TP RII (FILTRAGE RECURSIF)

(langage C, sans E/S) → dossier : rii_C

Objectifs du TP

- Découvrir l'environnement de développement **ide** de ***VisualDSP++***
- Se familiariser avec le **compilateur C** et le **simulateur**
- Etudier le passage des paramètres aux fonctions dans des registres

Explication du thème

Pour découvrir l'environnement intégré de **VisualDSP++**, nous allons étudier sous forme d'exemple l'implantation d'un filtre récursif en C. Un filtre récursif est caractérisé par l'équation de récurrence suivante :

$$s_n = \sum_{j=0}^{q} b_j * x_{n-j} + \sum_{i=1}^{p} a_i * s_{n-i}$$

La sortie s_n est obtenue par sommation pondérée d'un certain nombre d'échantillons d'entrée et d'un certain nombre de valeurs de sortie antérieures à s_n.
Dans le cadre de ce programme, on s'intéresse à un filtre récursif d'ordre 1. Le nombre de coefficients de pondération est réduit à 2.
L'équation de récurrence est :

$$s_n = b * x_n + a * s_{n-1}$$

L'exemple est écrit en C en respectant la norme ANSI. Il est donc portable et peut être exécuté par une cible quelconque. Il est constitué d'un seul fichier : ***"rii.c"***.
Le listing du programme est joint à ce document.
Les coefficients sont fixés directement dans le programme principal.
Les échantillons x_n à filtrer sont stockés dans un fichier texte.
Trois fichiers sont fournis pour servir d'exemples : ***« carre_100b.dat »***, ***« high.dat »*** et ***« low.dat »***.

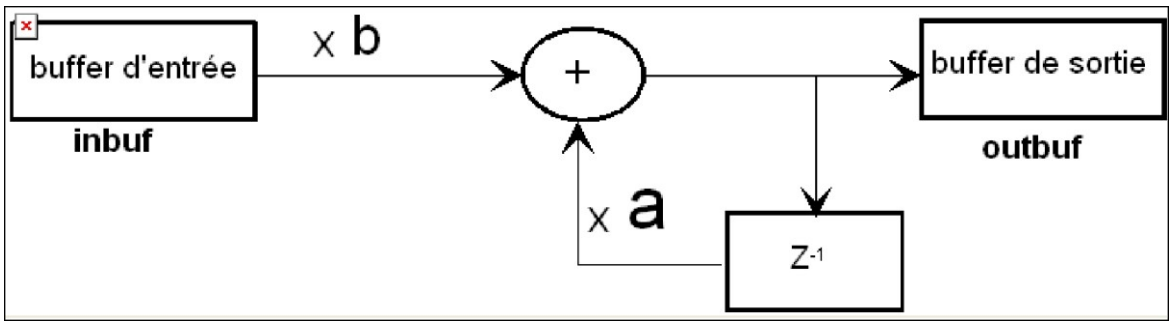
Le contenu du fichier de données est chargé dans le buffer **inbuf** à la compilation, grâce à l'utilisation de la directive d'inclusion du langage C :

```
float inbuf[] = {
        #include "high.dat"
};
```

Le programme appelle la fonction **rii()** pour traiter les échantillons. Les buffers ***inbuf*** et ***outbuf*** sont passés en arguments.

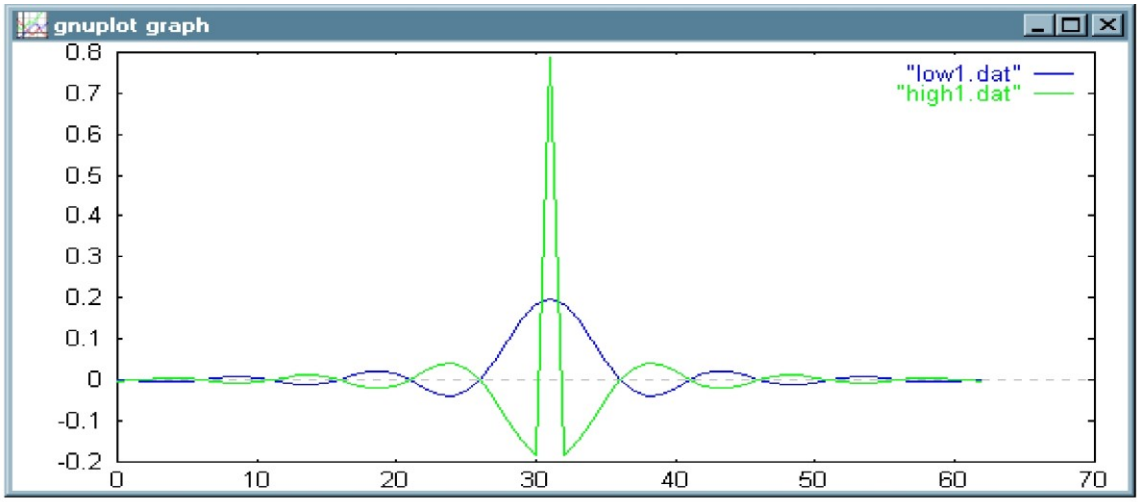
```
rii(coeff_x, coeff_s, &inbuf[0], SAMPLES, &outbuf[0]);
```

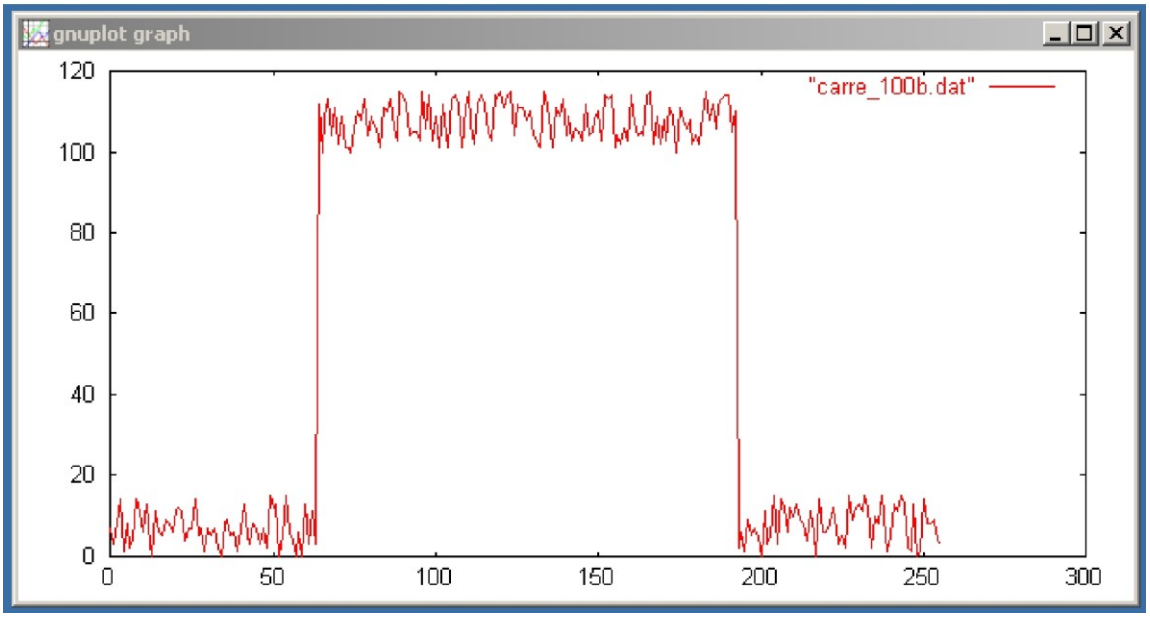
Les résultats du traitement sont renvoyés dans le buffer (***outbuf***).
La figure suivante montre le schéma de réalisation du filtre. Z^{-1} représente un tampon mémorisant le résultat antérieur à l'échantillon traité.



Un autre fichier de type LDF (Linker Description File) est nécessaire à l'édition de lien; il s'agit du fichier : « `ADSP-21262.LDF` ». Le fichier LDF (architecture) permet de déterminer les adresses absolues des différents segments à utiliser pour implanter les données et le code du programme. Le format LDF sert à développer des programmes pour des systèmes multiprocesseurs. Sa syntaxe permet de spécifier comment procéder pour générer le code exécutable à charger dans le système cible. Il est directement utilisé par l'éditeur de liens pour produire le code.

Exemples de signaux à traiter





Scénario du TP

❖ Lancer ***VisualDSP++***

➢ ***Menu démarrer ➔Analog Devices ➔VisualDSP++ (Environment)***

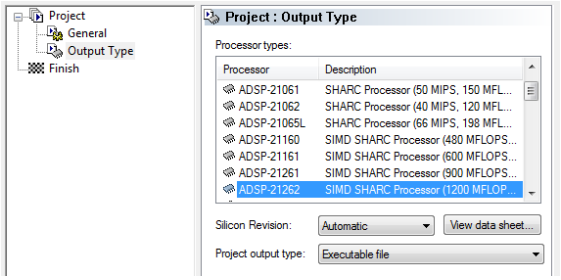
La fenêtre graphique qui se lance, ***VisualDSP++ Environment***, représente l'environnement de développement que bous allons utiliser pour les différents thèmes. **VisualDSP++** est un **environnement de développement intégré (***IDE* en anglais pour ***I****ntegrated* ***D****evelopment* ***E****nvironment*)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

❖ Fermer les fenêtres d'édition de texte.

❖ Ouvrir le groupe de projets crée lors du TP1 (rif_ASM) et créer un nouveau projet appelé « **rii_C** »

➢ **Project ➔New** → puis :

- Choisir les options suivantes :
  - Processeur : ***«ADSP-21262»***
  - Type de projet ***«Standard Application»***
- Parcourir les autres catégories pour découvrir les différentes options disponibles.



❖ Construire le projet en ajoutant les 2 fichiers : « ***ADSP-21262.LDF*** » et « **rii.c** »

➢ **Project➔add file(s)** → ou cliquer sur l'icône de la barre d'outils

↳ Editer les 2 fichiers et organiser votre environnement de travail à votre convenance.

❖ Compiler le projet et corriger le code, si erreurs et/ou warnings

➢ **Project ➔ Build project** (chercher l'icône correspondante dans la barre d'outils)

⇨ La simulation est lancée.

⇨Un point d'arrêt est automatiquement inséré sur la première ligne du programme principal (fonction main).

❖ Afficher dans la même fenêtre le code source en C et son équivalent en assembleur

➢ **<Bouton droit de la souris>** → ***Affichage d'un menu contextuel***

⇨ **(goto .., find .., source et mixed)**

↳ Une autre fenêtre affiche le code en assembleur en permanence (fenêtre ***Disassembly***).

↳ Rechercher le code assembleur de la fonction ***rii*** (généré par le compilateur) puis l'analyser.

❖ Utiliser l'outil de ***« Profiling »*** pour détecter les « boucles critiques »

↳ **Demander de l'aide au professeur présent si vous rencontrez des difficultés**

❖ Rechercher la ligne correspondant à l'appel de la fonction ***rii*** et placer un point d'arrêt

➢ *<bouton gauche de la souris>* ***puis*** cliquer 2 fois (et rapidement) sur la ligne

↳ Un point rouge doit s'afficher au début de la ligne.

❖ Afficher le contenu des buffers d'entrée et de sortie (***inbuf*** et ***outbuf***) dans 2 fenêtres mémoires séparées.

➢ **Memory ➔Two Column** ***puis*** **view➔goto ...** (utiliser le browser ou taper ***_inbuf*** et ***_outbuf***)

❖ Afficher le contenu des buffers en ***float*** 32 bits

➢ *<Bouton droit de la souris>* → ***menu contextuel*** → **Select Format** (.., .., **floating point 32 bits, ...**)

❖ Lancer l'exécution jusqu'au point d'arrêt

➢ **Debug ➔ run** (découvrir la variété des raccourcis dans la barre d'outils des modes d'exécution)

❖ Afficher les registres généraux
- ➢ **Register ➔ Core ➔ Register File**
  - ↳ Vérifier le contenu des registres utilisés dans le passage des arguments
  - ↳ Exécuter pas à pas quelques instructions du code de la fonction ***rii*,**

❖ Terminer l'exécution du programme
- ➢ **Debug ➔ run**
  - ↳ Vérifier que le contenu du buffer ***outbuf*** a bien été mis jour (résultat du traitement)

❖ Sauver le contenu des 2 buffers dans 2 fichiers textes (***"inbuf.dat"*** et ***"outbuf.dat"***)
- ➢ *<Bouton droit de la souris sur la fenêtre mémoire>* ➔ ***menu contextuel*** ➔ (.., **Dump**..., ...,)
  - ↳ Remplir la fenêtre de dialogue en vérifiant minutieusement les différents champs (ne pas oublier de décocher la case pour que le format ne soit pas écrit dans le fichier de sortie.

❖ Dans la rubrique ***tools*** des ***préférences***, ajouter un raccourcis "***plot"*** sur l'utilitaire **gnuplot.exe** (dossier UTILS). Ce dernier permet de tracer le contenu d'un fichier de donnees sous forme graphique (courbe).
- ➢ **Settings ➔ Preferences ➔ tools …**

❖ Tracer les courbes correspondantes aux données contenues dans les fichiers ***"inbuf.dat"*** et ***"outbuf.dat"***
- ➢ **Tools ➔ plot**
  - ↳ Il faut se positionner dans le bon répertoire en tapant dans la fenêtre ***gnuplot*** :
    gnuplot> *cd "nom_repertoire"*
  - ↳ Afficher les 2 courbes en tapant :
    gnuplot> *plot "inbuf.dat" with lines, "outbuf.dat" with lines*

❖ Créer une version optimisée du projet. Pour ce faire, il faut changer l'option de compilation du projet.
- ↳ Se placer dans la fenêtre ***Project*****,** puis :
- ➢ *<Bouton droit de la souris >* ➔ ***menu contextuel*** ➔ **Project Options…**
  - ⇨ Dans la rubrique ***project*** : cocher ***release*** concernant le champs ***Settings for configuration***
  - ⇨ Dans la rubrique ***Compile*** : cocher ***Enable optimization*** dans la catégorie ***General***

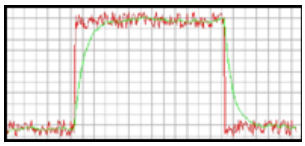
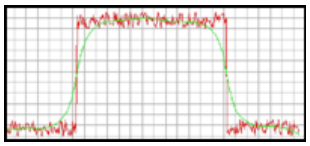
❖ Compiler le projet et vérifier que c'est l'exécutable du dossier **Release** qui est chargé dans le **simulateur.**

❖ Comparer le code assembleur ainsi généré avec le code de la version ***Debug*** (notamment les boucles).
- ➢ **File ➔ Load Program…**

❖ Exécuter le programme à nouveau et visualiser les registres compteur de boucles
- ➢ **Register ➔ Core ➔ Counters**

❖ Modifier le fichier ***"rii.c"*** pour traiter le signal ***"carre_100b.dat"***
- ↳ Le tracé du signal de sortie montre un signal asymétrique (ce qui n'est pas le cas du signal d'entrée)
- ↳ **Pourquoi ?** .....
  .....

❖ Pour compléter le code du filtre, ajouter la partie **anti-causale** à la fonction ***rii*** et vérifier que le résultat ressemble à celui de la figure ci-après.

❖ Ecrire une nouvelle fonction ***filtre_rii ()*** qui réalise le traitement complet en appliquant les deux parties **causal** et **anti-causal** sur le même buffer (pas de buffer tampon). **Cette fonction est à garder puisqu'elle sera utilisée plus tard (TP Deriche)** ➔ Prototype : ***void filtre_rii (float b, float a, float buffer[], int N);***

❖ Valider la fonction ***filtre_rii()*** en simulation (session ***simulator***) et en debug (session ***Ez-kit Lite***).

rii.c

```
#include <stdio.h> #include <stdlib.h>

/* Prototype de la fonction rii.c */ /*********************************/
extern void rii(float b, float a, float in[], int N, float out[]);

float inbuf[] =
{
        #include "high.dat"
};

float outbuf[sizeof(inbuf)]; /* sizeof: Calcule le nombre d'échantillons */
unsigned SAMPLES = sizeof(inbuf);

void main()
{
        float coeff_x = 0.125; /* coeff de xn */
        float coeff_s = 0.875; /* coeff de sn-1 */
        int i;

        rii(coeff_x, coeff_s, &inbuf[0], SAMPLES, &outbuf[0]);

        exit(0);
}

/***********************************************
Filtre recursif d'ordre 1 (sn = b*xn + a*sn-1)
Arguments : rii (b,a,in,N,out)
========
        a,b : coeffs du filtre
        in : tableau des echantillons N : Nbre d'echantillons
        out : tableau des echantillons filtres
***********************************************/
void rii(float b, float a, float in[], int N, float out[])
{
        int i;
        float sn_1=0.;

        for (i=0; i<N; i++)
                out[i] = sn_1 = b*in[i] + a*sn_1;
}
```

ADSP-21262.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, libcppehmt.dlb, libcpprtehmt.dlb;
#else
$LIBRARIES = libc26xmt.dlb, LIBIOMT, libdsp26x.dlb, libcppmt.dlb, libehmt.dlb, libcpprtmt.dlb;
#endif
#else
#ifdef __ADI_LIBEH__
$LIBRARIES = libeh.dlb, libc26x.dlb, LIBIO, libdsp26x.dlb, libcppeh.dlb, libcpprteh.dlb;
#else
$LIBRARIES = libc26x.dlb, LIBIO, libdsp26x.dlb, libcpp.dlb, libeh.dlb, libcpprt.dlb;
#endif
#endif

// Libraries from the command line are included in COMMAND_LINE_OBJECTS.
#ifdef __cplusplus
#ifdef EZKIT_MONITOR
# ifdef _ADI_THREADS
#   define CRT_HDR 262_cpp_hdr_ezkit_mt.doj
# else
#   define CRT_HDR 262_cpp_hdr_ezkit.doj
# endif // _ADI_THREADS
#else
# ifdef _ADI_THREADS
#   define CRT_HDR 262_cpp_hdr_mt.doj
# else
#   define CRT_HDR 262_cpp_hdr.doj
# endif // _ADI_THREADS
#endif //EZKIT_MONITOR
#else
#ifndef EZKIT_MONITOR
# define CRT_HDR 262_hdr.doj
#else
# define CRT_HDR 262_hdr_ezkit.doj
#endif //EZKIT_MONITOR
#endif
$OBJECTS = CRT_HDR, $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) }

#ifdef __cplusplus
   mem_ctdm { TYPE(DM RAM) START(0x000c0000) END(0x000c00ff) WIDTH(32) }
#ifdef IDDE_ARGS
#define ARGV_START 0xC0100
   mem_argv { TYPE(DM RAM) START(0x000c0100) END(0x000c01ff) WIDTH(32) }
   seg_dmda { TYPE(DM RAM) START(0x000c0200) END(0x000c7fff) WIDTH(32) }
#else
   seg_dmda { TYPE(DM RAM) START(0x000c0100) END(0x000c7fff) WIDTH(32) }
#endif
#else
#ifdef IDDE_ARGS
#define ARGV_START 0xC0000
   mem_argv { TYPE(DM RAM) START(0x000c0000) END(0x000c00ff) WIDTH(32) }
   seg_dmda { TYPE(DM RAM) START(0x000c0100) END(0x000c7fff) WIDTH(32) }
#else
   seg_dmda { TYPE(DM RAM) START(0x000c0000) END(0x000c7fff) WIDTH(32) }
#endif
#endif
 seg_sram { TYPE(DMAONLY DM) START(0x01000000) END(0x02ffffff) WIDTH(8) }
}
```

```
PROCESSOR p0
{
   #ifdef IDDE_ARGS
      RESOLVE(___argv_string, ARGV_START)
   #endif
   #ifdef __cplusplus
      KEEP( _main,___ctor_NULL_marker,___lib_end_of_heap_descriptions )
   #else
      KEEP( _main,___lib_end_of_heap_descriptions )
   #endif
      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

  #ifdef __cplusplus
           dxe_ctdm
           {
             __ctors = .;    /* points to the start of the section */
             INPUT_SECTIONS( $OBJECTS(seg_ctdm) $LIBRARIES(seg_ctdm))
             INPUT_SECTIONS( $OBJECTS(seg_ctdml) $LIBRARIES(seg_ctdml))
           } > mem_ctdm
  #endif

     .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))
  #ifdef __cplusplus
        INPUT_SECTIONS( $OBJECTS(.gdt) $LIBRARIES(.gdt))
        INPUT_SECTIONS( $OBJECTS(.gdtl) $LIBRARIES(.gdtl))
        INPUT_SECTIONS( $OBJECTS(.frt) $LIBRARIES(.frt))
        INPUT_SECTIONS( $OBJECTS(.rtti) $LIBRARIES(.rtti))
        INPUT_SECTIONS( $OBJECTS(.cht) $LIBRARIES(.cht))
        INPUT_SECTIONS( $OBJECTS(.edt) $LIBRARIES(.edt))
        INPUT_SECTIONS( $OBJECTS(seg_vtbl) $LIBRARIES(seg_vtbl))
  #endif
```

```
    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
 }
}
```