TP RIF (FILTRAGE NON-RECURSIF)

(assembleur, sans E/S) → dossier : rif_ASM

Objectifs du tutorial

- Se familiariser avec l'assembleur
- Se familiariser avec le simulateur
- Visualiser les différents parallélismes

Explication du thème

Le filtre calcule ***y(n)*** tel que :

$$y(n)= c0\ x(n) + c1\ x(n-1) + c2\ x(n-2) + c3\ x(n-3) + c4\ x(n-4)$$

Ce programme est un exemple qui peut facilement être modifié pour être utilisé dans un vrai système. Il est souhaitable de consulter les listings des pages suivantes pendant la lecture des explications.
Les données ***x*** à filtrer sont prédéfinies et stockées dans un buffer (***inbuf***). Les coefficients sont initialement stockés dans un fichier ***"rifcoeffs.dat"***. Le code exécute un filtre à 5 coefficients sur les 9 échantillons d'entrée prédéfinis. Les résultats sont sauvegardés dans le buffer (***outbuf***).
Le programme est constitué de 2 modules écrits en assembleur : ***"riftest.asm"*** et ***"rif.asm"***. Le fichier ***"riftest.asm"*** génère les valeurs des buffers et les pointeurs sur ces buffers puis appelle ***"rif.asm"***, ce module réalise la convolution par une succession de multiplications-accumulations.
Deux autres fichiers sont nécessaires : un fichier ***architecture*** et le fichier des coefficients.
Le fichier ***"ADSP-21262_ASM.LDF"*** est un fichier ***architecture*** et permet de déterminer les adresses absolues d'implantation des données et du code. Il définit le matériel (mémoires, périphériques) en terme d'espaces mémoires. Le fichier ***"rifcoeffs.dat"*** contient les coefficients du filtre.
La figure 1 montre les 4 buffers utilisés pour la réalisation du filtre.
Les deux tableaux ***coefs*** et ***dline*** sont 2 buffers gérés sous forme circulaire.

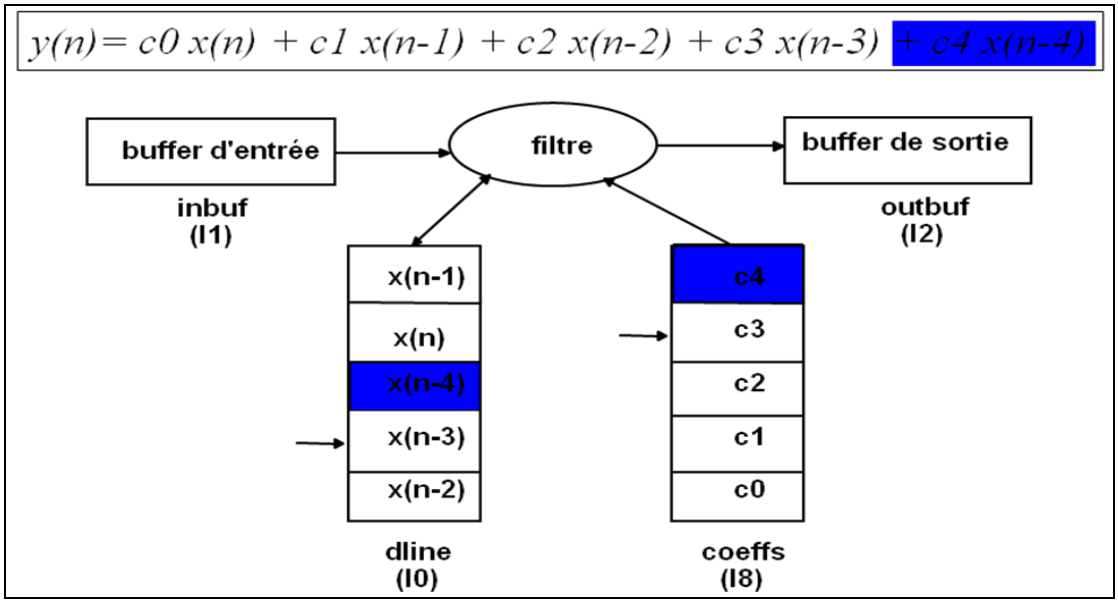


Figure 1

Au départ de l'exécution du filtre pour le calcul de ***y(n)***, l'échantillon ***x(n)*** est stocké dans ***dline***. Le pointeur **I0** pointe ensuite sur l'échantillon ***x*** le plus ancien. Pour le buffer ***coefs***, le pointeur **I8** pointe sur le coefficient d'indice le plus élevé. A chaque étape, ces deux pointeurs sont incrémentés pour réaliser l'ensemble des multiplications suivant l'équation du filtre. A la fin du calcul, **I0** pointe à nouveau sur l'échantillon ***x*** le plus ancien qui sera remplacé par ***x(n+1)*** lors du prochain calcul.
Nous laissons au lecteur le soin de vérifier, grâce au listing ***"rif.asm"***, que le nombre de cycles nécessaires au calcul d'un échantillon est de : **7 + (nombre de coefficients)** pour le sous-programme ***rif***.
On notera qu'un seul défaut de cache aura lieu pendant l'exécution de ce sous-programme (deux défaut dans le cas de la version optimisée).

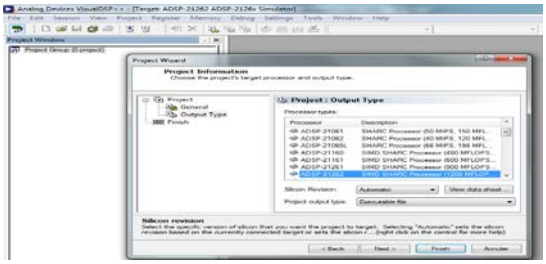
- Nombre de cycles par échantillon : .....
- Lignes mises en cache : .....

Scenario du TP

1. Lancer ***VisualDSP++*** :
   - ***Menu démarrer ➔ Analog Devices ➔ VisualDSP++ Environment***

   La fenêtre graphique qui se lance, ***VisualDSP++ Environment***, représente l'environnement de développement que nous allons utiliser pour les différents thèmes. **VisualDSP++** est un **environnement de développement intégré (***IDE* en anglais pour ***I****ntegrated* ***D****evelopment* ***E****nvironment*) constitué d'un ensemble d'outils pour le développement d'applications pour les processeurs de la famille SHARC d'Analog Devices.

2. Créer un projet nouveau appelé **rif_ASM** dans le sous-répertoire **rif_ASM**
   - **Project ➔ New** ⇨ puis :
   - Choisir les options suivantes : le processeur ***« ADSP-21262»*** et le type du projet ***« standard application »***
   - Parcourir les autres catégories pour découvrir les différentes options disponibles



3. Construire le projet ***rif_ASM*** en ajoutant deux fichiers sources écrits en assembleur (fichiers se trouvant dans le sous-dossier **rif_ASM**).
   - **Project ➔ add file(s)**, ou utiliser l'icône correspondante de la barre d'outils, pour ajouter au projet les deux fichiers sources : ***« rif.asm »*** et ***« riftest.asm »*** ainsi que le fichier ***« ADSP-21262_ASM.LDF »***, le fichier "architecture" qui se trouve dans le dossier ***« LDF »***.

4. Editer les trois fichiers et organiser votre espace de travail à votre convenance

5. Compiler le projet et corriger le code, si erreurs et/ou warnings
   - **Project ➔Build project** (chercher l'icône correspondante dans la barre d'outils)

6. Vérifier que le fichier exécutable est chargé dans le simulateur à la fin de la compilation, sinon le charger :
   - **File ➔ Load Program…** ***"debug\rif_ASM.dxe"*** ⇨ Une fenêtre assembleur s'ouvre automatiquement et pointe sur la première instruction à exécuter. L'exécution du programme en pas à pas peut commencer (exploiter les fonctionnalités du simulateur).

7. Afficher les registres de données
   - **Register ➔ Core ➔ Register File ➔ Register File PEx**

8. Afficher les registres d'adresse DAG1
   - **Register ➔ Core ➔DAG1 (DM)**

9. Afficher les registres d'adresse DAG2
   - **Register ➔ Core ➔ DAG2 (PM)**

10. Afficher le registre USTAT2
    - **Register ➔ Core ➔ Status (USTAT)**

11. Afficher les registres de compteur boucle
    - **Register ➔ Core ➔ Counters ➔ Loop Counters**

12. Afficher les registres de pile de compteur boucle
    - **Register ➔ Core ➔ Stacks ➔ Loop Counter**

13. Afficher les registres de pile d'adresse de boucle
    - **Register ➔ Core ➔ Stacks ➔ Loop Address**

14. Afficher les registres systèmes
    - **Register → System → System Control**

15. Ouvrir une fenêtre mémoire pour la visualisation du code
    - **Memory → Three Column**

16. Ouvrir une fenêtre mémoire pour visualiser les coefficients du filtre
    - **Memory → Two Column**
    - ***Bouton droit de la souris*** (sur la fenêtre) **→ *Affichage d'un menu contextuel***
      ⇨ Puis **Go to … (ou CTRL+G)** pour sélectionner le tableau ***coefs***
      ⇨ ***CTRL+T*** permet d'obtenir un affichage en flottant
      ↳ Observer les valeurs des coefficients : 0.1 | 0.2 | 0.4 | 0.2 | 0.1

17. Dans la même fenêtre, visualiser les tableaux d'entrée, de sortie et de travail
    - ***Bouton droit de la souris*** (sur la fenêtre) **→ *menu contextuel***
      ⇨ Puis **Go to …** sélectionner ***dline***
      ⇨ **CTRL+T** permet d'obtenir un affichage en flottant
      ↳ Observer les valeurs d'initialisation des entrées et des sorties

18. Placer un point d'arrêt à l'adresse ***0x80306***
    - Positionner le curseur sur l'instruction se trouvant à cette adresse (mémoire de code)
    - Puis appuyer sur **F9**.
    - Exécuter le programme en mode **run** en appuyant sur **F5**

19. Exécuter le programme en pas à pas (**F11**) jusqu'à l'étiquette ***begin***

20. Exécuter en pas à pas jusqu'à l'étiquette ***rif_init***
    - Observer les écritures dans les registres d'adresse de **DAG1** et **DAG2** et l'exécution des deux instructions qui suivent le ***call rif_init***
    - Pourquoi sont-elles exécutées ?

21. Après l'arrêt du programme, exécuter en pas à pas jusqu'à l'étiquette ***filtering***
    ↳ Observer la gestion des boucles et l'exécution d'un calcul du filtre

22. Placer un point d'arrêt à l'étiquette ***filtering*** et exécuter la suite du programme en mode **run**.

23. Tracer la courbe des résultats du filtrage
    - **View → Debug windows → plot → New …**

24. Réinitialiser le processeur et recharger le programme
    - **Debug → Reset** (pas toujours indispensable)
    - **File → Reload Program (CTRL+R)**

25. Modifier les valeurs d'entrées et/ou des coefficients
    ⇨ Cliquer sur la ligne à modifier avec le bouton droit de la souris → **edit**
    ⇨ Modifier et valider avec la touche entrée.

26. Ré-exécuter le programme et observer le résultat du filtrage
    Remarque : Avec la même méthode, on peut modifier le contenu de tous les registres du processeur.

27. Avant de quitter ***VisualDSP++***, faire une sauvegarde du groupe de projets (***Project Group***) que vous allez utiliser dans les prochains TP (fichier ***"votre_binomes.dpg"*** à placer dans le dossier TP_SHARC).

28. Désactiver les buffers circulaires et vérifier le fonctionnement non linéaire des buffers (DAG1 & DAG2).

29. Tester la version optimisée du filtre (vue en cours).

RIFTEST.ASM

```
/************************************************************************
        PROGRAMME DE TEST DU FILTRE A REPONSE IMPULSIONNELLE FINIE
        Fourni par Analog Devices
************************************************************************/

/************************************************************************
File Name : RIFTEST.ASM
Version  ADSP-21262

Purpose This program is a shell program for the RIF.ASM code. This program
initializes buffers and pointers. It also calls the RIF.ASM code.
************************************************************************/
#include "def21262.h"
#define SAMPLES 0x9 /*nombre d'échantillons à filtrer*/
#define TAPS 5          /* nombre de coefficients*/

.EXTERN rif;   /* étiquette définie dans un autre fichier */
/************************************************************************/
.SEGMENT /PM seg_pmda;         /* les données accessibles par le bus PM */
        .VAR coefs[TAPS] = "rifcoefs.dat";    /* coefficients du RIF stockés initialement dans un fichier texte */
.ENDSEG;
/************************************************************************/
.SEGMENT /DM seg_dmda;    /* les données accessibles par le bus DM */

        /* buffer qui contient les 5 derniers échantillons d'entrée nécessaires au calcul de la réponse du filtre */
        .VAR dline[TAPS];

        /* buffer entrée simulant un échelon initialisé à 1 */
        .VAR inbuf[SAMPLES]=1., 1., 1., 1., 1., 1., 1., 1., 1.;

        /* buffer de sortie initialisé à -99 */
        .VAR outbuf[SAMPLES]=-99.,-99., -99.,-99.,-99.,-99.,-99.,-99.,-99.;
.ENDSEG;


/************************************************************************/
/* Exécuté au RESET */
.SEGMENT /PM seg_rth;
    initial_setup:          /* début zone vecteur IT : RESET */
                    nop; nop; nop; nop;
                    NOP;              /* introduit un waitstate  */
                    BIT SET MODE1 CBUFEN;    /* Active les buffers circulaires */
                    JUMP begin;    /* 1st instr. exécutée après reset */
.ENDSEG;
/************************************************************************/

.SEGMENT /PM seg_pmco;

      begin:
          l0=TAPS;                /*initialisation pointeur sur buffer dline*/
          b0=dline;
          m0=1;

          l1=0;                   /*initialisation pointeur sur buffer d'entrée*/
          b1=inbuf;
```

```
        l2=0;                   /*initialisation pointeur sur buffer de sortie*/
        b2=outbuf;

        l8=TAPS;                /*initialisation pointeur sur buffer des coefficients*/
        b8=coefs;

        call rif_init (db);     /*initialise le buffer dline à zéro */
        m8=1;
        r0=TAPS;

        lcntr=SAMPLES, do filtering until lce;      /* le filtrage */
                call rif (db);  /*échantillon d'entrée dans F0, sortie renvoyée dans F0*/
                r1=TAPS;
                f0=dm(i1,1);
        filtering: dm(i2,1)=f0;     /*stockage du résultat dans outbuf*/

        done: jump done;            /* programme terminé */

        rif_init:       /* sous programme d'initialisation de dline à zéro */
            lcntr=r0, do zero until lce;
                zero: dm(i0,m0)=0;      /*initialise la ligne à retard à 0*/
        rts;
.ENDSEG;
```

RIF.ASM

```
/*****************************************************************************
        FILTRE A REPONSE IMPULSIONNELLE FINIE (Fourni par Analog Devices)

        File Name : RIF.ASM
        Version  ADSP-21262

        This is a subroutine that implements RIF filter code given coefficients and samples.
*****************************************************************************
Equation Implemented
y(n)=Summation from k= 0 to M of H sub k times x(n-k)

Calling Parameters
f0 = input sample x(n)
r1 = number of taps in the filter minus 1
b0 = address of the delay line buffer
m0 = modify value for the delay line buffer
l0 = length of the delay line buffer
b8 = address of the coefficent buffer
m8 = modify value for the coefficent buffer
l8 = length of the coefficent buffer

Return Values
f0 = output sample y(n)

Registers Affected
f0, f4, f8, f12, i0, i8

Cycle Count
6 + (Number of taps -1) + 1 cache miss (optimize : 2 caches misses)

# PM Locations
7 instruction words
Number of taps locations for coefficents

# DM Locations
Number of taps of samples in the delay line
*****************************************************************************/
```

```
/*************************************************************************
f12 : résultat multiplication
f8 : accumulation résultat
f0 : échantillons entrant (xn) et précédents en cours de calcul puis résultat du calcul
f4 : coefficients
r1 : nombre de coefficients moins 1
*************************************************************************/

.GLOBAL rif;
.EXTERN coefs, dline;
.SEGMENT /PM seg_pmco;

    rif:
        r12=r12 xor r12;
        dm(i0,m0)=f0;                   /* r12 =0 et stockage échantillon entrant dans dline */
        r8=r8 xor r8;
        lcntr=r1, do macs until lce;    /* boucle itérant nb_coeff fois*/
                f0=dm(i0,m0);  /* r8=0 et chargement donnée et coeff  issus de dline et coeff */
                f4=pm(i8,m8);
                f12=f0*f4;
        macs: f8=f8+f12;     /* multiplie, accumule et charge donnée et coeff. itération suivante */
        f0=f8;     /*stockage résultat dans f0 */
        rts;
    rif.END:

.ENDSEG;
```

RIFCOEFFS.DAT

```
0.1
0.2
0.4
0.2
0.1
```

ADSP-21262_ASM.LDF

```
ARCHITECTURE(ADSP-21262)

//
//  ADSP-21262 Memory Map:
//  ----------------------------------------------
//              0x0000 0000 to 0x0003 FFFF  IOP Regs
//        Block0   0x0004 0000 to 0x0004 3FFF  Long word (64) Space (1MB RAM)
//        Block0   0x0004 4000 to 0x0005 7FFF  Internal Memory (Reserved 5MB)
//        Block0   0x0005 8000 to 0x0005 FFFF  Long word (64) Space (2MB ROM)

//        Block1   0x0006 0000 to 0x0006 3FFF  Long word (64) Space (1MB RAM)
//        Block1   0x0006 4000 to 0x0007 7FFF  Internal Memory (Reserved 5MB)
//        Block1   0x0007 8000 to 0x0007 FFFF  Long word (64) Space (2MB ROM)

//        Block0   0x0008 0000 to 0x0008 7FFF  Normal word (32) Space (1MB RAM)
//        Block0   0x0008 8000 to 0x000A FFFF  Internal Memory (Reserved 5MB)
//        Block0   0x000B 0000 to 0x000B FFFF  Normal word (32) Space (2MB ROM)
//  ----------------------------------------------
//              0x000A 0000 to 0x000A AAAA  48-bit addresses
//  ----------------------------------------------
//        Block1   0x000C 0000 to 0x000C 7FFF  Normal word (32) Space (1MB RAM)
//        Block1   0x000C 8000 to 0x000E FFFF  Internal Memory (Reserved 5MB)
//        Block1   0x000F 0000 to 0x000F FFFF  Normal word (32) Space (2MB ROM)
//  ----------------------------------------------
//              0x000E 0000 to 0x000E AAAA  48-bit addresses
//  ----------------------------------------------
//        Block0   0x0010 0000 to 0x0010 FFFF  Short word (16) Space (1MB RAM)
//        Block0   0x0011 0000 to 0x0015 FFFF  Internal Memory (Reserved 5MB)
//        Block0   0x0011 0000 to 0x0015 FFFF  Internal Memory (Reserved 5MB)
//        Block0   0x0016 0000 to 0x0017 FFFF  Short word (16) Space (2MB ROM)

//        Block1   0x0018 0000 to 0x0018 FFFF  Short word (16) Space (1MB RAM)
//        Block1   0x0019 0000 to 0x001D FFFF  Internal Memory (Reserved 5MB)
//        Block1   0x001E 0000 to 0x001F FFFF  Short word (16) Space (2MB ROM)

//  ----------------------------------------------
//  External memory  0x0100 0000 to 0x02FF FFFF
//  ----------------------------------------------

// This linker description file allocates:
//       Internal 256 words of run-time header in memory block 0
//                256 words of initialization code in memory block 0
//                16K words of C code space in memory block 0
//                 7K words of C PM data space in memory block 0
//                16K words of C DM data space in memory block 1
//                 8K words of C heap space in memory block 1
//                 8K words of C stack space in memory block 1

#ifndef __NO_STD_LIB
SEARCH_DIR( $ADI_DSP/212xx/lib )
#endif

// The I/O library provides support for printing hexadecimal constants
// using the "%a" conversion specifier, and for registering alternatives to
// the default device driver that is supported by the VisualDSP++ simulator
// and EZ-KIT Lite systems. Those applications that do not require this
// functionality may define the macro __LIBIO_LITE which will select an
// alternative I/O library and lead to reduced code occupancy.
```

```
#ifdef __LIBIO_LITE
        # define LIBIO libio_lite.dlb
        # define LIBIOMT libio_litemt.dlb
#else
        # define LIBIO libio.dlb
        # define LIBIOMT libiomt.dlb
#endif

#ifdef _ADI_THREADS
        #ifdef __ADI_LIBEH__
                $LIBRARIES = libehmt.dlb, libc26xmt.dlb, LIBIOMT, libdsp26x.dlb;
        #else
                $LIBRARIES = libc26xmt.dlb, LIBIOMT, libdsp26x.dlb;
        #endif
#else
        #ifdef __ADI_LIBEH__
                $LIBRARIES = libeh.dlb, libc26x.dlb, LIBIO, libdsp26x.dlb;
        #else
                $LIBRARIES = libc26x.dlb, LIBIO, libdsp26x.dlb;
        #endif
#endif

// Libraries from the command line are included in COMMAND_LINE_OBJECTS.
$OBJECTS = $COMMAND_LINE_OBJECTS;

MEMORY
{
 seg_rth  { TYPE(PM RAM) START(0x00080000) END(0x000800ff) WIDTH(48) }
 seg_init { TYPE(PM RAM) START(0x00080100) END(0x000801ff) WIDTH(48) }
 seg_int_code { TYPE(PM RAM) START(0x00080200) END(0x000802ef) WIDTH(48) }
 seg_pmco { TYPE(PM RAM) START(0x000802f0) END(0x000841ff) WIDTH(48) }
 seg_pmda { TYPE(PM RAM) START(0x00086300) END(0x00087fff) WIDTH(32) }

   seg_dmda { TYPE(DM RAM) START(0x000c0000) END(0x000c7fff) WIDTH(32) }
 seg_sram { TYPE(DMAONLY DM) START(0x01000000) END(0x02ffffff) WIDTH(8) }
}


PROCESSOR p0
{
   LINK_AGAINST( $COMMAND_LINE_LINK_AGAINST)
   OUTPUT( $COMMAND_LINE_OUTPUT_FILE )

   SECTIONS
   {
      // .text output section
      seg_rth
      {
         INPUT_SECTIONS( $OBJECTS(seg_rth) $LIBRARIES(seg_rth))
      } > seg_rth

      seg_init
      {
         ldf_seginit_space = . ;
         INPUT_SECTIONS( $OBJECTS(seg_init) $LIBRARIES(seg_init))
      } > seg_init

      seg_int_code
      {
         INPUT_SECTIONS( $OBJECTS(seg_int_code) $LIBRARIES(seg_int_code))
      } > seg_int_code
```

```
seg_pmco
      {
        INPUT_SECTIONS( $OBJECTS(seg_pmco) $LIBRARIES(seg_pmco))
      } > seg_pmco

      seg_pmda
      {
        INPUT_SECTIONS( $OBJECTS(seg_pmda) $LIBRARIES(seg_pmda))
      } > seg_pmda


      .bss ZERO_INIT
      {
       INPUT_SECTIONS( $OBJECTS(.bss) $LIBRARIES(.bss))
      } > seg_dmda

      seg_dmda
      {
        RESERVE(heaps_and_stack, heaps_and_stack_length = 16K)
        INPUT_SECTIONS( $OBJECTS(seg_dmda) $LIBRARIES(seg_dmda))
        RESERVE_EXPAND(heaps_and_stack, heaps_and_stack_length)
        ldf_stack_space = heaps_and_stack;
        ldf_stack_end = ldf_stack_space + ((heaps_and_stack_length * 8K) / 16K);
        ldf_stack_length = ldf_stack_end - ldf_stack_space;
        ldf_heap_space = ldf_stack_end;
        ldf_heap_end = ldf_heap_space + ((heaps_and_stack_length * 8K) / 16K);
        ldf_heap_length = ldf_heap_end - ldf_heap_space;
      } > seg_dmda

      seg_sram
      {
        INPUT_SECTIONS($OBJECTS(seg_sram) $LIBRARIES(seg_sram))
        PACKING(5   B0 B0 B0 B4 B0
                    B0 B0 B0 B3 B0
                    B0 B0 B0 B2 B0
                    B0 B0 B0 B1 B0)
      } > seg_sram
   }
}
```