Institut Supérieur des Etudes Technologiques en Communications de Tunis

Guide de l'élève

Module

PRATIQUE DES DSP – AD BLACKFIN

Promotion

Mastère : Smart'Com

Enseignant

Lassaad JEMAI

© ISET'COM - Tunis – Janvier 2016

Présentation du module

Le module « Pratique des Processeurs Numériques du Signal » est destiné aux élèves du Mastère SMART'COM de L'ISET'COM et se présente sous la forme d'un cours/TP. Le premier objectif de ce module est de familiariser les élèves à l'architecture et au jeu d'instructions d'un DSP 16 bit de la famille Blackfin d'Analog Devices puis d'exploiter un environnement de développement logiciel et matériel intégré (IDE) pour implanter des fonctions de traitement de base puis des fonctions avancées de traitement de signal et de communications numériques.

Ce module est programmé au 2ème semestre des études de mastère SMART'COM à ISET'COM compte tenu de la nécessité d'un certain nombre de modules pré-requis notamment en traitement du signal, électronique numérique, microprocesseurs, théorie de l'information et communications numériques.

De point de vue structuration, ce module est scindé en différentes parties qui ont trait à l'architecture et l'environnement de développement du processeur AD-BF533 ainsi que des travaux d'application permettant aux élèves-ingénieurs de bien manipuler le DSP ainsi que son environnement de développement pour mettre au point des modules DSP de certains algorithmes des applications de télécommunications.

De point de vue organisation, les activités du module de TP sont réparties sur un volume horaire de 21 Heures soit 7 séances de 3 Heures chacune. Les quatre premières séances sont des travaux pratiques et les trois dernières séances sont réservées au développement d'un projet d'application.

La 1ère séance sera consacrée à la familiarisation avec l'environnement de développement intégré VISUAL DSP++. Les activités de cette séance consistent à effectuer des présentations, avec la manipulation en parallèle par les élèves, des différentes étapes de création d'un projet DSP avec le test de quelques fonctions élémentaires. A partir de la 2ème séance, l'initiative est laissée aux élèves-ingénieurs qui ayant disposé des connaissances de base suffisantes et moyennant toute la documentation et les ressources de développement nécessaires ils sont appelés à mener dans le cadre d'un travail d'équipe trois travaux d'application d'implantation sur DSP.

Le travail demandé aux élèves est réparti en trois TP dont les détails sont fixés par des cahiers des charges. Le 1er TP concerne l'utilisation des unités de traitement du Blackfin pour effectuant des opérations de base sur des tableaux. Le 2ème TP vise à développer et implanter un programme DSP pour filtre FIR. Enfin le 3ème TP d'application vise l'étude, le développement et le test d'une application de communication relative à un modulateur DMT.

SOMMAIRE

Partie 1

Présentation et Initiation à L'environnement VDSP++

1.1 Ressources logicielles de développement sur DSP

1.1.1 Ressources de compilation

Afin de faciliter la programmation de ses DSP, Analog Devices a mis au point un environnement logiciel de développement complet, fiable et d'utilisation souple. Cet environnement permet de supporter les tâches d'édition de code source en langage "C" ou en assembleur, la compilation, l'édition des liens, la simulation système et la génération de code en format compatible à la mémoire système. Le développement se fait selon un ordre chronologique en partant d'un code source en assembleur ou en C pour générer un code exécutable près à être chargé sur le DSP cible.
Les fonctions et les caractéristiques de ces outils de développement logiciel sont les suivantes :

- ***System Builder :*** Il permet la génération de fichier de description d'architecture qui fournit ainsi des informations concernant le système hardware cible. Cette description inclue la spécification de la taille de la mémoire externe utilisée ainsi que la configuration des ports d'entrée/sortie et des mémoires additionnelles. Le fichier architecture est utile pour l'éditeur de liens, le simulateur, et l'émulateur.

- ***Assembler :*** L'assembleur a la particularité de supporter un jeu d'instructions avec une syntaxe de haut niveau. L'assembleur DSP fournit des "*Macros*" flexibles et des librairies des fichiers "*Include*" pour le développement de programmes plus structurés et plus performants.

- ***Compilateur C/C++ :*** Le compilateur C/C++ lit un code source ANSI C/C++ et fournit à sa sortie un code source AD-DSP assemblé. Le compilateur C contient des librairies adaptées à l'environnement de AD-DSP et qui peuvent assurer des routines DSP.

- ***Linker :*** Il assure l'édition des liens entre les modules d'un programme assemblés séparément. Le linker organise le code source et les données de sortie du système hardware cible selon la spécification faite au niveau du "System Builder".

1.1.2 Simulateur Système

Il s'agit d'un outil logiciel qui permet de simuler le fonctionnement interne du DSP afin de tester et valider les codes sources développés avant de passer à la phase d'implantation sur le système cible. Cet outil de simulation système reproduit le fonctionnement réel de toutes les unités du DSP car sa modélisation a été effectuée en tenant compte de l'architecture spécifiée par l'utilisateur à l'aide du "System Builder". De point de vue exploitation, cet outil offre au programmeur une souplesse d'utilisation. En effet, il dispose d'une interface utilisateur permettant au concepteur d'observer et d'altérer le contenu des registres, des cases mémoires, et offrir ainsi un environnement de débuggage puissant. Les possibilités de traitement offertes par le simulateur incluent:

- la simulation de mémoire données (DM) et de mémoire programme (PM),
- la simulation de cases mémoire configurées comme des ports d'Entrées/Sorties,
- la simulation des interruptions,
- le profil d'exécution du code programme,
- la reconfiguration des fenêtres.

1.2 Environnement Visual DSP ++

L'environnement de développement intégré (IDE) Visual DSP++ est un gestionnaire de projet et un environnement de debugage pour des processeurs de traitement de numérique du signal d'Analog Devices. C'est un environnement qui fournit un contrôle graphique complet des différentes étapes de création d'un projet, d'édition des codes sources, d'assemblage, d'édition des liens et de l'ensemble des processus de debugage.

1.2.1 Caractéristiques de l'environnement Visual DSP++

L'environnement Visual DSP++ est un gestionnaire de projet flexible, à interface graphique conviviale (figure 1.1), pour les applications DSP permettant une multitude de fonctionnalités dont notamment :

- La création de projet, la compilation, l'édition de liens
- La simulation de l'exécution
- L'émulation de niveau élevé

L'IDE VDSP++ supporte les différents DSP d'ADI et principalement les DSP Blackfin de référence composant: BF531, BF532, BF533, BF535.

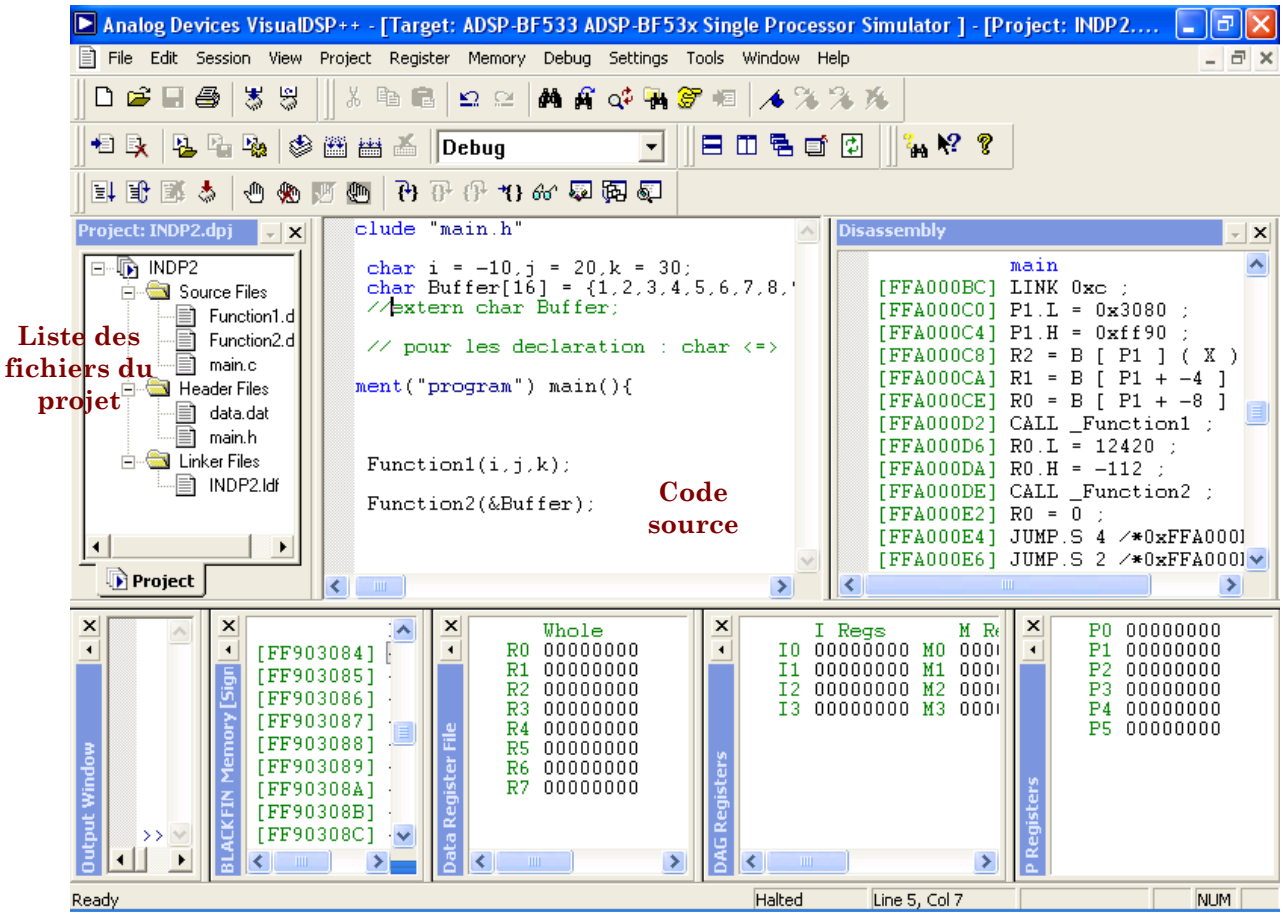


Figure 1.1 : Interface graphique du VDSP++

Un projet dsp est composé d'au moins les trois fichiers suivants:

- Un fichier projet «***.dpj***» comme fichier de gestion du projet
- Un fichier architecture «***.ldf***» pour décrire l'architecture du processeur et la répartition des zones mémoires.
- Au moins un fichier source «***.c***» ou «***.dsp***» ou «***.asm***» constituant le noyau du projet et contenant les instructions du programme.

L'interface est munie d'un menu déroulant et un ensemble de barres d'outils dont les boutons concernent les opérations les plus utilisées. On y trouve également un ensemble de fenêtres dont :

- La fenêtre projet : elle affiche toutes les ressources du projet organisées en dossiers, fichiers sources (C et dsp), fichiers headers (*.h) et le fichier d'architecture (*.ldf).
- La fenêtre d'édition.
- Une ou plusieurs fenêtres pour les ressources DSP (Mémoire, registres données, registres pointeurs, …)
- La fenêtre Output qui affiche un compte rendu sur toute opération effectuée (compilation, build, load, recherche, …).

L'environnement Visual DSP++ permet de mener les activités suivantes nécessaires pour créer et debuger un projet DSP:

- **Edition de fichiers texte**

A partir de l'interface graphique de l'environnement Visual DSP++, on peut créer et modifier un fichier source ou visualiser les fichiers générés par les outils DSP. Les fichiers sources peuvent êtres écrits en C/C++ ou en assembleur. Il y a d'autres fichiers tel que les fichiers de données ou les fichiers (*.ldf)* qui contiennent des commandes utiles pour l'éditeur de liens.

- **Accès et gestion des outils**

L'environnement Visaul DSP++ fournit une interface utilisateur à travers la quelle on peut spécifier les options pour les outils utilisés. On peut définir ces options une fois pour toutes ou les adapter selon les besoins.

- **Visualisation des résultats de compilation**

L'environnement Visual DSP++ offre la possibilité de visualiser les résultats du processus de compilation et de l'arrêter si nécessaire. En cliquant sur le message d'erreur dans la fenêtre de sortie on peut voir l'erreur correspondante dans le fichier source.

- **Debugage :**

Une fois le processus de compilation est terminé avec succès on peut lancer le debugage du programme écrit.

1.2.2 Opérations sur un projet

Le terme projet se réfère à une collection de fichiers source et des outils de configurations utilisés pour la création d'un programme DSP.

a. Création d'un nouveau projet

Pour créer un nouveau projet il faut sélectionner, dans le menu de l'interface graphique (figure 1.1), la commande ***new*** sous le menu ***File*** puis ***project***. Une fenêtre de sauvegarde s'affiche pour préciser le lieu et le nom du projet dont l'extension est «*.dpj* ». Une fois le projet créé un panneau de configuration de projet s'affiche (Figure 1.2). Ce panneau permet de sélectionner les options de tout le projet en choisissant par exemple l'architecture cible et les options de compilation.

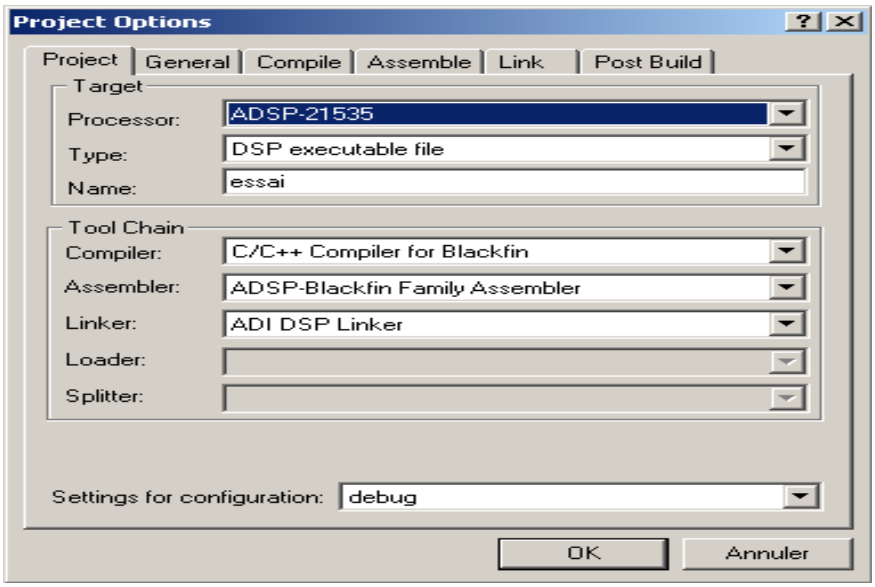


Figure 1.2 : Panneau d'options de projet

b. Ajout de fichiers au projet

Un projet doit au moins contenir un fichier «*.ldf* » nécessaire à l'étape d'édition de liens et un ou plusieurs fichiers sources. Cette opération peut être réalisée par la commande ***add to project*** sous le menu ***Project***.

c. Compilation et debugage d'un projet

Une fois le projet est complet il faut passer à la phase de compilation grâce à la commande ***Build project*** sous le menu ***project***. Si l'opération se réalise avec succès il faut vérifier le fonctionnement du programme avec la commande ***debug*** sous le menu

project. L'environnement de debugage donne un aperçu sur l'état des différents registres du processeur, de la mémoire programme (PM), de la mémoire données (DM), des générateurs d'adresses, des interruptions, … (Figure 1.3).

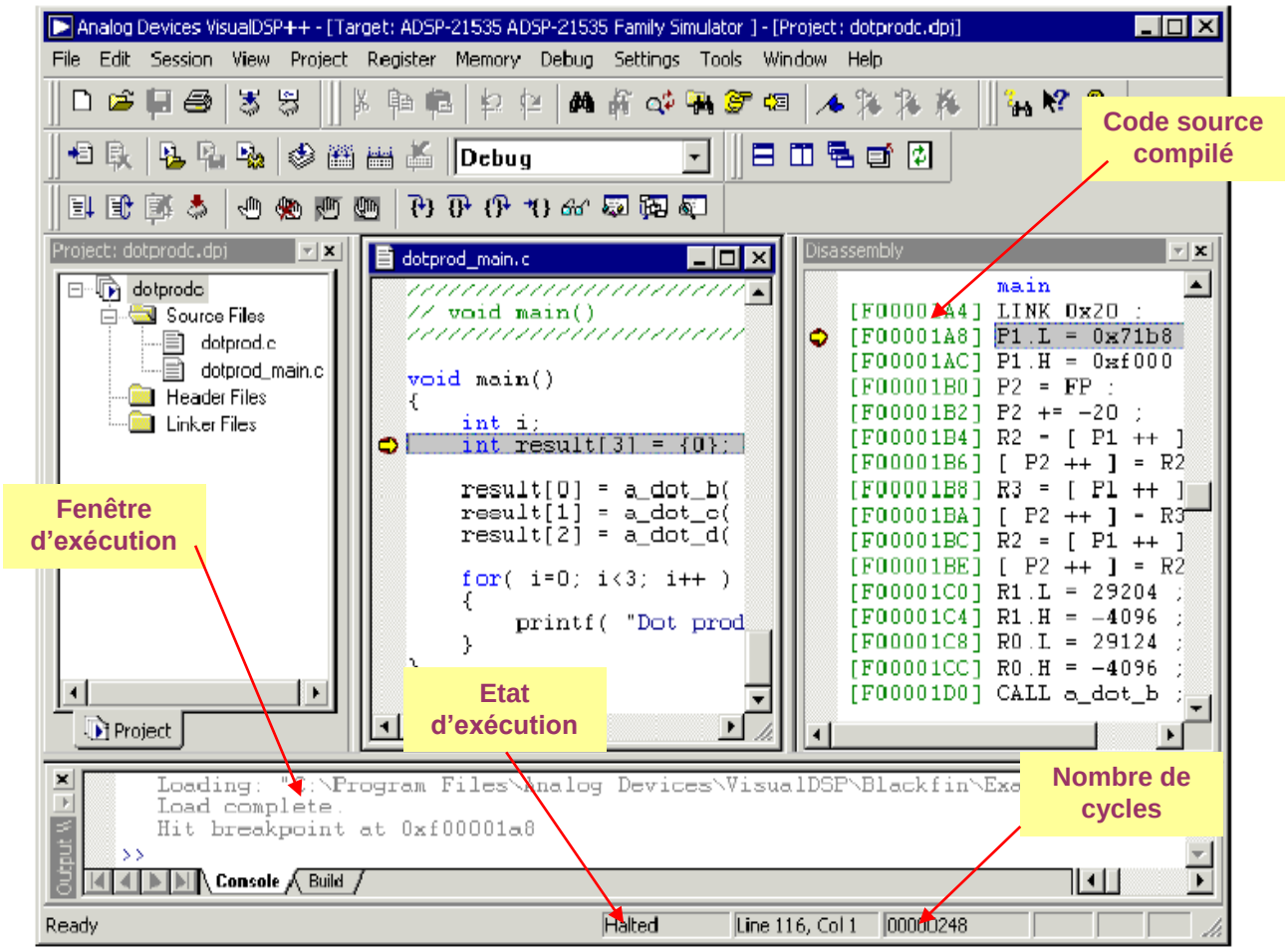


Figure 1.3 : Interface graphique de debugage du VISUAL DSP++

d. Emulation et chargement

La même interface peut servir pour les tâches d'émulation sur les cartes DSP. En effet, VISUAL DSP++ permet le chargement direct des programmes dans la mémoire interne du composant DSP. L'environnement Visual DSP++ permet aussi le chargement des programmes dans le DSP à travers un émulateur (ICE: In Circuit Emulator). Ce dernier permet le suivi de l'exécution du programme en temps réel en décrivant l'état des différents registres et mémoires du processeur comme dans le cas de la simulation.

1.3 Exemple de projet d'application

Pour se familiariser avec l'environnement VDSP++, nous proposons un projet *INDP2.dpj* qui comprend trois fichiers sources :

- ***tp0.c*** *:* programme principal qui contient la déclaration de trios tableaux ***x[8], y[8]*** **et** ***z[8]*** et appelle une fonction ***my_c_sum*** qui réalise ***z=x+y*** *;*
- ***my_c_funtion.c*** *:* contient le code C de la fonction ***my_c_sum***.
- ***my_asm_function.asm*** *:* contient le code assembleur de la fonction ***my_asm_sum***.

Il est demandé d'effectuer les étapes suivantes :

1- Créer un projet INDP2.dpj en choisissant le dsp ADSP-BF533 comme processeur cible.
2- Ajouter au dossier Source Files les deux fichiers tp0.c et my_c_funtion.c.
3- Lancer Build File pour les deux fichiers source et Build Project par la suite.
4- Afficher la Fenêtre Blackfin Memory et les variables x, y et z. Choisir le format signed integer 32 bits.
5- Effectuer l'exécution step into (Touche F11) et vérifier le fonctionnement du programme.
6- Eliminer le fichier my_c_funtion.c du projet et ajouter le fichier my_asm_function.asm. Modifier alors le programme principal pour qu'il appelle m_asm_sum.
7- Refaire le Build File et le Build project.
8- Reprendre l'exécution pas à pas et suivre le déroulement de l'exécution. Pour cela afficher les fenêtres des Data Registers et des DAG.
9- Enlever la déclaration de x, y et z du programme principal et l'effectuer dans le fichier assembleur. Pour cela déclarer une section de données : .section L1_data_a et utiliser le type .byte4.
10- Refaire Build File et Build Project et l'exécution
11- Changer les types de x et y en .byte et z en .byte2. Effectuer les modifications nécessaires au code assembleur et reprendre les étapes d'exécution.

/************ tp0.c ***************/***

```
int x[8]={5,-10,115,-23,9,47,11,-123};
int y[8]={-15,-23,106,41,119,-19,20,-95};
int z[8];
void my_c_sum(void);
void main( void )
{
     my_c_sum();
}
```

/***** my_c_funtion.c ******/***

```
extern int x[8], y[8], z[8];
void my_c_sum(void)
{
int i;

for(i=0; i<8;i++)
  {
       z[i]=x[i]+y[i];
 }

}
```

/***** my_asm_funtion.asm ******/***

```
.extern _x,_y,_z;
.global _my_asm_sum;

.section L1_code;

_my_asm_sum:
nop;

I0.L=_x;
I0.H=_x;

I1.L=_y;
I1.H=_y;

I2.L=_z;
I2.H=_z;

P0=8;

LSETUP (debut, fin) LC0=P0;

debut: R0=[I0++];
         R1=[I1++];
         R2=R0+R1;
fin:   [I2++]= R2;

RTS;

_my_asm_sum.end:
```

Partie 2

Projet d'application N°1

Familiarisation avec le jeu d'instructions du DSP Blackfin

2.1 Objectif du projet

L'objectif de ce 1er projet d'application est d'apprendre aux élèves ingénieurs de bien manipuler les unités de calcul des DSP Blackfin et ceci à travers le développement et le test de codes DSP supportant quelques opérations de base de traitement arithmétique tout en faisant appel à l'utilisation des accès mémoires.

2.2 Présentation des unités de calcul du DSP Blackfin

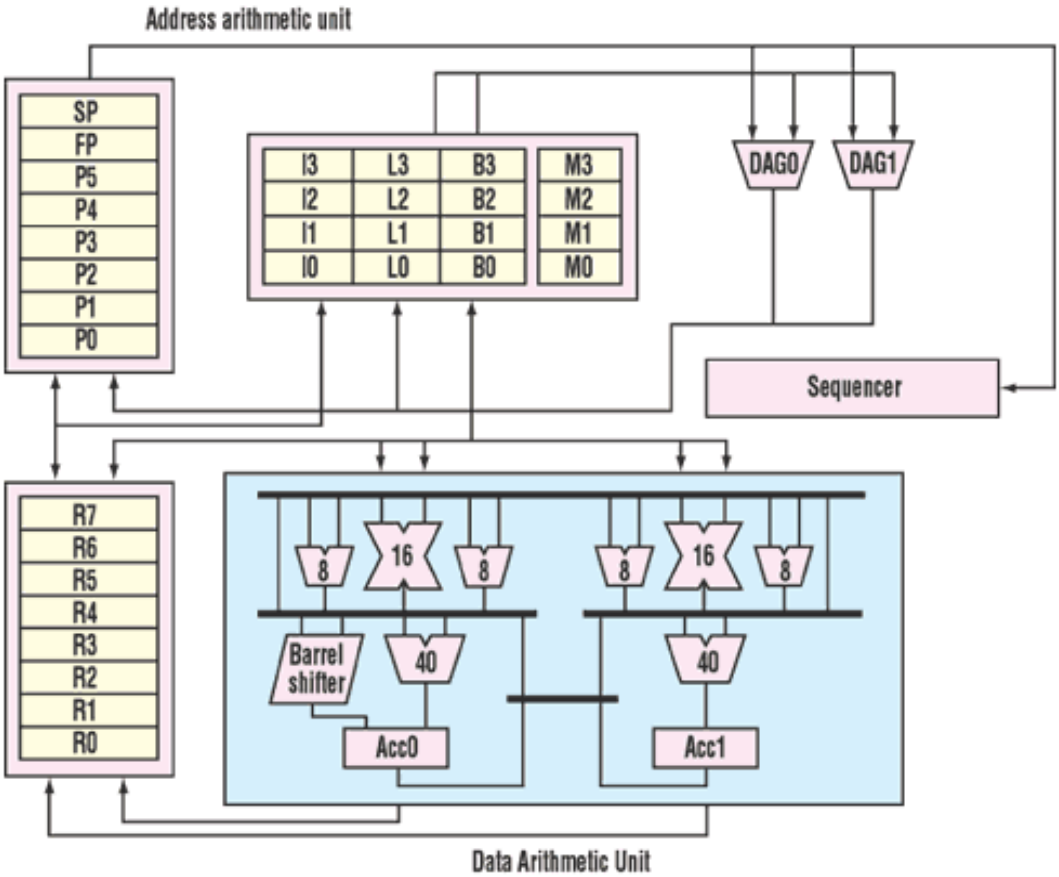


Figure 1 : architecture des unités de traitement des DSP AD-Blackfin

Comme le montre le schéma bloc de la figure 1 les DSP AD-Blackfin offre les unités de traitement de données suivantes :

- Deux multiplieurs câblés pour traitement de données sur 16 bits
- Deux accumulateurs pour traitement de données sur 40 bits
- Deux unités arithmétiques et logiques ALU pour traitement de données sur 40 bits
- Quatre ALU vidéo pour traitement de données sur 8 bits
- Un shifter pour permettre des décalages des bits des données

Ces unités de traitement assurent la manipulation des données 8, 16 ou 32 bits à partir d'un banc de registres de calcul.

2.3 Spécifications du projet

Les activités de ce projet concernent la création d'un nouveau projet dont un programme principal en C appelle une fonction DSP et qui à chaque fois réalise ce qui suit :

- Déclaration des deux tableaux *Tab1* et *Tab2* dont les valeurs sont données ci-dessous
- Comptage des valeurs paires dans *Tab1*
- Somme des éléments de *Tab1*
- Produits des éléments de *Tab1* et *Tab2* et accumulation. Distinguer les deux cas où les éléments sont *integer* ou *fractionnels*.

Tab1	*Tab2*
0xA120	0x22FF
0x 3B11	0x100B
0xC37D	0xCC54
0x98D1	0x1115
0x35CC	0x2BEA
0xD379	0xCACA
0x54B6	0x3346
0x32E5	0x77DD

4. Travail demandé

En considérant les spécifications du projet on demande de:

1. Donner vos schémas de conception des traitements de données sur les tableaux *Tab1* et *Tab2* sur DSP : organigramme et graphe flot de données en indiquant les ressources DSP utilisées.
2. Ecrire les codes source DSP correspondant à votre conception
3. Utiliser les ressources de l'environnement VISUAL DSP++ pour la compilation, l'édition des liens puis la génération du fichier exécutable.
4. Faire appel au simulateur pour analyser le fonctionnement de votre programme et vérifier les résultats de calcul obtenus.
5. Rédiger votre compte rendu de manipulation.

Partie 3

Projet d'application N°2

Implantation d'un filtre à réponse impulsionnelle finie (FIR)

3.1 Objectif du projet

L'objectif de ce 2ème projet d'application est l'étude, la synthèse et le développement de codes sources DSP d'un filtre numérique de type FIR utilisé dans des applications audio. Les codes sources développés pour ce filtre seront validés par une implantation sur le DSP (Blackfin 533) de la firme Analog Devices.

3.2 Etude des filtres numériques

Un filtre numérique est totalement décrit par son équation aux différences. Celle-ci établit la relation entre l'entrée et la sortie du filtre, comme l'illustre la relation (3.1):

$$Y(n)=1+a_1.x(n-1)+a_2.x(n-2)+ \dots + a_N.x(n-N)-b_1.y(n-1)-b_2.y(n-2) - \dots - b_M.y(n-M) \qquad \textbf{(3.1)}$$

Où les $x(n)$ sont les échantillons d'entrées et les $y(n)$ représentent la séquence de sortie. La fonction de transfert d'un filtre numérique est donnée par la relation (3.2) :

$$H(z)=\frac{1+\sum_{i=1}^{N} a_i z^{-i}}{\sum_{j=0}^{M} b_i z^{-j}}=\frac{Y(z)}{X(z)}=\frac{1+a_1z^{-1}+a_2z^{-2}+\dots+a_Nz^{-N}}{1+b_1z^{-1}+b_2z^{-2}+\dots+b_Mz^{-M}} \qquad \textbf{(3.2)}$$

On distingue deux types de filtres :

a. Etude des filtres FIR

La fonction de transfert d'un filtre à réponse impulsionnelle finie est de la forme (3.3):

$$y(n)=\sum_{k=0}^{N-1} h(k).x(n-k) \qquad \textbf{(3.3)}$$

L'équation aux différences d'un tel filtre s'écrie sous la forme (3.4) :

$$H(z)=\frac{Y(z)}{X(z)}=1+a_1z^{-1}+a_2z^{-2}+...+a_Nz^{-N} \qquad \textbf{(3.4)}$$

C'est la raison pour laquelle on appelle, également, ce type de filtre, filtre à convolution. Grâce à leur structure non-récursive, les filtres à réponse impulsionnelle finie sont toujours stables et assurent une phase linéaire tant que leurs coefficients sont symétriques. Ils sont donc tout à fait adaptés pour des applications audio en général, où la distorsion de fréquences due aux non-linéarités de phases est à bannir. La performance des filtres à réponse impulsionnelle finie est limitée par leur ordre et par la quantification de leurs coefficients, qui se traduit par une grande difficulté quant à leur implantation dans des composants à virgule fixe.

L'ordre du filtre est fonction des ondulations maximales dans la bande passante (δ_p) et dans la bande atténuée (δ_s), de la largeur de la bande de transition (Δf). L'ondulation dans la bande passante est donnée par la relation (3.5) :

$$A_p=20.\log_{10}\left(\frac{1+\delta p}{1-\delta p}\right)[dB] \qquad \textbf{(3.5)}$$

Et l'atténuation dans la bande atténuée est donnée par l'expression (3.6) :

$$A_s=20.\log_{10}(1+\delta s)[dB] \qquad \textbf{(3.6)}$$

La largeur de la bande de transition, normalisée par rapport à la fréquence d'échantillonnage d'entrée f_{sa}, est donnée par la relation (3.7):

$$\Delta f=\frac{f_s+f_p}{f_{sa}} \qquad \textbf{(3.7)}$$

Où f_p est la fréquence de coupure de la bande passante, f_s la fréquence de début de la bande atténuée. L'ordre du filtre peut être déterminé de manière approchée par la relation (3.8):

$$N=\frac{-13-20\log_{10}\sqrt{\delta p.\delta s}}{14.6.\Delta f}+1 \qquad \textbf{(3.8)}$$

Comme le montre l'équation (3.8), l'ordre du filtre dépend pour beaucoup de la largeur de la bande de transition. Plus les spécifications du filtre sont sévères, plus l'ordre du filtre sera important, ce que l'on cherche à éviter. Dans le cas d'un filtre divisé en plusieurs

étages, chaque filtre a alors ses propres spécifications. L'expression donnant l'ordre du ième filtre devient alors :

$$N_i = \frac{-13 - 20\log_{10}\sqrt{\delta p_i.\delta s_i}}{14.6.\Delta f_i} + 1 \qquad \textbf{(3.9)}$$

$$où \qquad \Delta f_i = \frac{f_{s,i} + f_{p,i}}{f_{sa,i}}$$

On voit bien alors que l'on pourra diminuer les contraintes sur chaque filtre, ce qui revient en fait à diviser la tâche en plusieurs parties.

Seul inconvénient, à fonctions de transfert équivalentes, l'ordre du FIR sera plus élevé que l'ordre du IIR et donc à profil de filtre identique, la complexité calculatoire du FIR est plus élevée.

b. Etude des filtres IIR

La fonction de transfert d'un filtre à réponse impulsionnelle infinie est de la forme (3.10):

$$H(z) = \frac{1 + \sum_{i=1}^{N} a_i z^{-i}}{\sum_{j=0}^{M} b_i z^{-j}} = \frac{Y(z)}{X(z)} = \frac{1 + a_1 z^{-1} + a_2 z^{-2} + ... + a_N z^{-N}}{1 + b_1 z^{-1} + b_2 z^{-2} + ... + b_M z^{-M}} \qquad \textbf{(3.10)}$$

Le nombre de coefficients nécessaires est plus faible pour une structure IIR. A surface égale, les produits IIR sont donc plus performants. Cependant, les techniques de taux multiples envisageables avec des filtres FIR (structure cascadé) permettent de réduire de manière sensible le nombre de coefficients. Un filtre analogique peut être très simplement synthétisée en une structure IIR, les structures FIR n'ayant pas d'équivalents analogiques. D'un point de vue logiciel, la synthèse d'un filtre FIR est plus simple et sa structure est plus stable.

3.3 Techniques d'implantation d'un filtre FIR sur DSP

La figure 1 illustre un schéma simplifié d'un filtre FIR d'ordre 4. Il est constitué par des unités de retard, correspondantes à la fréquence d'échantillonnage, des multiplieurs et des additionneurs.

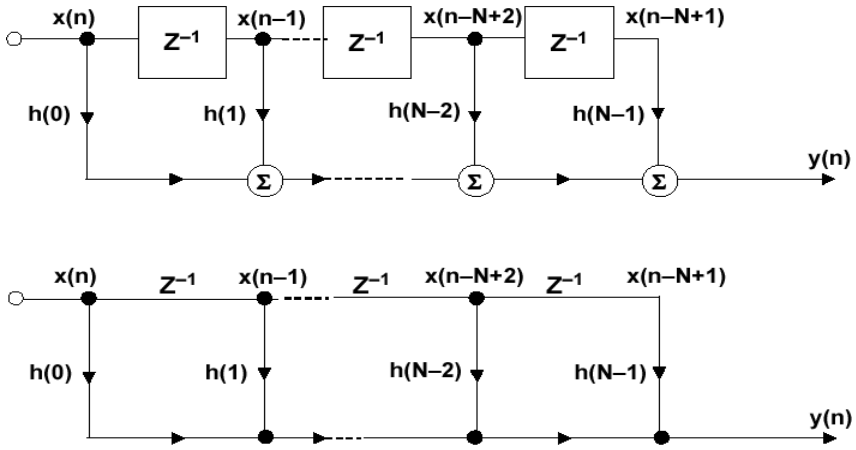


Figure 1 : architecture de réalisation d'un filtre FIR

L'algorithme utilisé pour l'implémentation du filtre est illustré dans la figure 2. Il s'agit du principe « Circular Buffering » :

- Lire les données en entrée
- Placer ces données dans une mémoire de même taille que l'ordre du filtre (4 dans ce cas)
- Charger les valeurs des coefficients
- Effectuer la convolution
- Remplacer la valeur la plus ancienne du signal par une nouvelle valeur
- Recalculer la convolution

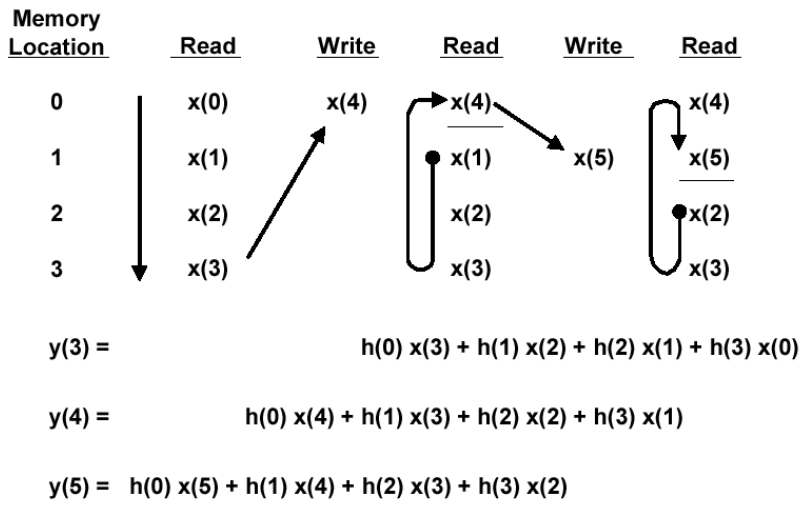


Figure 2 : algorithme DSP d'un filtre FIR

3.4 Spécifications du projet d'application

Dans une chaîne de traitement numérique d'un signal audio, on a besoin d'effectuer un filtrage passe bas pour éliminer des composantes parasites causées par le disfonctionnement du Convertisseur Analogique Numérique (ADC). La figure 3 illustre son allure temporelle, alors que la figure 4 montre son spectre en amplitude.

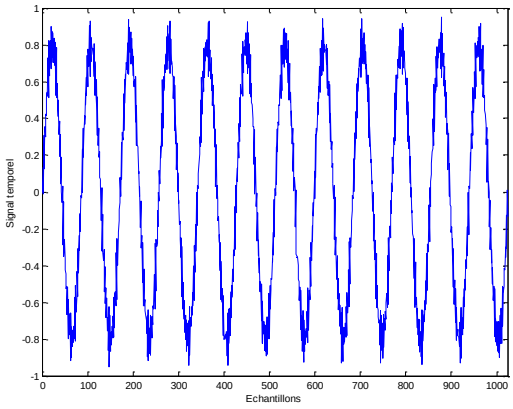


Figure 3

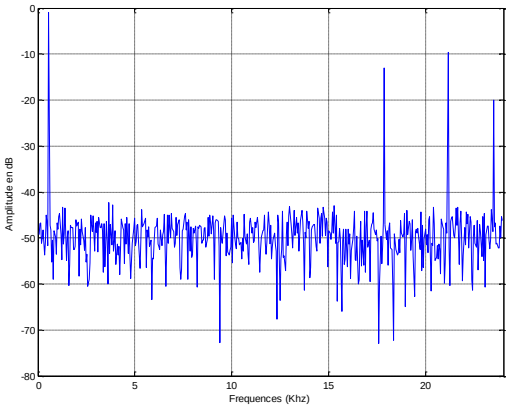


Figure 4

Avant d'effectuer un traitement sur ce signal audio, on demande d'effectuer un filtrage de type passe-bas. Notre choix s'est porté sur une topologie FIR pour sa phase linéaire. La figure 5 montre l'allure du gabarit du filtre voulu. Lors de la synthèse, on a utilisé la méthode des fenêtres (rectangulaire) et limité l'ordre à 6.

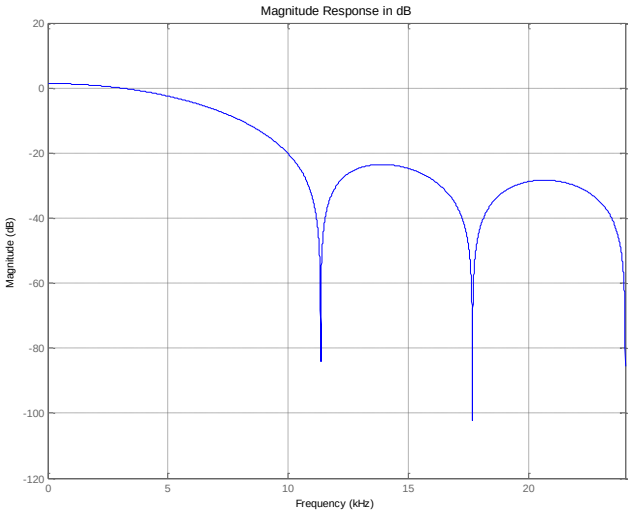


Figure 5

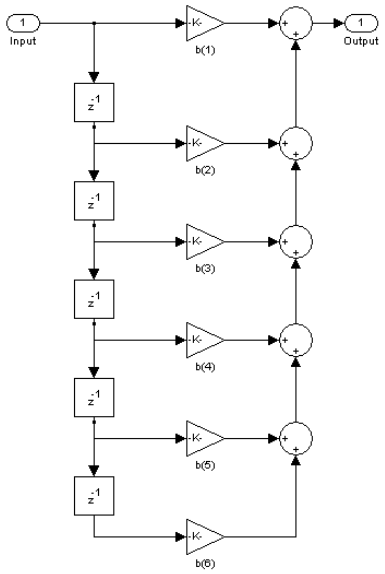


Figure 6

Les simulations sous MATLAB montrent les résultas suivants :

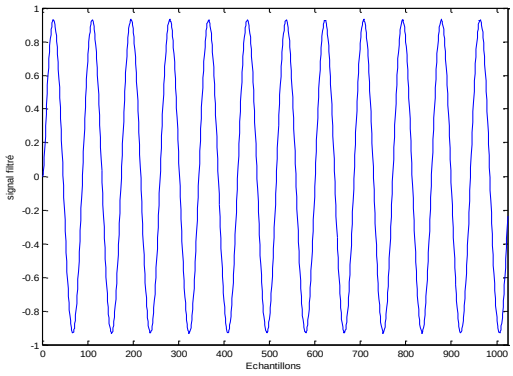


Figure 7

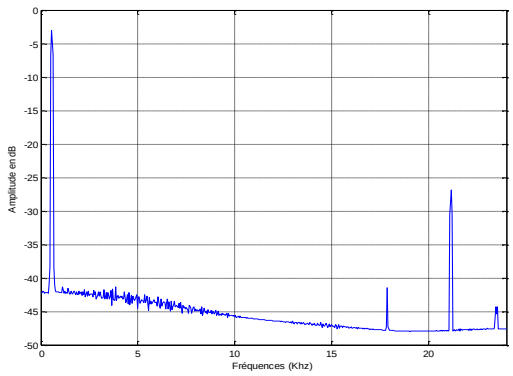


Figure 8

Les valeurs des coefficients sont :

3.5 Travail demandé

En considérant les spécifications du projet on demande de:

6. Donner vos schémas de conception du traitement de filtrage FIR sur DSP : organigramme et graphe flot de données en indiquant les ressources DSP utilisées.
7. Ecrire le code source DSP correspondant à votre conception
8. Utiliser les ressources de l'environnement VISUAL DSP++ pour la compilation, l'édition des liens puis la génération du fichier exécutable.
9. Faire appel au simulateur pour analyser le fonctionnement de votre programme et tracer l'allure des signaux temporels et les spectres fréquentiels avant et après filtrage.
10. Rédiger votre compte rendu de manipulation.

Remarque: Les échantillons du signal à appliquer à l'entrée du filtre sont fournis dans fichier *data.txt*.

Partie 4

Projet d'application N°3

Chargement des porteuses d'un modulateur DMT

4.1 Objectif du projet

L'objectif de ce 3ème projet d'application est le développement, le test et la validation du code DSP qui permet le chargement des porteuses d'un modulateur DMT à 16 voies, à partir d'un tableau qui représente la taille de la constellation QAM à utiliser pour chaque porteuse. Les codes sources développés pour cette application seront validés par une implantation sur le DSP (Blackfin 533) de la firme Analog Devices.

4.2 Etude de la Modulation DMT

La DMT (Discrete MultiTone) est une technique de modulation utilisée dans les modems ADSL. Elle se base sur une modulation multiporteuses OFDM (*Orthogonal Frequency Division Multiplex*) associée à un algorithme chargement de bits (*bit loading*).

a. Systèmes OFDM

Les systèmes OFDM sont proposés pour une transmission sur canaux sélectifs en fréquences (ce sont des canaux qui présentent un comportement différent selon la fréquence d'émission choisie). Il s'agit d'une transmission utilisant N porteuses comme le montre la figure 1.

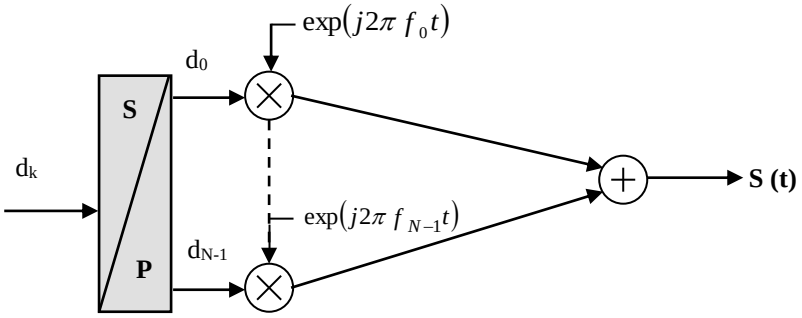


Figure 1 : Structure du modulateur OFDM

Considérons une séquence de données $\mathbf{d_0}$, $\mathbf{d_1}$, $\mathbf{d_2}$, …, $\mathbf{d_{N-1}}$ où $\mathbf{d_n}$ représente un symbole complexe obtenu après l'opération de "*Mapping*" des bits α_k tel que $\alpha_k \in \{0,1\}$. La séquence de symboles peut être la sortie d'un Mapping **QAM**, **PSK**,…

Pour tout instant t, le signal émis est donné par l'expression (4.1):

$$S(t) = \sum_{n=0}^{N-1} d_n \exp(j2\pi f_n t) \qquad \textbf{(4.1)}$$

Lorsque le nombre de porteuses est puissance de 2, la mise en œuvre du modulateur se fait au moyen de l'opération FFT Inverse des symboles complexes $\mathbf{d_k}$.

b. Bit Loading

Lorsque le canal est sélectif en fréquences, sa réponse fréquentielle présente des atténuations plus ou moins importantes. La technique de bit loading consiste en le chargement des porteuses de l'OFDM par un nombre de bits qui varie avec l'atténuation. En d'autres termes les symboles $\mathbf{d_k}$ de la figure 1 ne proviennent pas de la même constellation. Souvent on utilise une constellation M-QAM avec M=4,16,32,64,… Pour les fréquences correspondant à une atténuation importante du canal, il est préférable de choisir une constellation M-QAM avec M de faible valeur et là où le canal présente un comportement peu hostile, il est intéressant de transmettre un nombre maximal de bits (voir l'exemple de la figure 2).

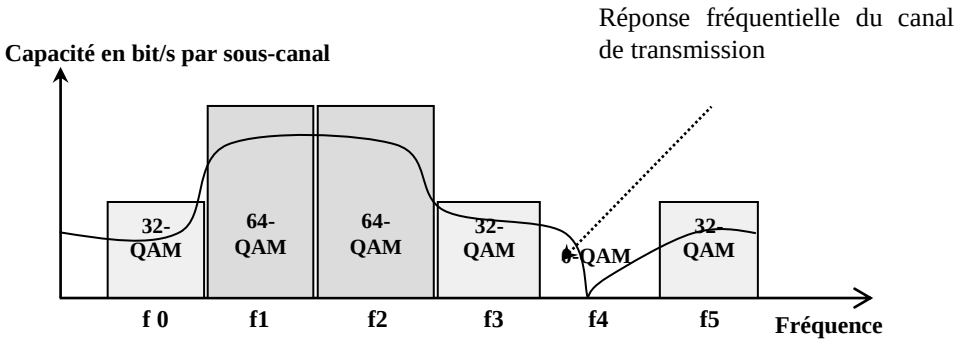


Figure 2 : Principe du bit loading

c. Structure de mise en œuvre du modulateur DMT

La mise en œuvre du modulateur DMT est décrite par la figure 3.

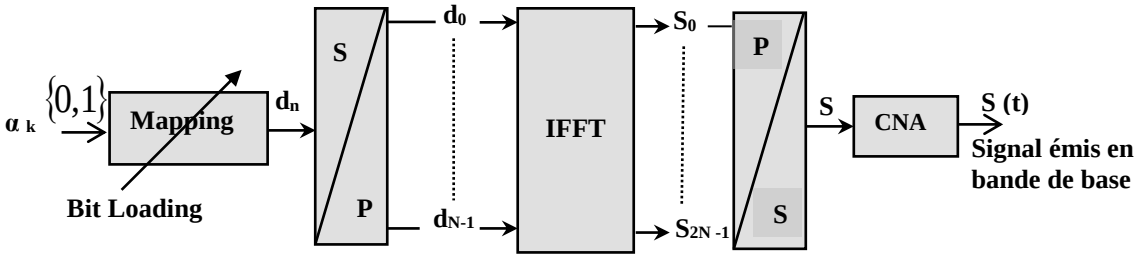


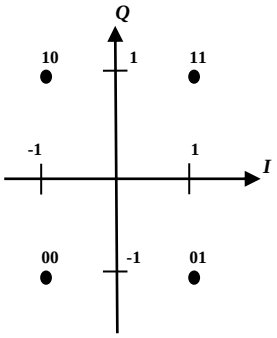
Figure 3 : Modulateur DMT

4.3 Spécifications du projet

4.3.1 Constellation M-QAM

a. Constellation 4-QAM

Elle est identique à celle de la QPSK



Le code assembleur à développer pour la fonction ***_QAM4*** doit lire les 2 bits de faible poids de ***R0*** et renvoie dans ***d_I*** et ***d_Q*** les valeurs réelles et imaginaires du symbole correspondant de la constellation 4-QAM

Pour cela on utilise deux *LUT (Look Up Tables)* de la constellation 4-QAM qui sont *Re4* et *Im4* données par :

Bits	Re4	Im4
00	-1	-1
01	1	-1
10	-1	1
11	1	1

b. Constellation 16-QAM

Elle se déduit en dupliquant la constellation 4-QAM sur les 4 quadrants et en ajoutant à chaque mot binaire deux bits en position fort poids comme indiqué sur la constellation de la constellation 4-QAM. L'alphabet de la constellation 16-QAM est : {-3,-1,1,3}.

1. Dresser la constellation de la16-QAM.
2. Déduire les deux LUTs *Re16* et *Im16* à utiliser.
3. Ecrire le code assembleur de la fonction ***_QAM16*** qui lit les 4 bits de faible poids de ***R0*** et renvoie dans ***d_I*** et ***d_Q*** les valeurs réelles et imaginaires du symbole correspondant de la constellation 16-QAM

c. Constellation 64-QAM

Elle se déduit en dupliquant la constellation 16-QAM sur les 4 quadrants et en ajoutant à chaque mot binaire deux bits en position fort poids comme indiqué sur la constellation de la constellation 4-QAM. L'alphabet de la constellation 64-QAM est : {-7,-5,-3,-1,1,3,5,7}.

1. Dresser la constellation de la 64-QAM.
2. Déduire les deux LUTs *Re64* et *Im64* à utiliser.
3. Ecrire le code assembleur de la fonction ***_QAM64*** qui lit les 6 bits de faible poids de ***R0*** et renvoie dans ***d_I*** et ***d_Q*** les valeurs réelles et imaginaires du symbole correspondant de la constellation 64-QAM.

4.3.2 Chargement des porteuses de la DMT

La réponse fréquentielle du canal est traduite sur le tableau 2 en une correspondance entre le numéro de porteuse du modulateur DMT et le type de constellation à utiliser.

N° porteuse	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Type QAM	4	4	64	64	16	4	4	16	16	16	16	64	64	64	16	4
Nombre de bits	2	2	6	6	4	2	2	4	4	4	4	6	6	6	4	2

Donc 64 bits sont nécessaires pour construire un symbole DMT. Ils sont déclarés dans la variable *binary_sequence*.

Ecrire une fonction assembleur **_Bit_load** qui permet à partir de binary_sequence et en se référant au tableau de chargement des porteuses de générer les symboles complexes pour la modulation DMT.

4.4 Travail demandé

En considérant les spécifications du projet on demande de:

1. Donner vos schémas de conception des traitements demandés dans le paragraphe 4.3 : organigrammes et graphes flot de données en indiquant les ressources DSP utilisées.
2. Ecrire les codes source DSP correspondant aux fonctions définies dans la partie spécifications du projet.
3. Utiliser les ressources de l'environnement VISUAL DSP++ pour la compilation, l'édition des liens puis la génération des fichiers exécutables.
4. Faire appel au simulateur pour analyser le fonctionnement de vos programmes et tracer les signaux utiles.
5. Rédiger votre compte rendu de manipulation.