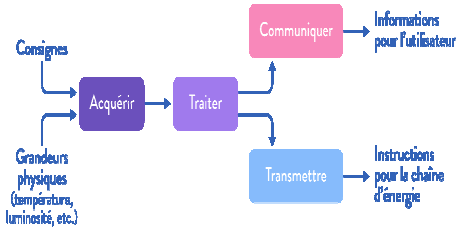
Université de Jijel
Faculté des Sciences & de la Technologie
Département d'Electronique
MASTER 1 : Electronique des Systèmes Embarqués
Intitulé de la matière :



TP Processeurs des signaux numériques (ESE 16)

Travaux Pratiques I
Code Composer Studio (CCS)

Apprendre à utiliser Code Composer Studio (CCS5.5); se familiariser avec le développement d'un système DSP (digital signal processing) et l'analyse d'un code et de présenter les résultats en temps réel.

Prérequis : Programmation en Langage C ou C++ et microprocesseurs

Introduction

Le CCS fournit plusieurs outils pour simplifier le développement et la réalisation des codes/programmes de DSP. Il comprend un éditeur de code source, un compilateur de langage C/C++, un assembleur de code re-localisable, un éditeur de liens, et un environnement d'exécution qui permet de télécharger un code/programme exécutable sur une carte cible (DSK6713), de l'exécuter et de le déboguer. CCS comprend aussi des outils qui permettent l'analyse en temps réel d'un programme en cours d'exécution et des résultats produits.

Exemple d'utilisation : L'exemple qui suit montre comment un algorithme multifilaire simple peut être compilé, assemblé et lié en utilisant CCS. D'abord, plusieurs valeurs de données sont écrites en mémoire. Ensuite un pointeur est assigné au début des données de sorte qu'elles puissent être traitées comme elles sont présentées. Finalement des fonctions simples sont ajoutées en C et en assembleur pour illustrer la façon dont celles-ci sont exécutées.

Installation du logiciel :

1. Installer le CCS V5.1 dans un répertoire TI.
2. Copier les deux répertoires C6xCSL et DSK6713 dans le même répertoire.
3. Lancer le CCS V5,

LANGAGE C

- Créer un nouveau projet nommée TP1
- Dérouler les programmes suivants et afficher les résultats de simulation

Exemple 1

```
#include<stdio.h>
Int main()
        {
        printf("Salam ");
        return 0;
        }
```

Exemple 2

```
#include<stdio.h>
intmain()
        {
        int num, count, sum = 0;
        printf("Enter a positive integer: ");
        scanf("%d", &num);
        for(count = 1; count <= num; ++count)
                {
                 sum += count;
                 }
        printf("Sum = %d", sum);
        return 0;
        }
```

Exemple 3

```
#include<stdio.h>
intmain()
        {
        int x = 2;
        switch (x)
                {
        case 1: printf("Choice is 1");
        break;
        case 2: printf("Choice is 2");
        break;
        case 3: printf("Choice is 3");
        break;
        default: printf("Choice other than 1, 2 and 3");
        break;
        }
        return 0;
        }
```

Exemple 4

```
#include<stdio.h>
main()
        {
         int m=4;                        /*Lenght of i/p samples sequence*/
        int n=4;                         /*Lenght of impulse response Co-efficients */
        inti=0,j;
        intx[10]={1,2,3,4,0,0,0,0};      /*Input Signal Samples*/
        int h[10]={1,2,3,4,0,0,0,0};     /*Impulse ResponseCo-efficients*/
        int *y;
        y=(int *)0x0000100;
        for(i=0;i<m+n-1;i++)
            {
             y[i]=0;
             for(j=0;j<=i;j++)
                 y[i]+=x[j]*h[i-j];
            }
            for(i=0;i<m+n-1;i++)
            printf("%d\n",y[i]);   }
```

ASSEMBLEUR

Partie 1

- **Créer un fichier source**

Le nom du fichier sera : initmem.asm pour notre exemple puis écrivons le code suivant dans cette fenêtre d'éditeur :

```
        .sect".mydata"
.short 0
.short 7
.short 10
.short 7
.short 0
.short -7
.short -10
.short -7
.short 0
.short 7
```

Ce code déclare 10 valeurs numériques en utilisant la directive « .short». La directive « .sect» spécifie que les 10 valeurs doivent résider dans une section de la mémoire du DSP appelée « .mydata »; les adresses physiques qui correspondent à cette section seront définies plus tard dans un fichier d'édition de liens « .cmd ». Vous sauvegardez le fichier source créé en choisissant l'élément de menu « File → Save »

- **Créer un fichier de commande pour l'éditeur de liens**

En plus des fichiers sources, un fichier de commande d'éditeur de liens «Linker» doit être créé pour générer un fichier exécutable et pour se conformer aux spécifications de mémoire du DSP et de la cible sur laquelle le fichier exécutable va être compilé. Un fichier de commande d'éditeur de liens «Linker» peut être créé en utilisant la même procédure que pour le fichier précédant (en choisissant « File → New File »). Pour l'exemple en cours, nous allons considérer le fichier de commande suivant créé sous le nom de : lab1.cmd

```
MEMORY
        {
        IVECS:org = 0h, len = 0x220                     /*vector section*/
        IRAM:org = 0x00000220, len = 0x00002FDE0        /*internal memory*/
        SDRAM:org = 0x80000000, len = 0x00010000        /*external memory*/
        MYDATA:org = 0x80010000, len = 0x00100000       /*external memory*/
        FLASH:org = 0x90000000, len = 0x00020000        /*flash memory*/
        }
SECTIONS
        {
        .vectors :> IVECS
        .text :> IRAM
```

```
.bss :> IRAM
.cinit :> IRAM
.stack :> IRAM
.sysmem :> SDRAM
.const :> IRAM
.switch :> IRAM
.far :> SDRAM
.cio :> SDRAM
.mydata :> MYDATA
}
```

Puisque notre objectif est de placer les valeurs définies dans le fichier initmem.asm dans la mémoire, un espace qui ne sera pas recouvert par le compilateur devrait être choisi. Les espaces de données externes SDRAM ou MYDATA peuvent être utilisés à cette fin. Commençons par assembler les données à l'adresse de mémoire 0x80010000 (0x dénote l'hexadécimal), située en mémoire externe. Pour faire ceci, assignez la section nommée « .mydata » à MYDATA en ajoutant « .mydata :> MYDATA » dans la partie SECTIONS du fichier de commande, comme déjà indiqué dans le programme ci-dessus. La fenêtre d'édition sera sauvegardée dans le fichier de commande d'éditeur de liens «Linker» en choisissant « File→Save ».

- **Ajout de fichiers de support au projet**

En plus des fichiers existants « initmem.asm » et « lab1.cmd », des fichiers supports pour le TMS320C6713et la cible DSK6713 devront être inclus au projet sous forme de librairie et répertoire « Include». Ces fichiers seront implantés au projet dans les prochains laboratoires.

Créer un fichier main.c

```
#include<stdio.h>
Void main()
        {
        printf("BEGIN\n");
        printf("END\n");
        }
```

Après avoir ajouté tous les fichiers sources, fichier de commande et fichier de bibliothèque au projet, vous pouvez construire le projet et créer un fichier exécutable pour le DSP-cible. Pour cela, choisissez l'élément de menu «Project→Build Active Project». Cette fonction permet au CCS de compiler, assembler, et joindre tous les fichiers dans le projet.Si le processus de construction est complété sans erreur, le fichier exécutable«lab1.out» est produit. Il est également possible de compléter cette opération par incréments; c'est-à-dire en recompilant ou en assemblant seulement des fichiers changés depuis la dernière construction, en choisissant l'élément de menu « Project→Rebuild Active Project ».

- **Le suivi de l'exécution (mode Debug)**

Une fois que le processus de construction est complété sans aucune erreur, le programme peut être chargé et exécuté sur la carte DSP cible ou un simulateur. Pour l'installation sur la carte cible DSK TMS320C6713 :

1. Cliquer sur « Target→New Target Configuration… ».
2. Entrer le nom de la cible dans le champs «File name » par exemple DSK6713.ccmxl et cocher «Use shared location» suivi de «Finish».
3. Sélectionner la cible *Spectrum Digital DSK-EVM eZdsponboard USB emulator* dans la fenêtre «Connection», cocher le *DSK6713* dans la fenêtre «Device» et terminer en sauvegardant la configuration avec «Save». La carte DSP cible «DSK6713.ccmxl» sera sélectionnée automatiquement par défaut pour tous les nouveaux projets

Pour télécharger et émuler le programme sur la carte DSP cible, faire les étapes suivantes :

1. Choisissez « Target→Lauch TI Debugger ». Code composer ouvrira une page avec laperspective du mode «Debug».
2. 2. Choisissez « Target→Connect Target » pour le branchement de la carte cible DSK6713.
3. 3. Choisissez « Target→Load Program… » pour transférer le programme compilé (Ex : TP1.out) vers la cible.

Pour revenir en perspective d'édition C/C++, terminer la session «Debug» en choisissant « Target→Terminate All » ou encore en cliquant sur le carré rouge du menu.Pour exécuter le programme, choisissez l'élément « Target→Run » pour un modecontinu et les éléments«Target→Step… » pour le mode étape par étape. Cescommandes sont aussi disponibles sous forme d'icônes sur la ligne de menu rapide. Lorsde l'exécution «Run», vous devriez voir apparaître « BEGIN » et « END » dans lafenêtre console « Stdout » (en bas de l'écran)

Maintenant, allons vérifier si l'ensemble des valeurs est assemblé dans l'emplacementde mémoire indiqué. CCS permet de visualiser le contenu de la mémoire à unemplacement spécifique. Pour visualiser le contenu de la mémoire à l'adresse0x80010000, choisissez « View→Memory » à partir du menu. La fenêtre de dialogue«Memory » permet de spécifier divers attributs d'affichage de la fenêtre. Allez à la zone« AdressText », écrivez **0x80010000** et sélectionnez « 16bit Signed Integer » dans lechamp du format. D'autres options sont disponibles à partir de la barre menue poursauvegarder ou charger des données d'une plage mémoire à partir d'un fichier du PC etremplir une plage mémoire avec une valeur prédéterminée (voir Help)

- **L'outil d'affichage graphique**

Un affichage graphique des données fournit souvent un meilleur aperçu du comportement d'un programme. CCS fournit une interface d'analyse de signal pour surveiller un signal ou ses données. Commençons par afficher le choix de valeurs à l'adresse 0x80010000 comme un signal ou un graphique de temps. Pour ce faire, choisissez « Tool→Graph→Single Time » pour visualiser la fenêtre de propriété du graphique. Allez au «**Start Address Field** », cliquez dessus, puis tapez

0x80010000.

Ensuite, allez à «Acquisition Buffer Size » et «Display Data Size », cliquez dessus et écrivez " 10". Finalement, cliquez sur «DSP Data Type», choisissez «16-bit Signed Integer» à partir de la liste déroulante, et cliquez sur OK. Alors, une fenêtre du graphique apparaîtra avec les propriétés choisies comme illustré ci-dessous

Partie 2

- Créer un nouveau projet. Ecrire les deux programmes : le premier en C permet d'appeler un programme en assembleur (Sumfunc) qui permettre le calcul de la somme *n+(n-1)+(n-2)+… +1*.

Sum1.c

```
#include<stdio.h>
short n=6, result;
extern short sumfunc();
main()
{
result=sumfunc(n);
printf("sum = %d", result);
}
```

Sumfunc.asm

```
Sumfunc.asm          ;Sumfunc.asmfunction en assembleur qui calcule la somme
        .def _sumfunc    ;fonction appelé a partir de c
_sumfunc:MV .L1 A4,A1    ;setup n as loop counter
        SUB .S1 A1,1,A1  ;décrémenter n
LOOP:ADD .L1 A4,A1,A4    ;accumuler/add dans A4
        SUB .S1 A1,1,A1  ;décrémenter le compteur
        [A1] B .S2 LOOP  ;brancher a loop si A1#0
        NOP 5            ;5 NOPs pour intervalles de retard
        B .S2 B3         ;retourne a la routine d'appel
        NOP 5            ; 5 NOPs pour intervalles de retard

        .end
```

COMPTE RENDU

- Créer un nouveau projet nommée TP1_1
- Ecrire les trois programmes suivants : main.asm ; vectors.asm et link.cmd
- Exécuter les programmes étape/étape

main.asm

```
                .text
                .def main
main:   MVK .S1 0x34,A1         ; mettre le nombre dans le registre A1
                MVK .S1 0x25,A2         ; mettre le nombre dans leregistre A2
                MPY .M1 A1,A2,A1        ; multiplier les deux nombres dans A1, A2 et mettre
                        ;le résultat dans A1
                NOP 5
```

Vectors.asm

```
        .global main
        .sect"vectors"          ; permet de mettre ce code dans une section en mémoire
_reset:    MVKL .s2 main,b0     ; mettre l'addresse dans b0
        MVKH .s2 main,b0        ; mettre l'addresse dans b0
b    .s2 b0                     ; branchement au programme
        NOP 5
```

Link.cmd

```
; fichier de lien permettre de stocker les données et les instructions dans une partie de mémoire
MEMORY
        {
        VECS:  origin=0x00000000 length=0x00000220
        IPRAM: origin=0x00000240 length=0x0000FDC0
        }
        SECTIONS
        {
        vectors> VECS
        .text> IPRAM
        }
```

- Ecrire un programme assembleur qui calcule le produit point/point de deux vecteurs suivants :

  B = [2 5 7 9] et b = [3 6 5 2]

- Utiliser la commande ADD pour ajouter les données et SUB pour actualiser le compteur et le conditionneur B pour la boucle. Sauvegarder le fichier TP2_ 2.asm. Exécuter le programme.