TP Linux Embarqué

L'ensemble des ressources nécessaires à ce TP est disponible sur
→ ce lien ←

1 Prise en main

Note : en cas de doute, appelez votre prof **avant** de faire une connerie !

1.1 Préparation de la carte SD

Avant de commencer, il faut flasher la carte SD avec l'image fournie par Terasic. Elle est disponible sur le lien ci-dessus. Il s'agit du fichier suivant :

```
VEEK_MT2S_LXDE/VEEK_MT2S_LXDE.img
```

Sous Windows Utilisez l'outile Win32DiskImager. Des tutoriels sont disponibles en ligne. Il faudra l'installer au besoin.

Sous Linux C'est un peu plus compliqué (mais y a rien à installer).

D'abord, il faut identifier le device correspondant à la carte SD. Ce device est dans le dossier `/dev/`

On peut donc regarder le contenu de ce dossier avant d'insérer la carte SD, puis après, et comparer :

```
ls /dev
```

Plusieurs fichiers devraient être apparus, quelque chose du type `/dev/sdX` `/dev/sdX1` `/dev/sdX2`... Notez bien la lettre (sda, sdb, sdc ou autre). Si vous vous trompez, vous risquez d'écraser le disque dur du PC. Et ce serait **très mal**. En cas de doute, faites appel au prof.

Pour flasher la carte SD, tappez simplement les lignes suivantes dans un terminal :

```
cd <chemin_vers_img>
sudo dd if=VEEK_MT2S_LXDE.img of=/dev/sdX bs=4k status=progress
sync
```

Croisez les doigts et priez tous les dieux disponibles.

1.2 Démarrage

Insérez la carte SD fraichement programmée, branchez la carte VEEK-MT2S et allumez-la : **bouton rouge !** Ça clignote de partout et un linux se lance.

Prenez un peu de temps pour explorer les différents programmes fournis sur le bureau de Lxde.

Mais sans clavier, ce n'est pas très pratique pour programmer cette « brique ». Nous allons y remédier.

1.3 Connexion au système

1.3.1 Liaison série

Le premier moyen pour se connecter sur un objet embarqué, c'est très souvent par le port série. Une fois que l'on aura eu accès au DE-10, on configurera le réseau, pour pouvoir ensuite y accéder via `ssh`.

Tout d'abord, déterminer le port à utiliser pour se connecter à la carte.

Il y a plusieurs ports USB sur la carte :

— 2 hôtes usb A
— 1 usb B : max blaster pour la programmation
— 1 usb mini : uart to usb ← **c'est celui-là qui nous intéresse**.

1.3.2 Utilisez un logiciel de liaison série

Sous Windows : Utilisez le logiciel PuTTy pour vous connecter grace au port série. Sélectionnez `serial`, et à l'aide du panneau de configuration de windows, cherchez le port COM de votre carte SoC. Pour la vitesse, entrez 115200.

Sous Linux : Utilisez minicom pour vous connecter grace au port série.

```
minicom -D /dev/ttyUSBx -b 115200
```

Avec x le numéro du port.

Vous devriez voir des choses s'afficher après un appui sur <Entrée>. Si rien ne s'affiche, essayez la combinaison suivante :

```
Ctrl+A puis O > Configurer le port série > F
```

Une fois connecté au SoC :

Pour vous identifier :

— login : root
— password : aucun (vraiment rien, ne tapez que sur entrée)

Pour l'exercice, nous allons redémarrer le SoC pour observer la séquence de démarrage.

```
reboot
```

Loggez-vous à nouveau. L'image copiée précédemment n'occupe pas la totalité de la carte SD. Quelle est la taille occupée ? Utilisez la commande suivante :

```
df -h
```

Des scripts sont fournis pour augmenter la taille du système de fichiers et profiter de l'ensemble de la carte SD :

```
./expand_rootfs.sh
```

Rebootez la carte (**proprement !**), puis, une fois loggé :

```
./resize2fs_once
```

Vérifiez que vous avez bien 32GB de disponible sur la carte SD.

1.3.3 Configuration réseau

- Branchez la carte VEEK sur le switch *via* un câble réseau,
- À l'aide de la commande `ifconfig`, vérifiez que la carte dispose d'une adresse IP,
- Éditer le fichier `/etc/network/interfaces` de la manière suivante :

```
# interfaces(5) file used by ifup(8) and ifdown(8)
# Include files from /etc/network/interfaces.d:
source-directory /etc/network/interfaces.d

auto eth0
iface eth0 inet dhcp
allow-hotplug eth0
```

Pour ces modification, vous allez devoir utiliser `vim` de la façon suivante :

- Éditez le fichier avec la commande `vim /etc/network/interfaces`,
- Effacez toutes les lignes inutiles en appuyant deux fois sur la touche *d*
- Insérez une nouvelle ligne en appuyant sur `o`. Cette touche vous bascule en mode `-- INSERT --`
- À partir de là vous pouvez taper le contenu du fichier,
- Quand c'est terminé, tapez sur la touche Echap pour revenir en mode `command`
- Tapez `:wq` puis sur entrée pour sauvegarder et quitter `vim`.

Rebootez une nouvelle fois, et là normalement vous allez avoir une adresse IP correcte. Vérifiez avec `ifconfig`.

- Si tout est correct, vous devez pouvoir pinger avec le PC dans les 2 sens

```
ping <adresse_ip>
```

— Vérifiez que dans le fichier `/etc/ssh/sshd_config`, la ligne suivante est présente :

```
PermitEmptyPasswords yes
```

— Vous devriez pouvoir vous logger en ssh sur la carte VEEK, avec Putty ou par le terminal :

```
ssh root@<IP_VEEK>
```

Une fois connecté en ssh, vous pouvez fermer la liaison série. Sur minicom ça se fait avec la combinaison suivante :

```
Ctrl+A puis Q
```

1.4 Découverte de la cible

1.4.1 Exploration des dossiers `/sys/class` et `/proc`

Explorez un peu votre environnement, par exemple :
— Répertoires présent sous la racine
— Dans `/proc` : `cpuinfo`, `ioports`, `iomem`. Utilisez les commandes `cat`, `less` ou `more` pour voir le contenu des fichiers.
— Le répertoire `/sys/class` contient des entrées nouvelles (par raport à un PC classique), saurez vous les reconnaître ? En particulier, explorez les répertoires suivants :
  — `/sys/class/leds/fpga_ledX/`
  — `/proc/ioport`
  — `/proc/iomem`
  — `/proc/device-tree/sopc@0` à comparer avec le fichier `iomem`.

1.4.2 Compilation croisée

Il existe deux méthodes pour compiler un programme sur le SoC :
— Directement sur les SoC à l'aide du gcc qui y est installé (`gcc -v`)
— Sur le PC (beaucoup plus puissant), en utilisant un chaine de compilation croisée sous linux (`apt install gcc-arm-linux-gnueabihf` sous Ubuntu).

Vous allez utiliser la deuxième solution. Pour cela, une VM contenant un linux déjà configuré va vous permettre de faire la compilation directement sur le PC :
— Lancez VirtualBox et importez la VM suivante `VM-SOC-2019.ova`
— Sur les PC Windows de l'école, modifiez le répertoire "Dossier de base" (Machine Base Folder) dans `D:\SOC-2021\VirtualBox VMs` pour éviter d'exploser vos quotas ENSEA.
— Lancez l'importation.
— Avant de lancer la VM, modifiez le dossier partagé en le faisant pointer sur un dossier dans "Mes Documents" (ou ailleurs).

— Lancez la VM, loggez vous (login : ensea, password : ensea).

Dans la VM, le répertoire `src` dans le home de ensea, est le dossier partagé. Tout ce que vous y placer est visible depuis la VM et depuis le système Hôte.

1.4.3 Hello world !

Réalisez un programme "Hello World !", compilez-le et testez-le sur la carte SoC.

— Pour compiler sur la VM, utilisez le cross-compilateur :

```
arm-linux-gnueabihf-gcc hello.c -o hello.o
```

Vous pouvez vérifier le type de vos exécutables avec la commande `file`. Essayez de l'exécuter dans la VM. Que se passe-t-il ?

Comme la carte SOC est sur le réseau, vous pouvez copier l'exécutable directement sur la cible :

```
scp chemin_sur_VM root@IP_DE_LA_CARTE_SOC:chemin_sur_SOC
```

Tester sur la carte.

1.4.4 Accès au matériel

Un certain nombre de drivers sont fournis. Comme tous les drivers sous Linux, ils sont accessible sous forme de fichiers. Par exemple pour allumer l'une des LED rouge de la carte, il suffit d'écrire un '1' dans le bon fichier.

```
echo "1" > /sys/class/leds/fpga_led1/brightness
```

Tester d'allumer et d'éteindre d'autres LED.

1.4.5 Chenillard (Et oui, encore !)

Plutôt que de passer par la commande `echo`, on peut écrire un programme C qui va ouvrir et écrire dans ces fichiers. Écrire un programme en C qui réalise un chenillard.

2 Modules kernel (TP2)

2.0 Reprise du TP1

Assurez vous de pouvoir communiquer avec la carte VEEK en ssh ou via le port série. Vous devez pour cela reprendre la configuration du réseau faite au TP1.

2.1 Accès aux registres

Avant de travailler avec les modules, vous allez créer un programme qui accède directement aux registres depuis l'espace utilisateur.

À cause de la virtualisation de la mémoire, il n'est pas possible d'écrire facilement dans un registre comme nous en avons l'habitude. Il faut en effet remapper la mémoire pour demander à l'OS de nous fournir une adresse virtuelle.

Pour cela, on utilisera la fonction `mmap()`

Le registre du GPIO connecté aux LED est disponible à l'adresse suivante :

— `0xFF203000`

Cette méthode permet de prototyper rapidement, mais pose quelques problèmes et limites. Quels sont-ils ?

2.2 Compilation de module noyau sur la VM

Pour ce TP, vous allez développer vos propres modules noyau. Vous allez avoir besoin des sources du noyau cible (en fait en théorie il faut seulement les includes). Il nous faut les sources **exactes** du noyau sur lequel le module va être chargé.

Pour compiler des modules noyau dans la VM, vous aurez besoin des paquets suivant :

```
sudo apt install linux-headers-amd64
sudo apt install bc
```

À partir du `Makefile` et du fichier source `hello.c` disponibles sur moodle, compilez votre premier module.

Utilisez `modinfo`, `lsmod`, `insmod` et `rmmod` pour tester votre module (à utiliser avec `sudo`) : chargez le et vérifiez que le module fonctionne bien (`sudo dmesg`).

Pour la suite, tester les programmes suivants (voir cours)

— utilisation de paramètres au chargement du module

— création d'un entrée dans /proc

— utilisation d'un timer

2.3 CrossCompilation de modules noyau

À cause de la puissance limitée du processeur de la carte cible, la compilation, en particulier la compilation de modules noyau, est relativement longue. Nous allons donc, une fois encore, *cross-compiler* les modules noyau pour la carte SoC, à l'aide de la VM.

2.3.0 Récupération du Noyau Terasic (c'est déjà fait dans la VM !)

Les commandes suivantes permettent de récupérer les sources du noyau actuellement en fonctionnement sur la carte VEEK :

```
git clone https://github.com/terasic/linux-socfpga/
git checkout 6b20a2929d54
git config core.abbrev 7
```

Pourquoi ces 2 commandes en plus ? Faites `uname -a` sur la carte VEEK, et vous obtenez :

```
Linux DE10-Standard 4.5.0-00198-g6b20a29 #32 SMP Thu May 4...
```

Le `-g6b20a29` signifie que ce noyau a été compilé avec le commit git 6b20a29, c'est à dire hash limité à 7 caractères. Or aujourd'hui, on utilise des hash limités à 12 caractères... Voir le mail de Linus Torval à ce sujet.

Remarque : Ce noyau est une version un peu ancienne de celle d'Altera (Altera/Intel est un bon contributeur au noyau Linux).

2.3.1 Préparation de la compilation

```
sudo apt install bc
sudo apt install crossbuild-essential-armhf
sudo apt install binutils-multiarch
```

Notez le chemin vers ces compilateurs : `whereis arm-linux-gnueabihf-gcc`

2.3.2 Récupéreation de la configuration actuelle du noyau

Sur la carte SoC, récupérez le contenu du fichier `/proc/config.gz` dans le dossier des sources du noyau.

Décompressez ce fichier dans le dossier `~/linux-socfpga/` et renommez le en `.config`.

```
gunzip config.gz
mv config .config
```

Lancez les lignes suivantes depuis le dossier `~/linux-socfpga/` :

```
export CROSS_COMPILE=<chemin_arm-linux-gnueabihf->
export ARCH=arm
make prepare
make scripts
```

Le `<chemin_arm-linux-gnueabihf>` est le chemin noté plus haut sans le gcc final. Par exemple : `/usr/bin/arm-linux-gnueabihf-`

— Quel est le rôle des lignes commençant par `export` ?

— Pourquoi le chemin fini par un tiret "-" ?

2.3.3 Hello World

Nous allons refaire les exercices précédents (Module, timers...) sur la plateforme VEEK.

Copiez le code source de la partie précédente. Modifiez le `Makefile` pour l'adapter à votre situation :

— Mettre à jour le chemin vers le noyau
— Ajouter `CFLAGS_MODULE=-fno-pic`

Compilez le module avec un simple `make`.

Si vous avez changé de terminal, il faudra à nouveau tapper les lignes suivantes :

```
export CROSS_COMPILE=<chemin_arm-linux-gnueabihf->
export ARCH=arm
```

La compilation risque d'échouer car `make` n'arrive pas à compiler dans le répertoire partagé avec Windows ("opération non permise"). Il faudra donc copier tout le répertoire en dehors du dossier `~/src`. Le plus simple reste de copier le dossier ailleurs :

```
cp -r ~/src/TP2 ~/
```

Une fois complié, copiez le module sur la carte, et charger le. Vérifiez que le module fonctionne bien avec la commande `dmesg`.

Essayez de compiler vos autres module pour la carte SoC.

2.3.4 Chenillard (Yes !)

On veut créer un chenillard dont on peut modifier :

— Le pattern depuis le fichier : `/proc/ensea/chenille`
— La vitesse au moment du chargement du module.

Dans un premier temps, vous vous interesserez surtout à la structure du code. Ne cherchez pas pour cette séance à piloter les LED, utilisez la `dmesg` pour vérifier le bon fonctionnement de votre module.

Créez un module respectant ces conditions.

3 Device tree (TP3)

. L'objectif de ce TP est de définir son propre périphérique, et de programmer un module qui identifie la présence du périphérique et se configure automatiquement en sa présence. Cet automatisme s'appuye sur le `Device Tree`.

Le fichier DTS/DTB (`.dts`) est un fichier qui est installé à coté du noyau sur la partition de boot, et qui lui indique quels sont les périphériques à sa disposition.

Nous voulons accéder aux LED en direct. Or le device-tree étant déjà configuré, ces leds sont déjà utilisées par les drivers fournis par Altera, ce qui nous empêche d'y accéder. Pour éviter cela, 2 possibilités :

— Modifier le fichier dts existant ;
— Recréer son propre design avec QSYS, puis ce qui va générer un nouveau fichier dts.

Pour ce TP, nous allons utiliser la première solution :

— Recherchez le fichier texte

```
VEEK-MT2S_v.1.0.3_SystemCD/Demonstration/SoC_FPGA/
    ControlPanel/Quartus/soc_system.dts
```

— Copiez ce fichier dans votre répertoire de travail et ouvrez le.
— Recherchez l'entrée 'ledr' et remplacez :

```
ledr: gpio@0x100003000 {
                compatible = "altr,pio-16.1", "altr,pio-1.0";
                reg = <0x00000001 0x00003000 0x00000010>;
                clocks = <&clk_50>;
```

par :

```
ledr: ensea {
                compatible = "dev,ensea";
                reg = <0x00000001 0x00003000 0x00000010>;
                clocks = <&clk_50>;
```

— Installer le compilateur de device-tree sur la VM :

```
sudo apt install device-tree-compiler
```

— compiler le fichier `.dts` en un fichier `.dtb` (lisible par le noyau) :

```
dtc -O dtb -o soc_system.dtb soc_system.dts
```

— sur la VEEK, pour accéder à la partion de boot (là où il y a le `zImage`), vous allez "monter" la partition :

```
mkdir /root/mntboot
mount /dev/mmcblk0p1 mntboot
```

— dans ce répertoire /root/mntboot, renommez le fichier dtb en .old, et copiez le nouveau fichier dtb provenant du PC
— rebootez la carte VEEK : `reboot`
— explorez le nouveau device-tree (/proc/device-tree/sopc@0).

3.1 module accedant au LED via `/dev`

— Récupérez le fichier `gpio-leds.c` sur moodle et compiler le.
— Étudiez ce fichier. Quel sont les rôles des principales fonctions (`probe`, `read`, `write`, `remove`), et quand entrent-elles en action ?

3.2 Module final

3.2.1 Cahier des charges

Réaliser un chenillard qui rempli les conditions suivantes :
— Choix de la vitesse de balayage par une option au moment du chargement du module
— Récupération de la vitesse courante par lecture du fichier `/proc/ensea/speed`
— Modification de la patern par écriture dans le fichier `/dev/ensea-led`
— Récupération du patern courant par lecture du fichier `/dev/ensea-led`
— Modification du sens de balayage par écriture du fichier `/proc/ensea/dir`
— Récupération du sens de balayage par lecture du fichier `/proc/ensea/dir`

La solution la plus rapide pour implémenter ce chenillard sera d'utiliser un timer (voir le cours) ou un thread noyau (voir sur internet).

4 Petit projet : Afficheurs 7 segments

L'objectif de ce petit projet est d'écrire une application permettant d'afficher l'heure courante sur les afficheurs 7 segments.

4.1 Prise en main

En vous inspirant du début du TP2 (Section 2.1), faites fonctionner les afficheurs 7 segments. Vous utiliserez la fonction `mmap()`.

La lecture du fichier suivant indique l'adresse de base des afficheurs 7 segments et donnera quelques informations quant à son utilisation.

```
VEEK-MT2S_v.1.0.3_SystemCD/Demonstration/SoC_FPGA/
        ControlPanel/ControlPanel_QT/fpga.cpp
```

Les afficheurs 7 segments sont appelés "HEX" dans le code (ce qui n'est pas très malin...).

4.2 Device Tree et module

Écrire un module permettant d'afficher l'heure sur l'afficheur.

Dans le device tree fourni, l'afficheur n'est pas présent. Il faudra donc rajouter un nœud pour y faire référence. Inspirez vous du TP3.

Faites attention aux adresses. En particulier, faites le lien entre le fichier consulté précédemment (`fpga.cpp`) et le nœud `ledr`.

Il y a plusieurs possibilités pour réaliser ce projet :

— Concevoir une interface entre le noyau et l'espace utilisateur (procfs, device driver...) et coder l'application du point de vue utilisateur,

— Récupérer l'heure courante dans le noyau et tout réaliser dans le module.