TP4 : La carte Armadeus APF28

Blottière Paul <blottiere.paul@gmail.com>

Table of Contents

1. Généralités

1.1. embsys

L'ensemble des cours, exemples, PDF et TP est disponible sur le dépôt github https://github.com/pblottiere/embsys.

Si vous voulez cloner entièrement le dépôt :

```
$ git clone https://github.com/pblottiere/embsys
```

Si vous voulez cloner le dépôt mais avoir simplement les labs dans votre répertoire de travail :

```
$ git clone -n https://github.com/pblottiere/embsys --depth 1
$ cd embsys
$ git checkout HEAD labs
```

Si vous souhaitez mettre à jour un dépôt que vous avez préalablement cloné :

```
$ cd embsys
$ git pull
```

Si vous n'avez pas **git**, téléchargez le ZIP sur la page d'acceuil de **embsys**.

Le TP d'aujourd'hui se trouve ici : https://github.com/pblottiere/embsys/labs/4_armadeus.

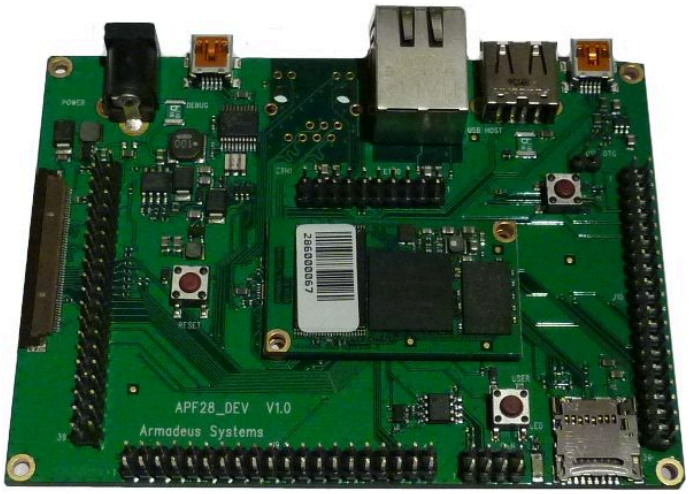
1.2. Attention particulière

Lors de ce TP, vous aurez les droits **root**. Vous devez donc être particulièrement **VIGILANT** à ce que vous faites ainsi qu'aux consignes des exercices afin de ne pas faire d'erreurs malencontreuses…

Il existe une règle simple pour éviter de telles erreurs : ne jamais être identifié root lorsque ce n'est pas nécessaire! Il ne s'agit donc pas ici d'être en root tout le long du TP…

1.3. Objectifs du TP

Dans le cadre de ce TP, nous allons travailler sur la carte Armadeus APF28 à travers sa plateforme de développement :



Le but est ici d'appréhender les points suivants :

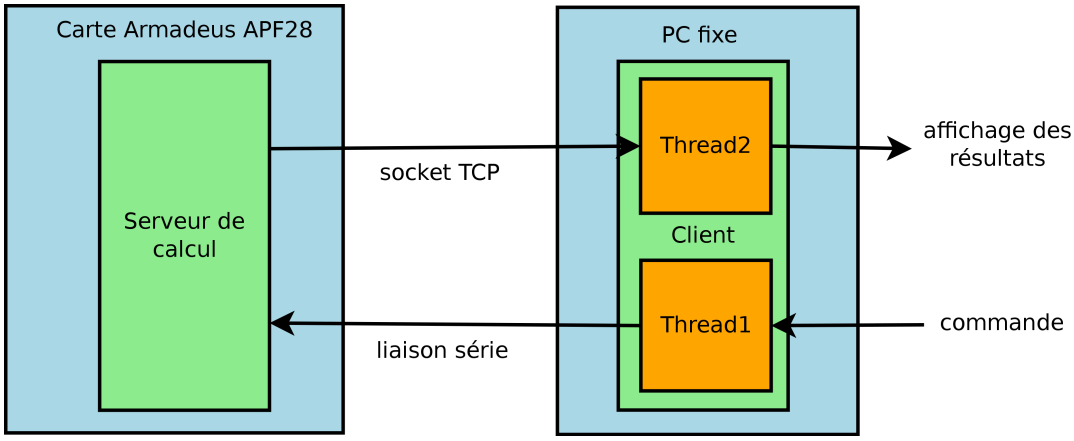
- la lecture de datasheet
- la connexion à une carte par liaison série
- l'upload et le flashage d'images (bootloader, kernel et rootfs)
- la génération d'une chaîne de cross-compilation et des images
- la cross-compilation d'un logiciel utilisateur
- l'utilisation des GPIO dans l'espace utilisateur pour gérer une LED lors de la phase de boot
- une révision de la liaison série, des sockets et des threads

Pour aborder chacun de ces points, nous allons réaliser une calculatrice distante.

Dans un premier temps, nous allons flasher la carte avec le bootloader, le kernel et le RFS. Ensuite nous développerons un programme calculatrice qui recevra des commandes (ajouter, soustraire, …) par liaison série et qui renverra les résultats sur un socket. Ce programme devra être cross-compilé.

Il faudra donc aussi un programme côté utilisateur permettant d'envoyer les commandes à la calculatrice et de lire les résultats.

L'architecture en image :



2. Exercice 1 (1h45) : Flashage de la carte APF28

2.1. Description de l'exercice

Pendant cet exercice, nous allons suivre le déroulement classique de travail sur une nouvelle carte :

- trouver le site web du constructeur et télécharger le BSP
- analyse rapide du contenu (système d'automatisation utilisé, bootloader, …)
- les différentes règles de compilation disponibles
- compilation de la distribution
- flashage de la carte avec les images compilées
- boot de la carte pour vérifier le bon fonctionnement

En réalité, pendant cet exercice, nous n'allons pas compiler le BSP, simplement pour une question de temps de compilation. Nous allons donc partir d'images déjà compilées. Cependant, nous compilerons plus tard dans le TP la chaîne de cross-compilation.

2.2. Questions

Dans un premier temps, rendez-vous sur le site ci-dessous et récuperez le BSP/SDK (Board Support Package) stable, actuellement la version 6.0 :

http://www.armadeus.com/wiki/index.php?title=Toolchain

Détarrez la tarball. Suite à l'extraction, vous pouvez observer divers répertoires/fichiers.

Question 1 (1 min) : *D'après une analyse rapide du contenu, quel type de système de build automatisé est utilisé?*

Question 2 (5 min) : *Selon vous, que contient le répertoire **patches**? Quels sont les différents sous répertoires et leurs utilités?*

Question 3 (1 min) : *D'après l'étude du répertoire **patches** de la question précédente, que pouvez vous dire sur le bootloader utilisé?*

Maintenant que l'on connait le système de build automatisé ainsi que le bootloader utilisé, nous allons regarder comment compiler notre distribution.

De manière générale, il y a globalement trois étapes lors de la compilation :

1. une règle de configuration générale, indiquant par exemple le type de carte sur laquelle nous allons travailler
2. une phase de configuration plus précise ouverte à l'utilisateur

3. une règle de compilation se basant sur les éléments configurés lors des étapes précédentes.

Question 4 (2 min) : *En étudiant le contenu du fichier **Makefile**, indiquez quelle règle de **configuration** doit être utilisée pour la carte APF28.*

Lancez la commande de configuration de la plateforme trouvée lors de l'étape précédente.

Question 5 (15 min) : *Qu'observez vous? Fouillez le contenu de l'interface, discutez et faite le rapprochement avec les notions vues en cours. Trouvez la version du kernel ainsi que le type/la version de libc.*

Lorsque vous êtes dans l'interface, allez sur **exit** et appuyez sur **entrée**. À partir de ce moment, un fichier de configuration **buildroot/.config** est créé et sera utilisé lors de la compilation de la distribution.

Question 6 (5 min) : *En étudiant ce fichier de configuration, déterminez l'ABI utilisée (vue en cours). Par rapport aux discussions eues en cours sur les différentes ABI d'une architecture ARM, que pensez vous de ce choix?*

Question 7 (2 min) : *Toujours par rapport aux notions vues en cours, rappellez l'utilité de **busybox**.*

Question 8 (2 min) : *En étudiant le fichier **Makefile**, déterminez quelle commande faut-il utiliser pour ouvrir le menu de configuration de busybox?*

Lancez cette commande de configuration. Dans une telle interface, l'utilisateur peut utiliser la commande / pour faire une recherche sur une chaine de caractères (comme dans vi).

Question 9 (5 min) : *Dans le menu de configuration de busybox, déterminez si busybox fournira **fdisk** et **ping**. Rappellez au passage l'utilité de ces deux comandes.*

Nous sommes maintenant à un stade où on connaît globalement le BSP de notre carte.

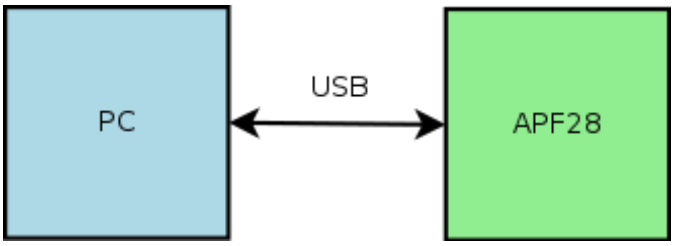
En deux commandes simples, nous pouvons donc configurer la carte et lancer la phase de compilation :

```
> make apf28_defconfig
> make
```

Cependant, la phase de compilation étant trop longue, nous n'allons pas réaliser cette étape, mais plutôt partir d'images précompilées disponibles ici : **embsys/labs/4_armadeus/images**. Dans le cas d'une compilation via le BSP, ces images seraient disponibles ici : **buildroot/output/images**.

Maintenant que nous avons les images, nous allons commencer à travailler avec la carte en tant que telle. Tout d'abord, nous allons essayer de communiquer par liaison série. La carte peut être alimentée par USB et c'est par ce même port USB que nous allons discuter avec l'APF28.

Réalisez donc le montage suivant :



Suite à cela, une entrée est créée dans le répertoire **/dev** grâce au mécanisme de hotplug du kernel et à udev.

Question 10 (1 min) : *Quel est l'entrée dans le /dev permettant de comuniquer avec l'APF28?*

Comme vue en cours, une configuration classique de communication série est du 8N1 (bits de données / bits de parité / bit d'arrêt). La carte APF28 n'échappe pas à la règle, mais il nous manque encore le débit de la ligne pour établir une communication.

Question 11 (3 min) : *Utilisez la commande **stty** pour trouver le débit de communication de la carte.*

Question 12 (10 min) : *Utilisez minicom pour vous connecter à la carte et appuyez sur le bouton **reset** de la carte. Que voyez vous? Que se passe t-il?.*

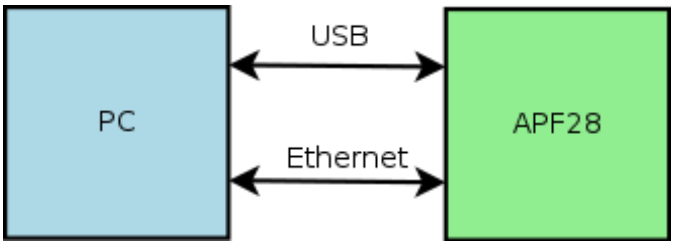
Refaîte une reset de la carte mais cette fois-ci intérompez la phase de boot afin d'arriver dans le prompt de U-boot. La commande **printenv** permet d'afficher les variables d'environnement/macros actuellement enregistrés sur la flash de la carte.

Question 13 (10 min) : *Grâce à la commande **printenv**, déterminez les macros permettant de télécharger et d'enregistrer de nouvelles versions du bootloader, du kernel et du RFS.*

Question 14 (10 min) : *En étudiant de plus prêt les commandes permettant de flasher le kernel et le RFS, déterminez les adresses flash où doivent être enregistrée ces images.*

Pour télécharger les images sur la carte, il faut que celles-ci soient sur un serveur TFTP (voir cours). Dans notre cas, le PC fixe sera ce serveur. Les images doivent être copiées directement dans le répertoire **/tftpboot** du serveur. Réalisez cette opération.

Ensuite, faite le montage suivant :



Question 15 (2 min) : *Déterminez l'adresse IP du serveur TFTP.*

Dans le prompt U-boot, la commande **setenv** permet de fixer une variable d'environnement et la commande **saveenv** de l'enregistrer sur la flash :

```
> setenv myvar coucou
> saveenv
```

Question 16 (5 min) : *En analysant dans U-boot les commandes de téléchargement, déterminez dans quelle variable d'environnement l'adresse IP du serveur est enregistrée.*

Grâce aux commandes **setenv** et **saveenv**, sauvegarder en flash dans la variable d'environnement correspondante (question précédente) l'adresse IP du serveur TFTP. Faite un reset de la carte et observez le fait que l'adresse IP est bien persistante.

Dans un premier temps, nous pouvons utiliser les commandes **download_** dans le prompt U-boot pour tester la connexion avec le serveur TFTP. Si le téléchargement se passe bien, nous pouvons lancer la phase d'update des images. Surtout, n'éteignez pas la carte pendant la phase d'enregistrement des images en flash!

Dans le prompt U-boot, lancez les commandes d'update (question 14) du bootloader, du kernel puis du RFS.

Ensuite, débranchez l'USB et l'Ethernet et rebootez la carte. Connectez vous avec minicom et laissez la phase de boot se déroulez jusqu'au boot. Vous arrivez alors à un prompt de login de votre distribution! La phase de flashage est alors terminée.

Jouez un peu avec votre distribution fraichement installée. Login : **root**.