

TP 1

Langages, techniques et outils

Objectif

Étapes après étapes, ce TP fait le tour d'un panel d'outils de base nécessaires au développement d'un programme embarqué. Vous les utiliserez pour développer, tester et déboguer un programme destiné à une cible LEON.

Vous produirez un rapport de TP contenant vos réalisations et observations.

>> Les éléments indispensables à ce rapport sont signalés de cette manière (en bleu).

Créez un projet sous Eclipse

File > New > C project

Donnez lui un nom. Sélectionnez le type Exécutable > Hello World ANSI C Project. Sélectionnez la chaîne de compilation « Cross GCC ». Indiquez ensuite le chemin du dossier « bin » de bcc, ainsi que le préfix par lequel commence tous les exécutables (ex: « sparc-gaisler-elf- »).

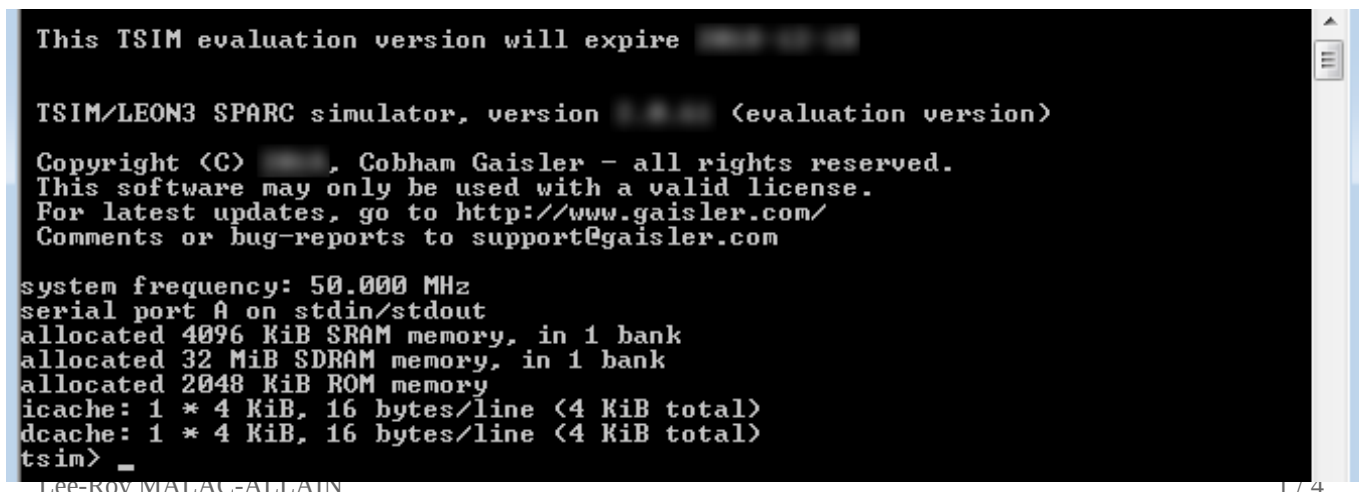
Le projet apparaît dans l'onglet « Project Explorer ». D'un clic droit sur celui-ci, lancez la compilation « Build Project » (ou via l'icône de marteau).

L'onglet « Console » indique les commandes jouées par la chaîne de compilation. Les fichiers compilés ont été créés dans le projet.

Exécutez un programme sur simulateur

Démarrez `tsim-leon3.exe`

Il est possible d'obtenir une brève documentation des commandes TSIM en tapant « help » dans le terminal du simulateur.



```
This TSIM evaluation version will expire 2005-12-31

TSIM/LEON3 SPARC simulator, version 3.3.3 (evaluation version)

Copyright (C) 2005, Cobham Gaisler - all rights reserved.
This software may only be used with a valid license.
For latest updates, go to http://www.gaisler.com/
Comments or bug-reports to support@gaisler.com

system frequency: 50.000 MHz
serial port A on stdin/stdout
allocated 4096 KiB SRAM memory, in 1 bank
allocated 32 MiB SDRAM memory, in 1 bank
allocated 2048 KiB ROM memory
icache: 1 * 4 KiB, 16 bytes/line (4 KiB total)
dcache: 1 * 4 KiB, 16 bytes/line (4 KiB total)
tsim> _
```

Chargez l'exécutable :

```
tsim > load c:\workspace\nom_du_projet\chaine_de_compilation\
nom_du_projet
```

Démarrez l'exécutable :

```
tsim > run
```

Le message « Hello world » apparaît : tout fonctionne.

Passons aux choses sérieuses.

Développez le générateur de données

Déclarez un tableau global de 20 `uint32_t`, type défini dans la bibliothèque `stdint.h`.

Dans un fichier `*.c` dédié, développez une fonction répondant au prototype suivant .

```
/**
 * Remplis un tableau avec la suite de Fibonacci.
 * @param output_array tableau où la suite est enregistré
 * @param size nombre de valeur à inscrire dans \a output_array
 * @param min_value valeur jusqu'à laquelle les nombres ne sont pas enregistrés
 */
void fibonacci(uint32_t output_array[], uint32_t size, uint32_t min_value);
```

Les fonctions que vous développerez devront être documentées, au minimum, au format Javadoc.

>> [Quel est le code de cette fonction ?](#)

Dans le main, appelez cette fonction avec le tableau global en paramètre.

Faire suivre l'appel à `fibonacci()` de l'appel à une fonction de test qui ne fait rien.

Celle-ci servira à mettre un point d'arrêt.

N.B. Il est possible de désactiver certaines des vérifications effectuées par l'interpréteur intégré à Eclipse. En particulier lorsque celles-ci sont fausses :

`Windows > Preferences > C/C++ > Code Analysis`

Les vérifications pertinentes sont faites par les outils de la chaîne de compilation que l'on utilise. Ces warnings et erreurs se trouvent dans la console.

Examinez des données grâce à un script GDB

Tel que défini dans le cours (diapositives 92, 93, 95), lancez le simulateur TSIM avec un serveur GDB, puis démarrez GDB (`C:\opt\bcc-2.2.0-gcc\bin\sparc-gaisler-elf-gdb.exe`) et connectez celui-ci au serveur GDB.

Définir un point d'arrêt sur la méthode de test et afficher les données du tableau.

N.B. Dans un script GDB, toute fin de ligne derrière « # » est un commentaire.

Développez le consommateur de données

Dans un fichier *.c dédié, développez une fonction prenant en entrée deux tableaux de float et un entier positif. Considérant le premier tableau comme contenant des valeurs de rayon, la fonction itérera le nombre de fois défini en entrée et enregistrera dans le second le résultat du calcul du périmètre.

N.B. : une bonne approximation de π s'obtient du produit 103993 / 33102

Définissez un second tableau global, de type float.

Considérant la suite de Fibonacci comme une suite pseudo-aléatoire, castez ce tableau ((float*)output_array) pour le mettre en paramètre d'entrée de la nouvelle fonction. Le second tableau est un paramètre de sorti.

>> Quel est le script minimal nécessaire à l'affichage des deux tableaux, avant après le calcul des périmètres ?

Créez des sections

Pour ajouter des sections au linker script par défaut, commencez par le récupérer.

Pour cela, commencer par récupérer dans la console de build de votre projet le ligne relative au linkage. On la trouve en fin de build faisant référence à tous les fichiers *.o du projet.

Rejouez cette commande dans un terminal en utilisant l'exécutable de linkage de votre bcc (bcc-2.2.0-gcc\bin\sparc-gaisler-elf-ld.exe) et en ajoutant l'option « — verbose ».

La sortie obtenue contient le linker script par défaut, délimité par des lignes de « = ». Le copier dans un fichier et le spécifier dans les options de linkage du projet (cf. slide 40) :

```
Propriétés du projet > C/C++ Build > Settings > Cross GCC Linker >
Miscellaneous
```

Si à la recompilation une erreur se produit sur la section « .init », la commenter dans le linker script.

Via le mot clé « `__attribute__` », placez les tableaux dans une zone mémoire que vous aurez ajouté au linker script sur le modèle de la section « `.bss` ». Fixez dans le linker script l'adresse de cette section.

Produire la linker map.

>> Quel(s) passage(s) avez-vous du ajouté ou modifié dans le linker script par défaut ?

>> Quelles parties de la linker map donnent des renseignements sur les fonctions que vous avez écrites ? Pour vos tableaux globaux ? Qu'apprenez-vous concernant la fonction calculant le périmètre ? À propos du tableau de float ?

Générer fichier SREC

Produisez le fichier SREC de l'exécutable au moyen de l'utilitaire objcopy (`bcc-2.2.0-gcc\bin\sparc-gaisler-elf-objcopy.exe`).

Testez son chargement dans TSIM comme un exécutable et vérifiez, en affichant les zones mémoire concernées (commande `mem`), que les tableaux globaux ont bien été alimentés.

>> D'après l'image de votre exécutable au format SREC, quelle est l'adresse du premier octet de cette image et l'adresse du dernier octet ?


Optimiser et débbuguer

En plaçant dans le script GDB un appel à `mon_perf_reset` avant le « `start` » et `mon_perf` après, on obtient le temps d'exécution du programme.

Le programme travaillant avec des nombres flottants, une opération importante pour être représentatif de la cible est d'utiliser la Floating Point Unit (FPU) de la cible.

Dans les paramètres d'appel de `tsim-leon3.exe`, ajouter « `-grfpu` » pour utiliser l'IP core FPU fournis avec les processeurs LEON vendu par Gaisler.

Une donnée empêche alors le programme de s'exécuter correctement.



```
Program received signal SIGSEGV, Segmentation fault.  
start() at 0x0000000000000000 (no symbol)  
_start() at 0x0000000000000000 (no symbol)
```

>> À l'aide de points d'arrêts, identifiez l'instruction et les données en cause.

Produisez une sortie entrelaçant le code C et le code assembleur.

>> Quelle est l'instruction C et les instructions assembleur où se produit l'erreur.

>> Interpréter le problème et proposez une solution.