Devoir surveillé de Traitement du Signal-DSP
I3 Electronique
10 janvier 2006 – Benoît Decoux
Durée : 2 heures - Tous documents et calculatrice autorisés
Le corrigé de cet examen sera disponible à la fin de la journée sur le site Internet
www-dsp.efrei.fr (login : rdf, password : rdf)

Exercice 1 : Arithmétique des DSP (4 points)

1) Donner la valeur décimale, exprimée au format à virgule fixe [1, 23], correspondant à la valeur hexadécimale $000800.

En binaire, cette valeur est égale à

0000100000000000000000000

En virgule fixe, on rajoute la virgule :

0,000100000000000000000000

On n'a plus qu'a multiplier les bits à 1 par leur poids :

$1\times2^{-4}=0,0625$

2) Dans le format mixte à virgule fixe [9, 47], donner la valeur binaire et la valeur hexadécimale correspondantes à la valeur décimale fractionnaire 64,75.

Dans le format mixte [9, 47], la partie entière du codage comporte 9 bits, dont un bit de signe. La partie entière de la valeur à coder est $64=2^6$, donc seul le 7^e bit de la partie entière du codage doit être égal à 1.

Pour la partie fractionnaire, on peut utiliser la méthode systématique, mais on peut remarquer que 0,75=0,5+0,25, donc on peut en déduire directement le résultat binaire:

0010 0000 0,110 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

et le nombre hexadécimal correspondant :

$20 600000 000000

Voir paragraphe « Conversion entre formats » dans le document « Arithmétique des DSP ».

Exercice 2 : Programmation de base en assembleur DSP56303 (4 points)

1) Ecrire un programme assembleur (au moyen d'instructions) qui copie en mémoire X, à partir de l'adresse 0, les valeurs hexadécimales $00000F à $00000, dans l'ordre décroissant, en utilisant un registre d'adresse Rn (n=0,…,7).

```
        org     X:$0
memx    dsm     16              ;réservation de 16 espaces en mémoire (=tableau)

        org     P:$0            ;adresse du programme
        move    #memx,R0        ;R0 (registre d'adresse) pointe sur la première case du tableau
        move    #$0F,A0         ;$F dans A0 (A0=24 bits de poids faibles de A)
        do      #16,fin         ;boucle de 16 itérations (jusqu'à "fin")
        move    A0,x:(R0)+      ;A0 dans X et post-incrémentation de R0
        dec     A               ;décrémentation du contenu de A
fin                             ;fin de la boucle
        end                     ;fin du programme
```

2) Modifier le programme pour que ces valeurs soient multipliées par 0,8.

Les lignes ajoutées sont mises en gras :

```
        org     X:$0
memx    dsm     16              ;réservation de 16 espaces en mémoire (=tableau)
        org     P:$0            ;adresse du programme
        move    #memx,R0        ;R0 pointe sur la première case du tableau
        move    #0.8,X0         ;0,8 dans X0
        do      #16,fin         ;boucle de 16 itérations (jusqu'à "fin")
        move    A0,Y0           ;A0->Y0
        mpy     X0,Y0,A         ;A=X0xY0=0.8xA
        move    A0,x:(R0)+      ;A0 dans X et post-incrémentation de R0
        dec     A               ;décrémentation du contenu de A
fin                             ;fin de la boucle
        end                     ;fin du programme
```

Exercice 3 : Synthèse de filtres RIF avec Scilab (4 points)

3.1) Donner le programme Scilab permettant de calculer les coefficients d'un filtre RIF passe haut de fréquence de coupure 480Hz, par la méthode du développement en série de Fourier de la réponse en fréquence, et fenêtrage. La fréquence d'échantillonnage sera prise égale à 48kHz. La réponse en fréquence du filtre devra être améliorée par fenêtrage de Hamming.

Les coefficients du filtre, avant décalage, sont définis par :

$$g_k = -\frac{1}{k\pi}\sin(2\pi k F_c) = -2F_c \sin c(2kF_c)$$

où F_c est la fréquence de coupure relative (ou normalisée), c'est à dire divisée par la fréquence d'échantillonnage. Ici : F_c=480/48000=0,01.

Les valeurs finales des coefficients sont obtenues par décalage

$$h_k = g_{k-p}$$

avec p=N/2 si N pair, p=(N-1)/2 si N impair. D'où le programme :

```
N=45
Fc=0.01
if modulo(N,2)==0               //si N pair
        p=N/2
else                            //si N impair
        p=(N-1)/2;
end
for k=0:N-1
        w(k+1)=0.56+0.44*cos(2*%pi*(k-p)/N);        //Fenêtrage de Hamming
        if k~=p
                h(k+1)=-1/(k-p)/%pi*sin(2*%pi*Fc*(k-p));
        else                                        //cas où k==p, traité à part
                h(k+1)=Fc;
        end
end
somme=sum(h);
h=h/somme;                      //normalisation des coefficients
```

Une partie de ce programme était donnée dans le document : "Traitement du signal : algorithmique et programmation".

3.2) Ecrire le programme permettant d'appliquer ce filtre à 1000 échantillons de signal sinusoïdal de fréquence 440Hz et d'amplitude égale à 1. On supposera qu'il existe une fonction appelée "rif" (qu'on ne demande pas d'écrire) possédant 2 arguments, le premier étant l'adresse du tableau des coefficients et le 2e celui des échantillons du signal à traiter, et retournant l'adresse du signal résultat.

```
fe=48000;               //fréquence d'échantillonnage (nombre d'échantillons par seconde)
```

```
Te=1/fe;              //période d'échantillonnage
N=1000;               //nombre d'échantillons calculés
t=(0:N-1)*Te;         //indices de temps discret des échantillons : de 0 à (N-1)*Te
fre=440;              //fréquence du signal
a=1;                  //amplitude
e=a*sin(2*%pi*fre*t); //échantillons du signal
s=rif(e,h);
```

En fait, la fonction *rif* peut être constituée par les quelques lignes de programme ci-dessous. Le principe de cette méthode est d'utiliser la fonction de filtrage *flts*, qui nécessite comme 2^{e} argument un objet de type "système linéaire". Ce dernier est obtenu par la fonction *syslin*, qui nécessite comme arguments le polynôme du numérateur et celui du dénominateur. Ces polynômes sont obtenus pas la fonction *poly*. Il est à remarquer que, comme il n'y a pas de récurrence dans ce filtre, le polynôme du dénominateur se limite à la valeur 1. Cette partie de programme n'était pas demandée.

```
d(N)=1;
num=poly(h, "z", "coef");     //définition du polynôme en z du numérateur
den=poly(d, "z", "coef");     //définition du polynôme en z du dénominateur
sys=syslin('d', num, den);    //définition d'un système linéaire pour la fonction flts
s=flts(e,sys);                //génération du signal de sortie par la fonction de filtrage
```

Exercice 4 : Filtrage en assembleur DSP56303 (4 points)

4.1) Le programme donné ci-dessous permet de réaliser un filtrage de type RIF (à Réponse Impulsionnelle Finie) avec un filtre à 3 coefficients, réalisant un moyenneur. Le compléter pour qu'il permette de réaliser un filtrage de type RII (à Réponse Impulsionnelle Infinie) du 2^{e} ordre. Seules les parties ajoutées ou modifiées sont à spécifier. On prendra des valeurs quelconques pour les coefficients.

```
...                             ;les "..." représentent des parties du programme "pass.asm"
        org     X:$20           ;attention, pass.asm met des choses en X:0
h0      dc      0.333           ;liste des coefficients
h1      dc      0.333
h2      dc      0.333
en      dsm     3               ;3 emplacements mémoire pour e(n), e(n-1) et e(n-2)
...
        move    #en,R0          ;registre d'adresse R0 pointe sur en
        move    #2,M0           ;R0 en modulo 3
        move    #h0,R1          ;registre d'adresse R1 pointe sur h0
        move    #2,M1           ;R1 en modulo 3
        clr     A               ;A=0
        do      #3,finraz       ;initialisation :
        move    A,X:(R0)+       ;0->e(n-i), i=0,1,2
finraz
...
        jsr     moy             ;appel dans boucle infinie de lecture-écriture de pass.asm
...
moy                             ;début routine (on suppose que B contient e(n))
        move    B,X:(R0)        ;e(n)->X:(R0) (mémorisation de e(n), écrase e(n-3))
        clr     B               ;B=0
        do      #2,finmac       ;boucle sur les 2 premiers mac
        move    X:(R0)+,X0      ;e(n-i)->X0 (i=0,1)
        move    X:(R1)+,Y0      ;h(i)->Y0 (i=0,1)
        mac     X0,Y0,B         ;B=B+X0*Y0 <-> s(n)=s(n)+h(i)*e(n-i) (i=0,1)
finmac
        move    X:(R0),X0       ;e(n-2)->X0
        move    X:(R1)+,Y0      ;h(2)->Y0
        mac     X0,Y0,B         ;B=B+X0*Y0 <-> s(n)=s(n)+h(2)*e(n-2)
        rts                     ;fin routine
```

Du point de vue des symboles utilisés, les coefficients h_i des filtres RIF sont remplacés par les ceux des filtres RII, appelés généralement a_i et b_i.

```
a_0     dc      0.00038/2       ;coefficients utilisés en TP
```

```
a_1     dc      0.00077/2
a_2     dc      0.00038/2
b_1     dc      -1.99064/2
b_2     dc      0.99218/2
...
sn      dsm     2               ;emplacements mémoire pour s(n-1) et s(n-2)
...
        move    #sn,R2          ;registre d'adresse R2 pointe sur sn
        move    #1,M2           ;R2 en modulo 2
...
        move    #4,M1           ;R1 en modulo 5
...
        do      #2,raz2
        move    A,X:(R2)+       ;0->s(n-i)
raz2
...
        move    X:(R2)+,X0      ;s(n-1)->Y0
        move    X:(R1)+,Y0      ;b(1)->X0
        mac     -X0,Y0,B        ;A=A+X0*Y0 <-> s(n)=s(n)+b(1)*s(n-1)
        move    X:(R2),X0       ;s(n-2)->Y0
        move    X:(R1)+,Y0      ;b(2)->X0
        mac     -X0,Y0,B        ;A=A+X0*Y0 <-> s(n)=s(n)+b(2)*s(n-2)
        asl     B               ;s(n)=s(n)*2
        move    B,X:(R2)        ;s(n) ecrase s(n-2)
...
```

4.2) Donner les modifications à apporter à ce programme pour réaliser un filtrage du 4e ordre, obtenu par une double exécution de ses opérations (par exemple en les plaçant dans une boucle), l'entrée du 2e filtrage étant constitué par la sortie du 1er. Cette répétition est équivalente à la mise en série (=en cascade) de 2 filtres identiques du 2e ordre. Il sera nécessaire de mémoriser des échantillons supplémentaires : en sortie du 1er filtre et/ou en entrée du 2e (ils sont identiques mais mémoriser les deux simplifie la programmation).

Le problème peut être traité par une boucle de 2 itérations dans laquelle est exécutée 2 fois les différentes opérations de la routine *moy*. Mais il est nécessaire de mémoriser des échantillons supplémentaires : 3 pour l'entrée et 2 pour la sortie. De plus, il faut se déplacer différemment dans ces buffers d'échantillon. On utilise pour cela des registres Ni (i=0,…,7) associés aux registres Ri. Voici ce qui change par rapport au programme précédent :

```
en   dsm       6
sn   dsm       4
     ...
     move      #en,R0
     move      #5,M0           ;R0 en modulo 6
     move      #sn,R2
     move      #3,M2           ;R2 en modulo 4
     move      #2,N0           ;pour le passage d'un filtre à l'autre : buffer d'entrée
     move      #2,N2           ;idem pour buffer de sortie
     ...
     do        #2, fin_fltr
     move      B,X:(R0)+N0     ;e(n)->X:(R0) (mémorisation de e(n), écrase e(n-3)) ;
                               ;déplacement pour changement de filtre
     ...
     move      B,X:(R2)+N2     ;s(n) écrase s(n-2) ; déplacement pour changement de filtre
fin_fltr
```

Exercice 5 : Filtrage en C++ (4 points)

5.1) Ecrire une fonction en C++, réalisant un filtrage de type RIF. Cette fonction devra :
- s'appeler "trait_ech " ;
- être membre d'une classe appelée " Rif " ;
- utiliser un tableau d'échantillons, appelé "em", de façon circulaire, pour mémoriser et relire les échantillons nécessaires (on supposera que l'allocation mémoire aura déjà été faite pour

ce tableau ; de même, on supposera que les coefficients du filtre auront préalablement été mémorisés dans un tableau appelé "h") ;
- posséder comme seul argument l'échantillon d'entrée de type float, nommé "e" ;
- renvoyer l'échantillon de sortie en retour de la fonction.

L'équation devant être implémentée par cette fonction est :

$$s(n)=h_0.e(n)+h_1.e(n-1)+\dots h_N.e(n-N)$$

où N est l'ordre du filtre. La fonction peut donc s'écrire :

```
float
Rif::trait_ech(float e)
{
	int i;
	float s;

	s=h[0]*e;
	for(i=0 ; i<nh-1 ; i++)
		s+=h[i+1]*em[(nh-1-i+ih)%(nh-1)];
	em[ih]=e;
	if(++ih>=nh-1)
		ih=0;
	return s;
}
```

On pouvait s'inspirer des différents exemples de programmes donnés dans le document : "Traitement du signal : algorithmique et programmation", pour le cas des filtres RII.

5.2) Ecrire une fonction C++ calculant les coefficients d'un filtre de type RIF de type passe-bande, par la méthode de décomposition en série de Fourier. Les paramètres de la fonction seront les deux fréquences de coupure du filtre.

```
void
passebande_rif(int N, float fc1, float fc2)
{
	if(n%2==0)
		p=N/2
	else
		p=(N-1)/2
	for(k=0 ; k<N ; k++)
	{
		if(k!=N/2)
			h(k+1)=1/(k-N/2)/%pi*(sin(fc1*(k-N/2)*%pi)-sin(fc2*(k-N/2)*%pi));
		else
			h(k+1)=fc1-fc2;
	}
}
```

5.3) Ecrire une fonction C++ calculant les coefficients d'un filtre de type RII du 2^e ordre, de type passe-bande, ces coefficients étant obtenus par la transformée bilinéaire. Les paramètres de la fonction seront la fréquence de coupure et le coefficient de résonance désirés (habituellement symbolisé par ξ ; on l'appellera "xi").

```
void
passebas_2deg_iir(float fc, float xi)
{
	FLOAT k, k1, k1c, k2, a[3], b[2];

	k1= fe/M_PI/fc;
	k1c=k1*k1;
	k2=2*xi*k1;
	k=1+k2+k1c;
	a[0]=1./k;
	a[1]=2.*a[0];
	a[2]=-a[0];
	b[0]=-2.*(1.-k1c)/k;
	b[1]=-(1-k2+k1c)/k;
}
```