Tutorial 5

Realizzazione di un sistema di elaborazione audio su NIOS-2

Descrizione: La DE1, oltre all'FPGA monta un DECODER Audio 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 l'SOPC Builder realizzare (come da precedente tutorial) un'architettura composta dai seguenti elementi:

- 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 questo in particolare sia configurato
    - Perché sia sincronizzato sul fronte di discesa del segnale
    - Perché generi un interrupt sincronizzato sul "fronte del segnale"

Si aggiunga inoltre

- External Clocks for DE Board Periherals . Questo blocco integra nel suo interno delle PLL che generano gli opportuni clock per alcune le periferiche (SDRAM, VGA, DECODER)

  `University Program > External Clocks for DE 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

- Blocco di interfaccia verso il decoder. 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

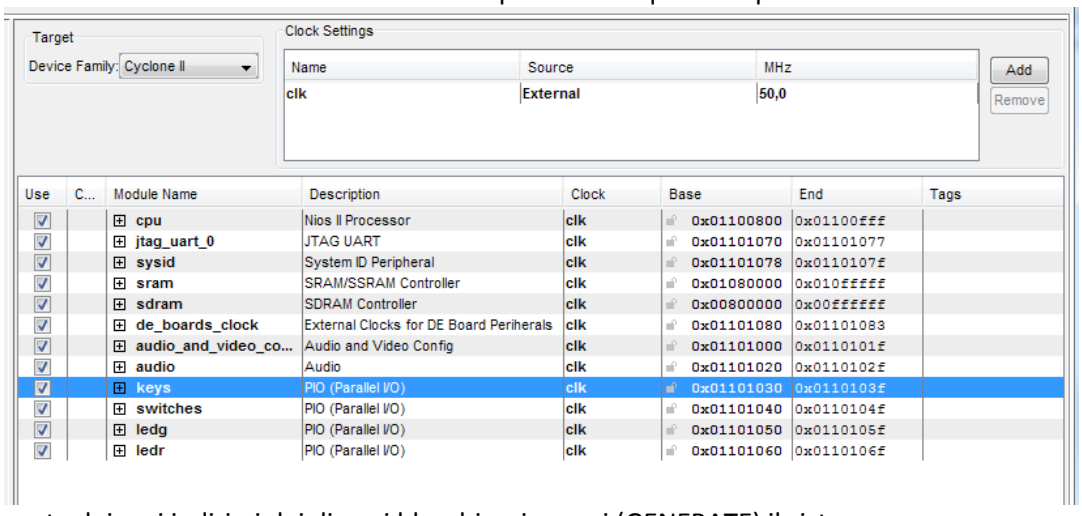
- Blocco di configurazione per il decoder. Questo blocco genera gli opportuni segnali I2C per configurare il decoder Audio

  `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. Base Over / Samplig Rate : 250fs/256fs
    h. Sampling Rate: 0

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.



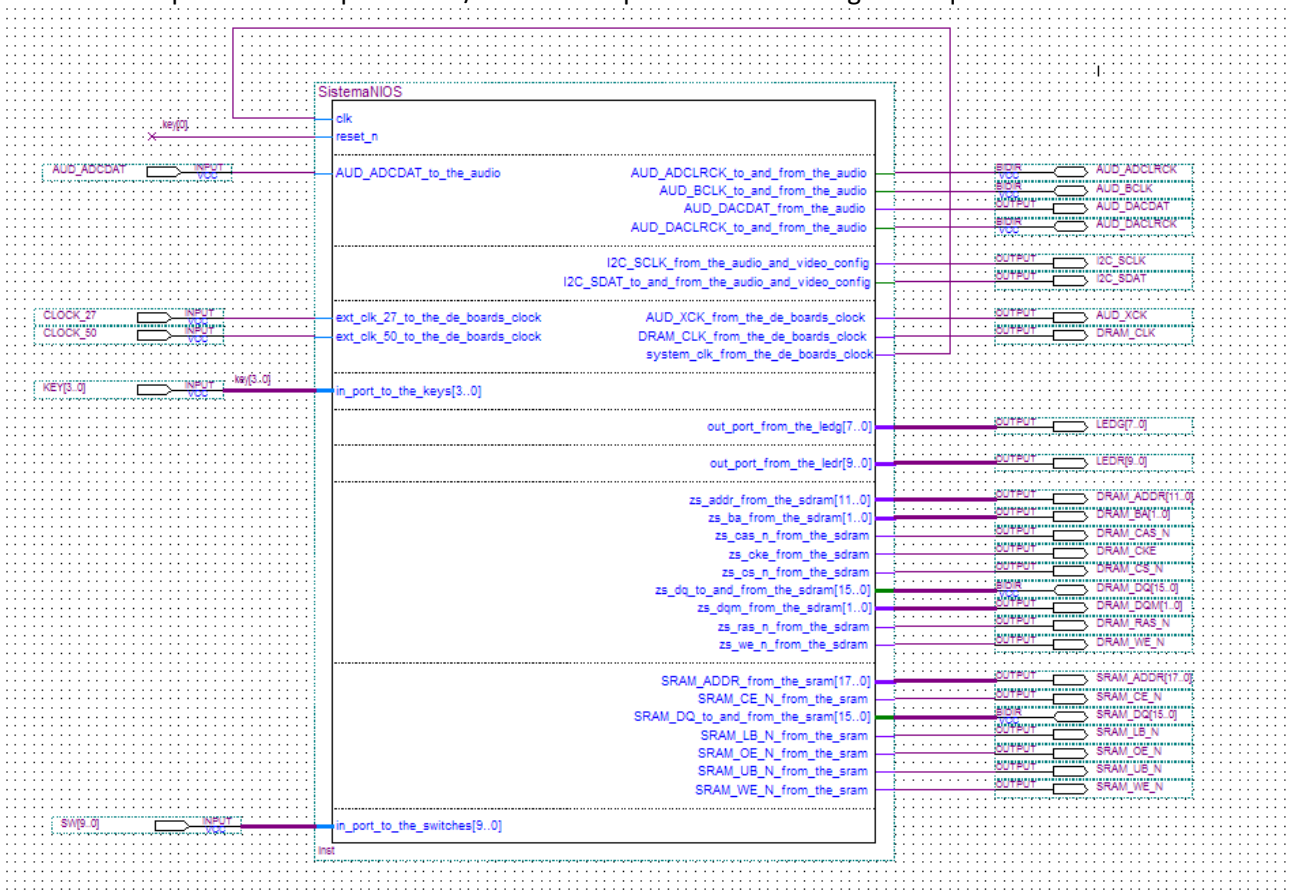
Si prenda nota dei vari indirizzi dei diversi blocchi e si generi (GENERATE) il sistema.

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 tre clock in uscita: uno per la SDRAM, uno per il DECODER ed il terzo da usarsi come clock per il Processore.
Si noti bene che i segnali dati di SRAM e SDRAM devono essere configurati come bidirezionali, così come pure i segnali AUD_BCLK, AUD_DACLRCK, AUD_ADCLRCK. 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.

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.

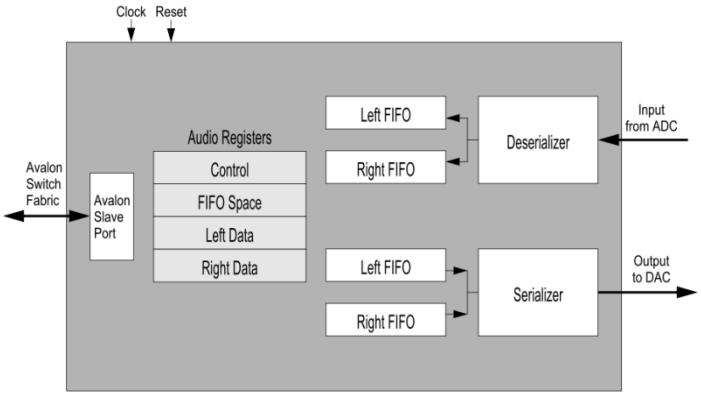
Il sistema completo di tutti i piedini di I/O dovrebbe più o meno assomigliare a questo:



Si salvi lo schematico, lo si configuri come "Top Level Entity", si importi un opportuno file di vincoli, 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 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 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 che poi si andrà a compilare e a caricare nella memoria del NIOSII

```
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);
                    }
          }


     }
}
```

Si noti 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.

Si noti altresì che 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 che per valori di "n" pari a 0 o 1 il sistema funzionerà regolarmente, mentre per valori superiori la FIFO in ingresso si riempirà immediatamente ed il segnale risulterà distorto, non riuscendo il sistema ad elaborare correttamente il flusso di campioni. L'operatore moltiplicazione è un'operazione piuttosto lenta (specie per il processore NIOS-II (e)) che ritarda il ciclo di lettura scrittura dei campioni.

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 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 Nios
    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 IMPEDISCE DI MINIMIZZARE A ICONA QUARTUS)
    e. A questo punto ci sono due possibilità
        i. o (passando per il desktop) si apre L'altera Monito Program e si prosegue
        ii. Oppure si stacchi il cavo USB, si chiuda la finestrella di controllo (e alla bisogna anche "Quartus"), si riconnetta il cavo USB. Ora il sistema funziona solo per un tempo limitato. (1 ora c.a)
- 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 da tabella:

SAMPLING RATE		MCLK FREQUENCY	SAMPLE RATE REGISTER SETTINGS					DIGITAL FILTER TYPE
ADC	DAC							
kHz	kHz	MHz	BOSR	SR3	SR2	SR1	SR0	
48	48	12.288	0 (256fs)	0	0	0	0	1
		18.432	1 (384fs)	0	0	0	0	
48	8	12.288	0 (256fs)	0	0	0	1	1
		18.432	1 (384fs)	0	0	0	1	
8	48	12.288	0 (256fs)	0	0	1	0	1
		18.432	1 (384fs)	0	0	1	0	
8	8	12.288	0 (256fs)	0	0	1	1	1
		18.432	1 (384fs)	0	0	1	1	
32	32	12.288	0 (256fs)	0	1	1	0	1
		18.432	1 (384fs)	0	1	1	0	
96	96	12.288	0 (128fs)	0	1	1	1	2
		18.432	1 (192fs)	0	1	1	1	
44.1	44.1	11.2896	0 (256fs)	1	0	0	0	1
		16.9344	1 (384fs)	1	0	0	0	
44.1	8 (Note 1)	11.2896	0 (256fs)	1	0	0	1	1
		16.9344	1 (384fs)	1	0	0	1	
8 (Note 1)	44.1	11.2896	0 (256fs)	1	0	1	0	1
		16.9344	1 (384fs)	1	0	1	0	
8 (Note 1)	8 (Note 1)	11.2896	0 (256fs)	1	0	1	1	1
		16.9344	1 (384fs)	1	0	1	1	
88.2	88.2	11.2896	0 (128fs)	1	1	1	1	2
		16.9344	1 (192fs)	1	1	1	1	

Table 18 Normal Mode Sample Rate Look-up Table

- 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=-a_2=1, a_1=0, 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|< 2 sqrt(b_2) questo parametro 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 (anzicchè 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 canalo L e R stereo.
Utilizzando Altera Monitor Program e si compili e si implementi sul NIOS. Si noterà subito che la FIFO in ingresso tenderà subito a saturare ed il suono risulterà fortemente distorto. Per ovviare all'inconveniente si deve riconfiguare 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 la strada indicata 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 (se si è scelto il parametro per il registro SR del codec uguale a 0) – 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 si 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 si riempirà in brevissimo tempo. Per aggirare l'inconveniente si potrebbe ad esempio scegliere una frequenza di campionamento del segnale audio inferiore (settando l'opportuno registro SR), 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".

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

Il sistema ha il pregio di mettere a disposizione dello sviluppatore di software alcune funzioni 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 NIOS II – Software Build Tool for Eclipse (l'ambiente è basato di diversi workspaces) che mantengono memoria dei vari progetti sviluppati
- `File > New > Application for Nios II and BSP application`
- Si selezioni opportunamente il file `.sopcinfo` che contiene le informazioni HW del nostro sistema (generato dal SOPCBUILDER)
- 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_avalon_pio_regs.h` che contiene alcune funzioni utili nella gestione delle periferiche di I/O.

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:

```
#include "sys/alt_stdio.h"
#include "altera_up_avalon_audio.h"
#include "altera_avalon_pio_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 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");
```

```
  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)
  {
  int fifospace = alt_up_audio_read_fifo_avail (audio_dev, ALT_UP_AUDIO_RIGHT);
  IOWR_ALTERA_AVALON_PIO_DATA(LEDR_BASE, fifospace);

   // demo swithes and leds
  sw=IORD_ALTERA_AVALON_PIO_DATA(SWITCHES_BASE);
  IOWR_ALTERA_AVALON_PIO_DATA(LEDG_BASE, sw);

  if ( fifospace > 16 ) // check if data is available
  {
  // 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);

  }
  }

  return 0;
}
```

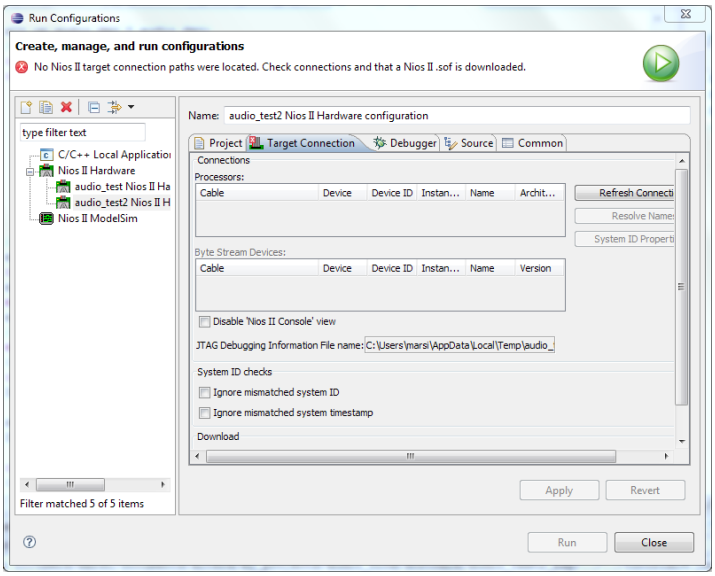
Ricordarsi di includere i `file.h` utili al nostro progetto e si noti l'uso di funzioni predefinite per le operazioni di I/O verso le periferiche.

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

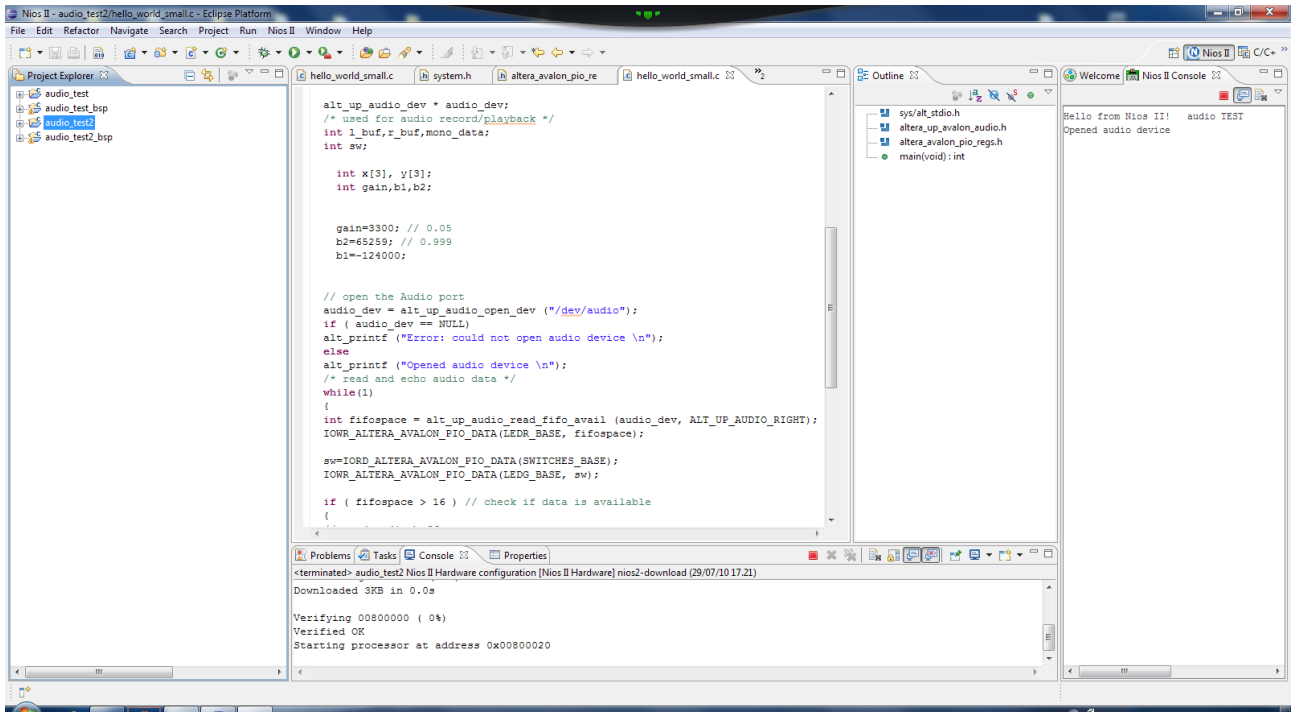
- Si evidenzi (`click`) la prima delle due cartelle (<nome_progetto>)
- `Run > Run As > Nio II Hardware`

Dopo aver compilato tutti i file inclusi nel progetto la prima vola si apre una finestra di comunicazione che chiede di attivare la connessione al processore;

- Si attivi la Tab - `Targhet Connection`
- Click on `Refresh Connection`
- Click on `RUN`



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.