Travaux Pratiques

DSP TMS320C54

1 Implantation assembleur du filtrage FIR

1.1 Objectifs

Cette première partie du TP permet de :

- Parcourir l'architecture du noyau de calcul du DSP C54 (Architecture Harward, mémoire programme et mémoire donnée, multiplieur, accumulateurs, scaler, bits de garde) et le mode d'adressage indirect.
- Prendre en main une partie de l'environnement de développement du *CodeComposerStudio* : Edition, Build, Load, Run, Debug, Probe, Graph, Display
- Comprendre l'implantation assembleur d'un filtre FIR en utilisant l'instruction MACD (Cycle Read/Write) associé à un Repeat Buffer (via l'instruction RPTZ)
- Revoir les notions de réponse impulsionnelle, indicielle, fréquentielle et l'opération de convolution.

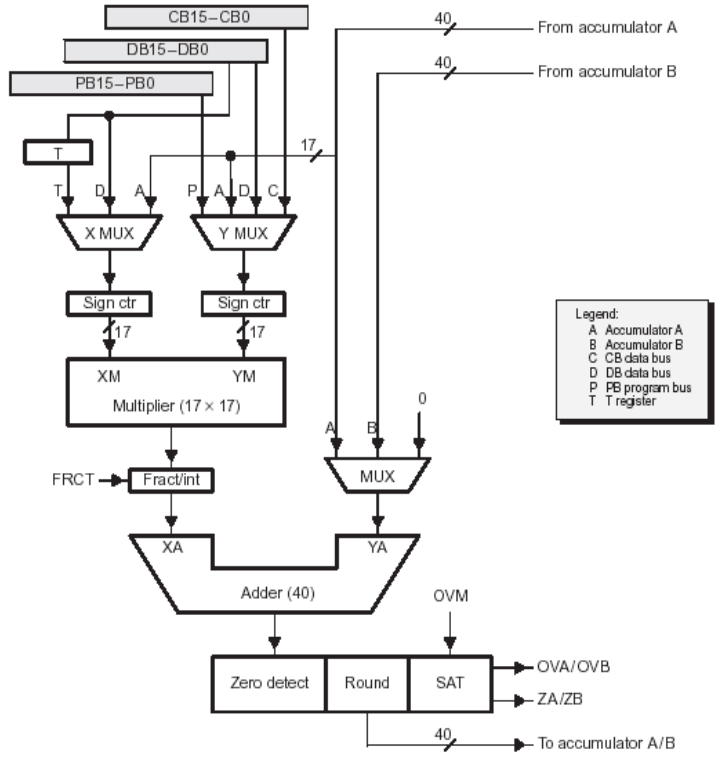

Figure 1 : Autour du multiplier du TMS320C54.

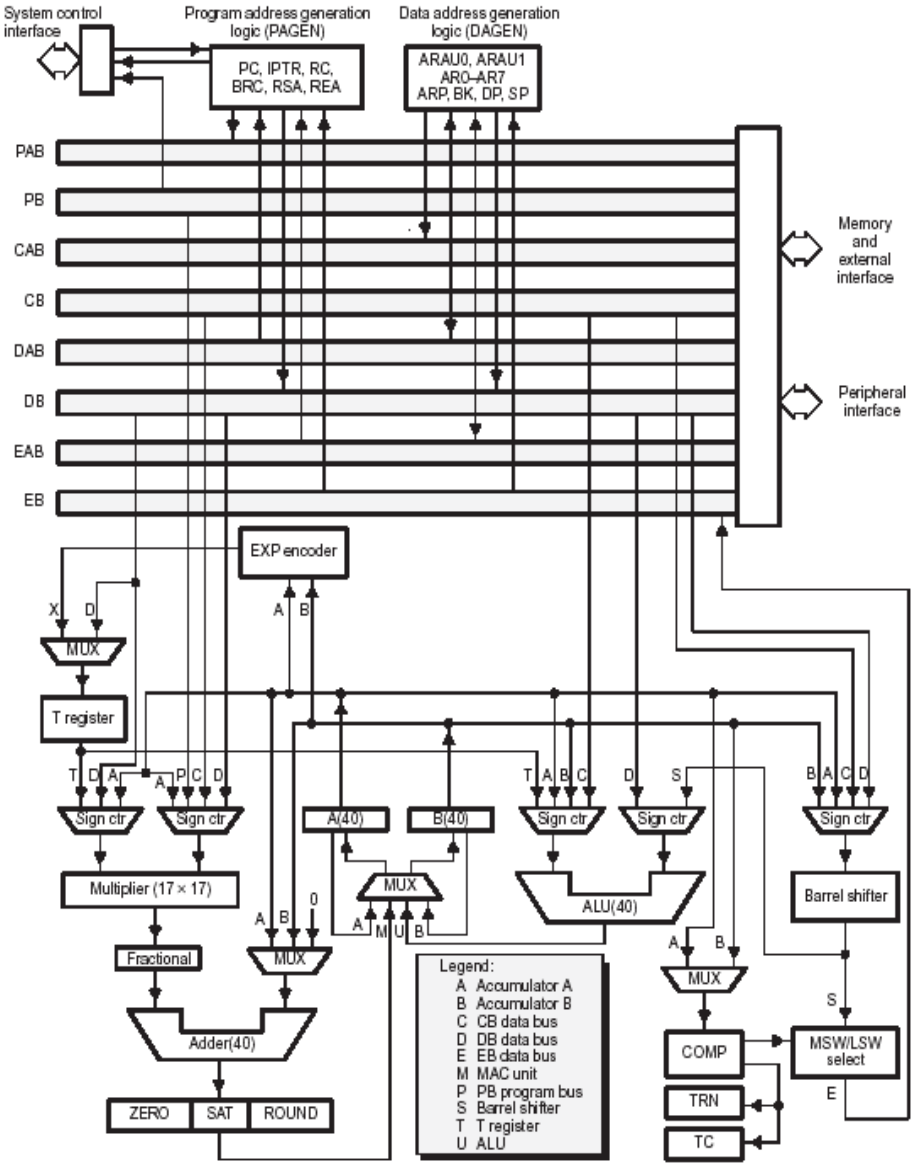

Figure 2 : Diagramme d'ensemble du noyau de calcul du TMS320C54.

1.2 Architecture du C54

En analysant le diagramme du noyau de calcul du C54 (Figure 1 et Figure 2), et en allant fouiller si besoin dans la documentation (*spru131g-reference.pdf*), répondez aux questions suivantes :

1) Combien y a t-il de bus dans le C54 et quels sont leurs rôles ?
2) Combien y a t-il d'accumulateurs et quelle est leur taille ?
3) De quels types peuvent être les entrées du multiplieur ?

1.3 Code Composer Studio et filtrage en assembleur

L'ensemble du développement d'applications est réalisé avec le logiciel *CodeComposerStudio* (*CCS*). Ce logiciel est très complet et nous n'en verrons qu'une petite partie en TP. Il permet de

programmer et configurer l'environnement logiciel complexe associé à la carte DSP (Figure 3). N'hésitez pas à vous reporter à l'aide (touche F1). Vous pouvez accéder à de nombreuses commandes via les icônes de l'interface.

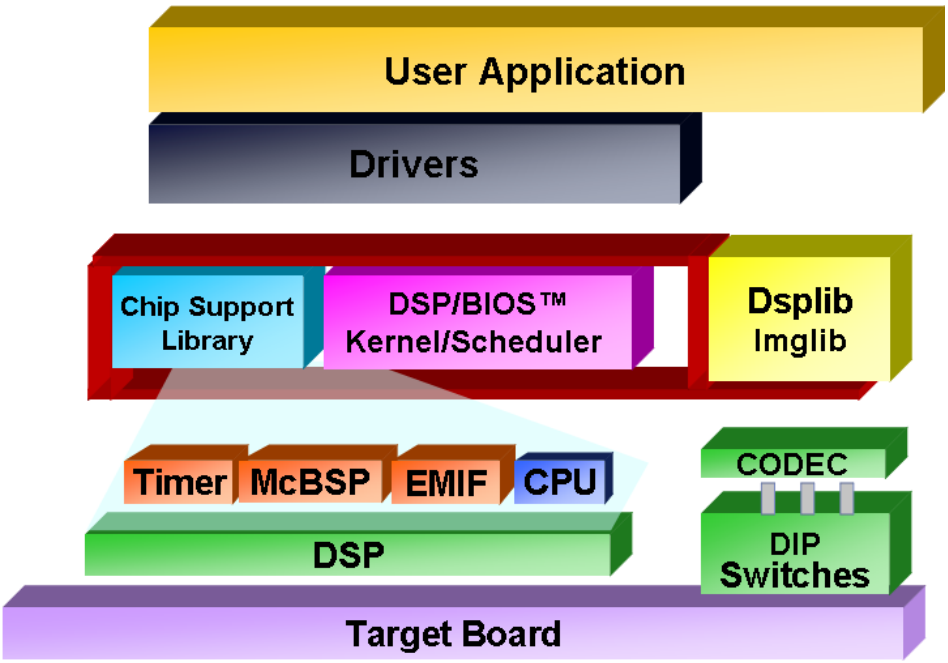

Figure 3 : Contexte de programmation.

Commencer par copier les fichiers sources dans un répertoire personnel. Changer les permissions sur ces fichiers : autoriser l'écriture.
Connectez la carte au PC via le câble USB, une fois le PC allumé, alimentez la carte DSP, puis lancez CCS. L'objectif est d'analyser l'implantation du filtrage FIR en assembleur et de tester cette implantation avec CCS. Au final, on obtient ce qui est représenté Figure 4.

1.3.1 Project

- Via moodle, récupérez les sources du TP : TP-C54.zip et désarchivez.
- Ouvrez le projet *firmacd2.pjt* qui se trouve dans le répertoire *TPC54/firmacd2* via le menu *Project/Open.* Un projet peut être associé à un makefile. Il contient les fichiers sources (C, assembleur), les librairies, les fichiers de configuration de la mémoire de la carte DSP et tout ce qui est nécessaire pour construire un programme exécutable.
- Ouvrez (double click) le fichier *firmacd.asm* qui se trouve dans *source*.

1.3.2 Routine de filtrage

4) Analysez la **structure** générale de ce fichier. Détaillez ce qui se passe dans la boucle qui commence à *debut*. On détaillera le contenu des mémoires, les cases pointées par les registres auxiliaires (AR_i). Très utile, après avoir sélectionné une instruction, l'appui sur F1 vous donne l'explication de l'instruction. On se servira de son cours, mais également de la documentation pdf, pour revoir l'aspect adressage indirect (AR*). Faire un schéma des opérations et transferts mémoires effectués dans la boucle de traitement.

1.3.3 Build / View / Run

- Fabriquez l'exécutable via la commande *Project/Build*. Le fichier exécutable *firmacd2.out* a été créé dans le répertoire *debug* du projet.
- Chargez le programme dans le DSP via la commande *File/Load Program*. Après chargement, la petite flèche jaune indique la position du Programme Counter (PC) qui pointe sur la prochaine ligne à exécuter.
- Affichez les registres CPU via la commande *View/CPU registers*. Repérez les registres intervenant dans votre programme.
- Affichez la mémoire data via la commande *View/Memory*. Dans le champ *adresse*, on saisira l'étiquette *adr_debut_dat*.
- Affichez la mémoire programme via la commande *View/Memory*. Dans le champ *adresse*, on saisira l'étiquette *adr_coef*, et on choisira *program* dans le champ *page*.

Vous devez avoir (sauf les deux graphes) l'écran ci-dessous (Figure 4).

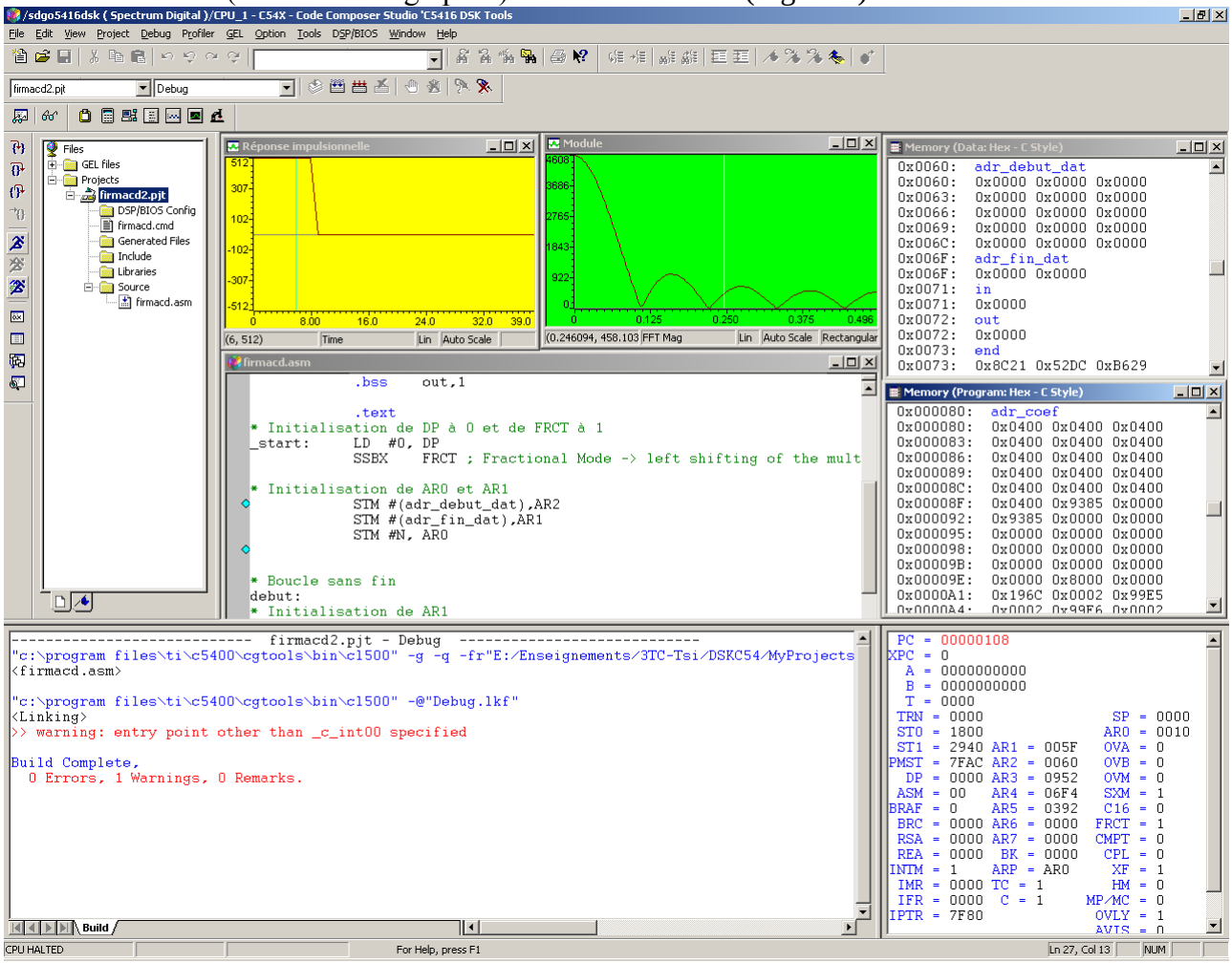

Figure 4 : Filtrage FIR sous CodeComposerStudio. Visualisation de la mémoire programme et data, des registres CPU, du fichier source, de la réponse impulsionnelle et fréquentielle du filtre implanté.

Votre code ayant été chargé sur le DSP, il est prêt à être exécuté. Il suffit de lancer *Debug/Run.*

Il ne se passe rien sauf l'affichage de *CPU RUNNING* en bas à gauche de la fenêtre. Arrêtez le calcul via *Debug/Halt.* S'affiche alors *CPU HALTED.*
Le DSP a bien travaillé mais vous n'avez pas mis de signal à l'entrée de la carte, ni récupéré la sortie sur un oscilloscope. Quand bien même, vous n'auriez rien vu, car votre boucle de calcul prend en entrée ce qui se trouve à une case mémoire étiquetée *in* et envoie la sortie sur une case étiquetée *out*. Or, rien ne relie ces deux cases mémoires aux registres d'entrée (DRR) et de sortie (DXR) de la carte.
Nous verrons dans la deuxième partie du TP comment accéder aux entrée/sortie analogique de la carte, mais ici nous allons étudier la fonctionnalité *Probe* offerte par le *CCS.*

1.3.4 Probe / Pas à pas

Un *Probe point* est une ligne du programme où vous allez brancher une sonde (en anglais probe :-). Lors de l'exécution du programme, chaque fois que l'on rencontre un *Probe point*, le système va lire des valeurs dans un fichier sur le PC et les transférer dans la mémoire du DSP ou bien écrire dans un fichier le contenu de la mémoire du DSP. C'est très utile pour tester des algorithmes à partir de données parfaitement définies dans un fichier qui peut par ailleurs être utilisé sous *Matlab* ou dans tout autre logiciel.
L'utilisation des *Probe points* nécessite le positionnement dans les fichiers source de *probe points*, le choix de fichiers d'entrée et/ou de sortie, l'association d'un fichier à une case mémoire et à un *probe point*.
Dans notre exemple, nous allons utiliser les probe points pour remplir la case mémoire *in* (devant contenir l'échantillon courant x(n)) à l'aide d'un fichier correspondant à un signal de test. De même, nous enverrons dans un fichier la case mémoire *out* (devant contenir l'échantillon de sortie y(n)).

- Positionnez un *probe point* avec l'icône *Toggle probe point* sur la ligne 37 « STM #(adr_fin_dat) ». Un losange bleu apparaît.
- Via le menu *File/File IO*, choisissez comme fichier d'entrée *infir.dat* (il se trouve dans le répertoire du projet), mettez le champ *address* à *in*, et le champ *length* à 0x0001 puis cliquez sur *Add Probe point.*
- Sélectionnez le *probe point* de la ligne 37 dans la liste, via le champ *Connect to*, associez le *probe point* à *infir.dat*. Cliquez sur le bouton *Replace* pour valider cette association. *OK – OK*. Un 'player' de fichier apparaît.
- Exécuter le programme. Ce dernier s'arrête quand la fin du fichier est atteinte. Rembobinez le fichier en faisant 'play' puis 'rewind'.
- Ajoutez un nouveau *Probe point* sur l'instruction de fin de boucle. Connectez à ce *probe point* le fichier *outfir.dat* via *File/File IO/ OutputFile*. Champ *Address = out* et *Length=0x0001*. Au passage, cliquez sur le bouton *Aide* et regardez le contenu de la rubrique File IO control. Dans la fenêtre File IO, cliquez sur *Add Probe point* pour connecter votre fichier.
- Sélectionnez le *probe point* de la ligne 52 dans la liste, via le champ *Connect to*, associez le *probe point* à out*fir.dat*. Cliquez sur le bouton *Replace* pour valider cette association. *OK – OK*. Un deuxième 'player' de fichier apparaît.

Nous allons maintenant exécuter le programme en mode pas à pas (touche F10). Le fichier *infir.dat* contient un échantillon de valeur 0x4000, suivi de 39 échantillons de valeur nulle.

- En mode pas à pas, faites dérouler sur une dizaine d'échantillons *infir.dat*. Observez l'évolution de la mémoire et des différents registres. Commentaires ?
- Rajoutez un point d'arrêt avec l'icône 'main' sur la dernière ligne du code. Continuez à dérouler le fichier *infir.dat* en utilisant l'icône *Run*.
- Rembobinez *infir.dat*.

1.3.5 Graphes

CCS inclut un outil graphique permettant de représenter des zones de mémoire sous forme de graphe temporel, fréquentiel, constellation ….

- Insérez un graphe via le menu *View/Graph/TimeFrequency*. On positionnera les champs suivants : *StartAddress = out, Page=Data, AcquisitionBufferSize=1, IndexIncrement=1, DisplayDataSize=40, DSPDataType=16 bit integer signed*. N'hésitez pas à cliquez sur Help pour avoir des informations sur les différents champs.
- Insérez un deuxième graphe identique au précédent en modifiant le champ *DisplayType* à *FFT magnitude*.
- Le fichier ayant été rembobiné et un point d'arrêt en fin de boucle ayant été positionné, ré exécuter votre programme en utilisant *Restart* dans le menu *Debug*.

On peut également utiliser les graphes sur les fichiers de sortie associés à un *Probe point*. Au préalable, il suffit de charger ce fichier en mémoire.

- Chargez le fichier *outfir.dat* en mémoire à l'adresse 0x8000 via le menu *File/Data/Load*. Address=0x8000, Length=0x0029.
- Créez un nouveau graphe avec les paramètres de votre choix.

1.3.6 Filtrage

5) Que venez-vous de tracer ?
6) Quelles sont les caractéristiques du filtre implanté dans ce code ?

- Copiez le fichier *infir.dat* dans un autre fichier. Modifiez-le, de telle sorte que vous puissiez tracer la réponse indicielle de ce filtre.

7) Réalisez les manipulations nécessaires pour tracer la réponse indicielle. Quelle est sa forme ?

En fin du fichier firmacd.asm, on donne les coefficients d'un autre filtre.

- Modifiez le fichier pour que ce soit ce nouveau filtre qui soit utilisé. Réalisez les manipulations nécessaires (Build, Load ….) pour tracer sa réponse impulsionnelle et fréquentielle sous *CCS*.

8) Quelles sont les propriétés de ce filtre ?

1.4 Bilan partie 1

Vous devez être à l'aise avec les principaux éléments du *CCS* manipulés dans cette partie (Build, Run, Load, Probe, Point-d'arret, File IO, Graph).

Vous devez être capable de comprendre un petit code assembleur qui implante du filtrage et de l'adapter pour changer de filtre.
Vous devez avoir en tête les principaux éléments concernant les filtres FIR (Coeff de la fonction de transfert, réponse impulsionnelle, réponse fréquentielle, phase linéaire).

2 Partie 2 : Programmation C et filtrage FIR

2.1 Objectifs

Cette deuxième partie va vous permettre de découvrir :

- comment accéder en C au hardware de la carte (codec, switch, LED),
- comment implanter la convolution en C,
- comment implanter un filtre en C
- comment caractériser en fréquence un système linéaire.

2.2 Codec, LED et Interrupteurs

La carte associe au DSP C5416 (Figure 5) un codec stéréo le PCM3002 associé à des entrées sortie audio, 4 interrupteurs (DIP switches) et 4 LEDs. Un grand nombre de fonctions C permet d'accéder aux fonctionnalités de ce codec (initialisation, configuration, lecture, écriture, …), à l'état des interrupteurs ou encore de contrôler l'allumage des LEDs. La position des interrupteurs peut être observée en temps réel et interprétée pour changer le déroulement d'un programme et le rendre interactif.

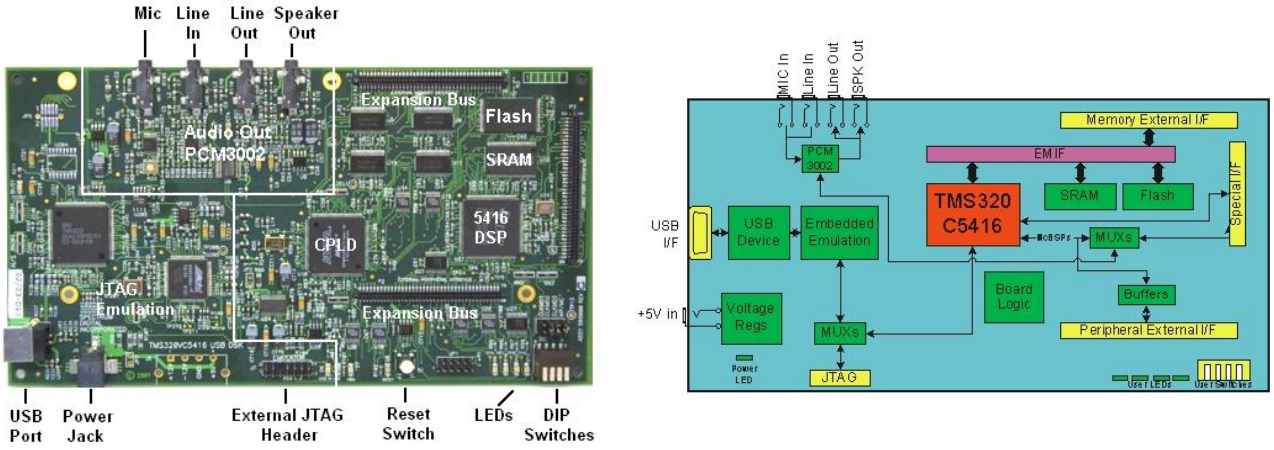

Figure 5 : Vue du starter kit TMS320VC5416

Le petit exemple ci-dessous va permettre interactivement de choisir le nombre de bits de quantification à appliquer au signal en entrée avant de l'envoyer ainsi quantifié sur la sortie.

- Fermez tous les projets en cours sous CCS (*Project/Close*)
- Ouvrez le projet *TPC54/sampling_bits /sampling_bits.pjt* (*Project/Open*)
- Editez le fichier *sampling_bits.c*

La fonction *UserTask()* est la fonction principale qui va être appelée par le gestionnaire de tâche du DSP.

9) Repérez dans cette fonction les appels de fonctions associées au codec, aux interrupteurs et aux LED.

- Générez l'executable via la commande *Build*. Chargez le programme sur le DSP (*File/LoadProgram*)
- Connectez un signal en entrée (signal audio, ou signal du générateur), observez la sortie (via un oscilloscope ou avec vos oreilles)
- Exécutez le programme (*Debug/Run*).
- Manipulez les interrupteurs pour décroître progressivement le nombre de bits de quantification

10) A partir de combien de bits percevez vous une différence (visuelle sur l'oscilloscope, qualitative avec vos oreilles)

- Fermez ce projet.
- Vous pouvez faire rapidement la même manipulation avec le projet *sampling_rate* pour étudier l'effet de la fréquence d'échantillonnage et du repliement de spectre associé.

2.3 Filtrage C, assembleur

En utilisant le même type de programmation que dans le projet précédent, on va appeler interactivement différents filtres utilisant des implantations de la routine de filtrage en C et en assembleur.

2.3.1 Filtrage FIR

11) En utilisant un oscilloscope et un générateur de fonction, quelle manipulation simple permet rapidement d'estimer la réponse en fréquence d'un filtre implanté sur la carte ?

- Fermez tous les projets en cours sous CCS (*Project/Close*)
- Ouvrez le projet *TPC54/ fir_windows /fir_windows.pjt* (*Project/Open*)
- Ouvrez le fichier *fir_windows.c*.

12) Analysez le contenu de la fonction UserTask(). Quelles sont les fonctions appelées pour réaliser le filtrage ? Où sont définis les coefficients des filtres ?

13) Mais au fait, pourquoi utilisez des fenêtres pour synthétiser des filtres FIR ?

- Ouvrez le fichier *fir_filters_asm.asm*. Dans la section *_FIR_filter_asm*, repérez la boucle de convolution.

14) Quelles sont les instructions utilisées ? Quel adressage spécifique est utilisé ?

15) Ouvrez le fichier *fir_filters.c*. Dans la fonction *FIR_filter()*, comment est calculé le produit de convolution ?

- Générez le programme exécutable (*Build*)
- Chargez le sur la carte (File/Load) et exécutez le (Debug/Run). On mettra les switchs en position 0.

16) En utilisant la méthode proposée à la question précédente, tracez la réponse en fréquence du système correspondant à la carte. Un filtre est-il présent ? Si oui, d'où vient-il ?

17) Positionnez les switchs pour choisir un filtre. Tracez en la réponse en fréquence.

18) Pour une fréquence dans la bande passante du filtre, quel phénomène observez vous si vous augmentez progressivement l'amplitude du signal d'entrée ?

Pour la question 17), on pourra vérifier les résultats obtenus en utilisant les *probe points* et les graphes vues dans la partie 1 pour tracer les réponses des filtres.

- L'utilisation des *probe points* perturbe la lecture des switchs. Dé-commentez la ligne switch_value=6, (Mettez la valeur du filtre que vous voulez étudier)
- *Build, Load …*
- Positionnez un *probe point* sur la ligne 124 à l'appel de *bargraph_6dB().*
- Connectez le au fichier *inpulse.dat* (il contient une impulsion et 80 zéro), avec *Address=&mono_input* et *Length=1*
- Positionnez un point d'arrêt sur le premier *if ()*
- Créer deux graphes, l'un en temps, l'autre en FFT magnitude avec les réglages suivants : *Start_Address=&left_output, Acq.BufferSize=1, DisplayDataSize=82, DSPDataType=16-bit signed integer.*
- On peut faire un *View/Memory* à partir de l'adresse *&left_input.*
- *Run, Run, Run …. Run* et on obtient quelque chose comme la Figure 6. On rembobine, et encore un coup pour le plaisir !

On cherche à rajouter un nouveau filtre à tester. Il a été défini dans *monfiltre.h.*

19) Modifiez *fir_windows.c* pour intégrer ce nouveau filtre et le rendre appelable dans le cas ou les switchs valent 15.

20) Construisez votre nouvel exécutable, chargez le, et vérifiez la réponse en fréquence de *monfiltre.*

2.3.2 Filtrage IIR

- Fermez tous les projets en cours sous CCS (*Project/Close*)
- Ouvrez le projet *TPC54/ IIR_order /iir_order.pjt* (*Project/Open*)
- Ouvrez le fichier *iir_order.c.*

21) Analysez le contenu de la fonction UserTask(). Quelles sont les fonctions appelées pour réaliser le filtrage ? Où sont définis les coefficients des filtres ?

22) Ouvrez le fichier *IIR_filters_first_order.c*. Analyser les fonctions de calcul *first_order_IIR_direct_form_I()* et *first_order_IIR_direct_form_II()*. Quels sont les différences entre ces deux modes de calcul ?

On pourra aller voir les modifications de ces fonctions pour des filtres d'ordre 2 et 4 en allant voir les fichiers *IIR_filters_second_order.c* et *IIR_filters_fourth_order.c* .

- Générez le programme exécutable (*Build*)

- Chargez le sur la carte (File/Load) et exécutez le (Debug/Run). On choisira la position ad hoc pour les switchs.

23) En utilisant la méthode proposée à la question précédente, tracez la réponse en fréquence d'un des filtres implanté.

2.4 Bilan partie 2

Vous avez compris comment accéder aux échantillons d'entrée/sortie via les fonctions du codec.
Vous comprenez les grandes lignes d'un code en C qui implante la convolution. Vous savez le modifier pour utiliser un filtre préalablement défini.
Vous avez compris le lien entre la réponse d'un système linéaire à une sinusoïde et la réponse en fréquence.
Le filtrage n'a plus de secret pour vous … et vous commencez à vous sentir maître de *CCS* …

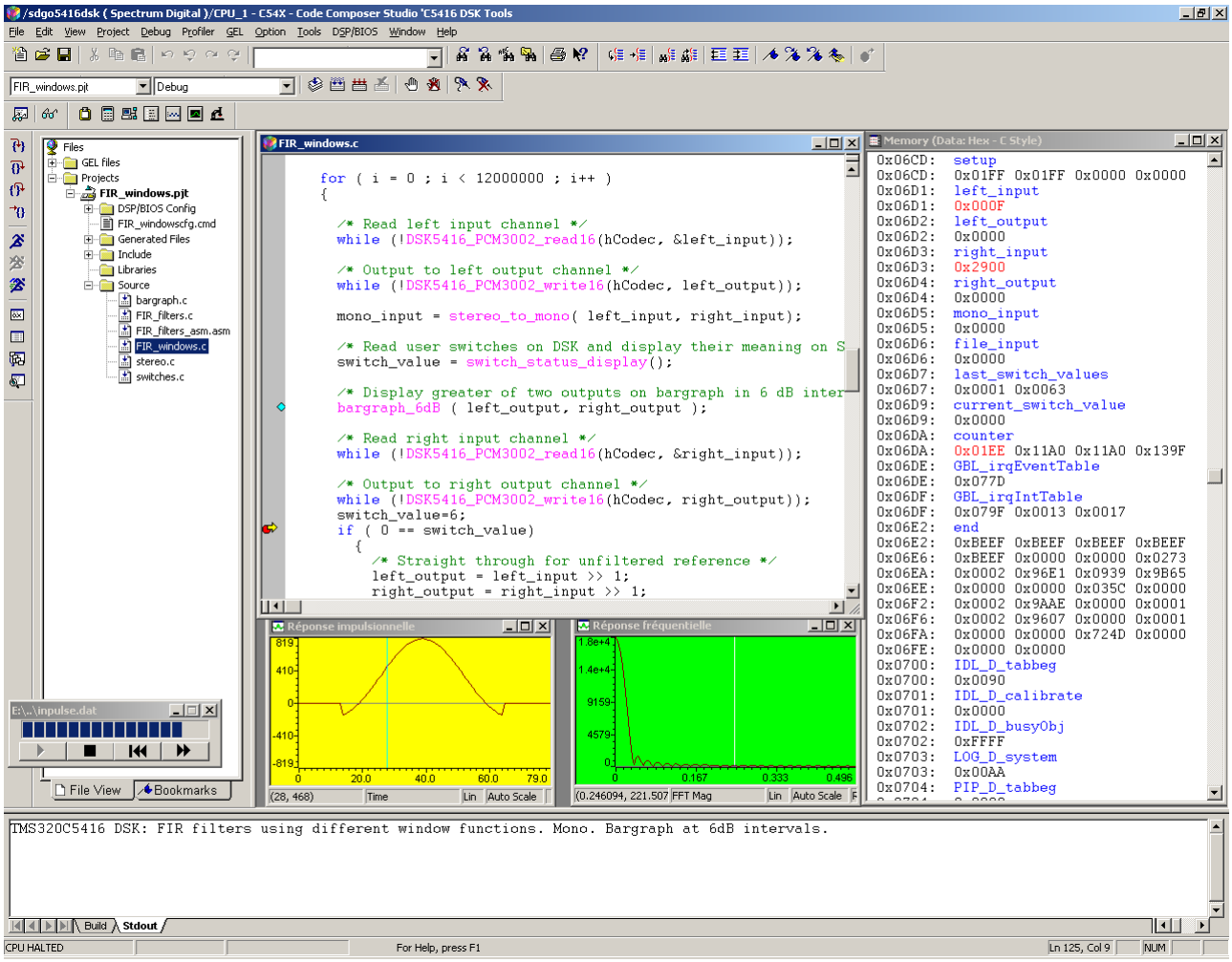

Figure 6 : Réponse impulsionnelle et fréquentielle d'un FIR synthétisé par une méthode de fenêtre.

3 Conclusion

Vous avez mis un orteil dans le monde des DSP, à suivre …

4 Annexe : Architecture mémoire du C54

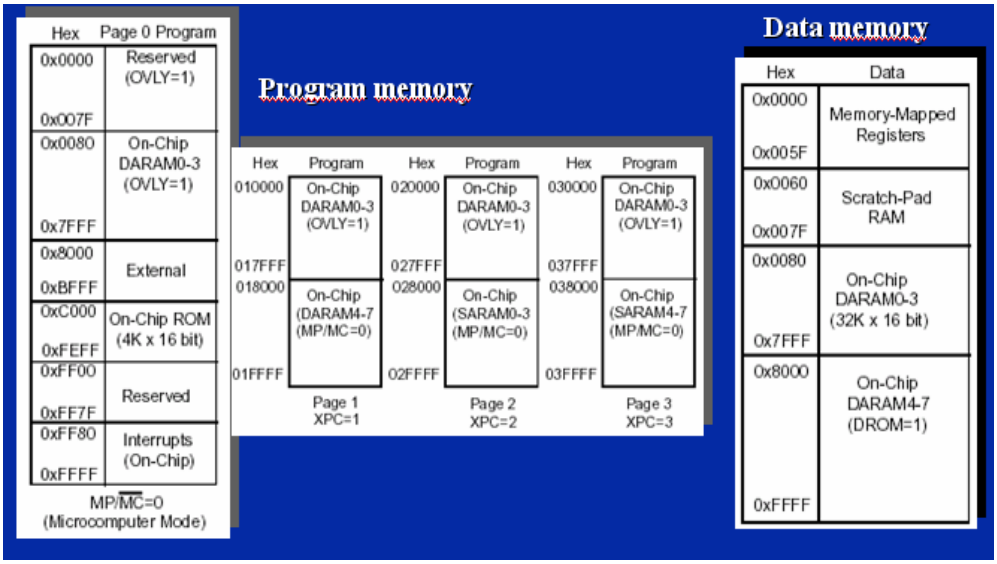

Figure 7 : Plan de la mémoire du C54.

5 Annexe : Fichier *firmacd.asm*

```
                .mmregs
                .def _start
                .global     adr_debut_dat, adr_fin_dat, adr_coef,_start
                .global     adr_coef

                .sect ".coef"
adr_coef    .word 0X400
            .word 0X400,0X400,0X400,0X400,0X400
            .word 0X400,0X400,0X400,0X400,0X400
            .word 0X400,0X400,0X400,0X400,0X400

N               .set  16
                .bss  adr_debut_dat,N+1
adr_fin_dat .set  adr_debut_dat+N-1
                .bss  in,1
                .bss  out,1

                .text
* Initialisation de DP à 0 et de FRCT à 1
_start:         LD    #0, DP
                SSBX  FRCT ; Fractional Mode -> left shifting of the
multiplier

* Initialisation de AR0 et AR1
                STM   #(adr_debut_dat),AR2
                STM   #(adr_fin_dat),AR1
                STM   #N, AR0

* Boucle sans fin
debut:
* Initialisation de AR1
                STM   #(adr_fin_dat),AR1

* Lecture de x(n) à l'adresse DRR à partir du fichier firin.dat
                LD    *(in), A
            STL   A,*AR2

* Boucle de filtrage
                RPTZ  A, #N-1
                MACD  *AR1-, adr_coef, A

* Sauvegarde de la'accumulateur dans la case tampon
* à l'adress adr_fin_dat+1 puis sortie de y(n) dans fichier outfir.dat
                STH   A,*(out)

* Retour au début de la boucle sans fin
                B     debut
```

6 Annexe : Fichier *fir_filter_asm.asm*

```
;***************************************************************************
;* FUNCTION DEF: _FIR_filter_asm                                           *
;***************************************************************************
_FIR_filter_asm:

        PSHM      ST0                   ; Keep original values of flags
        PSHM      ST1
        PSHM      AR3                   ; Keep original values of registers
        PSHM      AR4

        SSBX      OVM                   ; Prevent overflow
        SSBX      FRCT                  ; Extra shift for multiplications
        SSBX      SXM                   ; Turn on sign-extension mode

        FRAME     #-1                   ; Create stack frame

        STLM      A,AR3                 ; AR3 now points to _coefficients

;Start by shuffling values in buffer along and inserting new value at buffer[0]

        STM       #_buffer1+N-2, AR4    ; AR4 points to buffer[8]
          RPT       #(N-2)             ; Shuffle all the values.# is important!
          DELAY     *AR4-

        MVDK      *SP(4+3),*(_buffer1)   ; New input to beginning of buffer
        STM       #_buffer1, AR4         ; AR4 now points to buffer[0]

        ; Multiplications with accumulation

        RPTZ      A, #(N-2)              ; Clear A then repeat N - 2 times
        MAC       *AR3+, *AR4+, A        ; Multiply and accumulate in A

        MACR      *AR3+, *AR4+, A        ; Round up the last one

        SFTA      A, -16, A              ; Remove fractional part
                                         ; Ready to return output in
Accumulator A

        FRAME     #1
        POPM      AR4                    ; Restore registers
        POPM      AR3
        POPM      ST1                    ; Restore flags
        POPM      ST0

        FRET                             ; Far return
```

7 Annexe : Fichier *fir_filter.c*

```
short int FIR_filter(signed int * filter_constants, signed int input)
{
 static int value[FIR_FILTER_SIZE]; /* Retain value between calls */
 signed int return_value;
 signed int i;  /* Index into filter constants */
 long product;
 static signed int j = 0; /* Index into input array */

 product = 0;

 for ( i = 0 ; i < FIR_FILTER_SIZE ; i++)
 {
  /* Generate sum of products. Shift right to prevent overflow */
  /* This is first part of divide by 32767 */

  product +=  ( (long)(value[j] * filter_constants[i]) >> 5);

  if ( j < FIR_FILTER_SIZE-1) /* Next item in circular buffer */
    {
     j++;
    }
  else
    {
     j = 0;                    /* Go back to beginning of buffer */
    }
 }

 if ( j < FIR_FILTER_SIZE -1) /* Loop done. Last filter element */
    {
     j++;
    }
 else
    {
     j = 0;   /* Point to new value */
    }

 product >>= 10;   /* Second part of divide by 32768 */

 value[j] = input; /* Read in new value for next time. */

 return_value = (signed int) product;

 return(return_value);
}
```

8 Annexe : Fichier *IIR_filters_first_order.c*

```
/*************************************************************************/
/* A first order infinite impulse response (IIR) filter can be represented   */
/* by the following equation:                                                */
/* H(z) = b0 + b1.z^-1                                                       */
/*        ------------                                                       */
/*        a0 + a1.z^-1                                                       */
/* where H(z) is the transfer function. a0 is always 1.000                   */
/* In order to implement this function on a fixed point processor, the value */
/* 32767 is used to represent 1.000                                          */
/*************************************************************************/

#include <stdio.h>
/* Numerator coefficients */
#define B0 0
#define B1 1
/* Denominator coefficients */
#define A0 2
#define A1 3

/*************************************************************************/
/* first_order_IIR_direct_form_I()                                       */
/* First order direct form I IIR filter.                                 */
/* PARAMETER 1: Coefficients in order B0 B1 A0 A1.                       */
/* This implmentation uses two buffers, one for x[n] and the other for y[n]  */
/*************************************************************************/

signed int first_order_IIR_direct_form_I( const signed int * coefficients,
signed int input)
{
  long temp;
  static signed int x[2] = { 0, 0 };  /* x(n), x(n-1). Must be static */
  static signed int y[2] = { 0, 0 };  /* y(n), y(n-1). Must be static */

  x[0] = input; /* Copy input to x(n) */
  temp =  ( (long) coefficients[B0] * x[0]) ;   /* B0 * x(n)   */
  temp += ( (long) coefficients[B1] * x[1]);    /* B1 * x(n-1) */
  temp -= ( (long) coefficients[A1] * y[1]);    /* A1 * y(n-1) */

  /* Divide temp by coefficients[A0] */
  temp >>= 15;
  /* Range limit temp between maximum and minimum */
  if ( temp > 32767 )
   {
     temp = 32767;
   }
  else if ( temp < -32767)
   {
     temp = -32767;
   }
  y[0] = (short int) ( temp );

  /* Shuffle values along one place for next time */
  y[1] = y[0];                 /* y(n-1) = y(n)   */
  x[1] = x[0];                 /* x(n-1) = x(n)   */
 return ( y[0] );
}
```

```
/*************************************************************************/
/* first_order_IIR_direct_form_II()                                      */
/* First order IIR (canonic) filter.                                     */
/* Uses 32767 to represent 1.000.                                        */
/* Note that input is divided by 2 to prevent overload.                  */
/*************************************************************************/

signed int first_order_IIR_direct_form_II ( const signed int * coefficients,
signed int input)
{
  long temp;
  static signed int delay[2] = { 0, 0 }; /* Must be static */

 /* Process using denominator coefficients */
  temp = (((long)coefficients[A0] * input ) >> 1 ); /* Divide by 2 */
  temp -= ((long)coefficients[A1] * delay[1] );  /* A1 */
  temp >>= 15;  /* temp /= coefficients[A0] */

  /* Limit output to maximum and minimum */
  if ( temp > 32767)
   {
     temp = 32767;
   }
  else if ( temp < -32767)
   {
     temp = -32767;
   }
  delay[0] = (signed int) temp;

  /* Process using numerator coefficients */
  temp = ((long)coefficients[B0] * delay[0] );
  temp += ((long)coefficients[B1] * delay[1] ) ;  /* B1 */
  delay[1] = (signed int)( delay[0] );

  /* Scale output. Divide by temp by coefficients[A0] then multiply by 2 */
  temp >>= ( 15 - 1 );

  /* Range limit output between maximum and minimum */
  if ( temp > 32767 )
   {
     temp = 32767;
   }
  else if ( temp < -32767)
   {
     temp = -32767;
   }

  return ( (signed int) temp );
}
/*************************************************************************/
/* End of IIR_filters_first_order.c                                      */
/*************************************************************************/
```