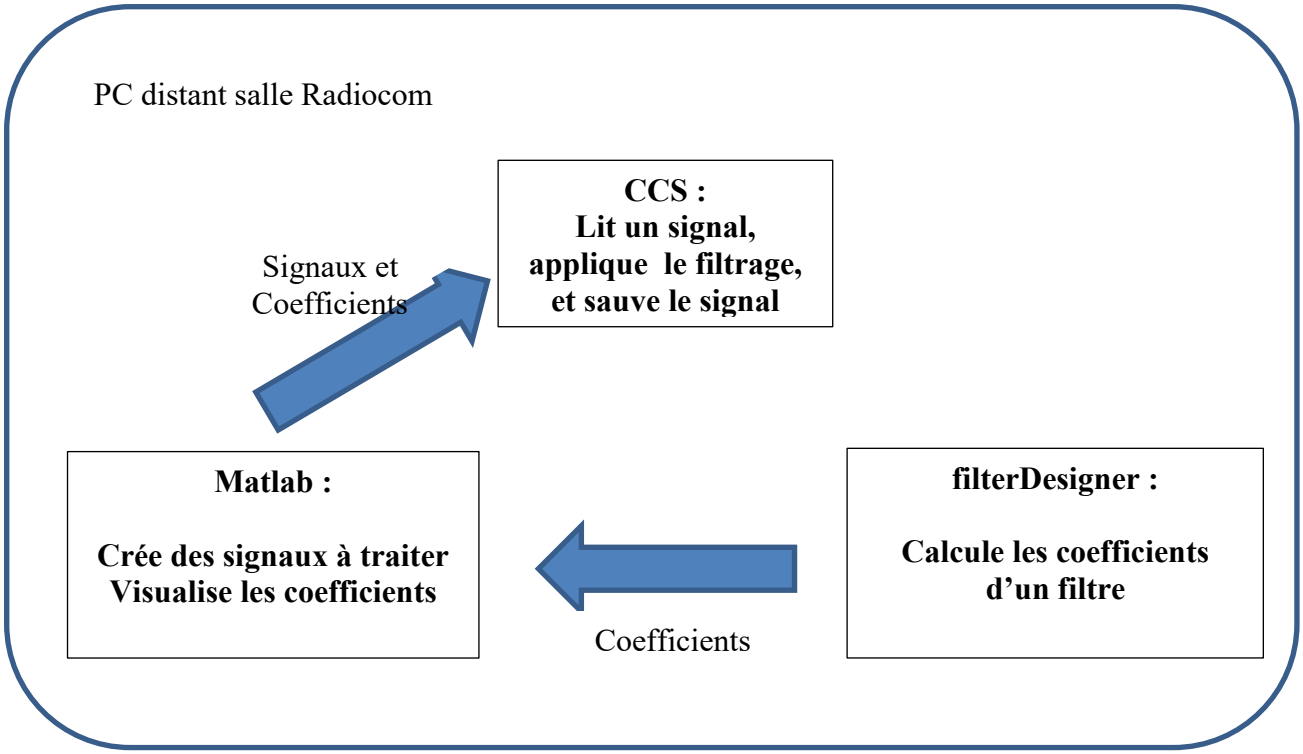
TP 7/8 : Conception et implémentation de filtres sur le DSP 6747 de Texas Inst. OMAP137

Préparation

1. Réviser un minimum le cours de TSA (notamment le poly 1).
2. Revoir le principe du buffer circulaire
3. Revoir le principe des méthodes OLS et OLA (poly 2)
4. Procéder à la création de votre environnement de travail (créer les répertoires sur C :)

Environnement logiciel de travail



Objectifs :

Il y en a 3 :

1. Il s'agit de concevoir et d'implémenter un filtre en mettant en œuvre le système de développement de l'OMAP137. 2 projets sont fournis, contenant le code de base pour réaliser le filtrage. Dans le contexte de travail à distance sur les cartes DSP les signaux manipulés sont dans des fichiers qu'il faudra lire, traiter, puis sauvegarder les résultats également dans des fichiers.

Pour cela 3 outils logiciels seront utilisés : **Code Composer Studio** pour écrire, exécuter et débugger les programmes, **filtreDesigner** pour trouver les coefficients des filtres recherchés et **Matlab** pour créer/manipuler des signaux.

Chaque outil a son répertoire de travail, donc bien vérifier où vous travaillez, sinon des erreurs vont apparaitre.

2. Programmer l'utilisation d'un buffer circulaire dans un code de filtrage échantillon par échantillon
3. Programmer un filtrage en mode trame en gérant les effets de bords par la méthode OLS ou OLA

Manipulations :

Ce sujet est celui des TP7 et 8, donc sur 2 séances de 4h

L'installation de l'environnement étant faite, Récupérer et décompresser les projets TP7-projet1 et TP7-projet2. Lancer ensuite CCS et déclarer le workspace puis importer les projets TP7-projet1 et TP7-projet2 (project/importing existing CCS eclipse project, puis choisir le répertoire où vous avez stocké les projets, afin que CSS crée les projets dans votre espace de travail et cocher : copy into workspace).

Attention : Les 2 projets changeront de noms une fois chargés (resp. tp6-projet1 et TP6-V1), ne pas en tenir compte.

I. tp6_projet1- : Programme FIR avec lecture du signal depuis un fichier

A. Prise en main des programmes et signaux à utiliser

1. Pour commencer, et avant de lancer toute exécution, analyser les programmes main.c , utility.c et fir_basic.c et repérer les différentes fonctions qui y sont implémentées. Analyser les fichiers entête filter_coefficients.h et filter_algorithme.h. L'objectif est essentiellement de repérer les différentes fonctions, et comprendre ce qu'elles font.
2. Tracer un schéma fonctionnel montrant les différentes étapes de l'exécution de ce projet (quelles fonctions, quelles données échangées, etc.)
3. La première étape dans ce projet est de prendre connaissance des signaux sinusoïdaux qui vous sont fournis (s_*.txt), et d'en créer d'autres. Le but étant de matérialiser la notion de signal numérique et de fréquence normalisée

   Repérer où est donné dans le « main » le nom du fichier qui contient le signal à traiter et qui est lu par le programme de filtrage. Modifier le chemin pour que les signaux soient lus depuis le répertoire où ils se trouvent. Bien mettre un double antislash (\\) dans le chemin.

Pour chacun des signaux qui vous sont donnés :

- Le lire avec le programme « main », en donnant le nom du signal à lire
- Afficher le signal avec CCS en temporel et en fréquentiel

CCS vous permet de visualiser les signaux en temporel et en fréquentiel (choisir Tool/Graph/(Time ou FFT) donner la taille du buffer à visualiser : 200, donner le type de données :32bits Floating point, et l'adresse de début du signal (que vous trouvez en visualisant les variables sous débuggeur).
Attention1 : une erreur d'une position mémoire donnera des résultats complètement aberrants.
Attention 2 : Il faut visualiser la mémoire avant que celle-ci ne soit libérée, donc avant la fin du programme. Il convient de mettre un point d'arrêt à la ligne 26 du « main » si on veut juste visualiser le signal d'entrée, et à la ligne 37 pour le signal de sortie.

- Visualiser le contenu du fichier à l'aide du bloc-note et vérifier que les valeurs hexadécimales enregistrées en mémoire sont bien celles données par bloc-note (on pourra utiliser L'outil http://babbage.cs.qc.cuny.edu/IEEE-754/index.xhtml qui permet de convertir simplement un code en virgule flottante en sa valeur décimale)
- Vérifier que la formule donnant la fréquence normalisée d'un signal : fnorm=1/No est bien valide, No étant le nombre de points pour une période.

4. Vous allez à présent créer des signaux carrés pour différentes fréquences (normalisées) à l'aide de Matlab. On vous donne le programme **sig_carre.m**. Il faut le modifier pour créer le signal voulu (Attention la première valeur écrite dans le fichier correspond au nombre de points). Créer ainsi des signaux carrés aux fréquences normalisées 0.01, 0.05, 0.1 et 0.2.
   Avec Matlab, visualisez les signaux et leur spectres (plot(sig) et plot(abs(fft(sig))) et vérifiez qu'ils sont conformes à vos attentes (le script **lect_affichage.m** vous aide pour cela)
   Ceci vous sera également utile pour la question 4 de la partie B.
   Faites attention au choix du répertoire de travail de Matlab, pour que CCS puisse lire vos signaux.

L'idée principale est de vous amener à bien comprendre le lien entre fréquence normalisée et fréquence vraie, et que cette dernière devient effective, lorsqu'on décide du temps séparant 2 échantillons successifs.

Ainsi, un même vecteur (signal), correspond à plusieurs fréquences réelles, selon la fréquence d'échantillonnage, c'est-à-dire la vitesse avec laquelle on lit le vecteur

B. Filtrage de signaux

1. Ouvrir le projet **tp6_projet1** et prendre en entrée le signal s_0.0078.txt, de fréquence normalisée 0.0078 (donc très basse fréquence), et utiliser le filtre passe bas de fréquence de coupure Fc=0.1 et qui a 6 coefficients.

    a. Exécuter le projet **tp6_projet1** en pas à pas (touche F5 instruction par instruction) et vérifier à chaque pas les valeurs des variables (indice, nom de fichier, valeur de signal, … .Penser à placer des points d'arrêt pour sauter les boucles et aller plus vite puis arrêtez-vous à la ligne 37 (libération de la mémoire)

    b. Tracer dans CCS les signaux d'entrée et de sortie et commentez.

2. Vous disposez de 8 signaux sinusoïdaux et de 6 filtres (3 passe bas, 2 passe bande et 1 passe haut). Pour différents filtres et signaux, prédire le résultat attendu par le filtrage puis vérifiez que celui-ci est cohérent par rapport à la fréquence du signal et la nature du filtre (on ne fera que les combinaisons pertinentes…)

3. Nous allons ici, vérifier que la réponse impulsionnelle est bien constituée des coefficients du filtre. Pour cela, créer une impulsion (avec le programme **impulsion.m**) et prendre ce signal en entrée.
   On analysera la réponse impulsionnelle de 2 façons :
   - Visualiser avec CCS le signal de sortie après filtrage (donc la réponse impulsionnelle du filtre) et vérifier que l'on trouve bien les coefficients du filtre utilisé.
   - Ouvrir le fichier out_impulsion.txt avec bloc-note, et vérifier que l'on trouve bien les coefficients du filtre utilisé.

   Enfin, vérifier ensuite si la réponse en phase est linéaire ou pas, en cohérence avec la nature de la réponse impulsionnelle.

4. A présent, nous allons travailler avec un signal carré, et le but sera d'éliminer une ou des harmoniques. On procédera de la manière suivante :
   Pour chacun des scénarios suivants :
   - supprimer le fondamental
   - supprimer le premier harmonique

- supprimer tous les harmoniques
  vous devez :
  a. Choisir parmi les signaux carrés créés précédemment, celui qui convient le mieux à notre étude (ça demande un peu de réflexion…)
  b. Utiliser FilterDesigner pour créer le filtre FIR correspondant
  c. Passer par Matlab pour récupérer les coefficients du filtre
  d. Copier les coefficients dans coefficient.h (attention à la syntaxe : le script **coeff_for_CCS.m** vous aide pour cela)
  e. Exécuter le programme et vérifier les résultats. Il est probable que ça ne marche pas du premier coup, auquel cas, il faut remodifier le filtre…
  f. Remplir les lignes du tableau suivant **et faire valider par l'enseignant**. Le tableau devra être déposé sous Moodle à la fin du TP.

Objectif	Signal choisi	Type de filtre	Ordre	Fréquences de coupure	Commentaires
Supp. Fondamt.					
Supp. 1° Harm					
Supp. Tous Harm					

II. tp6_projet1 : Utilisation de l'adressage circulaire

Il faudra compléter le programme fourni, pour travailler avec un buffer circulaire.

Nous ne travaillons pas en assembleur, et il n'y a pas d'utilisation des registres du DSP (AMR), mais juste une gestion des pointeurs du buffer en C.

1. Compléter le programme fir_circular.c pour implémenter la gestion d'un buffer circulaire. Une fois terminé, **faire valider par l'enseignant**. Le code devra être déposé sous Moodle en fin de TP
2. Comment peut-on vérifier que le programme fonctionne correctement ? Mettre ceci en œuvre.

III. TP6_V1 : Programme FIR en mode trame avec OLS ou OLA

Dans cette partie on passe à un filtrage en mode trame. Nous allons donc éliminer les effets de bords qui vont apparaitre par la méthode OLS ou OLA.

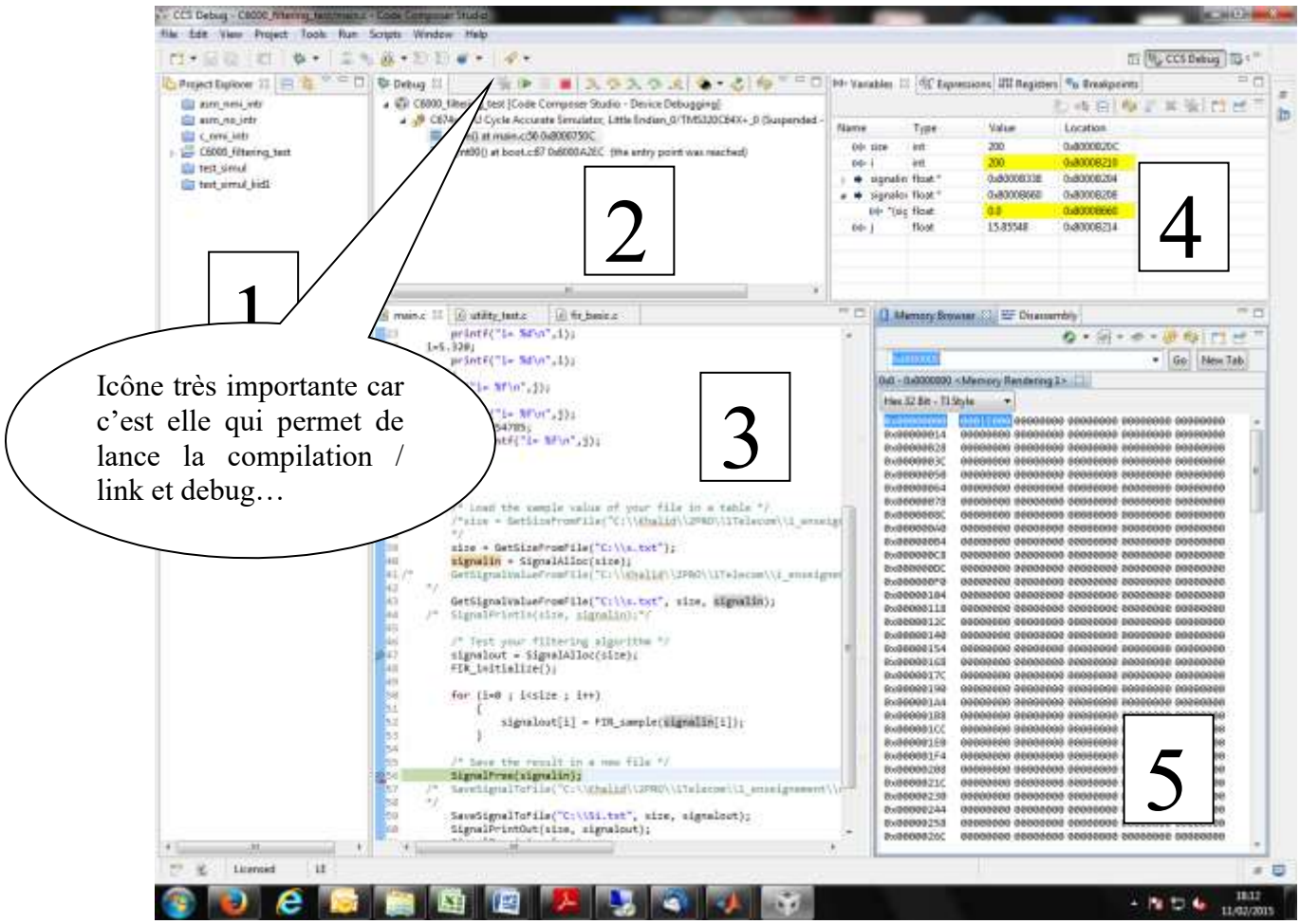
Choisir un couple (signal / filtre) adéquat.

1. Le corps du programme est donné. Vous devrez le compléter pour implémenter les méthodes OLS et OLA

   Pour chacune des 2 méthodes, une fois terminée, **faire valider par l'enseignant.** Le code devra être déposé sous Moodle en fin de TP
2. Vérifier que les résultats obtenus pour le signal et le filtre choisis sont identiques à ceux obtenus précédemment en mode échantillon par échantillon. Faire valider par l'enseignant.

Les fenêtres de l'interface graphique de CCS

Fenêtre 1 : gestionnaire de projet (ouvrir / fermer / activer / etc.)
Fenêtre 2 : le debugger (lance / arrêter / exécuter pas à pas avec F5 ou F6 / etc.)
Fenêtre 3 : éditeur de fichier (voir et modifier les sources / suivre l'exécution pas à pas / etc
Fenêtre 4 : Pour visualiser les registres / les variables / des expressions / etc.
Fenêtre 5 : pour visualiser la mémoire / pour voir le code assembleur correspondant au code machine (disassembly)
D'autres fenêtres peuvent être utilisées, notamment celle qui permet de tracer un signal, son spectre, etc…



Exécution d’un projet sous CCS :

Vous pouvez, en cliquant sur l’icône du debug, lancer la compilation, l’édition de lien et le debugger. Ainsi vous pouvez commencer l’exécution du programme soit en pas à pas (touche F5), soit en lui demandant de s’exécuter jusqu’au prochain point d’arrêt (si vous en avez créé, en vous plaçant là ou voulez en mettre un, puis bouton droit de la souris et choisir Toogle Breakpoint).

Dans la fenêtre 3 vous voyez votre programme, et vous voyez l’exécution en pas à pas. En vous positionnant sur une variable, vous pouvez en voir la valeur.

Dans la fenêtre 4, vous pouvez visualiser le contenu des registres, les valeurs des variables, etc., Dans notre cas, c’est là qu’il est possible de récupérer les adresses des variables, et notamment des signaux d’entrées et de sorties pour pouvoir les visualiser grâce à l’outil Graph.

Dans la fenêtre 5, vous pouvez voir le contenu de la mémoire. Il suffit pour cela de donner l’adresse à partir de laquelle on veut visualiser la mémoire**. ATTENTION : En TD, il a déjà été montré comment les données sont rangées en mémoire, et plus exactement, la manière avec laquelle la mémoire est affichée par le CCS :**