TP DSP n°1
Programmes d'initiation et filtrage

Modalités de déroulement

Page Internet DSP

De nombreuses informations nécessaires aux travaux pratiques (TP) de DSP sont accessibles sur le site Internet du module DSP :

www-dsp.efrei.fr

Compte-rendu de TP

Un compte-rendu de TP est à remettre à la fin de la séance, comportant :

- les réponses aux questions posées dans ce sujet ;
- les programmes-sources mis en forme (mis en page, et seulement les parties modifiées par rapport aux programmes de départ).

Evaluation

L'évaluation des TP portera sur :

- les comptes-rendus
- les exercices validés (montrer à l'enseignant tout résultat obtenu)

Matériel et logiciels nécessaires

- PC sous Windows
- Scilab
- Visual C++
- Assembleur pour DSP56303 et debugger pour carte d'évaluation
- Carte d'évaluation EVM56303
- Câble série mâle-femelle
- Alimentation stabilisée 9V
- Oscilloscope
- Générateur de fonctions
- 2 câbles BNC-jack stéréo
- Enceintes stéréo ou casque

Exercice 1 : Calcul de la somme de 2 nombres sur DSP

Le programme suivant effectue une addition de 2 nombres. Le sauvegarder dans un fichier portant l'extension ".asm" et l'assembler en utilisant la procédure donnée dans le document "Traitement du Signal sur DSP 56303".

```
              org     Y:$0            ;positionnement à l'adresse $0 du champ mémoire Y
oper1         dc      $1234           ;définition d'une constante "oper1" (opérande 1)
result        ds      1               ;réservation d'un emplacement mémoire pour le résultat

              org     X:$0            ;positionnement à l'adresse $0 du champ mémoire X
oper2         dc      $DEC0           ;définition d'une constante "oper2" (opérande 2)

              org     P:s0            ;adresse $0 de la mémoire programme (champ P)
                                      ;P:0 correspond à l'adresse du vecteur RESET
              jmp     begin           ;1ère instruction après le RESET : saut

              org     P:$40           ;programme principal commençant à l'adresse P:$40
begin
              move    Y:oper1,Y0      ;chargement du registre Y0 avec la valeur "oper1"
              move    X:oper2,A       ;chargement du registre A avec la valeur "oper2"
              add     Y0,A            ;oper1+oper2, résultat dans A
              move    A,Y:result      ;stockage du résultat à l'adresse "result" dans Y
              jmp     begin           ;retour à "begin"
```

1.1) En s'aidant du debugger, donner les valeurs décimales (fractionnaires) des 2 opérandes, ainsi que le résultat en hexadécimal et en décimal.

1.2) Ecrire un programme qui copie en mémoire X les valeurs hexadécimales $0 à $F (à l'aide d'instructions assembleur), en utilisant l'adressage indexé. Vérifier son bon fonctionnement en précisant la procédure.

Exercice 2 : Entrées/sorties analogiques sans traitement sur DSP

Pour cette exercice, on a besoin du programme "pass.asm". Ce programme configure correctement le CODEC et réalise des entrées/sorties analogiques sans traitement. On pourra l'appeler également programme "suiveur".

Le cœur de ce programme est une boucle infinie comportant une synchronisation sur les échantillons du signal d'entrée, leur lecture dans les buffers d'entrée, un saut à la routine de traitement, et l'envoi du résultat dans les buffers de sortie :

```
...
loop_1
        jset    #2,x:M_SSISR0,*         ;wait for frame sync to pass
        jclr    #2,x:M_SSISR0,*         ;wait for frame sync

        move    x:RX_BUFF_BASE,a        ;receive left
        move    x:RX_BUFF_BASE+1,b      ;receive right
        jsr     process_stereo
        move    a,x:TX_BUFF_BASE        ;transmit left
        move    b,x:TX_BUFF_BASE+1      ;transmit right

        move    #TONE_OUTPUT,y0         ;set up control words
        move    y0,x:TX_BUFF_BASE+2
        move    #TONE_INPUT,y0
        move    y0,x:TX_BUFF_BASE+3

        jmp     loop_1
...
process_stereo                          ;début de la routine de traitement
        ...
        ...
        rts                             ;fin de la routine
        ...
        end                             ;fin du programme
```

Tous les exercices utilisant les entrées/sorties analogiques pourront utiliser le programme suiveur.

Pour tester ce programme, effectuer les opérations suivantes :

1) Appliquer un signal sinusoïdal issu du générateur de fonctions, dans l'entrée analogique de la carte d'évaluation (connecteur jack) ;
2) Connecter un oscilloscope en sortie analogique ligne (connecteur jack) ;
3) Lancer le programme suiveur ;
4) Visualiser le signal et au besoin diminuer son amplitude si une saturation du signal de sortie apparaît ;
5) Faire varier la fréquence du signal sinusoïdal en sortie du générateur de fonctions ;
6) Relever grossièrement la réponse en fréquence de la carte (amplitude du signal de sortie en fonction de la fréquence) et relier cette réponse à la fréquence d'échantillonnage programmée.

Exercice 3 : Dérivée d'un signal avec Scilab

La dérivée s(t) d'un signal analogique e(t) est définie par :

$$s(t) = \frac{de(t)}{dt} = \lim_{\tau \to 0} \frac{e(t) - e(t-\tau)}{\tau}$$

En calcul numérique, le plus petit pas temporel est la période d'échantillonnage T_e. La dérivée s'obtient donc en calculant :

$$s(t) = \frac{e(t) - e(t-T_e)}{T_e}$$

où T_e est l'unité de temps élémentaire.

Le programme Scilab suivant implémente un dérivateur et l'applique à un fichier-son. La méthode utilisée est l'implémentation directe, en temps-différé.

```
//Calcul de la dérivée d'un signal
//Méthode par implémentation directe (temps différé)
fe=44100;              //fréquence d'échantillonnage (nombre d'échantillons par seconde)
Te=1/fe;               //période d'échantillonnage
N=fe/10;               //nombre d 'échantillons étudiés
t=(0:N-1)*Te;          //temps discret : N valeurs de 0 à (N-1)*Te
fre=440;               //fréquence du signal
a=1;                   //amplitude
e=a*sin(2*%pi*fre*t); //signal
s(1)=0;                              //condition initiale
for n=2:N
    s(n)=(e(n)-e(n-1))/Te;           //dérivée
end;
xbasc                          //effacement de l'écran graphique
xsetech([0,0,1,1/2]);          //définition de la zone graphique
plot(e(1:N/10));               //affichage
xsetech([0,1/2,1,1/2]);
plot(s(1:N/10));
REP_WAV="c:\";
nom_fich=REP_WAV+"test1.wav";
savewave(nom_fich, s, fe);    //sauvegarde dans un fichier WAV
```

Tester ce programme et le modifier pour simuler du temps-réel ("simuler" car on n'aura jamais du vrai temps-réel ni même "pseudo temps-réel" avec Scilab), en considérant qu'à chaque calcul de s(n), e(n-1) n'est plus accessible et donc doit avoir été mémorisé.

Exercice 4 : Dérivée d'un signal sur DSP

4.1) Programmer la dérivation d'un signal, la tester avec un signal carré, un signal triangulaire et un signal sinusoïdal. Commenter la forme des signaux obtenus et comparer avec les résultats théoriques. Dans le cas sinusoïdal, vérifier également que l'amplitude du signal de sortie est correcte. L'échantillon passé e(n-1) nécessaire au traitement devra être stocké (provisoirement) dans un des accumulateurs 24 bits disponibles : X0, X1, Y0 et Y1.

4.2) Même chose que dans la question précédente mais en stockant l'échantillon e(n-1) en mémoire (par exemple dans le bloc mémoire X, à l'adresse 0) au lieu d'un accumulateur.

Exercice 5 : Filtre moyenneur qur DSP

Pour réaliser un filtre moyenneur défini par exemple par :

$$s(n) = \frac{e(n) + e(n-1) + e(n-2)}{3}$$

il faut modifier le programme dérivateur étudié précédemment en utilisant un buffer circulaire, l'adressage indexé avec modulo, et l'instruction MAC. C'est ce que fait le

programme suivant, dont seules les parties importantes sont données (les … correspondent à des parties du programme "pass.asm") :

```
        org     X:$0
em      dsm     2               ;2 emplacements mémoire pour e(n-1) et e(n-2)
...
        move    #em,R0          ;?
        move    #1,M0           ;?
        do      #2,finraz       ;?
        move    0,X:(R0)+       ;0->en (valeur initiale de e(n-i))
finraz
...
        jsr     moy             ;appel dans boucle de lecture-écriture
...
moy                             ;début routine (A contient e(n))
        move    #0,Y0           ;?
        do      #2,finmac       ;boucle sur les 2 premières additions
        move    X:(R0)+,X0      ;?
        add     X0,Y0           ;X0+Y0->Y0 <-> s(n)=s(n)+e(n-i)
finmac
        move    X:(R0),X0       ;?
        add     X0,Y0           ;?
        move    #0.33333,X0     ;?
        mpy     X0,Y0           ;X0=X0*Y0; <-> s(n)=s(n)*0.33333
        move    A,X:(R0)        ;?
        rts                     ;fin routine
```

5.1) Analyser ce programme, compléter ses commentaires et le tester. Commenter la forme du signal obtenu dans le cas où le signal d'entrée est carré. Quel est le type de ce filtre ?

5.2) L'une des principales propriétés des DSP est de posséder une instruction MAC (Multiplication et ACcumulation), optimisée pour le calcul rapide des équations de récurence. Modifier le programme ci-dessus pour qu'il utilise l'instruction MAC à la place de l'instruction ADD, les coefficients du filtre étant préalablement stockés en mémoire.

5.2) Tester ce programme en relevant les caractéristiques de sa fonction de transfert à l'aide d'un générateur de fonction et d'un oscilloscope. La comparer avec la fonction de transfert théorique obtenue à l'aide du programme Scilab suivant :

```
h0=0.3333;
h1=0.3333;
h2=0.3333;
h=[h2, h1, h0];
num=poly(h, "z", "coef");
den=1;
sys=syslin('d', num, den)
bode(sys, 0.0001, 0.3);
```

Exercice 6 : Synthèse d'un filtre RIF avec Scilab

6.1) Calcul des coefficients

On souhaite calculer les coefficients d'un filtre FIR, en utilisant la méthode de la Transformée de Fourier Discrète inverse. Cette méthode est basée sur le fait que :

- la fonction de transfert du filtre est la Transformée de Fourier de sa réponse impulsionnelle d'une part, et que
- ses coefficients sont un échantillonnage de sa réponse impulsionnelle.

On peut démontrer (cf document "Traitement du Signal : algorithmique et programmation") que cette méthode conduit à une expression donnant les coefficients du filtre RIF :

$$h_k = \frac{2}{N}\sum_{n=0}^{\frac{N}{2}-1} H_n . \cos\left(\frac{\pi n}{N}(2k - N + 1)\right) \text{ avec k=0,…,N/2-1}$$

N est le nombre de coefficients choisi pour le filtre. H_n correspond aux valeurs de la fonction de transfert désirée, échantillonnée. De plus, ces coefficients doivent être normalisés, c'est à dire divisés par leur somme.

Implémenter cette expression sous Scilab en complétant le programme ci-dessous (on pourra s'aider du programme Scilab utilisé dans l'exercice 5). L'appliquer à la fonction de transfert d'un filtre passe-bas de fréquence de coupure d'environ 1000Hz.

```
pi=3.14159;
N=15;
p=(N-1)/2;            //variable utilisée pour le décalage de la réponse impulsionnelle
h=zeros(1,N);
d=zeros(1,N);
H=zeros(1,N);
//Codage de la réponse fréquentielle désirée (exemple d'un passe-bas ici)
fc=...;                       //compléter les ...
for n=0:N-1
       if(n<fc)
              H(n+1)=1;
       end,
       if(n>N-fc)
              H(n+1)=1;       //symétrie % (N-1)/2 !
       end,
end
//Calcul des coefficients du filtre h(k)
...                           //calcul des coefficients à partir d'ici
```

6.2) Filtrage d'un signal de test

6.2.1) Pour tester notre filtre, on génère d'abord un signal défini par la somme de 2 sinusoïdes : l'une de 500 Hz (*sig1*) et l'autre de 5000 Hz (*sig2*), par le programme ci-dessous. Le compléter.

```
t=0.01:0.01:3;                  // fréquence d'échantillonnage : (à compléter)
sig1=sin(...);                  // compléter les ...
sig2=sin(...);                  // compléter les ...
inp=sig1+sig2;
xsetech([0,0,1,1/2]);           // affichage sur la moitié haute de la fenêtre
plot2d(...,inp)                 // compléter les ...
xtitle('signal original')
```

6.2.2) Ensuite, on applique le filtre au signal à l'aide du programme suivant :

```
out=flts(inp,sys);
xsetech([0,1/2,1,1/2])
plot2d(...,inp)                 // compléter les ...
xtitle('filtered signal')
```

Observer et commenter les résultats, au besoin en testant d'autres valeurs pour la fréquence de coupure.

6.2.3) Recommencer la même opération mais avec un filtrage éliminant le signal de fréquence supérieure et conservant l'autre.

6.2.4) Recommencer les 2 filtrages précédents avec un filtre d'ordre plus important. Conclure sur l'influence de l'ordre du filtre sur son efficacité, et sur l'inconvénient que cela représente.

6.2.5) L'appliquer à un son chargé à partir d'un fichier-son, en utilisant le programme de filtrage réalisé précédemment. Ecouter et interpréter le résultat.

Exercice 7 : Implémentation du filtre FIR sur DSP

7.1) Ecrire le programme assembleur implémentant le filtre synthétisé précédemment sous Scilab. On pourra s'inspirer du filtre moyenneur implémenté précédemment.

7.2) Tester ce filtre avec un générateur de signaux. Relever sa fréquence de coupure à –3dB et sa pente.

Exercice 8 : Filtrage RIF en C++

Pour cet exercice , on pourra s'inspirer largement du projet "Fifich" disponible sur le site Internet du module.

8.1) Implémenter le programme de calcul des coefficients du filtre FIR précédemment développé en langage C++, sous forme d'une routine.

8.2) Implémenter également la fonction de filtrage RIF en temps-réel.

8.3) Modifier le projet Visual C++ "Fifich" pour appliquer ce filtrage à un fichier-son. Comparer le résultat avec celui obtenu sous Scilab.