Rapport de stage de fin d'etudes

13 June 2018

Corentin Gay

Contents

1 Resume **3**

2 Introduction **4**
2.1 Rappel du sujet de stage 4
2.2 Presentation de l'entreprise 6

3 Aspects organisationnels **7**
3.1 Decoupage du stage 7
3.2 Diagramme de Gantt, Kanban ?? 7
3.3 Points de controle 7
3.4 Situations de Crise ????? Kesako 7

4 Aspects techniques **7**
4.1 Objectifs 8
4.1.1 Alternatives 8
4.2 Cadre du stage dans l'entreprise 8
4.3 Propositions retenues ou pas 8

5 Bilan **8**

1 Resume

J'ai decouvert Ada lors des cours a EPITA donnes par Raphael Amiard. J'ai trouve le langage tres interessant car les concepts (notamment l'oriente objet) sont assez differents de leurs equivalents en C++. Lors de ce cours, nous avions essaye de faire fonctionner un emulateur Gameboy code en C++ avec un programme Ada qui devait se charger de l'affichage, tout cela tournant sur une carte STM32F729. Malheureusement le projet n'a pas abouti mais j'en ai tout de meme garde

Parler des trucs rtos ??

parler de faire des trucs embedded mais d'avoir a les integrer dans un IDE ?? possibilite d'en apprendre plus sur le fonctionnement des runtimes et de leur role dans le developpement embarque en Ada

Plusieurs facteurs ont affecte mon choix pour ce stage. Le premier est l'opportunite d'integrer mon travail dans un outil pre-existant. Ce point me semble important car il permet de comprendre le contexte technique dans lequel sera utilise mon travail. Le second point est le sujet qui me permet de toucher a plusieurs technologies tout en restant dans mon domaine de predilection, le developpement embarque. Par exemple, en apprendre plus sur les differents processeurs ARM et leurs assembleurs differents me paraissait etre une bonne experience a avoir.

En Ada, il existe un concept de runtime, cette runtime est le logiciel qui va permettre d'utiliser certaines fonctionnalites du langage (par exemple : un allocateur memoire).

Plusieurs types de runtime, 2 principaux types:

- runtimes ravenscar : possedent presque les features d'un RTOS
- runtimes ZFP : minimum pour faire tourner du code Ada

Par exemple, la ZFP fournit un allocateur memoire utilisant une pile, avec une fonction `free` qui ne fait rien. Etant donne cet allocateur memoire il est impossible d'avoir une propagation d'exception efficace car on ne peut pas recupere la memoire allouee lorsqu'une exception est levee.

Dans le contexte de ce stage nous ne parlerons que de runtimes ZFP.

C'est quoi le linker script et le startup code ????

Dans ce cadre la, lorsque je veux developper pour une nouvelle cible en

compilation croisee, il faut recompiler une runtime pour en generer une adaptee a la cible. Pour cela il faut modifier le `linker script` pour representer le `mapping memoire` et le `startup code` afin d'utiliser l'assembleur de la cible.

```
with Ada.Text_IO;

procedure Test (Entier : Integer);
Test : Integer := 8;
Test : Integer := 8;
Test : Integer := 8;
Test : Integer := 8;
```

On peut diviser le contenu d'une runtime en trois grandes parties :

- cpu specifiques : architecture and float handling
- device specific : memory mapping
- board specific : additionnal mem mapping + device mapping

Dans le cas de ZFP, le code de la runtime elle-meme est identique d'un materiel a l'autre, c'est le contenu du `linker script` et du `startup code` qui va changer.

Heureusement, ARM a creer un standard qui permet de decrire le materiel d'une famille de boards et de devices. Ce sont les CMSIS-Packs

A partir de ces packs, on a toute les informations necessaires pour generer un linker script et le startup code pour une board donnee.

2 Introduction

Contexte et complexite du stage

2.1 Rappel du sujet de stage

Mon sujet de stage s'intitulait 'Improve baremetal support in GPS' et comportait plusieurs axes. Par exemple : ameliorer la stack view ou bien developper une register view ou encore, 'investiguer' comment rendre la creation d'un projet plus simple et plus generique.

J'ai choisi de m'attaquer a ce dernier. L'idee etait de fournir les outils a l'utilisateur fin qu'il puisse choisir sa cible de developpement et que GPS genere les fichiers necessaires afin de pouvoir commencer a developper apres la creation du projet.

Cependant pour pouvoir executer du code Ada sur une cible donnee, il faut avoir un logiciel appelle une `runtime`. Ce logiciel fournit des fonctionnalitees du langage qui sont utilisee par le programme comme le support multi-taches, la propagation des exceptions ou un allocateur memoire.

Etant donne que le code etant dans la runtime doit tourner sur la cible, il faut adapter la runtime a chaque cible. Actuellement, c'est une etape qui est fait manuellement. Il faut egalement savoir qu'il y a plusieurs types de runtime qui ne fournissent pas toutes les memes fonctionnatlites. Dans le cas de mon stage je me suis attaque a la question des `runtimes` dites `ZFP` pour `zero footprint`. Ce type de runtime est le minimum pour pouvoir faire tourner du code Ada. Par exemple, elle n'a pas de propagation d'exception, pas de support multi-taches et ne possede qu'un allocateur memoire naif.

Dans le cas decrit ci-dessus, les modifications a effectuer dans le code de la runtime elle-meme sont nulles. Cependant, il faut tout de meme modifier le `crt0` qui permet de preparer les differentes memoires et qui appelle la fonction `main`. Il faut egalement modifier le `linker script`, le fichier responsable de la cartographie memoire, afin de specifier les zones memoires et quelles portions du code y mettre.

Ces modifications dependent de la cible et necessite actuellement de lire la documentation afin de recuperer les infos utiles. Dans le cas d'une cible possedant un processeur cortex-m, ARM a creer un standard qui permet de decrire le materiel d'une `board` ou d'un `device` et `packageant` ces infos dans une archive zip. On appelle ces archives des CMSIS-Packs.

La solution est donc d'utiliser ces packs pour automatiser le processus de modification du `startup code` et du `linker script`. Cette automatisation permet a l'utilisateur de choisir sa `board` de developpement lors de la creation d'un projet

2.2 Presentation de l'entreprise

En 1992, l'universite de New York conclut un contrat avec l'`US Air Force` afin de creer un compilateur libre et standard afin d'aider a la diffusion du nouveau standard Ada, Ada 9X (qui deviendra Ada 95). Suite a ce projet, la societe Ada Core Technologies est cree a New York et la societee soeur ACT-Europe est cree deux annees plus tard. Ce n'est qu'en 2012 que les deux societes sont unifiees.

AdaCore fournit un compilateur Ada appelle GNAT en plusieurs versions avec des licenses differentes. La version chaque version des fonctionnalite differentes, par exemple la version `Community` ne supporte que la derniere version du standard Ada, Ada 2012, alors que la version `Assurance` destinee aux projets de certifications ou a des projets de longues durees supporte jusqu'a Ada 83. De plus, avec la version `Community`, tout le code ecrit est soumis a la license GPL tandis qu'avec la version commerciale, une exception est presente dans la license permettant de ne pas etre soumis a la GPL.

Pour aller avec le compilateur, AdaCore peut egalement aider les clients avec des projets de certifications. En effet, une partie des outils fournis par AdaCore, comme GNATcoverage, est qualifie pour le developpement d'outil en DO-178B en DAL A. C'est a dire le niveau de criticite le plus eleve dans l'industrie avionique. GNATcoverage aide a l'analyse de couverture de code ce qui permet de garantie qu'il n'y a pas de code qui n'est jamais execute.

AdaCore a beaucoup de clients dans des domaines ou la presence d'erreurs n'est pas acceptable comme le domaine de l'avionique ou de la defense. Voici quelques projets que des clients d'AdaCore ont realises :

- MDA, une division de Maxar Technologies, va utiliser Ada ainsi que le produit GNAT Pro Assurance afin de realiser le logiciel en charge de la communication espace-terre a bord de l'ISS.
- Real Heart AB est une entreprise suedoise qui travaille sur un coeur totalement artificiel. Afin de garantir le bon fonctionnement du logiciel qui pilote le moteur de la pompe du coeur artificiel, elle a choisi d'utiliser Ada ainsi que le compilateur GNAT Pro fourni par AdaCore.

AdaCore travaille sur un compilateur Ada, GNAT. Ils fournissent des com-

pilateurs croises qui visent une plate-forme specifique a leurs clients. Ils offrent egalement du support pro sur tous leurs produits. Parler des differentes runtimes ? Domaines : defense, aeorospatiale, securite (voir site)

Mon stage se situe dans la perspective d'ameliorer l'experience des utilisateurs de GPS dans le domaine du `bare board`.

Thematiques du stage : bare board, IDE et CMSIS-Packs Actuellement pas d'integration des CMSIS-Packs dans GPS. Contrairement a Eclipse qui supporte parfaitement les pack.

Mon stage se deroule dans l'equipe IDE. Cette equipe s'occupe de la maintenace de l'IDE GPS. Cet IDE utilise les references croisees afin de fournir une meilleure experience de developpement a l'utilisateur. Mon stage s'insere donc parfaitement dans les thematiques de cette equipe.

Ce support est specifique au langage Ada, le concepte de runtime est plus ou moins unique a ce langage.

Rapport a epita: j'ai fait du bareboard et de l'Ada. J'avais deja essaye de faire un projet mixant Ada et C++, mais ce ne s'etait pas fini comme prevu. Pas de compilateur ds la toolchain d'AdaCore. Motivation: ca touchait a du bare metal, mais il fallait quand meme integrer ca dans un ide 'classique' (en Ada lol)

3 Aspects organisationnels

3.1 Decoupage du stage

3.2 Diagramme de Gantt, Kanban ??

3.3 Points de controle

3.4 Situations de Crise ????? Kesako

4 Aspects techniques

Liste:

- generation du startup code

- generation du linker script
- base de donnees representant les packs
- integration dans GPS

4.1 Objectifs

4.1.1 Alternatives

4.2 Cadre du stage dans l'entreprise

4.3 Propositions retenues ou pas

on ne genere pas des runtimes on prend celles de bb_runtimes probablement par raison politique, le code de la runtime n'est pas ouvert au public ## Difficultes eventuelles ## Resultats obtenus avancement

5 Bilan

- pretty good