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.

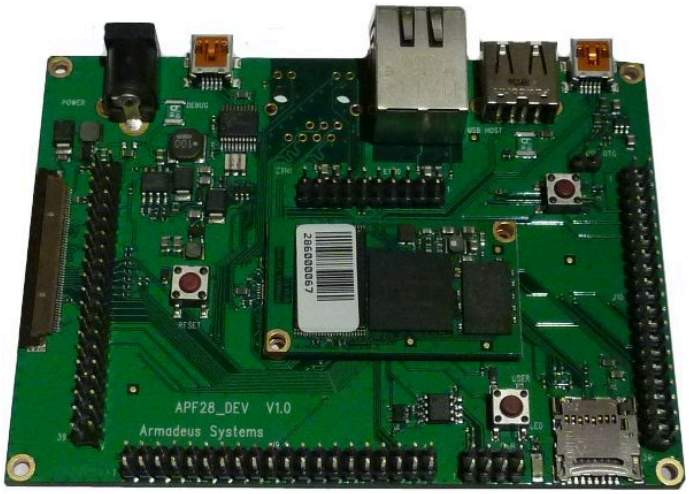
1.2. Attention particulière

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 :



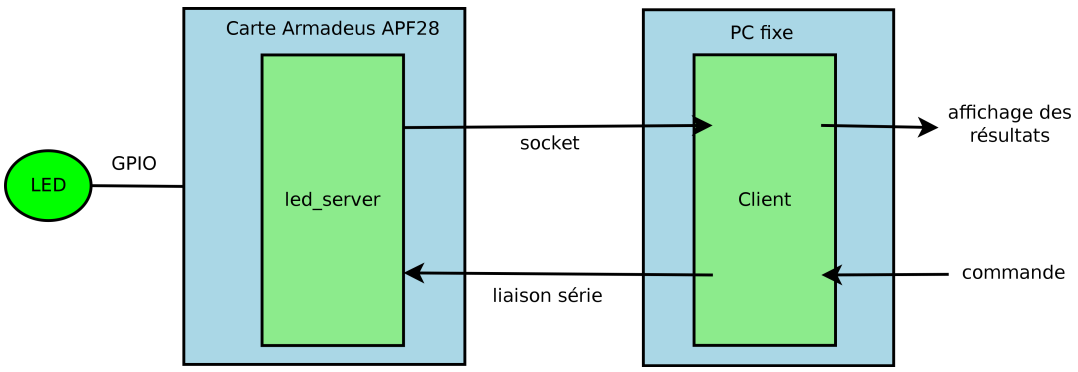
Le but est ici d'appréhender les points suivants :

- la lecture de datasheet
- la connexion à une carte par liaison série
- l'upload et le flashage d'images (kernel et rootfs)
- la cross-compilation d'un logiciel utilisateur
- l'utilisation des GPIO dans l'espace utilisateur
- une révision de syslog, de la liaison série et des sockets

Pour aborder chacun de ces points, nous allons réaliser un serveur embarqué sur la carte APF28 chargé d'allumer/éteindre periodiquement une LED. La fréquence d'allumage pourra être modifiée par liaison série et un message de status sera envoyé sur le réseau. Le tout devra être loggé sur le système grâce à Syslog.

Il faudra donc aussi un programme côté utilisateur permettant d'envoyer les commandes au serveur et de lire les messages envoyés par le serveur sur le réseau.

L'architecture en image :



2. Exercice 1 (1h30) : 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.

2.2. Questions

Découverte du BSP

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 (2 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 (2 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 (5 min) : *En étudiant le contenu du fichier **Makefile**, indiquez quelle règle de **configuration** doit être utilisée pour la carte APF28.*

Lancez cette commande de configuration.

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 (5 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 (10 min) : *Dans le menu de configuration de busybox, déterminez si busybox fournira **fdisk**, **ping** et **ssh**. Rappellez au passage l'utilité de ces 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 :

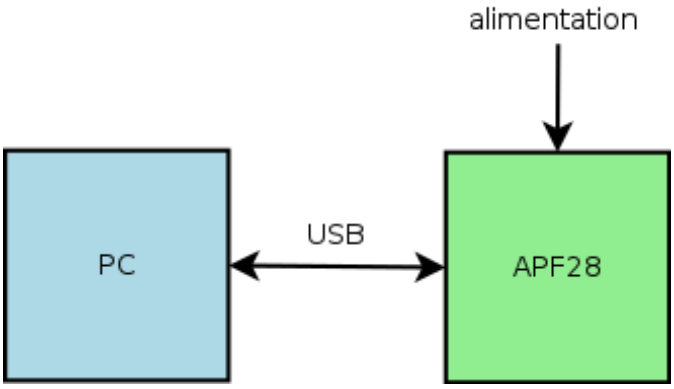
```
> make apf28_defconfig
> make
```

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

Flashage de la carte

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 (via cable USB).

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 (2 min) : *Rappellez l'utilité de la commande **dmesg**. 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. Ici, le débit est 115200 bauds.

Question 11 (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?*

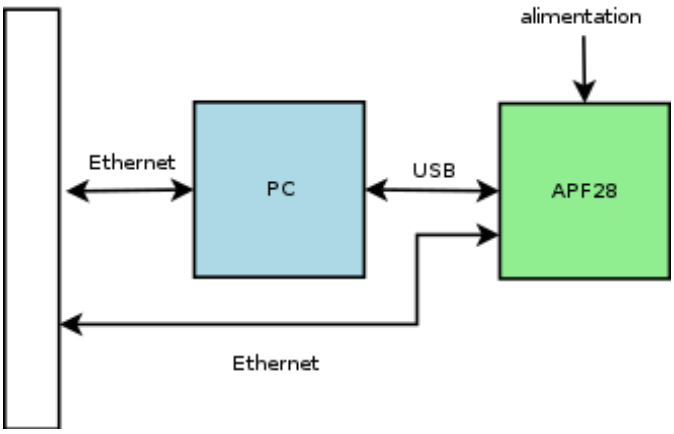
Refaite un reset de la carte mais cette fois-ci interrompez la phase de boot (en appuyant sur une touche pendant le compte à rebour) afin d'arriver dans le prompt de U-boot. La commande **printenv** permet d'afficher les variables d'environnement/macros actuellement enregistrés sur la flash de la carte.

Question 12 (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 13 (10 min) : *En étudiant de plus prêt les commandes permettant de flasher le kernel et le RFS, déterminez les adresses flash où doivent être enregistrées ces images.*

Pour télécharger les images sur la carte, il faut que celles-ci soient sur un serveur TFTP (voir cours). Dans notre cas, il existe un serveur TFTP distant ayant l'adresse IP : 172.20.5.2. Les images sont dans le répertoire /tftpboot.

Maintenant, connectez la carte APF28 au réseau de l'école par Ethernet.



Nous allons maintenant récupérer une adresse IP par DHCP afin que la carte APF28 soit sur le réseau.

Question 14 (2 min) : *Lancez la commande **dhcp** dans le prompt uboot. Quelle est l'IP de la carte à l'issue de cette commande?*

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 15 (5 min) : *En analysant dans U-boot les commandes de téléchargement des images, déterminez dans quelle variable d'environnement l'adresse IP du serveur TFTP est enregistrée.*

Grâce aux commandes **setenv** et **saveenv**, sauvegardez en flash dans la variable d'environnement correspondante (question précédente) l'adresse IP du serveur TFTP.

Nous pouvons utiliser la commande **ping** afin de voir si la liaison avec le serveur TFTP est opérationnelle :

```
> ping ${LA_VARIABLE_STOCKANT_L_IP_DU_SERVEUR_TFTP}
```

Dans le prompt U-boot, lancez les macros d'update du kernel et du RFS (on ne mettera pas à jour le bootloader pendant ce TP) :

```
> run update_kernel
> run update_rootfs
```

Ensuite, rebootez la carte et laissez la phase de boot se dérouler jusqu'au bout. 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 :

- changez le mot de passe root avec la commande **passwd**
- connectez vous en SSH sur la carte APF28
- copiez un fichier quelconque avec SCP
- …

3. Exercice 2 (1h30) : Cross-compilation, GPIO et syslog

3.1. Description de l'exercice

À partir du moment où on possède une carte fonctionnelle, la question de cross-compilation se pose. Dans le cas d'une compilation from scratch des images à flasher, le cross-compilateur est disponible à travers le BSP. Le but de l'exercice est donc :

- d'étudier une chaine de cross-compilation précompilée
- de compiler/cross-compiler un binaire hello-world

- d'apprendre à utiliser Syslog
- de jouer avec une LED pour découvrir l'utilisation des GPIO dans l'espace utilisateur

3.2. Questions

Cross-compilation

Dans un premier temps, écrivez un hello-world et utilisez le gcc natif du PC fixe. Utilisez SCP pour copier le binaire résultant sur la carte APF28.

Question 16 (10 min) : *Exécutez le binaire sur la carte. Qu'observez-vous? Lancez la commande **file** sur le binaire. À quoi sert cette commande selon vous?*

Un cross-compilateur est accessible sur la machine **saltp7-l** dans le répertoire /appli/arma/armadeus/buildroot/output/host/usr/bin. Connectez vous en SSH sur cette machine avec votre login personnel.

Question 17 (5 min) : *Dans le répertoire cité ci-dessus, déterminez le path du cross-compilateur pour du code C.*

Cross-compilez le hello-world pour l'APF28, copiez le sur la carte et exécutez le.

Syslog

Sous GNU-Linux, les logs enregistrés dans le répertoire /var/log sont gérés par le démon **syslog**. Vous pouvez trouver un exemple d'utilisation ici : **labs/4_armadeus/src/syslog**.

Question 18 (5 min) : *Compilez le code de syslog avec gcc et testez le.*

Question 19 (5 min) : *Utilisez **grep** dans /var/log pour déterminer dans quel fichier les messages sont écris.*

Question 20 (15 min) : *En étudiant le fichier syslog.c, décrivez l'utilisation des fonctions **setlogmask**, **openlog**, **syslog** et **closelog**.*

Cross-compilez le fichier syslog.c et testez sur la carte APF28.

GPIO

Nous allons utiliser un GPIO de l'APF28 pour allumer/éteindre la led présente sur la carte à côté du lecteur de mini SD.

Sous Linux, un GPIO est défini par un numéro. En revanche, côté matériel, un GPIO est défini par un numéro de banque et un numéro de pin. Dans le cas de l'APF28dev, le numéro sous Linux est calculé comme suit :

```
gpio_linux = (32 * bank_number) + pin_number
```

Question 21 (15 min) : *Trouvez sur internet la datasheet de la plateforme de développement de l'APF28 et fouillez dans la documentation pour trouver les numéros caractérisant le GPIO qui gère la LED utilisateur.*

Question 22 (2 min) : *En déduire le numéro du GPIO qui doit être utilisé sous Linux.*

Question 23 (15 min) : *Utilisez les notions vues en cours pour allumer et éteindre la LED dans le shell.*

Question 24 (20 min) : *Écrivez un code C permettant de gérer la LED. Ce programme doit agir comme un simple toggle.*