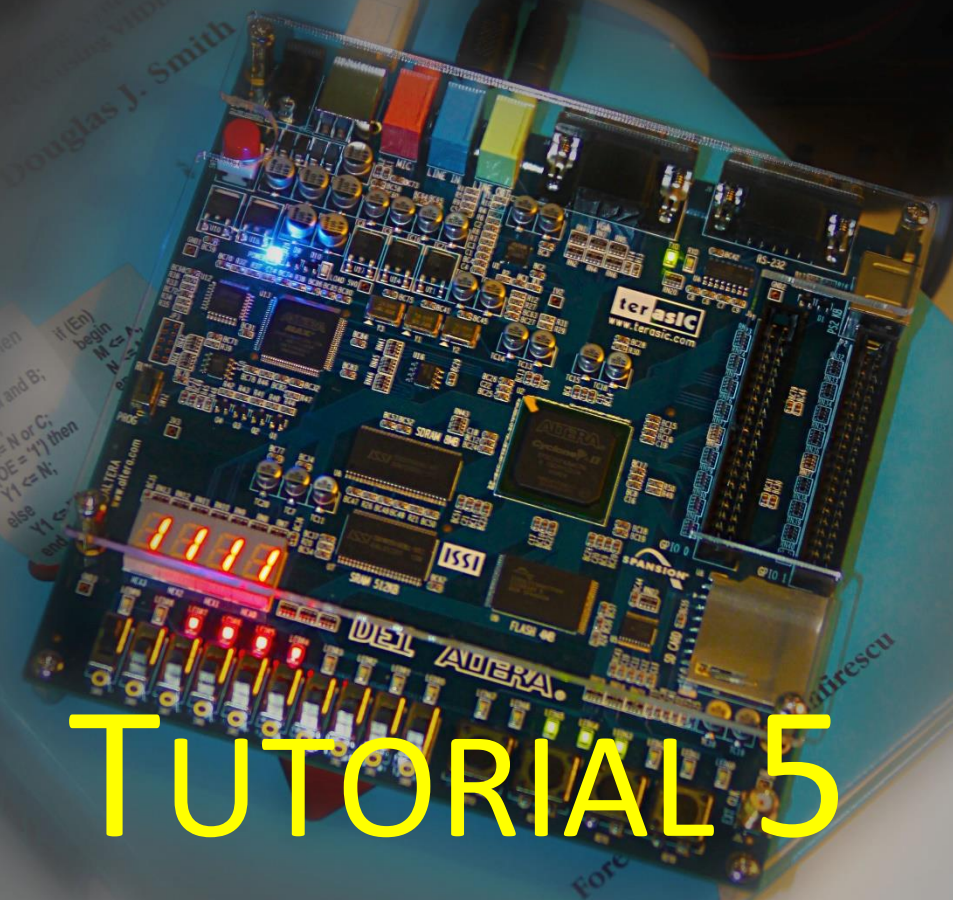
FPGA Tutorial on DE1 Board



TUTORIAL 5

Using Quartus II V13.0

Marsi 2014 - University of TRIESTE

Tutorial FPGA n.5

Realizzazione di un sistema di elaborazione audio su NIOS-2

Descrizione: La DE1, oltre all'FPGA monta un DECODER Audio [1][2] che si può interfacciare al Nios per realizzare un sistema di elaborazione digitale di segnali audio.

Scopo: Realizzazione di una sistema multimediale composto da un microprocessore e di alcune interfacce dedicate di I/O. Analisi di varie metodologie di programmazione

Apprendimenti previsti:

- Sviluppo di un sistema multimediale basato su NIOS II
- Gestione dei segnali audio tramite "polling"
- Colli di bottiglia nell'elaborazione in tempo reale e miglioramento delle prestazioni
- Tool di sviluppo software "Software Build Tool for Eclipse"

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 (Default)
- Nios II, versione (e)
- JTAG UART
- System ID
- SRAM Controller
- SDRAM Controller
- PIO per pilotare i LED verdi
- PIO per pilotare i LED rossi
- PIO di interfaccia verso i gli interruttori a scorrimento
- PIO di interfaccia verso i pulsanti

Si aggiunga inoltre

- External Clocks for DE Board Periherals [3]. Questo blocco integra nel suo interno due PLL che generano gli opportuni clock per alcune le periferiche (SDRAM, VGA, DECODER). Esso NON sostituisce il blocco Clock Source che (a dispetto del suo nome) viceversa si occupa fondamentalmente della gestione del segnale di reset (come visto nel tutorial precedente).

  `University Program > Clock Signals for DE-series Board Periherals`

  - Si configuri perché sia compatibile con la DE1
  - Generi i segnali di clock sia per SDRAM e Audio DECODER
  - Si fissi una frequenza di sistema si 12.288 MHz

Il system clock generato da questo blocco può essere utilizzato per alimentare tutti gli altri blocchi del sistema, mentre gli altri 2 segnali di clock generati vanno portati su collegamenti esterni per poter essere poi connessi all'opportuno dispositivo sulla scheda.

- Blocco di interfaccia verso il decoder [4]. Questo blocco gestisce i segnali in ingresso ed in uscita dal decoder Audio

  `University Program > Audio & Video > Audio`

  - Si configuri perché gestisca tanto i segnali in ingresso che in uscita
  - Si fissi la lunghezza di parola a 32 bit
  - Avalon Type: Memory Mapped

- Blocco di configurazione per il decoder [5]. Questo blocco genera gli opportuni segnali I2C per configurare il decoder Audio

  `University Program > Audio & Video > Audio and Video Config`

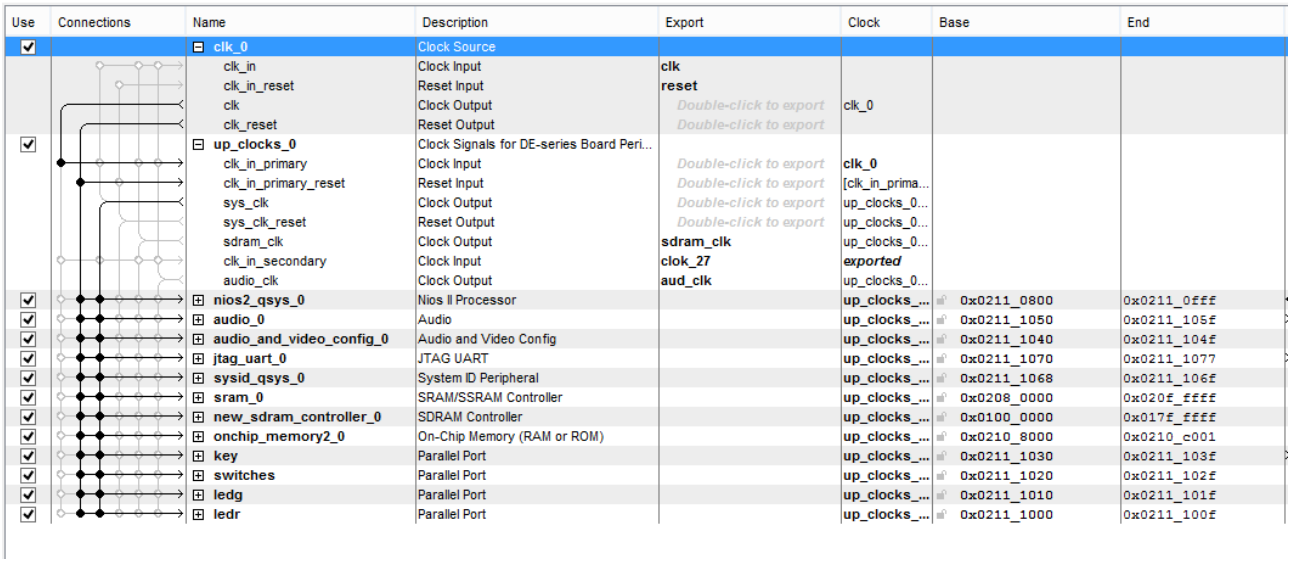
  - Si configuri perché sia compatibile con la DE1
  - Inizializzi automaticamente il dispositivo audio ed adotti i seguenti parametri
    a. Audio in path : Line in to ADC
    b. Enable DAC Audio
    c. Disable Microphone Bypass
    d. Disable Line In Bypass
    e. Format: Left Justified
    f. Bit length: 32 Bit
    g. Sampling Rate: 48 kHz

- Si stabiliscano tutte le opportune connessioni tra gli elementi con alcune attenzioni.
  - Nel blocco "Clock Source" entrino i segnali di clock e di reset esterno e le uscite di questo vengano impiegate come segue:
    - Il clock entra nel blocco "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 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 alimentare tutte le interfaccie con un ingresso Avalon 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 due blocchi: "Clock Source", e dal 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 forniscano dei nomi mnemonici ai vari blocchi.
Si assegnino automaticamente gli indirizzi di memoria ai veri blocchi.
Si assegni automaticamente il livello di interrupt ai vari blocchi.

Alla fine l'architettura del sistema dovrebbe avere più o meno questo aspetto.



Use	Connections	Name	Description	Export	Clock	Base	End
☑		⊟ clk_0	Clock Source				
		clk_in	Clock Input	clk			
		clk_in_reset	Reset Input	reset			
		clk	Clock Output	Double-click to export	clk_0		
		clk_reset	Reset Output	Double-click to export			
☑		⊟ up_clocks_0	Clock Signals for DE-series Board Peri...				
		clk_in_primary	Clock Input	Double-click to export	clk_0		
		clk_in_primary_reset	Reset Input	Double-click to export	[clk_in_prima...		
		sys_clk	Clock Output	Double-click to export	up_clocks_0...		
		sys_clk_reset	Reset Output	Double-click to export	up_clocks_0...		
		sdram_clk	Clock Output	sdram_clk	up_clocks_0...		
		clk_in_secondary	Clock Input	clok_27	*exported*		
		audio_clk	Clock Output	aud_clk	up_clocks_0...		
☑		⊞ nios2_qsys_0	Nios II Processor		up_clocks_...	0x0211_0800	0x0211_0fff
☑		⊞ audio_0	Audio		up_clocks_...	0x0211_1050	0x0211_105f
☑		⊞ audio_and_video_config_0	Audio and Video Config		up_clocks_...	0x0211_1040	0x0211_104f
☑		⊞ jtag_uart_0	JTAG UART		up_clocks_...	0x0211_1070	0x0211_1077
☑		⊞ sysid_qsys_0	System ID Peripheral		up_clocks_...	0x0211_1068	0x0211_106f
☑		⊞ sram_0	SRAM/SSRAM Controller		up_clocks_...	0x0208_0000	0x020f_ffff
☑		⊞ new_sdram_controller_0	SDRAM Controller		up_clocks_...	0x0100_0000	0x017f_ffff
☑		⊞ onchip_memory2_0	On-Chip Memory (RAM or ROM)		up_clocks_...	0x0210_8000	0x0210_c001
☑		⊞ key	Parallel Port		up_clocks_...	0x0211_1030	0x0211_103f
☑		⊞ switches	Parallel Port		up_clocks_...	0x0211_1020	0x0211_102f
☑		⊞ ledg	Parallel Port		up_clocks_...	0x0211_1010	0x0211_101f
☑		⊞ ledr	Parallel Port		up_clocks_...	0x0211_1000	0x0211_100f

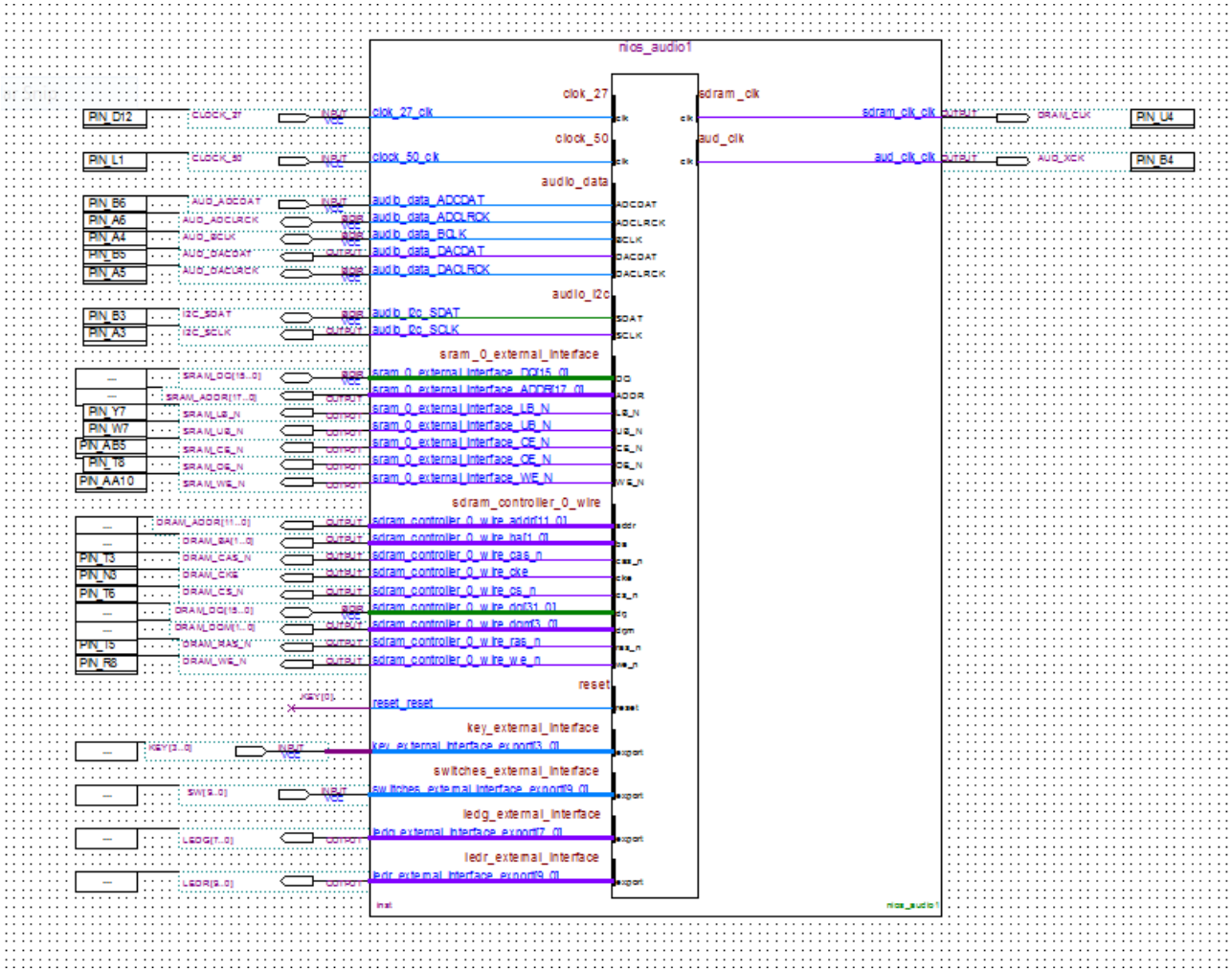
Si prenda nota dei vari indirizzi dei diversi blocchi e si generi (GENERATE) il sistema.

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.

Il blocco contenente le PLL di sistema avrà 2 clock in ingresso rispettivamente a 27MHz ed a 50MHz, che vanno collegati con gli opportuni oscillatori presenti sulla scheda, e due clock in uscita: uno per la SDRAM, uno per il DECODER mentre il terzo, usato per il processore è stato collegato internamente.
Si presti particolare attenzione ai versi dei segnali, si noti che i segnali dati di SRAM e SDRAM devono essere configurati come bidirezionali inoltre per quanto concerne il blocco audio: ADCDAT sarà un ingresso, mentre DACDAT sarà un'uscita, così come i segnali AUD_BCLK, AUD_DACLRCK, AUD_ADCLRCK e bene siano bidirezionali. Infatti all'interno del modulo che configura il decoder, non vi è la possibilità di un settaggio esplicito che lo configuri come Master oppure come Slave, pertanto la soluzione più opportuna è quella di tenere questi tre segnali come bidirezionali. Inoltre è opportuno tenere anche il segnale I2C_SDAT come bidirezionale.

Sebbene i quattro pulsanti (KEY) vengono visti come periferica di ingresso da parte del processore, è pur sempre comodo avere un tasto di reset, pertanto si deriva il tasto KEY[0] a funzionare anche da reset del sistema. Esso risulterà pertanto inutilizzabile in questa configurazione come ingresso generico.
Si importi un opportuno file di vincoli e si utilizzi l'opzione "`show location assignment`" per assicurarsi che tutti i piedini siano stati opportunamente vincolati.

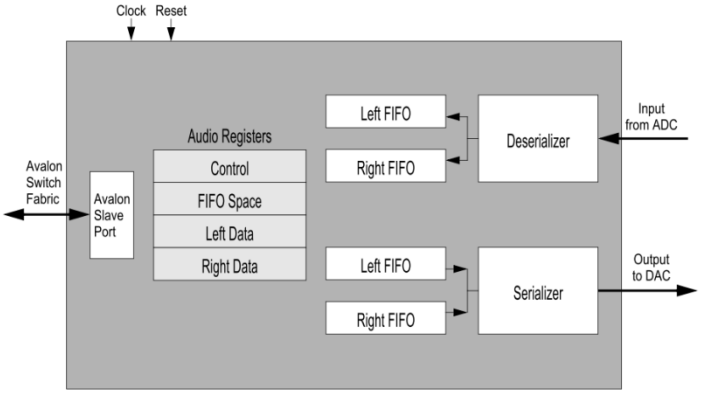
Il sistema completo di tutti i piedini di I/O dovrebbe più o meno assomigliare a questo.



Si salvi lo schematico, lo si configuri come “Top Level Entity”, si aggiunga al progetto il file .qip relativo al sistema sviluppato tramite Qsys, si compili l’intero progetto e si programmi opportunamente la scheda col risultato finale.

L’interfaccia di I/O del decoder AUDIO.

Il blocco che si occupa dell’interfacciamento [4] dei segnali audio ha una struttura come quella semplificata in figura: esso si compone di 2 FIFO in ingresso da 128 parole da 32bits e di due analoghe FIFO in uscita.



Inoltre vi sono 4 registri da 32 bits ciascuno per il controllo del sistema e dei dati: nel primo vi sono diversi bit di controllo, il secondo mantiene informazione sullo stato di riempimento delle FIFO in ingresso ed in uscita, mentre gli ultimi due registri quando usati in lettura forniscono i dati (rispettivamente del canale sinistro e destro) letti dal convertitore ADC, mentre se usati in scrittura forniscono i dati per il DAC.

Table 1. Audio Core register map

Offset in bytes	Register Name	R/W	Bit Description									
			31…24	23…16	15…10	9	8	7…4	3	2	1	0
0	control	RW	*(1)*			WI	RI	*(1)*	CW	CR	WE	RE
4	fifospace	R	WS LC	WS RC	RA LC			RA RC				
8	leftdata	RW *(2)*	Left Data									
12	rightdata	RW *(2)*	Right Data									

Codice per il NIOS – Stereo Spaziale

Spesso per enfatizzare l'effetto stereo si utilizza il semplice accorgimento di aumentare la separazione dei canali sommando a ciascun canale una porzione dell'altro in controfase, ovvero sottraendo a ciascun canale una porzione dell'altro `(R←R-n*L , L←L-n*R)`, si voglia scrivere un programma C che realizzi tale effetto:

Utilizzando l'Altera Monitor Program (come da tutorial precedente) si può scrivere il seguente programma (apportando gli opportuni cambiamenti che rispecchino le locazioni di memoria del sistema sviluppato tramite QSyS) che poi si andrà a compilare e a caricare nella memoria del NIOSII

```
#include "stdio.h"
#define BUF_THRESHOLD 96

int main(void)
{       volatile int * red_LED_ptr = (int *) 0x01101060;          // red LED address
        volatile int * green_LED_ptr = (int *) 0x01101050;        // green LED address
        volatile int * audio_ptr = (int *) 0x01101020;            // audio port address
                 int * sw_ptr = (int *) 0x01101040;               // sw address

        int fifospace;
        int left_data, right_data;
        char n;


        while(1)
        {
                fifospace = *(audio_ptr + 1);            // read the audio port fifospace register
                        if ( (fifospace & 0x000000FF) > BUF_THRESHOLD ) // check RARC
                        {
                                // processing data until the the audio-in FIFO is empty
                                while (fifospace & 0x000000FF)
                                {
                                        left_data = *(audio_ptr + 2);
                                        right_data = *(audio_ptr + 3);
                                        n=(char) *(sw_ptr);

                                        left_data=left_data-(right_data>>n);
                                        right_data=right_data-(left_data>>n);

                                        *(audio_ptr + 2) = left_data;
                                        *(audio_ptr + 3) = right_data;

                                        fifospace = *(audio_ptr + 1);

                                *(green_LED_ptr) = (fifospace & 0x00FF0000);
                                *(red_LED_ptr) = (fifospace & 0x000000FF);
                                }
                        }


        }
}
```

Il funzionamento è sinteticamente il seguente: durante un loop infinito (`while(1))` si controlla l'occupazione della FIFO in ingresso. Quado questa supera una certa soglia si entra in un secondo ciclo atto a vuotarla completamente mentre vengono eseguite le operazioni richieste. Quando la FIFO è vuota si sospendono le operazioni fintanto che non si torna a superare la soglia predeterminata. Ovviamente la velocità di svuotamento deve essere superiore a quella di riempimento affinchè il procedimento funzioni.

Si noti inoltre che sui LED vengono visualizzate per controllo le occupazioni delle FIFO rispettivamente in ingresso (Rossi) ed in Uscita (Verdi) , mentre gli swithes sono usati per dosare l'effetto.

Eventualmente si può provare a spostare l'istruzione di lettura della FIFO e di visualizzazione della medesima in diversi punti del codice per coglierne il funzionamento sopra esposto.

VU-meter

Per realizzare un semplice vu-meter sui led si deve ricordare che i segnali inerenti ai campioni sono sia positivi che negativi e rappresentati in complemento a due. Pertanto se si prova a visualizzare direttamente il valore assoluto di questi si noterà che i led relativi ai bit meno significativi risulteranno mediamente tanto accesi quanto spenti, mentre i bit più significativi saranno spenti se il segnale non raggiunge una certa ampiezza.

Pertanto un modo piuttosto semplice potrebbe essere sostituire alle istruzioni di accensione dei led le seguenti:

```
*(green_LED_ptr) = abs(left_data>>24);
*(red_LED_ptr)   = abs(right_data>>24);
```

Ove l'operazione di shift (>>) serve per visualizzare solo gli 8 bit più significativi della parola (a 32 bit).

Se si volesse avere una visualizzazione più aderente a quella che è la percezione umana dell'intensità si dovrebbe ricorrere ad un operatore non lineare (logaritmo o radice) modificando ad esempio il codice in

```
*(green_LED_ptr) = pow(abs(left_data>>24),.5);
*(red_LED_ptr)   = pow(abs(right_data>>24),.5);
```

Ricordandosi di aggiungere al sorgente, prima di compilarlo l'header

```
#include <math.h>
```

Oltre alla flag "`-lm`" tra le opzioni per il linker in fase di compilazione che sono accessibili in Altera Monitor Program

```
Setting > program Settings > Additional Linker Flags
```

Però così facendo ci si accorge che il suono risulta distorto. Infatti, ripristinando la visualizzazione di riempimento della FIFO in ingresso sui Led Rossi ci si accorge che questa è satura. Infatti le elaborazioni matematiche richieste richiedono un certo tempo e rallentano il processo a tal punto che il processore non riesce più ad elaborare il flusso dei dati in real-time.

Una soluzione molto semplice per ovviare a ciò potrebbe essere quello di portare le istruzioni inerenti ai vu-meters al di fuori del ciclo di elaborazione dei campioni (un attimo prima dello svuotamento della fifo), facendo sì che queste in pratica vengano raggiunte solo da un insieme sottocampionato (del fattore BUF_THRESHOLD) dei campioni in ingresso, ma mantenendo sempre una frequenza di aggiornamento ben superiore a quella percepibile dall'occhio umano.

Uso del moltiplicatore

Si noti altresì che nel precedente codice si è utilizzato l'operatore shift (>>) per pesare opportunamente i campioni del canale in controfase. Si provi ad impiegare ora l'operatore moltiplicazione (*) sostituendo le rispettive righe ad esempio con:

```
left_data=left_data-(right_data*n>>8);
right_data=right_data-(left_data*n>>8);
```

Si potrà notare il sistema smette di funzionare regolarmente. Ciò accade perché il sistema non riesce più ad elaborare in tempo correttamente il flusso di campioni. L'operatore moltiplicazione è infatti un'operazione piuttosto lenta (specie per il processore NIOS-II (e)) che ritarda il ciclo di scrittura dei campioni. Vi è inoltre da sottolineare che vi sono talune operazioni (quali l'operazione di moltiplicazione o Prinf possono mettere in crisi la versione (e) del Nios e portando ad un continuo "reset" del sistema)

Per ovviare a questo inconveniente vi sono tre strade:

- Impiegare un processore dalle prestazioni più spinte (NIOS-II(s) o NIOS-II(f)). Svantaggi in questo senso si hanno sia in termini di occupazione di risorse, sia perché l'uso di questi processori viene consentito dietro acquisto della rispettiva licenza, in assenza della quale il loro funzionamento è garantito a tempo indeterminato solo fintanto che viene mantenuto il collegamento via USB col PC, viceversa esso dura solo un tempo limitato quando questo collegamento venga staccato. Ove si volesse impiegare tali processori e gestire il sistema con Altera Monitor Program, si deve seguire la seguente strada:
    a. Compilare il progetto intero con Quartus
    b. Programmare la DE1 tramite il tool "Programmer" incluso in Quartus
    c. Il sistema avviserà della presenza di blocchi per i quali è richiesta la licenza
    d. Dopo il download comparirà una finestrella che consente di interrompere il collegamento USB. (E CHE TALVOLTA IMPEDISCE DI MINIMIZZARE A ICONA QUARTUS)
    e. A questo punto ci sono due possibilità
        i. o (passando per il desktop) si apre L'altera Monitor Program e si prosegue
        ii. Oppure si stacchi il cavo USB, si chiuda la finestrella di controllo (ed alla bisogna anche "Quartus"), si riconnetta il cavo USB. Ora il sistema funziona solo per un tempo limitato. (1 ora c.a) – evidentemente per staccare il cavo USB continuando a mantenere in funzione il sistema è essenziale alimentarlo attraverso l'alimentatore esterno in dotazione.
- Alternativamente si può impiegare una frequenza di campionamento del segnale audio inferiore a 48KHz. Si ricordi che vi è un registro apposito nel CODEC audio (SR) , controllato tramite il blocco "audio-controller" (Sampling Rate) che consente di predisporre a piacimento (secondo alcune preconfigurazioni) sia la frequenza di campionamento in ingresso che in uscita.

- Come ulteriore alternativa si potrebbe scrivere il codice direttamente in assembler ed ottimizzarlo i n modo tale da ridurre al minimo il suo tempo di esecuzione.

Codice per il NIOS – Filtro IIR

Si voglia ora realizzare un semplice filtro IIR di secondo ordine di tipo passa-banda con 2 zeri e 2 poli.
La sua equazione temporale sarà descritta da:

$$y_n = K(a_o * x_n + a_1 * x_{n-1} + a_2 * x_{n-2} - b_1 * y_{n-1} - b_2 * y_{n-2})$$

Ove "a_i e b_i" sono i coefficienti del filtro, x_i sono i campioni in ingresso ed y_i quelli in uscita e K serve a regolare opportunamente il guadagno.
Dato l'ordine piuttosto basso, per ottenere un effetto abbastanza evidente si suggerisce di realizzare un filtro abbastanza selettivo, per fare questo conviene scegliere opportunamente i valori:

- a_0=1, a_1=0, a_2=-1, [1 0 -1] in modo da garantire la presenza di due zeri rispettivamente alla frequenza 0 ed alla frequenza di Nyquist (1/2 della frequenza di campionamento)
- $b_2 < 1$; questo parametro regola la selettività del filtro: tanto più è prossimo a 1 tanto più stretta sarà la banda passante
- |b1| che deve rispettare il vincolo < 2 sqrt(b_2) regola la frequenza centrale di risonanza del filtro
- K deve essere regolato onde evitare che vi possa essere una saturazione sui campioni in uscita. Indicativamente si potrebbe regolare in modo da garantire il guadagno alla frequenza centrale prossimo a 1. Una scelta approssimata potrebbe essere K=1-b_2

Per eseguire questa operazione si possono seguire due strade: operare tramite codifica in virgola fissa oppure tramite codifica in virgola mobile.

Operazioni in FIXED Point

Tutti i parametri del filtro ed i campioni del segnale sono codificati come interi, l'operazione di moltiplicazione avviene tramite numeri interi (anziché tra float), e pertanto risulta più veloce per il processore. Ovviamente per evitare la saturazione del risultato si devono imporre opportuni scalaggi. Poiché i campioni in ingresso sono codificati a 32 bit si consiglia di adottare solo i 16 bit più significativi (ovvero di scalare i valori di 16 bit verso destra) e contemporaneamente di "moltiplicare" i coefficienti del filtro per 2^16 = 65.536.

Un eventuale codice C per implementare il filtro proposto potrebbe essere il seguente.

```
#include "stdio.h"
#define BUF_THRESHOLD 8

int main(void)
{

          volatile int * red_LED_ptr = (int *) 0x01101060; // red LED address
          volatile int * green_LED_ptr = (int *) 0x01101050;         // green LED address
          volatile int * audio_ptr = (int *) 0x01101020;             // audio port address

          int fifospace, left_data, right_data, mono_data;
          int x[3], y[3];
          int gain,b1,b2;


gain=3300; // 0.05
b2=65259;  // 0.95
b1=-124000; // -1.9


          while(1)
          {
                    fifospace = *(audio_ptr + 1);                           // read the audio port fifospace register
                              if ( (fifospace & 0x000000FF) > BUF_THRESHOLD ) // check RARC
                              {
                                        // store data until the the audio-in FIFO is empty
                                        while (fifospace & 0x000000FF)
                                        {
```

```
                                        left_data = *(audio_ptr + 2);
                                        right_data = *(audio_ptr + 3);

                                        mono_data = (left_data+right_data)>>2;

                                        x[2]=x[1];x[1]=x[0];
                                        x[0]= mono_data;

                                        y[2]=y[1];y[1]=y[0];
                                        y[0]=(gain*((x[0]-x[2])>>16))-(b1*(y[1]>>16))-(b2*(y[2]>>16));

                                        mono_data= y[0];

                                        *(audio_ptr + 2) = mono_data;
                                        *(audio_ptr + 3) = mono_data;

                                        fifospace = *(audio_ptr + 1); // read the audio port fifospace register

                                    *(green_LED_ptr) = (fifospace & 0x00FF0000);
                                    *(red_LED_ptr) = (fifospace & 0x000000FF);
                                    }
                                }

        }
}
```

Si noti in particolare le due linee che mantengono memoria dei campioni precedenti sia in ingresso che in uscita, e che lo stato di occupazione delle FIFO viene mappato sui LED. Si noti altresì che l'elaborazione avviene su di un segnale monofonico ottenuto miscelando opportunamente i due canali L e R stereo.
Utilizzando Altera Monitor Program e si compili e si implementi sul NIOS. Si noterà subito che l'utilizzo dell'operazione di moltiplicazione all'interno del processore Nios II(e) crea notevoli problemi. Per ovviare all'inconveniente si deve riconfigurare il sistema Hardware ed utilizzare un Processore più performante, ad esempio il Nios II (s) che presenta il notevole vantaggio di avere il moltiplicatore realizzato in Hardware, e pertanto l'operazione di moltiplicazione, collo di bottiglia del presente programma dovrebbe occupare molte meno risorse in termini temporali, durante l'esecuzione.

- Si riconfiguri l'architettura HW utilizzando il processore Nios II (s)
- Si ricompili il sistema
- Si operi il download su scheda seguendo il procedimento indicato a Pag.5
- Utilizzando Altera Monitor Program (che richiede un ri-settaggio dei parametri) si implementi il sorgente C sul sistema realizzato.

Qualche nota sui risultati conseguiti:

- Il filtro è stato reso molto selettivo e centrato sui 2KHz per evidenziarne il funzionamento, peraltro in queste condizioni anche piccoli ritocchi dei parametri risultano molto delicati e possono portare il sistema facilmente in instabilità
- Il filtro è molto selettivo, pertanto solo poca energia potrà raggiungere l'uscita. Il risultato sarà un volume d'uscita piuttosto basso. Si può ovviare ritoccando il guadagno (come peraltro si è fatto), però quando in ingresso è presente la frequenza di risonanza si otterrà una saturazione dei segnali con conseguente distorsione
- Il filtro può essere centrato per frequenze variabili tra 0 e 24KHz – estremi esclusi, d'altro canto merita ricordare che la percezione umana delle frequenze e delle ampiezze avviene su base logaritmica, mentre le regolazioni sono lineari, questo fa sì ad esempio che piccole regolazioni per il sistema centrato su basse frequenze comportino ampie variazioni nella percezione dei risultati, viceversa alle alte frequenza, ampie variazioni dei parametri si riflettono poco sul risultato percepito.

Operazioni in FLOATING Point

Una soluzione indubbiamente più comoda per i programmatori potrebbe essere quella di far ricorso ad un'aritmetica a virgola mobile (Floating point).

Il sorgente C precedente potrebbe ad esempio prevedere le seguenti modifiche:

```
float x[3], y[3];
float gain,b1,b2;

gain=0.05;
b2=-.98;
b1=1.2;
. . .

x[2]=x[1];x[1]=x[0];
x[0]= (float) mono_data;

y[2]=y[1];y[1]=y[0];
y[0]=(gain*(x[0]+x[2]))+(b1*y[1])+(b2*y[2]);

mono_data= (int) y[0];
```

Si potrà notare che se vi segue questa strada nuovamente il sistema, anche se dotato del processore Nios II-(s) non riuscirà ad eseguire l'elaborazione in tempo reale e la FIFO d'ingresso risulterà satura. Per aggirare l'inconveniente si potrebbe ad esempio scegliere una frequenza di campionamento del segnale audio inferiore, ma questo comporterebbe una limitazione alle prestazioni del sistema peraltro non legata a reali necessità, infatti, sebbene "comodo" dal punto di vista del programmatore, il ricorso all'aritmetica floating point in questo caso sarebbe, come dimostrato, ridondante.

Ambiente di sviluppo Eclipse

Un altro ambiente di sviluppo, più sersatile, che può essere di interesse per chi è abituato a sviluppare software è il tool: "NIOS II – Software Build Tool" sviluppato su piattaforma "Eclipse" [6][7][8].

Questo ambiente è più versatile e completo di quello messo a disposizione da "Altera Monitor Program" ma per contro è più complesso, pesante e talvolta può presentare curiosi malfunzionamenti.

Il sistema ha il pregio di mettere a disposizione dello sviluppatore di software alcune funzioni (HAL) predefinite da utilizzare con le varie periferiche, che se da un lato possono semplificare in parte lo sviluppo del software, per contro, a volte appesantiscono il codice al punto da comprometterne le prestazioni.

Per sviluppare ad esempio il filtro IIR realizzato poc'anzi in ambiente "Altera Monitor Program" si prosegua in tal modo:

- Si programmi innanzitutto la scheda DE1 col file .sof corrispondente al progetto
- Si apra l'ambiente di sviluppo (ciò può essere fatto direttamente da Quartus

  `Tools > NIOS II – Software Build Tool for Eclipse`

  (l'ambiente è basato di diversi workspaces) che mantengono memoria dei vari progetti sviluppati

  `File > New > Nios II Application and BSP from template`

- Si selezioni opportunamente il file `.sopcinfo` che contiene le informazioni HW del nostro sistema (generato dal Qsys)
- Si assegni un nome al progetto
- Si scelga un modello di file C al quale ispirarsi per eventuali modifiche (es. Hello World Small)
- Finish

Dopo un certo periodo si generano due cartelle <nome_progetto> e <nome_progetto_bsp>

La prima di esse contiene il file Hello world small.c Utile come punto di partenza.

Nella seconda è invece utile verificare la creazione di alcuni file .h e .c che verranno inclusi nel nostro proetto e che contengono alcune strutture e funzioni predefinite; alcuni esempi:

In `drivers > inc`

- Si trova il file `altera_up_avalon_audio.h` che al suo interno contiene le definizioni di alcune funzioni utili per la gestione della periferica audio.
- ancora vi si trova il file `altera_up_avalon_parallel_port_regs.h` che contiene alcune funzioni utili nella gestione delle periferiche di I/O specifiche per le schede DEx

Inoltre in `system.h` si trovano molti "`#define`" utili per sostituire indirizzi con nomi mnemonici di maggior comprensione.

Si può a questo punto editare il file Hello world Small.c e sostituirlo ad esempio con qualcosa di simile:

```
#define BUFFER_LENGHT 16

#include "sys/alt_stdio.h"
#include "altera_up_avalon_audio.h"
#include "altera_up_avalon_audio_regs.h"
#include "altera_up_avalon_parallel_port.h"
#include "altera_up_avalon_parallel_port_regs.h"


int main(void)
{
  alt_putstr("Hello from Nios II!   audio TEST\n");

  alt_up_audio_dev * audio_dev;
  int l_buf,r_buf,mono_data;
  int fifospace;

  int sw;

  int x[3], y[3];
  int gain,b1,b2;


  gain=3300; // 0.05
  b2=65259; // 0.999
  b1=-124000;

  // open the Audio port
  audio_dev = alt_up_audio_open_dev ("/dev/audio_0");

  if ( audio_dev == NULL)
        alt_printf ("Error: could not open audio device \n");
  else
        alt_printf ("Opened audio device \n");

  /* read and filter audio data */
  while(1)
  {
```

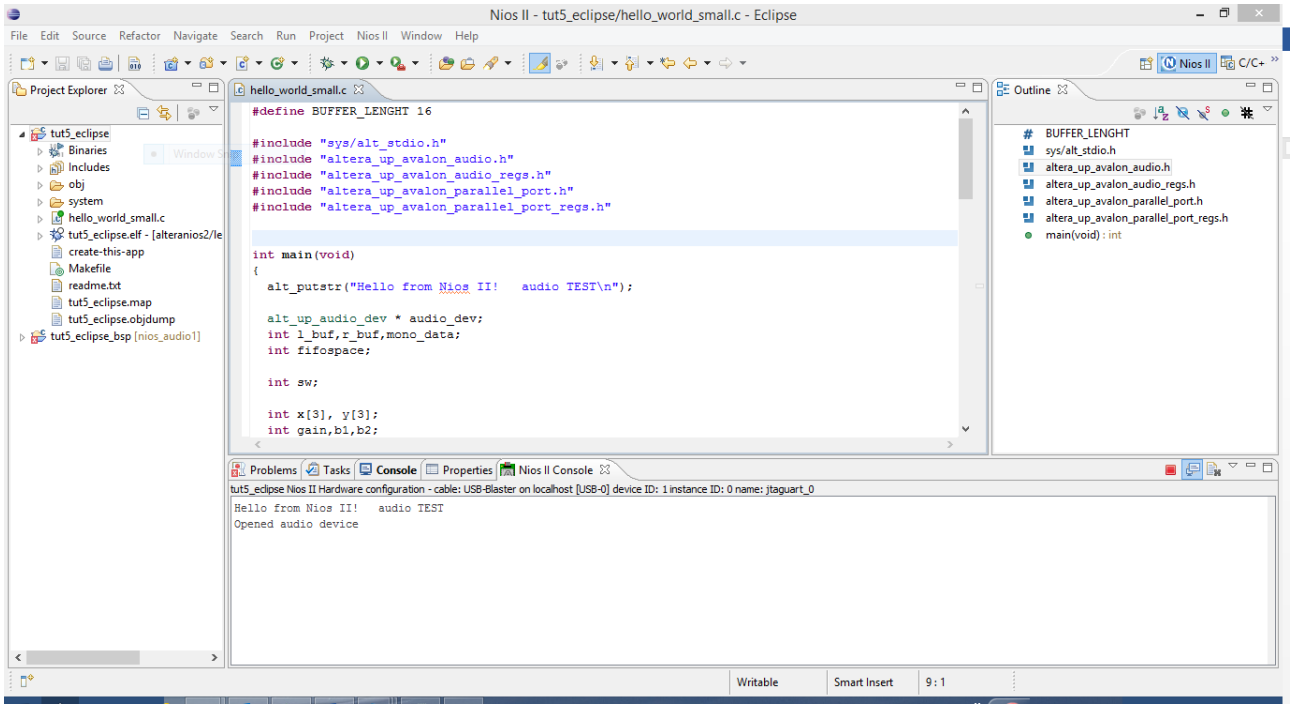
```
  fifospace = alt_up_audio_read_fifo_avail (audio_dev, ALT_UP_AUDIO_RIGHT);
  if (fifospace > BUFFER_LENGHT ) // check RARC
                {
                        // demo swithes and leds
                        sw=IORD_ALT_UP_PARALLEL_PORT_DATA(SWITCHES_BASE);
                        IOWR_ALT_UP_PARALLEL_PORT_DATA(LEDG_BASE, sw);

                        // selectivity tuned by switches
                        b2=65536-sw;

                        // store data until the the audio-in FIFO is empty
                        while (fifospace)
                        {
                        // read audio buffer
                        alt_up_audio_read_fifo (audio_dev, &(r_buf), 1, ALT_UP_AUDIO_RIGHT);
                        alt_up_audio_read_fifo (audio_dev, &(l_buf), 1, ALT_UP_AUDIO_LEFT);

                        mono_data = (l_buf+r_buf)>>2;

                        x[2]=x[1];x[1]=x[0];
                        x[0]= mono_data;

                        y[2]=y[1];y[1]=y[0];
                        y[0]=(gain*((x[0]-x[2])>>16))-(b1*(y[1]>>16))-(b2*(y[2]>>16));

                        mono_data= y[0];

                        // write audio buffer
                        alt_up_audio_write_fifo (audio_dev, &(mono_data), 1, ALT_UP_AUDIO_RIGHT);
                        alt_up_audio_write_fifo (audio_dev, &(mono_data), 1, ALT_UP_AUDIO_LEFT);

                        fifospace = alt_up_audio_read_fifo_avail (audio_dev, ALT_UP_AUDIO_RIGHT);
                        IOWR_ALT_UP_PARALLEL_PORT_DATA(LEDR_BASE, fifospace);
                        }
                }
}
  return 0;
}
```

Ricordarsi di includere i `file.h` utili al nostro progetto e si noti l'uso di funzioni predefinite per le operazioni di I/O verso le periferiche. Si noti anche che il nome della perifierica (in questo caso `/dev/audio_0` deve rispecchiare fedelmente il nome dato alla stessa all'interno del sistema QSys)

A questo punto si può compilare il sorgente ed effettuare il download nella memoria del processore.

- Si evidenzi (`click`) la prima delle due cartelle (<nome_progetto>)
- `Run > Run As > Nios II Hardware`

Dopo aver chiesto se salvare eventuali modifiche al file sorgente il sistema effettua la compilazione, seguita dal download nella memoria del processore e fa partire il medesimo.



Nella Finestra di Console di sistema appariranno le informazioni inerenti alle operazioni di download, mentre nella finestra di console del Nios Vi saranno i messaggi che il programma ha inviato su STDIO.

Nota su "ID mismatching":

Il sistema basato su Eclipse, onde garantire che il software sviluppato su di un sistema non venga erroneamente caricato su di un sistema diverso che potrebbe non avere le caratteristiche HW corrette, effettua un controllo sull'identificativo (sysid) del sistema.
Ciò può creare abbastanza problemi

1. Ad esempio se si aggiorna l'hardware del processore, un eventuale software, creato precedentemente non può essere fatto girare, a meno di non rigenerare il progetto.
2. L'analisi del sysid viene fatta via software sulla periferica denominata /dev/sysid_qsys, per cui bisogna garantire che all'interno del sistema sviluppato tramite QSys questa periferica sia presente ed abbia **ESATTAMENTE** questo nome.
3. Una valida alternativa può essere quella di disattivare tale "facility"

   `Run > Run Configuration`

   E nella sezione "Target Connection" attivare "`Ignore mismatched system ID`"

Bibliografia

[1] Wolfson Microelectronics: **WM8731 / WM8731L**
"**Portable Internet Audio CODEC with Headphone Driver and Programmable Sample Rates**"
http://www.cs.columbia.edu/~sedwards/classes/2008/4840/Wolfson-WM8731-audio-CODEC.pdf

[2] [Wolfson Microelectronics: "**Simplified Datasheet for WM8731 Audio Codec**"
http://diglab.ece.gatech.edu/downloads/project/WM8731_SimplifiedDatasheet.pdf

[3] 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

[4] Altera Corporation: Audio Core for Altera DE-Series Boards
ftp://ftp.altera.com/up/pub/Altera_Material/13.0/University_Program_IP_Cores/Audio_Video/Audio.pdf

[5] Altera Corporation: Audio/Video Configuration Core for DE-Series Boards
ftp://ftp.altera.com/up/pub/Altera_Material/13.0/University_Program_IP_Cores/Audio_Video/Audio_and_Video_Config.pdf

[6] Altera Corporation: Getting Started with the Graphical User Interface
http://www.altera.com/literature/hb/nios2/n2sw_nii52017.pdf

[7] Altera Corporation: Getting Started from the Command Line
http://www.altera.com/literature/hb/nios2/n2sw_nii52014.pdf

[8] Altera Corporation: Nios II Software Build Tools
http://www.altera.com/literature/hb/nios2/n2sw_nii52015.pdf