Filtres numériques (2 séances)

1. OBJECTIFS

- Comprendre la discrétisation d'un filtre et prendre conscience de ses effets.
- Comprendre la conception de filtres numériques IIR en effectuant une approximation à la main.
- Implémenter un filtre numérique en virgule fixe sur DSP.

2. INDICATIONS : FONCTIONS MATLAB UTILES

Matlab :	tfchk, zp2tf, filter, audioread, audioplayer, play, playblocking
Signal Processing Toolbox :	eqtflength, residuez, sos2tf, sos2zp, tf2sos, tf2zpk, zp2sos, freqz, freqs, impz, stepz, zplane, bilinear, ellipord, ellip, chirp
Control Toolbox :	bode, zpk, zpkdata, tf, tfdata, lsim

3. TRANSFORMATION S->Z

Soit un filtre analogique passe-bas de type Butterworth dont la fonction de transfert est :

$$F(s) = \frac{1}{\frac{s^2}{\omega_c^2} + s \cdot \frac{1.414}{\omega_c} + 1} \qquad \text{avec} \quad f_c = 1\ \text{kHz}$$

1) Choisir une fréquence d'échantillonnage pour discrétiser ce filtre.

2) A l'aide de MATLAB, déterminer sa fonction de transfert en z en appliquant la transformation **exacte aux pôles et aux zéros** :

$$p_{di} = e^{p_{ai} \cdot T} \qquad z_{di} = e^{z_{ai} \cdot T}$$

k_a, p_{ai}, z_{ai} : gain, pôles et zéros du filtre analogique dans la représentation "zpk", tels que retournés p.ex. par tf2zp. Attention : k_a n'est pas le gain DC.

k_d, p_{di}, z_{di} : gain, pôles et zéros du filtre numérique…

Pour un filtre passe bas, le gain "numérique" vaut :

Error! Bookmark not defined. $$k_d = k_a \cdot \prod_i \frac{(1 - e^{p_{ai} \cdot T})}{p_{ai}}$$

3) A l'aide de MATLAB, déterminer sa fonction de transfert en z en appliquant la transformation **bilinéaire aux pôles et aux zéros** :

$$p_{di} = \frac{1 + p_{ai} \cdot \frac{T}{2}}{1 - p_{ai} \cdot \frac{T}{2}} \qquad z_{di} = \frac{1 + z_{ai} \cdot \frac{T}{2}}{1 - z_{ai} \cdot \frac{T}{2}}$$

Cette fois, le gain "numérique" vaut :

$$k_d = k_a \cdot \frac{1}{\prod_i \left(2/T - p_{ai}\right)}$$

4) Comparer le comportement fréquentiel de ces deux filtres avec celui du filtre analogique, en particulier la limite supérieure de la bande passante (-3dB). En tirer des conséquences sur la méthode de conception de filtres à l'aide de la transformation bilinéaire.

5) Simuler le comportement de ces filtres (analogique (fct. lsim), transf. exacte, transf. bilinéaire) pour les signaux suivants :
   a) saut indiciel
   b) sinus de fréquence 100 Hz
   c) sinus de fréquence 2kHz

4. SYNTHÈSE D'UN FILTRE IIR

4.1 APPROXIMATION SEMI-MANUELLE (TRANSFORMATION BILINÉAIRE)

Soit le gabarit d'un filtre donné par la figure ci-dessous et les valeurs suivantes des paramètres: H0 = 0 dB, H1 = -3dB, H2 = -40 dB, f1 = 7 kHz, f2 = 9 kHz, fs = 48 kHz.

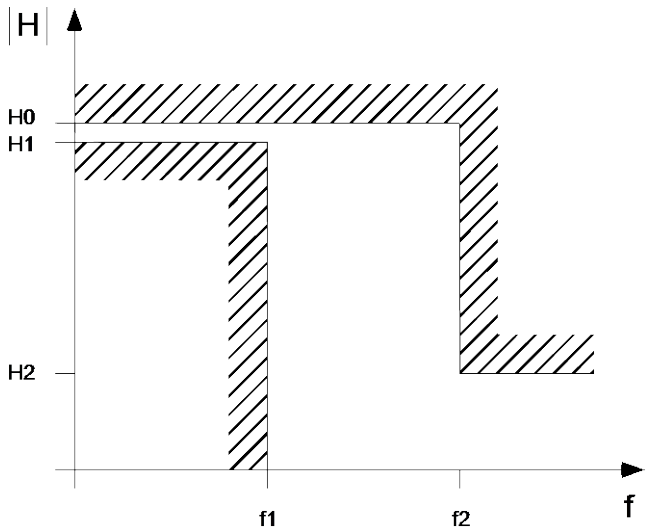


Fig. 1. Gabarit de filtre passe bas.

6) Transformer ce gabarit de façon à pouvoir utiliser les méthodes développées pour les filtres analogiques.
7) Effectuer l'approximation analogique à l'aide de MATLAB (caractéristique elliptique).
8) Dessiner la réponse en fréquence "analogique" (amplitude et phase).
9) Transformer cette fonction de transfert analogique en fonction de transfert numérique à l'aide de la fonction "bilinear" de MATLAB.
10) Dessiner la réponse en fréquence "numérique" et comparer au résultat souhaité.

4.2 VÉRIFICATION : APPROXIMATION À L'AIDE DE MATLAB

11) Effectuer la même approximation directement dans le domaine numérique à l'aide de MATLAB et comparer la fonction de transfert obtenue avec celle calculée au point 10).

4.3 SECTIONS DU 2È ORDRE

12) Séparer la fonction de transfert en sections du 2è ordre et calculer les coefficients correspondants (les a_i et b_i du paragraphe 5.1) qui seront directement utilisés lors de l'implémentation.

4.4 MESURE

13) A l'aide de MATLAB tester le filtre conçu plus haut en filtrant des signaux que vous pouvez enregistrer avec la carte son ou avec des signaux générés dans MATLAB (p. ex. sifflement dont la fréquence augmente progressivement).

5. RÉALISATION SUR DSP (TMS320C6211)

5.1 INTRODUCTION

Dans cette partie du laboratoire, nous allons réaliser le filtre numérique conçu précédemment, avec un programme écrit en C qui tournera en temps réel sur un DSP en virgule fixe. Nous utiliserons une structure cascade (2 étages du 2è ordre).

Soit la fonction de transfert en z universelle du 2è ordre :

$$H(z) = \frac{b0 + b1 \cdot z^{-1} + b2 \cdot z^{-2}}{a0 + a1 \cdot z^{-1} + a2 \cdot z^{-2}}$$

la manière la plus directe de la réaliser est l'équation aux différences.

La figure suivante représente le schéma bloc de cette réalisation pour le cas où a0 = 1 (rem : on peut toujours réécrire une fonction de transfert de manière à ce que a0 vaille 1). Cette structure est communément appelée "Forme directe I". Elle nécessite la mémorisation des valeurs passées de l'entrée et de la sortie.

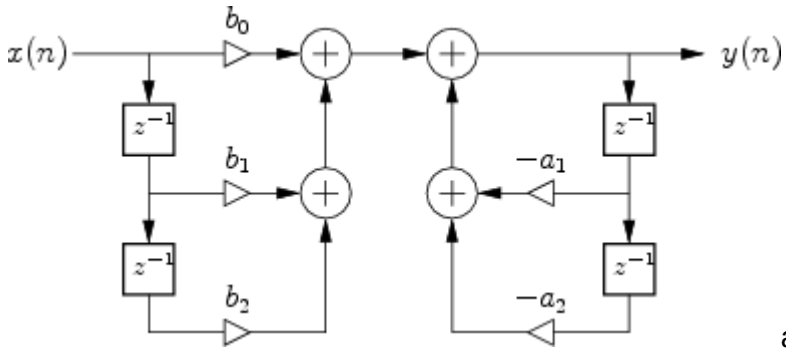


Fig. 2. Forme directe I

L'équation aux différences correspondante est :

$$y(n) = b0 \cdot x(n) + b1 \cdot x(n-1) + b2 \cdot x(n-2) - a1 \cdot y(n-1) - a2 \cdot y(n-2)$$

Il existe des structures dites canoniques réalisant la même fonction de transfert qui nécessitent 2 fois moins de mémorisations que la structure précédente. En voici deux :

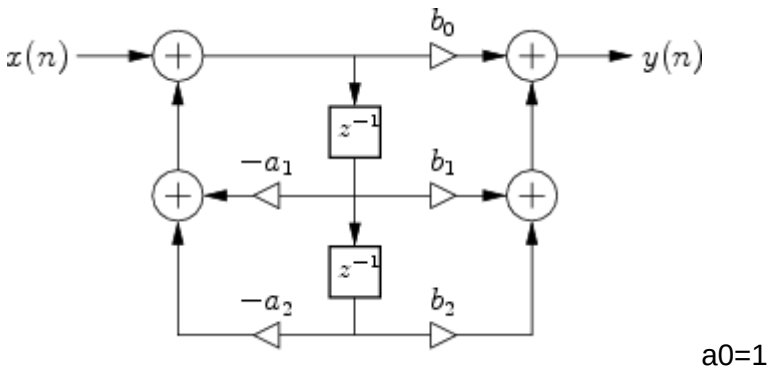
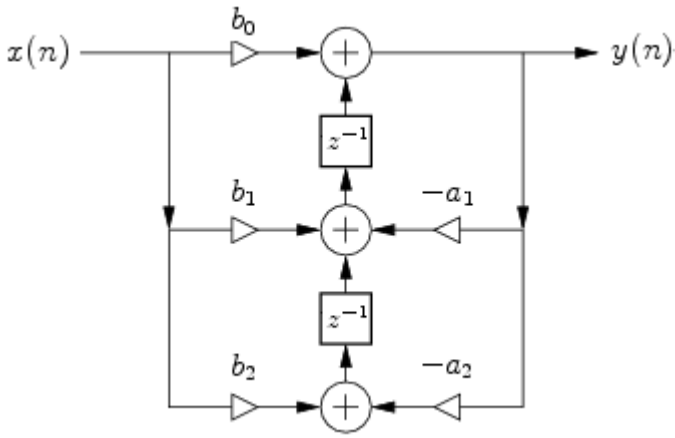


Fig. 3. Forme directe II



a0=1

Fig. 4. Forme directe II transposée

Comme ce n'est pas le propos d'entrer plus en détail dans ces réalisations, je citerai simplement qu'une des différences importantes entre ces formes est la gamme des signaux internes.

5.2 EQUATIONS DE RÉALISATION

14) Choisir une des 2 structures canoniques présentées au paragraphe 5.1 et écrire sur papier les équations partielles correspondantes **sans les combiner** (on retomberait sur l'équation aux différences).

Conseils et indications: nommer les signaux internes; l'ordre des équations est important; les opérateurs z^{-1} correspondent à une variable à mémoriser (en HW ce seraient des registres); donc pendant une période d'horloge on ne peut pas "traverser" un opérateur z^{-1}; à chaque période d'horloge ces équations seront évaluées et ensuite les "registres" mis à jour.

5.3 RÉALISATION

Comme point de départ, le code d'un filtre du premier ordre vous sera donné.

15) Prendre en main le système et faire fonctionner le filtre du 1er ordre (l'annexe 2 résume quelques points concernant le système de développement : importation du code fourni, connexion de la cible, compilation et exécution d'un programme sur la cible).
16) Dans le code fourni, comprendre les fonctions qui réalisent un filtre du 1er ordre (l'annexe 1 présente un bref rappel (ou introduction) sur le codage des nombres en virgule fixe).
17) Ecrire une fonction implémentant un filtre IIR du 2è ordre (équations préparées plus haut) et la tester sur le DSP en réalisant tout d'abord un seul étage du filtre.
18) Réaliser le filtre complet et le tester : Mesurer la réponse en fréquence et comparer avec le gabarit.

6. ANNEXE 1 : BREF RAPPEL (INTRODUCTION) SUR LE CODAGE DES NOMBRES EN VIRGULE FIXE

6.1 FORMAT DES NOMBRES

Dans l'exemple de filtre fourni, les nombres sont codés de la manière suivante :

Type de nombre	Gamme	Nombre de bits	Format:	Représentation
Coefficients	[-2 .. ~2]	16	Q14	SB.BBBBBBBBBBBBBB
Signaux	[-1 .. ~1]	16	Q15	S.BBBBBBBBBBBBBBB

Qn signifie que le nombre comporte n bits après la virgule.

B et S représentent respectivement un bit et un bit de signe.

6.2 CONSÉQUENCES

L'unité arithmétique du DSP, ne s'embarrasse pas de virgules et calcule avec des nombres entiers, donc :

- Les coefficients doivent être définis comme le coefficient réel multiplié par 2^14.
- Lorsqu'on multiplie en virgule fixe un signal 16 bits par un coefficient 16 bits codé en format Q14, on multiplie le signal par 2^14 fois le coefficient réel et le résultat est sur 32 bits. Pour retrouver le résultat à la position correcte sur 16 bits, il faut alors le diviser par 2^14, c'est à dire le décaler de 14 bits vers la droite. On peut alors supprimer les 16 bits de poids fort. A noter que ce décalage va entrainer un arrondi, qui se traduit comme un bruit sur le signal (bruit de quantification).
- Nous utiliserons donc des entiers 16 bits et des entiers 32 bits qui, pour le C5505, seront représentés respectivement en C par les types short et long.

Ainsi l'opération temp1=coeff1*sig1-coeff2*sig2, dans laquelle toutes les variables sont des entiers 16 bits, s'écrit de la manière suivante en C :

```
temp1 = (((long)coeff1 * (long)sig1) >> SHIFT) - (((long)coeff2 * (long)sig2) >> SHIFT);
```

A noter les "typecast" qui sont indispensables en C pour une compilation correcte.

Il est aussi possible d'effectuer la soustraction sur 32 bits et de faire la mise en échelle après.

```
temp1 = ((((long)coeff1 * (long)sig1) - ((long)coeff2 * (long)sig2)) >> SHIFT);
```

6.3 DÉPASSEMENT DE L'ARITHMÉTIQUE

Il n'y a pas de risque de dépassement pour la multiplication dans notre cas, car la sortie du multiplicateur comporte 32 bits.

Par contre, les additions et les soustractions sont sujettes aux dépassements car les opérandes et le résultat comportent le même nombre de bits. Ces dépassements peuvent se produire aux nœuds entre les biquads, mais aussi aux nœuds internes des biquads et lors d'additions partielles. Afin d'éviter tout dépassement dans le filtre, il faudrait idéalement, adapter l'échelle des signaux tout au long du filtre.

Différentes méthodes (normes) existent pour ceci. Une façon de faire assez simple consiste à se baser sur la réponse en amplitude en chaque nœud. Cependant, pour des raisons de temps à disposition dans le cadre de ce laboratoire, nous nous contenterons de limiter l'amplitude des signaux d'entrée de manière empirique.

7. APPENDIX 2 (ENGLISH ONLY): WORKING WITH CODE COMPOSER STUDIO (CCS) (ECLIPSE)

7.1 IMPORTING THE EXAMPLE PROJECT INTO CCS WORKSPACE

1) Copy the whole given directory to the local disk (project + lib + include).
2) CCS Menu: File -> Import -> Code Composer Studio -> Existing CCS Eclipse Projects -> Next

   Under "Select search-directory" enter the path of the directory in which you just copied the project => The folder and sub folders are automatically scanned to discover projects.

   Select the desired project (IIR_FILTER…). Do not check "Copy Projects into the workspace".

   Click Finish.
3) Make sure C5505_USBStick_ccs5.ccxml is the Active Target Configuration (else right click…)

Remark: If the Project Explorer window is not visible, select CCS Menu: View -> Project Explorer.

7.2 MAKING THE CONNECTION BETWEEN TI DEBUGGER AND TARGET

1) Plug the TMS320C5505 eZdsp USB Stick into a USB port (if a driver needs to be installed, wait until completion).
2) CCS project Explorer: right click on project -> Debug As -> Code Composer Debug Session or CCS Menu: Run-> Debug (F11).

   Note that the program will be automatically rebuilt and loaded into target.

Remark: If the debug window is not visible, select CCS Menu: View -> Debug.

7.3 BUILDING, LOADING AND RUNNING A PROGRAM

1) CCS Menu: Project -> Build or Right click on project -> Build Project.

   Note that the program will be automatically reloaded if the debug session is active.

2) CCS Menu: Run -> Resume (F8) or click on Resume icon.

To stop a program, click on the icon with the "pause" symbol (suspend). Clicking on the icon with the "stop" symbol terminates the debug session.