Programmation des DSP

Contraintes de vitesses, réduire puissance CPU, complexité des algorithmes.

Entre 2 et 4 mini-projets sur des sons, réverbération, saturation, etc.

Evaluation sur les codes et les rapports, qualité du code, des explications.

DIGITAL SIGNAL PROCESSOR : calculateur numérique (entrée/sortie numérique)

L’entrée est un flux continu de données, il travaille par paquets (puissances de 2)

Monter une octave <=> double fréquence

Bien adapté pour le traitement linéaire

Pourquoi numérique et pas analogique ? Programmable, on peut changer le code facilement. Il est stable (pas de contraintes température, vieillissement etc.). Un seul DSP peut faire plusieurs applications (filtre, synthétiseur, compression, décompression, etc.). Evolutif, flexible. Reproduction parfaite.

Certaines choses ne sont réalisables qu’en numérique :

* Compression
* Filtrage adaptatif (filtre anti-Larsen)
* Filtres à phase linéaire

Traitement temps réel :

Le traitement doit être réalisé dans un temps donné (assez faible) pour donner l’impression qu’il est absent. Il faut que : t < processing time < t + Te. On peut aussi dans certains cas être obligé de faire un traitement par paquets (buffer) pour réduire le temps de traitement.

On peut faire du traitement linéaire, filtre réponse impulsionnelle, convolution, FFT, transformée en ondelette.

Circuit DSP :

Mémoire + CAN + CNA + ports série/parallèle

Opération MAC : multiplication and accumulation A = B\*C + D

Permet de mesurer la puissance d’un DSP (1 MAC par cycle d’horloge pour un DSP classiquement)

Architecture :

* Harvard : mémoire programme et mémoire données distincts, accessibles en même temps
* Von-Neumann : principe du PC, mélange des données et programmes dans la mémoire
* Aujourd’hui : Harvard + Von-Neumann car mémoires caches L1 L2 L3

Il faut faire attention à savoir si on met les données dans le cache ou dans la RAM

En C : « FAR » dans mémoire externe

DSP utilisés : TMS320C6713

Configurer la structure du cache L2 !

ALU : deux registres A et B reliés à 4 unités de traitement (8 unités fonctionnelles, 8 opérations par cycle d’horloge environ)

* Deux unités M (2 multiplications par cycle d’horloge)
* Deux unités D : calcul d’adresse
* Deux unités L
* Deux unités S

Il faut réfléchir à paralléliser dans le code C !

Mémoire programme données 256ko

Mémoire externe 8mo

Echantillonner c’est multiplier par un peigne de Dirac.

Numérique échantillonnage à 44kHz ou 96kHz, on filtre toujours (en analogique) en entrée du CAN.

Il faut ensuite le quantifier (ici uniforme en binaire), il y a une erreur de quantification et ajout de bruit blanc. En général 16bits ou 24bits.

CAN à approximation successive, double rampe, flash, sigma delta (24 bits)

Langage C

Char : 8 bits

Short : 16 bits

Int : 32 bits (à prendre pour les DSP)

Float : 32 bits virgule flottante

Int t[10] : tableau de 10 valeurs

Int \* p ; c’est un pointeur qui va pointer sur des tableaux d’entiers

p = &t[0] ;

\*p = 1 ; // équivalent à t[0] = 1

p++ ;

\*p = 2 ; // équivalent à t[1] = 2

\*p++ = 2 ; // équivalent à t[1] = 2 ET incrémentation

c = \*p ; // c = t[2]

Si on a int t1[10], t2[10] et qu’on veut copier t1 dans t2 :

Boucle for(i=0 ; i<10 ; i++)

{

t2[i] = t1[i] ;

}

Ou encore :

int\* p1,p2 ;

p1 = &t1[0] ;

p2 = (int\*)t2 ;

for(i=0 ; i<10 ; i++)

{

\*p2++ = \*p1++ ;

}

Il vaut mieux définir les grosses variables en global, sinon les variables se mettent dans la pile et la font exploser. Et mot clé « FAR » en mémoire.

Premier exo : diminuer/augmenter graves, médiums, aigus

On va faire 3 filtres à la suite (2 d’ordres 1, 1 d’ordre 2)

1. On veut faire un filtre passe-bas pour les graves de paramètre « g » (+-10db)

On définit une première fréquence f0 où le gain vaut « sqrt(g) » (point d’inflexion de la courbe)

Filtre = (dp+a)\*(cp+b)

Si f = 0, a/b = g

Si f = infini, d = c = 1

Module ² = g

Module de (j\*W0+a)/(j\*W0+b) = g d’où (a² + W0²)/(b² + W0²) = g or a/b = g

D’où b=W0/sqrt(g) et a = W0\*sqrt(g)

Puis transformée bilinéaire p = (2/Te)\*(1-z^-1)/(1+z^-1)

On pose w0 = W0/fe (sans unité)

D’où filtre = (1+w0\*sqrt(g) + z^-1\*(w0\*sqrt(g)-2))/(2+w0/sqrt(g) + z^-1\*(w0/sqrt(g) -2))

On met sous forme canonique filtre = (c+z^-1\*d)/(1+z^-1\*e)

y[n] = c\*x[n] + d\*x[n-1] – e\*y[n-1]

1. Passe-haut pour les aigus

(1+a\*p)/(1+b\*p)

1. Pour les médiums

On pose une fréquence centrale f2 où le gain = g

On pose alpha\*f2 où le gain vaut sqrt(g) et alpha > 1 à fixer