di sincronizzazione in corrispondenza al fronte di salita di `lrck`
- Un contatore circolare a 64 bit utile per smistare i dati. Questo contatore viene risincronizzato ad ogni ciclo con l'impulso di cui sopra. (Anche se in teoria basterebbe fosse sincronizzato solo all'inizio, rimane aperta la possibilità che esso perda di sincronismo ad esempio in seguito ad un reset)
- Un processo che smista i dati dalla porta seriale in ingresso alle porte parallele in uscita. Esso usa i 4 bit meno significativi del contatore di cui sopra opportunamente invertiti in modo da avere un conteggio discendente.
- Ulteriori due processi che sincronizzano l'uscita dei dati paralleli sul segnale `lrck`

In modo più o meno duale si può realizzare anche il blocco per la conversione da parallelo a seriale:

```
module p2s
(
      // Input Ports
      input bck,lrck,
      input [31:0] left, right,

      // Output Ports
      output reg dac_dat
);

wire [63:0] data={left_reg[31:0],right_reg[31:0]};
reg [5:0] counter; // 64 bits
wire [4:0] counter1;
reg [31:0] left_reg, right_reg;
reg mem,pulse;


// generazione di un impulso unitario in corrispondenza al fronte di salita di lrck
// solo mentre il reset è attivo per allineare il contatore
always @(posedge bck)  mem=lrck;
always @(posedge bck) begin if (!mem & lrck) pulse=1; else pulse=0; end

// contatore
always @(negedge bck) begin
      if (pulse) counter=6'b000010;
      else  counter=counter+1;
end


always @(negedge bck) begin
                dac_dat=data[~counter];
end


always @(posedge lrck) right_reg=right;
always @(negedge lrck) left_reg=left;

endmodule
```

Nota 1: si noti che mentre la "lettura" dei dati presenti in ingresso viene fatta sul fronte di salita del clock (ovvero con dati stabili), è utile che la scrittura degli stessi (ovvero la loro rigenerazione) avvenga sul fronte di discesa del clock, come rappresentato nella figura sottostante.

Nota 2: Si noti che non è cosa semplice avere un contatore che inizi il conteggio sincronizzato sul fronte di salita del segnale `LRCK`. Qualsiasi processo infatti che attenda di vedere lo stato del segnale `LRCK` per azzerare il conteggio, inesorabilmente eseguirà questa operazione in ritardo rispetto all'istante desiderato. L'unica possibilità è basata sulla conoscenza a priori della ciclicità dei segnali in ingresso e pertanto si può opportunamente cercare di anticipare il conteggio. Nei listati sopra riportati si inizia il conteggio all'istante "2" in modo che dopo un ciclo completo ci si ritrovi al valore "0" quando arriva il primo bit della parola in ingresso.



Un ulteriore blocco utile per il nostro sistema è un blocco di sincronizzazione dei segnali (che introduce un ritardo di un ciclo di clock)

```
module delay
(
      // Input Ports
      input clk,
      input [31:0] din,

      // Output Ports
      output reg [31:0] dout
);

always @(posedge clk) dout=din;

endmodule
```

Si realizzino i blocchi corrispondenti ai tre moduli di cui sopra e si generi il simbolo corrispondente.

Definizione dell'architettura del Filtro.

Esistono molti sistemi dedicati al progetto di filtri digitali lineari FIR ed IIR. Per realizzare il filtro impiegato in questo tutorial è stato utilizzato il tool `filterbuilder` [2][3] di MATLAB, che genera un filtro di determinate specifiche, ne visualizza le caratteristiche e successivamente genera il file HDL che lo descrive e ne consente la sintesi da parte di molti sistemi EDA.

Nel nostro caso le specifiche sono state le seguenti:

- Tipo di flitro: LP (la scelta di un filtro passa-basso consente di contenere l'ordine al minimo, viceversa la scelta di un filtro bassa-banda impone in fase progettuale al ricorso ad ordini superiori che poi si traducono in una maggiore complessità strutturale) … d'altro canto potendo modificare in un secondo tempo tutti i coefficienti del filtro esso potrà essere trasformato a piacere in LP, HP, BP, BS, ecc. ecc.
- Utilizzo di un filtro ricorsivo (IIR)

- Ordine del filtro: 2
- Aritmetica in Fixed point
- Valori in ingresso ed in uscita a 32 bit
- Coefficienti del filtro a 8 bit

Altri parametri sono riportati nell'intestazione del filtro stesso:

```
// -------------------------------------------------------------
//
// Module: HlpB1
//
// Generated by MATLAB(R) 7.8 and the Filter Design HDL Coder 2.4.
//
// Generated on: 2010-08-09 16:40:08
//
// -------------------------------------------------------------

// -------------------------------------------------------------
// HDL Code Generation Options:
//
// TargetDirectory: C:\Users\marsi\Documents\Altera\my_projects\tutorial6
// Name: HlpB1
// CoefficientSource: ProcessorInterface
// ResetAssertedLevel: Active-low
// TargetLanguage: Verilog
// TestBenchName: Hlp_tb
//
// Filter Settings:
//
// Discrete-Time IIR Filter (real)
// -------------------------------
// Filter Structure    : Direct-Form II, Second-Order Sections
// Number of Sections  : 1
// Stable              : Yes
// Linear Phase        : No
// Arithmetic          : fixed
// Numerator           : s8,7 -> [-1 1)
// Denominator         : s8,6 -> [-2 2)
// Scale Values        : s8,8 -> [-5.000000e-001 5.000000e-001)
// Input               : s32,0 -> [-2.147484e+009 2.147484e+009)
// Section Input       : s32,-3 -> [-1.717987e+010 1.717987e+010)
// Section Output      : s32,-3 -> [-1.717987e+010 1.717987e+010)
// Output              : s32,-3 -> [-1.717987e+010 1.717987e+010)
// State               : s32,0 -> [-2.147484e+009 2.147484e+009)
// Numerator Prod      : s40,7 -> [-4.294967e+009 4.294967e+009)
// Denominator Prod    : s40,6 -> [-8.589935e+009 8.589935e+009)
// Numerator Accum     : s40,5 -> [-1.717987e+010 1.717987e+010)
// Denominator Accum   : s40,4 -> [-3.435974e+010 3.435974e+010)
// Round Mode          : convergent
// Overflow Mode       : saturate
// Cast Before Sum     : true
// -------------------------------------------------------------
```

Da cui si evincono le precisioni (in termini di bit per la parte intera e per la parte frazionaria) delle varie sezioni e dei vari operatori (sommatori, moltiplicatori) componenti il filtro.

Il codice Verilog del filtro realizzato (parzialmente modificato) può essere scaricato dal sito WEB del corso.

Rispetto alla versione generata da MATLAB si sono aggiunte alcune piccole variazioni:

- I dati in uscita dal filtro vengono opportunamente shiftati (aritmeticamente) verso sinistra di un numero di posizioni impostabili tramite uno specifico registro (denominata shift). Questo consente di aumentare considerevolmente il volume del segnale in uscita. Infatti Il filtro generato da MATLAB per precauzione e prevenire distorsioni non consente di raggiungere il volume massimo in uscita. Il segnale filtrato, ovviamente conterrà meno energia del segnale originale e pertanto il suo volume complessivo sarà ridotto.

- Per un funzionamento corretto del filtro eventuali variazioni a coefficienti del medesimo devono avvenire nel medesimo momento. Si è pertanto introdotto un registro che nel momento stesso nel quale esso viene scritto rende disponibili all'interno dell'architettura del filtro tutti i coefficienti che sono stati fino a quell'istante salvati in dei registri interni. Il filtro originale prevedeva un funzionamento simile ma attraverso linea dedicata. Con questa variante lo si è reso compatibile a poter essere configurato attraverso l'interfaccia Avalon MM.

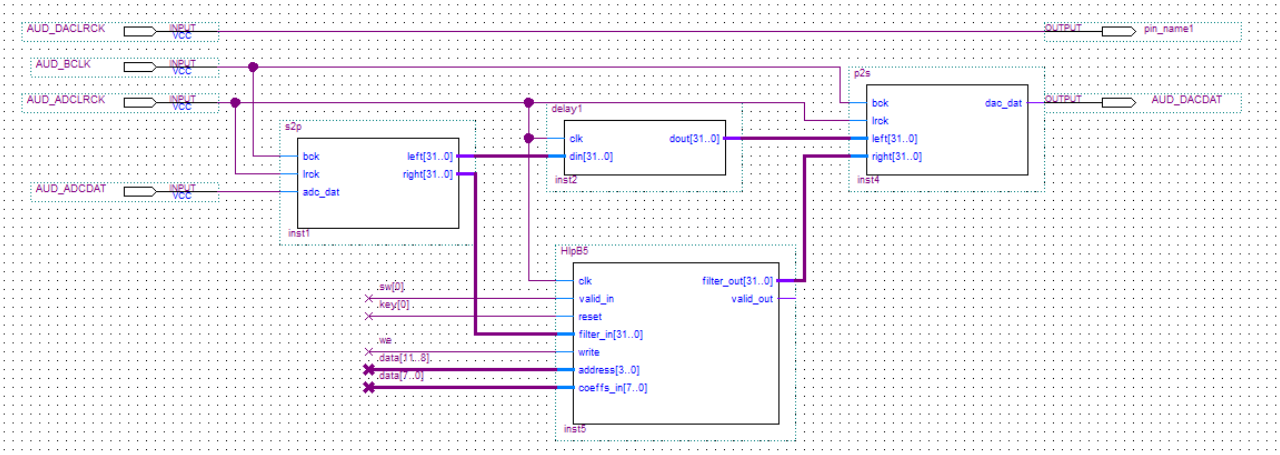
Si importi il file verilog rappresentante il Filtro IIR generato esternamente all'interno del progetto

```
Project > Add/Remove Files in Project
```

si realizzi quindi un ulteriore blocco relativo al filtro stesso e si generi il simbolo corrispondente.

Definizione dell'architettura completa del sistema.

Si può ora realizzare l'architettura completa del sistema: all'architettura del Nios si aggiunga nello schematico principale anche gli elementi qui sotto riportati:



Si noti in particolare:

1- I collegamenti opportuni tra le linee di controllo del NIOS ed il filtro
2- `SW[0]` è stato impiegato come `clock enable` del filtro
3- La linea `AUD_DACLRCK` anche se non viene impiegata in alcun modo deve trovare una sua ubicazione altrimenti il converitore non funziona.
4- Il filtro usa come segnale di clock il segnale di campionamento del segnale audio (AUD_ADCLRCK)

Giunti a questo punto si assegni un opportuno file di vincoli, si includa tra i file di sistema il file .qip del processore e si compili il progetto completo.

Definizione e memorizzazione dei coefficienti del filtro.

Il filtro realizzato dal blocco hardware [4] presenta un'evoluzione temporale modellizzata dalle seguenti equazioni:

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

Per scrivere i coefficienti nei registri di memoria preposti all'interno del blocco che realizza il filtro stesso si devono svolgere le seguenti operazioni:

1- Fornire sulla linea dati il dato da scrivere e sulla linea indirizzo l'indirizzo del coefficiente da modificare
2- Dare un impulso positivo sulla linea `we` per memorizzare il dato in una memoria temporanea
3- Ritornare al punto 1 per memorizzare un ulteriore coefficiente
4- Esauriti tutti i dati scrivere nel registro di locazione 0x8 per aggiornare contemporaneamente tutti i coefficienti e parametri del filtro.

Questa procedura è particolarmente utile perché evita di avere il filtro che funzioni con alcuni coefficienti aggiornati mentre altri non lo sono ancora (ed appartengono alla configurazione precedente). In questa situazione di incertezza non si può prevedere quale sarebbe la risposta del filtro che per esempio potrebbe entrare in instabilità.

Gli indirizzi ai quali fare riferimento per 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

Da notare che un impulso di reset (basso) azzera tutti i parametri.

Si dovrà pertanto programmare opportunamente il processore Nios II per fargli eseguire la sequenza di operazioni qui sopra richieste.

Software per la configurazione del Nios II

Sebbene le potenzialità del sistema messo a punto siano di molto superiori a quelle che si andranno a realizzare , nel presente tutorial si realizzarà un software dove all'interno di un ciclo infinito (`while (1)`) si leggerà il dato presente sugli switches e si aggiorneranno di conseguenza i coefficienti del filtro.

In particolare si realizzerà un filtro selettivo a banda stretta e frequenza di risonanza modificabile tramite gli swithes. Per fare questo si consiglia di predisporre i coefficienti come segue:

B0=1; B1=0; B2=-1; In modo da posizionare i due zeri del filtro rispettivamente in 0 ed in π (ovvero alle frequenze 0 e 24 kHz).

A2 abbia un valore abbastanza prossimo ad 1 per aumentare la selettività stando tuttavia attenti a non scivolare in condizioni di instabilità:

A1 sia di valore negativo e modificabile tramite gli swithes per regolare la frequenza di risonanza attorno ai valori piuttosto bassi della gamma di frequenze del filtro. Infatti, sebbene il filtro lavora nella gamma di frequenze lineare compresa tra 0 e 24KHz, va notato che il sistema uditivo umano presenta una percezione delle frequenze di tipo logaritmico ed inoltre è relativamente poco sensibile a frequenze superiori a 10 kHz.

Ancora qualche considerazione sui coefficienti: Come riportato nell'intestazione del filtro i coefficienti della parte non-ricorsiva sono di tipo "signed" a 8 bit di cui 7 per la parte frazionaria per cui coprono il range [-1, +1], mentre i coefficienti della parte ricorsiva sono "signed" a 8 bit di cui 6 per la parte frazionaria, per cui coprono il range [-2, +2] ecco pertanto che possibili valori devono venir così calcolati:

B0 =1	0,1111111	0x7F
B1=0	0,0000000	0x00
B2=-1	1,0000000	0x80
A1=-1,922	10,000101	0x85
A2=0,9375	00,111100	0xC3

Un esempio di codice C è il seguente, implementabile nel progetto attraverso "Altera Monitor Program"

```
#include <stdio.h>

#define Keys (volatile int *) 0x01111050
#define Switches (volatile long int *) 0x01111040
#define LEDG (int*) 0x01111030
#define LEDR (long int*) 0x01111020
#define AUD_COEFF (long int*) 0x01111010
#define AUD_WE (long int*) 0x01111000
#define DELAY 0

void delay(void);

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

printf("Start \n");

while (1)
{

ta = (a & 0x0000003E);
sa = (a & 0x000003C0)>>6;


dat=0x07F; // gain input
*AUD_COEFF=dat;
delay();*AUD_WE=1;delay();*AUD_WE=0;delay();

dat=0x17f; // b0 (=1)
*AUD_COEFF=dat;
delay();*AUD_WE=1;delay();*AUD_WE=0;delay();
```

```
dat=0x200; // b1 (=0)
*AUD_COEFF=dat;
delay();*AUD_WE=1;delay();*AUD_WE=0;delay();

dat=0x380; // b2 (=-1)
*AUD_COEFF=dat;
delay();*AUD_WE=1;delay();*AUD_WE=0;delay();

dat=0x485; // a1 (freq)  485
*AUD_COEFF=dat+ta;
delay();*AUD_WE=1;delay();*AUD_WE=0;delay();

dat=0x53c; // a2 (selettività) 53C
*AUD_COEFF=dat;
delay();*AUD_WE=1;delay();*AUD_WE=0;delay();

dat=0x600; // shift
*AUD_COEFF=dat+sa;
delay();*AUD_WE=1;delay();*AUD_WE=0;delay();

dat=0x77F; // gain output 77f
*AUD_COEFF=dat;
delay();*AUD_WE=1;delay();*AUD_WE=0;delay();

dat=0x800; // activate
*AUD_COEFF=dat;
delay();*AUD_WE=1;delay();*AUD_WE=0;delay();



a = *Switches;
*LEDR = a;

b = *Keys;
*LEDG = b;
}
}

void delay(void)
{
long int t;

/* modificare a piacere DELAY */
for(t=1; t<DELAY; ++t);
}
```

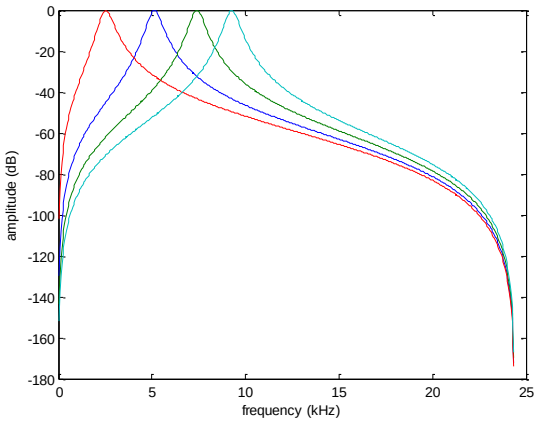
Si noti in particolare:

1- Gli indirizzi dei vari dispositivi devono ovviamente rispecchiare gli indirizzi del sistema sviluppato tramite QSyS.
2- Il sistema filtra solo i campioni del canale destro, mentre fa passare intonsi quelli del canale sinistro.
3- Il NIOS-II lavora a 50MHz, mentre il filtro a 48KHz, inoltre il NIOS-II risulta molto rapido dell'eseguire operazioni semplici, quali quelle di pilotare porte di I/O (tutt'altro discorso per operazioni quali moltiplicazioni, altre operazioni matematiche complesse, o scrittura di stringhe su `stdio`), pertanto è opportuno ritardare opportunamente i segnali che si portano sulla PIO per configurare il filtro e dare tempo al filtro (che funziona con una frequenza di 3 ordini di grandezza inferiore) di leggere tali segnali.
4- L'indirizzo del parametro ed il dato da scrivere sono composti sul medesimo bus, pertanto prendendo ad esempio il valore "`0x53c`" esso va interpretato come (addr=101, data=00111100) ovvero si scrive il valore 0x3C relativo al il parametro A2.

5- I valori ovviamente sono rappresentati in virgola fissa. Inoltre la modifica del parametro A2 risulta molto critica in quanto i poli realizzati sono davvero prossimi al cerchio unitario: un suo aumento porta un aumento della selettività (e può al limite cadere in instabilità) e con esso un consistente aumento del guadagno che può portare a forti distorsioni se non compensato da un debito abbassamento del volume, viceversa una sua riduzione oltre alla minor selettività porta una considerevole diminuzione del volume.
6- Il funzionamento degli swiths è il seguente:
    a. `SW[0]` attiva o disattiva il filtro
    b. `SW[1..5]` modifica la frequenza di risonanza del filtro
    c. `SW[6..9]` controlla il volume in uscita (ma un volume troppo alto provoca forti distorsioni – il segnale viene shiftato al di fuori del range utile)

Per differenziare il funzionamento degli switches si sono impiegate opportune funzioni di mascheratura ta ed sa.

7- La serie di filtri ottenibili è indicativamente riportata nella figura qui sotto



Bibliografia

[1] *Teich, J.* "Hardware/Software Codesign: The Past, the Present, and Predicting the Future" Proceedings of the IEEE, Volume: 100 , Issue: Special Centennial Issue, Digital Object Identifier: 10.1109/JPROC.2011.2182009, Publication Year: 2012 , Page(s): 1411 – 1430

[2] Matlab Corporation: Filter Design HDL Coder http://www.mathworks.com/products/datasheets/pdf/filter-design-hdl-coder.pdf

[3] Matlab Corporation: Signal Processing Toolbox – User Guide http://www.mathworks.com/help/pdf_doc/signal/signal_tb.pdf

[4] Giovanni Ramponi: Teaching slides on IIR digital filters http://www2.units.it/ramponi/teaching/DSP/materiale/Ch6(2).pdf