Università degli Studi del Sannio

Facoltà di Ingegneria Informatica

Corso di Elaborazione dei Segnali e delle Informazioni di Misura

Prof. Pasquale Daponte

A.A. 2007/2008

Effetti Audio Digitali: sintesi di un filtro digitale IIR ed implementazione dell'effetto "Wah wah" in C su DSP 'C6711

Bruno Raffaele (195/683)

Dell'Oste Vittorio (195/894)

Merola Marco (195/877)

Russo Angelo (195/932)

Indice

Appendici

1. Scopo del progetto

Lo scopo del progetto è stato quello di acquisire familiarità con:

- Le teorie alla base degli effetti audio digitali
- L'architettura dei processori DSP
- L'implementazione di uno specifico effetto audio (Wah wah) attraverso l'utilizzo di filtri digitali
- L'implementazione di un effetto audio digitale in MatLab ed in C
- L'ambiente di sviluppo Code Composer Studio

Lo sviluppo del progetto è stato articolato in tre fasi:

1) Analisi dell'effetto audio digitale "Wah wah" e di una sua possibile realizzazione

2) Implementazione in MatLab del programma che implementa l"effetto Wah e simulazione dello stesso per verificarne l'effettivo funzionamento

3) Implementazione in linguaggio C e testing su DSP dell'intero progetto

2. Introduzione all'effetto Wah wah

Il wah wah (conosciuto anche come wah-wah o semplicemente wah) in musica è un effetto musicale che prende il nome dal caratteristico suono prodotto dal taglio e dal reinserimento graduale delle frequenze alte, vagamente assimilabile a un miagolio o a un vagito, da cui il nome onomatopeico. Per questo motivo l'effetto viene anche chiamato cry baby ("pianto di bambino"), da cui crybaby, marchio di fabbrica della Jim Dunlop, che insieme alla Vox è il più famoso costruttore di pedali wah-wah per strumenti elettrici.

Un pedale wah wah modifica il suono dello strumento tramite un graduale cambiamento di tono tra acuti e bassi, regolati per mezzo di un apposito potenziometro azionato con la spinta del piede sul pedale rispettivamente verso la punta o verso il tacco.

Il wah wah è uno dei pochi effetti musicali tradizionali che non sono stati soppiantati da versioni digitali o strumenti software di elaborazione del suono (almeno nelle esibizioni dal vivo). A differenza di altri effetti, infatti, il wah-wah non deve essere semplicemente *inserito* e *disinserito* in funzione del brano musicale da eseguire, ma viene controllato in modo continuo e a totale discrezione di chi lo usa, applicando un effetto potenzialmente diverso a ogni singola nota. Esistono tuttavia in commercio **wah wah digitali**, a costo relativamente basso, che applicano un effetto prefissato a ogni suono riprodotto.

Particolari tipi di wah sono gli **auto wah**, che sono costituiti generalmente dallo stesso circuito di un normale wah; la differenza sta nell'apparato che varia il cambiamento di tono: esso è sostanzialmente costituito da un oscillatore che pilota una resistenza variabile al posto del potenziometro collegato meccanicamente al normale pedale. Altri tipi di wah sono i **wah dinamici**, che utilizzano come parametro di controllo il volume della nota suonata: a volume più alto corrisponde un tono più acuto o più basso e viceversa. Esistono altri tipi di wah, come i wah ottici, a sequencer e digitali. Spesso questo genere di effetto è già caricato in versione digitale nei preset di numerose apparecchiature multi-effetto per chitarra, basso o per il trattamento della voce.

Il wah wah è un effetto molto noto tra i chitarristi e anche uno dei più antichi assieme al distorsore, al tremolo e al vibrato. Il suo utilizzo risale infatti agli anni '60. A renderlo celebre hanno contribuito grandi chitarristi, il più famoso dei quali è Jimi Hendrix, in grado di utilizzarlo con particolare maestria (nei brani Voodoo Child, Belly Button Window, ecc.). Altri chitarristi lo hanno spesso sfruttato con successo, tra gli altri: Eric Clapton, Santana, Stevie Ray Vaughan, Zakk Wylde, Mick Ronson, Jeff Beck, Slash e Tom Morello. Un genere musicale in cui il wah wah è stato usato spesso è il reggae.

3. Scelte Progettuali

Per l'implementazione dell'effetto wah wah si è utilizzato un filtro tempo-variante; la scelta è ricaduta su un particolare tipo di filtro digitale, quello "a variabili di stato", un IIR derivato dal suo equivalente analogico. Questo filtro permette di avere simultaneamente un comportamento passa-basso, passa-alto e passa-banda, grazie al fatto di poter prendere l'uscita del filtro in più punti del circuito, come mostrato nello schema a blocchi di Fig.1, dove $y_h(n)$ è l'uscita del filtro passa-alto, $y_l(n)$ è quella del passa-basso, mentre $y_b(n)$ è l'uscita del filtro passa-banda, quello utilizzato per il nostro scopo. Per l'implementazione si è utilizzata l'uscita del passa-banda. Infatti, per ottenere l'effetto Wah si filtra il segnale audio mediante un filtro passa-banda la cui frequenza centrale viene modulata nel tempo con un'onda triangolare. La frequenza centrale è fatta variare da 500 Hz a 3kHz, frequenze medie dei segnali audio in alta fedeltà.

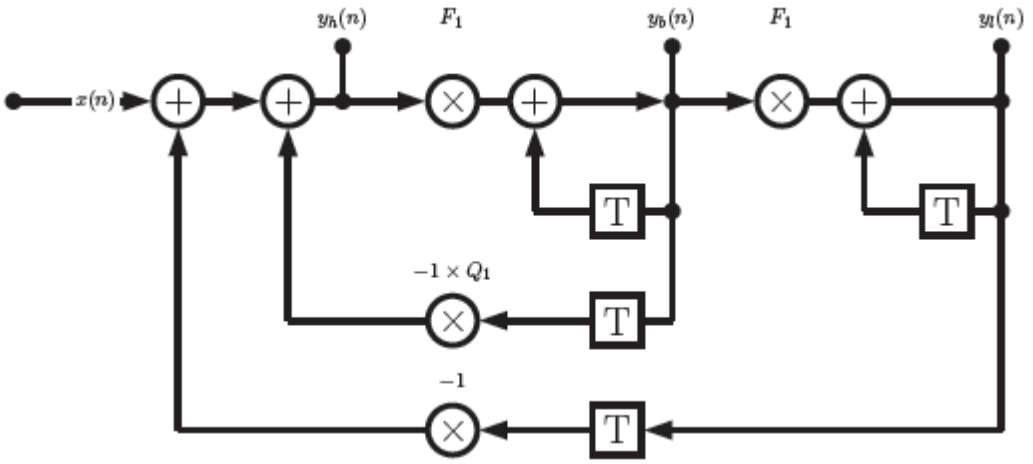


Fig. 1. Schema a blocchi del filtro a variabili di stato.

Il sistema di equazioni alle differenze del filtro a variabili di stato è:

$$\begin{cases} y_l(n) = F_1 y_b(n) + y_l(n-1) & (1) \\ y_b(n) = F_1 y_h(n) + y_b(n-1) & (2) \\ y_h(n) = x(n) - y_l(n-1) - Q_1 y_b(n-1) & (3) \end{cases}$$

Queste equazioni sono state ottenute dallo schema a blocchi del filtro a variabili di stato, applicando la Z-Antitrasformata.

La funzione di trasferimento discreta del nostro filtro passa-banda è mostrata nell'equazione (4), della quale viene tracciato il diagramma di Bode dei moduli e delle fasi, riportato in Fig.2:

$$H_b(z) = F_1 \frac{(1 - z^{-1})}{1 + (2 + F_1^2 + Q_1 F_1) z^{-1} + (1 - 2F_1 Q_1) z^{-2}} \qquad (4)$$

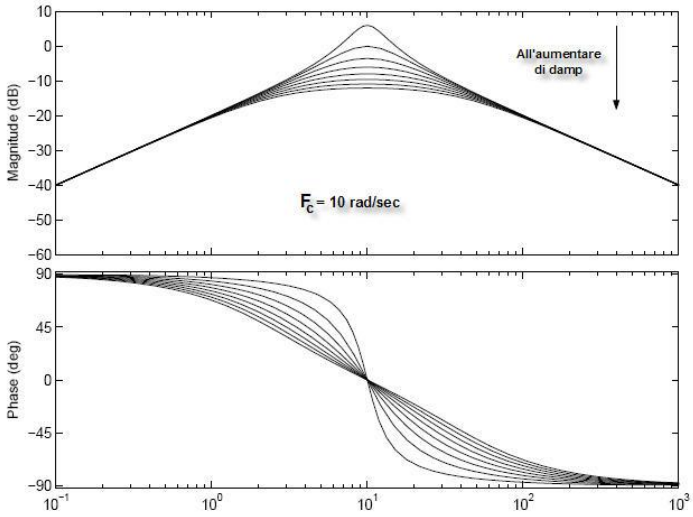


Fig.2. Diagramma di Bode del passa-banda con Fc = 10Hz e fattore di smorzamento variabile

Nel caso specifico si ha una frequenza centrale variabile nel tempo, e un fattore di smorzamento molto basso, così da avere un alto picco di risonanza intorno ad Fc e una banda passante molto stretta.

Lo schema di principio dell'effetto Wah è riportato in Fig.3

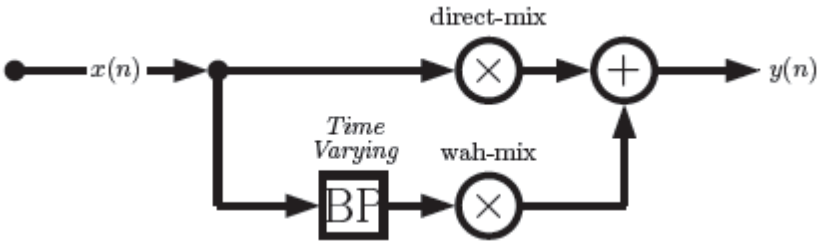


Fig.3 . Schema di principio dell'effetto Wah

Il segnale in ingresso al sistema x(n) verrà fatto passare attraverso il filtro passa-banda (BP) tempo – variante, e il segnale risultante dal filtraggio verrà moltiplicato per un guadagno. Infine l'uscita del sistema sarà la composizione del segnale x(n) originale con quello filtrato.

4. Simulazione in ambiente MatLab

Il primo problema affrontato è stato quello di simulare l'effetto Wah wah in MatLab per verificare la correttezza delle scelte implementative. Di seguito vengono riportati frammenti di codice MatLab.

Come primo passo è stata creata un'onda triangolare per poter modulare la frequenza centrale del filtro passa-banda, il cui andamento è mostrato in Fig.4:

```
Fc=minf:delta:maxf;
while(length(Fc) < length(x) )
    Fc= [ Fc (maxf:-delta:minf) ];
    Fc= [ Fc (minf:delta:maxf) ];
end
% Fissa la lunghezza dell'onda triangolare in base all'ingresso
Fc = Fc(1:length(x));
```

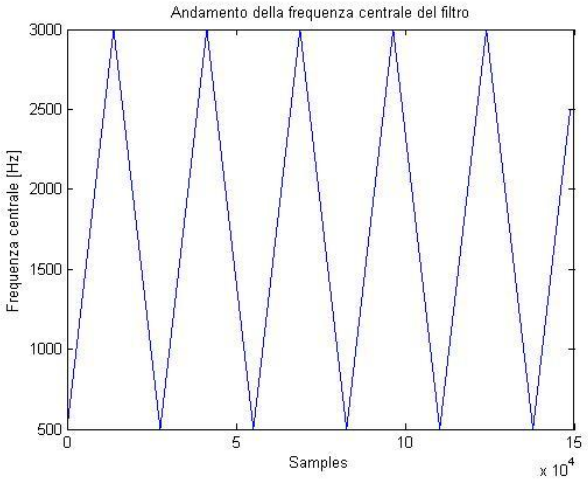


Fig.4 . Andamento frequenza centrale del filtro

In questo modo Fc potrà variare dalla frequenza minima a quella massima con passo prefissato. L'ultima istruzione invece serve a far sì che la lunghezza dell'onda triangolare sia fissata in base alla lunghezza dell'ingresso.

Il secondo passo è stato quello di implementare le equazioni alle differenze del filtro a variabili di stato:

```
% coefficienti delle equazioni alle differenza
% deve essere ricalcolata ogni volta che Fc cambia
F1 = 2*sin((pi*Fc(1))/Fs);
% impone la larghezza della banda passante
Q1 = 2*damp;
% applica le equazioni alle differenze ai campioni
for n=2:length(x),

    yh(n) = x(n) - yl(n-1) - Q1*yb(n-1);
    yb(n) = F1*yh(n) + yb(n-1);
    yl(n) = F1*yb(n) + yl(n-1);


    F1 = 2*sin((pi*Fc(n))/Fs);
end
% normalizza

maxyb = max(abs(yb));
yb = yb/maxyb;
```

Sono stati calcolati i cofficienti del filtro F1 e Q1 ed alla fine si è presa l'uscita del filtro passa-banda e la si è normalizzata, in quanto MatLab impone un range di valori in ampiezza per segnali audio tra [-1,1]. Alla fine è stato creato un file .wav contenente il file audio filtrato, ed il confronto tra i segnali originale e filtrato è mostrato nel grafico di Fig.5.

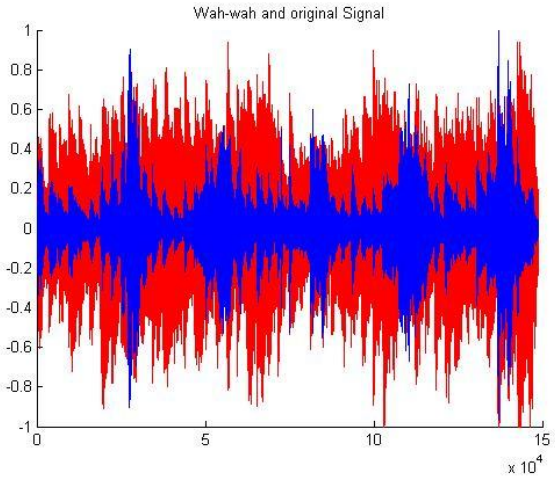


Fig.5 . Confronto tra segnale originale (rosso) e Wah wah (blu).

5. Implementazione in linguaggio C dell'effetto Wah wah

Dopo aver simulato in MatLab ed aver ottenuto risultati soddisfacenti, si è passati alla traduzione in linguaggio C per la creazione del programma da caricare su DSP 'C6711.

L'hardware utilizzato consiste in una DSK Texas Instruments TMS320 e una Audio DaughterCard PCM3003, quest'ultima necessaria per l'elaborazione di segnali in qualità stereo con frequenza di campionamento di 44100Hz, valore non raggiungibile usando il codec AD535 integrato sulla scheda TMS320.

Il passaggio dal codice in MatLab al C non ha causato molti problemi; per far variare la frequenza centrale del filtro passa-banda è stata creata una funzione *setCutOffFreq()*.

```
void setCutOffFreq() {

        if(Fc <= minf)

                delta = 0.1;

        else if(Fc >= maxf)

                delta = -0.1;

        Fc = Fc + delta;

}
```

La funzione che implementa il filtro IIR a variabili di stato è *wah_wah(short sample)* la quale viene invocata ad ogni interrupt generato dal McBSP1 ogni qual volta viene ricevuto in ingresso un nuovo campione.

```
short wah_wah(short sample) {

yh = sample - yl_1 - Q1*yb_1;

yb = F1*yh + yb_1;

yl = F1*yb + yl_1;

yh_1 = yh;

yb_1 = yb;

yl_1 = yl;

setCutOffFreq();

F1 = 2*sin(Fc*pi/Fs);

return (short)(yb);

}
```

Infine la funzione *c_int11()* è quella che viene invocata ad ogni interrupt11; essa si preoccuperà di prelevare un campione, filtrarlo e inviarlo all'output.

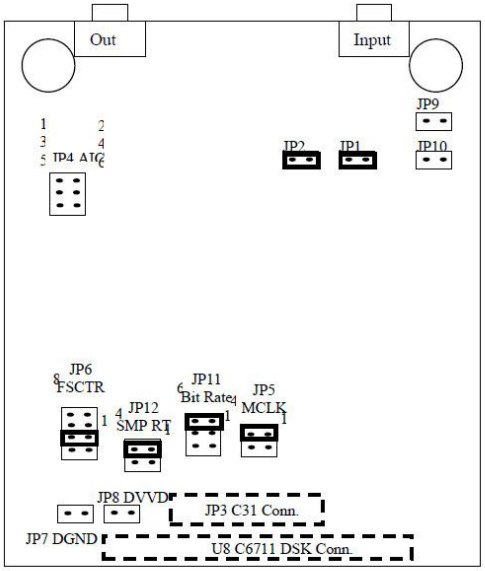
```
interrupt void c_int11()

{

short wah_sample = wah_wah(input_left_sample());

        output_left_sample(wah_sample);

        return;

}
```

6. Testing su DSP

I primi test su DSP hanno avuto esito negativo a causa delle difficoltà incontrate sia nell'interfacciare il programma C con la scheda Audio DaughterCard PCM3003, che nell'acquisizione del segnale audio da parte di quest'ultima.

Per poter interfacciare il programma alla scheda audio è stato configurato l' HWI (HardwareInterrupts) INT11. Tale interrupt è generato alla ricezione di un nuovo campione in ingresso sulla porta McBSP1 (Multichannel Buffered Serial Port). Per leggere tali campioni sono state utilizzate alcune funzioni della libreria CSL (Chip Support Library), quali `mcbsp1_init`, `mcbsp1_read`, `mcbsp1_write`. (Vedi Appendice B).

Per acquisire il segnale audio è stato necessario effettuare settaggi dei jumper sulla scheda soppalco al fine di abilitare la ricezione dai microfoni integrati o dalla linea d'ingresso. Si è dovuto, inoltre, configurare manualmente la scheda affinchè l' ADC campionasse ad una frequenza costante di 44.100kHz. Lo schema di configurazione dei jumper è mostrato in Fig.6.



Jumper Settings

JP1 – pin 1-2
JP2 – pin 1-2
JP3 –C31 DSK connector (bottom)
JP4– no connection
JP5 – pin 3-4

JP6 – PIN 3-4

JP7 – no connect, Digital Ground
JP8 – no connect, Digital Vdd (3.3V)
JP9 – no connect, Analog Ground
JP10 – no connect, Analog Vdd (3.3V)
JP11 – pin 5-6
JP12 – pin 3-4
U8 – C6711 DSK Connector (bottom)

Fig.6 . Jumper Settings

Appendice A – Codice MatLab

```
% wah_wah.m -- filtro passa banda a variabili di stato
%
% 08 Maggio 2008
%
% filtro BP con una stretta banda passante,
% Fc oscilla su e giu lungo lo spettro.
% Equazioni differenziali prese dal filtro analogico attivo
% a variabili di stato utilizzato per generare effetti audio
%
% yl(n) = F1*yb(n) + yl(n-1)
% yb(n) = F1*yh(n) + yb(n-1)
% yh(n) = x(n) - yl(n-1) - Q1*yb(n-1)
%
% Frequenze medie per i segnali audio:
% Fc varia da 500 a 5000 Hz
% 44100 campioni al secondo

clear all;
close all;

infile = 'acoustic.wav';

% legge  dal file audio wave
[ x, Fs, N ] = wavread(infile);

            % COEFFICIENTI DELL'EFFETTO AUDIO %
% coefficiente di smorzamento
% minore è il coefficiente di smorzamento minore è la banda passante
damp = 0.05;

% min e max centrano la frequenza di taglio variabile del
% filtro passa-banda

minf=500;
maxf=3000;


% frequenza effetto wah (Hz ciclati al secondo)
Fw = 2000;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Passo di variazione della frequenza centrale del filtro
% delta = 0.1;
delta = Fw/Fs;
% 0.1 => a 44100 campioni al secondo, Fc varia di 4.41kHz al sec


% crea un'onda triangolare di valori della frequenza centrale del filtro
Fc=minf:delta:maxf;
while(length(Fc) < length(x) )
    Fc= [ Fc (maxf:-delta:minf) ];
    Fc= [ Fc (minf:delta:maxf) ];
end

% Fissa la lunghezza dell'onda triangolare in base all'ingresso
Fc = Fc(1:length(x));
```

```
% coefficienti delle equazioni alle differenza
% deve essere ricalcolata ogni volta che Fc cambia
F1 = 2*sin((pi*Fc(1))/Fs);
% impone la larghezza della banda passante
Q1 = 2*damp;


yh=zeros(size(x));          % crea vettori vuoti
yb=zeros(size(x));
yl=zeros(size(x));

% condizioni iniziali per evitare riferimenti a segnali negativi
yh(1) = x(1);
yb(1) = F1*yh(1);
yl(1) = F1*yb(1);



% applica le equazioni alle differenze ai campioni
for n=2:length(x),

    yh(n) = x(n) - yl(n-1) - Q1*yb(n-1);
    yb(n) = F1*yh(n) + yb(n-1);
    yl(n) = F1*yb(n) + yl(n-1);


    F1 = 2*sin((pi*Fc(n))/Fs);
end

% normalizza

maxyb = max(abs(yb));
yb = yb/maxyb;

% scrive il file audio di uscita
wavwrite(yb, Fs, N, 'out_wah.wav');

figure(1)
hold on
plot(x,'r');
plot(yb,'b');
title('Wah-wah and original Signal');
```

Appendice B – Codice C

```
//loop_intr.c Programma che usa gli interrupt con codec stereo PCM3003

#include <math.h>

float Fs = 44100.0;           //settaggio irrilevante se il jumper è in 3-4
                              //sulla daughter card PCM3003
//PARAMETRI EFFETTO WAH
float damp = 0.05;
int minf = 500;
int maxf = 3000;

float Fc = 500;
float delta = 0.1;

int Fw = 2000;
float Q1;
float pi = 3.14159;
float F1;

float yh_1, yb_1, yl_1;
float yh, yb, yl;

//SETTO LA FREQUENZA CENTRALE DEL PASSA BANDA
void setCutOffFreq() {
      if(Fc <= minf)
            delta = 0.1;
      else if(Fc >= maxf)
            delta = -0.1;

      Fc = Fc + delta;
}

//FUNZIONE EFFETTO WAH
short wah_wah(short sample) {

      yh = sample - yl_1 - Q1*yb_1;
      yb = F1*yh + yb_1;
      yl = F1*yb + yl_1;

      yh_1 = yh;
      yb_1 = yb;
      yl_1 = yl;

      setCutOffFreq();
      F1 = 2*sin(Fc*pi/Fs);
      return (short)(yb/2);
}

//INTERRUPT SERVICE ROUTINE
interrupt void c_int11()
{
      short wah_sample = wah_wah(input_left_sample());
      output_left_sample(wah_sample); //IN/OUT from left

      return;                 //return dall' interrupt
}
```

```
void main()
{
      Q1 = damp*2;
      F1 = 2*sin(Fc*pi/Fs);
      yh_1 = yb_1 = yl_1 = 0;

      comm_intr();            //init DSK, codec, McBSP
      while(1);               //loop infinito in attesa dell'interrupt
}
```

```
//C6xdskinit_pcm.c Include delle funzioni standard della TI, init DSK ed McBSP

#include <c6x.h>
#include "c6xdsk.h"
#include "c6xdskinit_pcm.h"
#include "c6xinterrupts.h"


#define LEFT  1
#define RIGHT 0

float Act_Fs=0;
char polling=0;
extern float Fs;

volatile union {
      unsigned int uint;
      short channel[2];
      } mcbspData;

/* declare McBSP base addresses */
#define McBSP0_Base     ((McBSP *)0x18c0000)
#define McBSP1_Base     ((McBSP *)0x1900000)

/* declare McBSP data structure */
typedef volatile struct {
      unsigned int drr;   /* rx data reg */
      unsigned int dxr;   /* tx data reg */
      unsigned int spcr;  /* control reg */
      unsigned int rcr;   /* rx control reg */
      unsigned int xcr;   /* tx control reg */
      unsigned int srgr;  /* sample rate gen reg */
      unsigned int mcr;   /* multichannel reg */
      unsigned int rcer;  /* rx chan enable reg */
      unsigned int xcer;  /* tx chan enable reg */
      unsigned int pcr;   /* pin control reg */
} McBSP;

float SampleRate(float freq) { /* freq is desired sample freq */
      unsigned int divider;
      float clock_freq = 37500000.0;
      float clocks_per_sample = 256.0;

      *(volatile unsigned int *)TIMER0_CTRL = 0;     /* stop timer */
      if(freq == 0)
            freq = 1;
      divider = (clock_freq / (2 * clocks_per_sample)) / freq + (float)0.5;
```

```
      if(divider == 0) //* minimum divider in clock mode is 1 */
            divider = 1;
      freq = (clock_freq / (2 * clocks_per_sample)) / divider;
      *(volatile unsigned int *)TIMER0_PRD = divider;/* set period */
      *(volatile unsigned int *)TIMER0_CTRL = 0x3D1; /* clock mode, int clock */
                                                     /* TOUT = timer pin, PWID =
1 */
      return freq;
}

void mcbsp1_init(void)
{
      McBSP *port  = McBSP1_Base;

      port->spcr = 0;          /* reset serial port */
      port->srgr = 0;          /* sample generator not used */
      port->pcr  = 0x000C;     /* FSX,FSR,CLKS are inputs; FSR and FSX are
active low */
      port->rcr  = 0x000000A0; /* 1-phase 32 bits receive */
      port->xcr  = 0x000000A0; /* 1-phase 32 bits transmit */
      port->dxr  = 0;
      port->spcr = 0x00010001; /* enable tx/rx */
}

int mcbsp1_read()
{
      McBSP *port  = McBSP1_Base;

    if (polling)
      while ( !(port->spcr & 0x2)) ;
    mcbspData.uint=port->drr;
      return mcbspData.uint;
}

void mcbsp1_write(int outdata)
{
      McBSP *port  = McBSP1_Base;

    if (polling)
            while ( !(port->spcr & 0x20000)) ;
      port->dxr =outdata;
}

void c6x11_dsk_init()          /* dsp and periphiral initialization  */
{
      CSR=0x100;               /* disable all interrupts             */
      IER=1;                   /* disable all interrupts except NMI  */
      ICR=0xffff;              /* clear all pending interrupts       */
      *(unsigned volatile int *)EMIF_GCR = 0x3300;       /* EMIF global control
*/
      *(unsigned volatile int *)EMIF_CE0 = 0x30;          /* EMIF CE0control
*/
      *(unsigned volatile int *)EMIF_CE1 = 0xffffff23;    /* EMIF CE1 control,
8bit async */
      *(unsigned volatile int *)EMIF_SDCTRL = 0x57116000; /* EMIF SDRAM control
*/
      *(unsigned volatile int *)EMIF_SDRP = 0x61a;        /* EMIF SDRM refresh
period     */
      *(unsigned volatile int *)EMIF_SDEXT = 0x54529;     /* EMIF SDRAM
extension         */
      Act_Fs=SampleRate(Fs);
      mcbsp1_init();
```

```
}

void comm_intr()        /*added for communication/initialization using
interrupt*/
{
      c6x11_dsk_init();
      config_Interrupt_Selector(11, XINT1); /*using transmit INT11*/
      enableSpecificINT(11);                /*for specific interrupt*/
      enableNMI();                          /*Enable NMI*/
      enableGlobalINT();                    /*Enable GIE for global interrupt*/
      mcbsp1_write(0);                      /*write to SP0*/
}

int input_sample()                          /*funzione aggiunta per input*/
{
      return mcbsp1_read();
}

short input_left_sample()                   /*funzione aggiunta per input*/
{
      mcbspData.uint=mcbsp1_read();
      return mcbspData.channel[LEFT];
}

short input_right_sample()                  /*funzione aggiunta per input*/
{
      mcbspData.uint=mcbsp1_read();
      return mcbspData.channel[RIGHT];
}

void output_sample(int out_data)            /*funzione aggiunta per input*/
{
      mcbsp1_write(out_data);
}

void output_left_sample(short out_data)     /*funzione aggiunta per output*/
{
      mcbspData.uint=0;
      mcbspData.channel[LEFT]=out_data;
      mcbsp1_write(mcbspData.uint);
}

void output_right_sample(short out_data)    /*funzione aggiunta per output*/
{
      mcbspData.uint=0;
      mcbspData.channel[RIGHT]=out_data;
      mcbsp1_write(mcbspData.uint);
}
```

Manuale d'uso

Indichiamo di seguito le operazioni necessarie al montaggio dell'hardware ed all'esecuzione del programma su DSP per generare l'effetto audio digitale "Wah wah".

1. Collegare la *DaughterCard* alla scheda *DSK* tramite la porta seriale (*Multichannel Buffered Serial Port McBSP*);
2. Collegare la scheda *DSK* al PC attraverso la porta parallela;
3. Connettere le casse audio alla *DaughterCard* tramite la porta contrassegnata dall'etichetta *OUT*;
4. Connettere la porta di input della *DaughterCard* contrassegnata dall'etichetta *IN* con l'uscita audio del PC per acquisire il segnale audio da elaborare;
5. Alimentare la scheda DSK tramite apposito alimentatore in dotazione;
6. Lanciare il software *CCS* (*Code Composer Studio DSK v.2.10.00*);
7. Selezionare dal menu a tendina *Project -> Open*;
8. Aprire il file "Loop_intr_pcm.pjt" presente all'interno della directory "Progetto Wah";
9. Selezionare dal menu a tendina *File -> Load Program* per effettuare il caricamento del programma sul DSP;
10. Selezionare il file "Loop_intr_pcm.out" presente nella directory "Progetto Wah\Release";
11. Eseguire il programma selezionando dal menu a tendina *Debug -> Run*;
12. Lanciare un file audio dal PC per inviarlo al *DSP*;
13. Per terminare l'esecuzione del programma selezionare *Debug -> Halt.*