Systèmes d'exploitation et environnements d'exécution embarqués

Rapport de laboratoire

Master HES-SO

Émilie Gsponer, Grégory Emery

1er mai 2016
version 1.0

Table des matières

1 Introduction REPTAR

1.1 Mise en place de l'environnement, utilisation de git

a) Donnée : Il faut tout d'abord récupérer le dépôt étudiant pour les laboratoires SEEE à l'aide de la commande suivante (via une fenêtre de terminal) :

```
$ git clone firstname.lastname@eigit.heig-vd.ch:/home2/reds/seee/seee_student
```

Travail réalisé : Nous n'avions pas les droits d'accès pour le dépôt git, nous l'avons donc téléchargé, puis extrait depuis le lien : https://drive.switch.ch/index.php/s/TbHxQZtmO9IVdkb. Le dossier seee_student a ensuite été placé dans : /home/redsuser/

b) Donnée : Lancez Eclipse et ouvrez le workspace seee_student. Vous devriez obtenir la liste des projets (à gauche). Chaque projet a un lien symbolique dans la racine du workspace.

Travail réalisé : En introduisant le path du dossier seee_student comme workspace au lancement d'Eclipse, nous obtenons la liste de projets suivante :

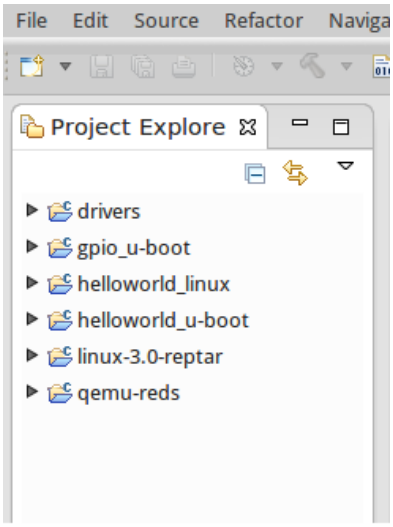

FIGURE 1 – Liste des projets

c) Donnée : Compilez maintenant l'émulateur Qemu. Dans une fenêtre de terminal, lancez la commande suivante à partir de votre répertoire seee_student :

```
$ make qemu
```

Travail réalisé : Vu que nous n'avons pas téléchargé le dossier de projets depuis git, il faut nettoyer le contenu du dossier avec clean ou distclean avant de compiler qemu. Le make qemu prend quelques instants.

```
$ cd ~/seee_student
$ make clean
$ make qemu
...
make[1]: Leaving directory '/home/redsuser/seee_student/qemu-reds'
$
```

1.2 Démarrage de Qemu

a) Donnée : Depuis Eclipse, lancez le debugger avec la configuration de debug « qemu-reds Debug ». Dans la fenêtre Console, vous pourrez entrer directement des commandes de U-boot (tapez help par exemple).

Travail réalisé :

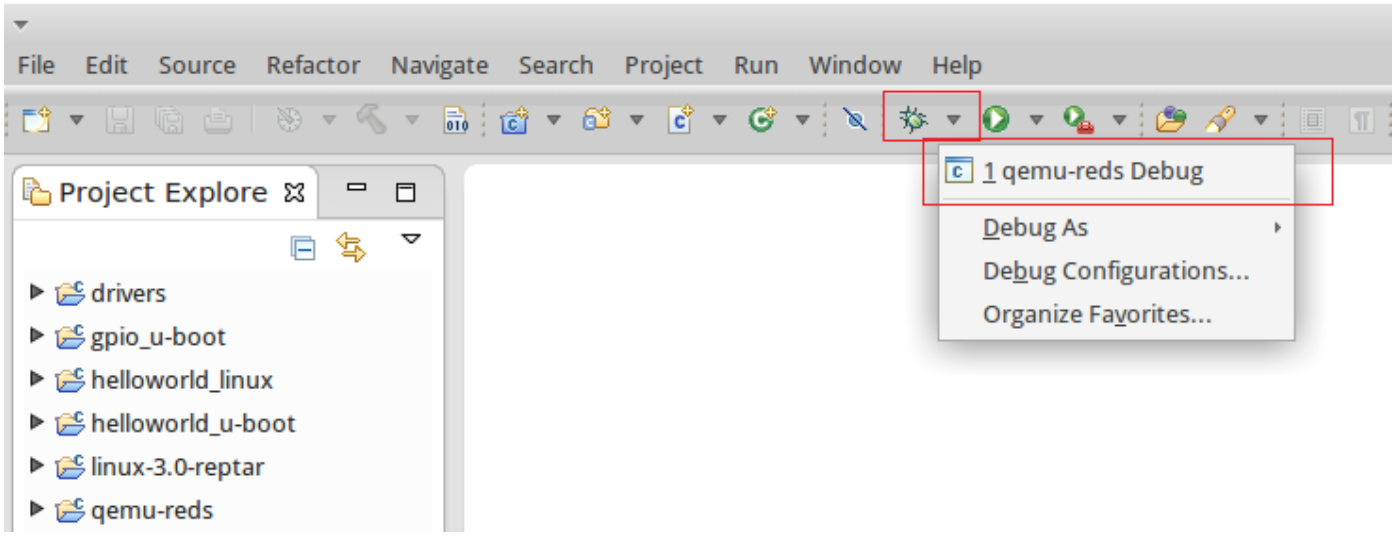

FIGURE 2 – Lancement d'Eclipse en mode Debug

Remarque : Après le lancement du Debug, il faut changer d'onglet en haut à droite en choisissant *Debug* pour avoir la console. Ce changement d'onglet ne se fait pas automatiquement.

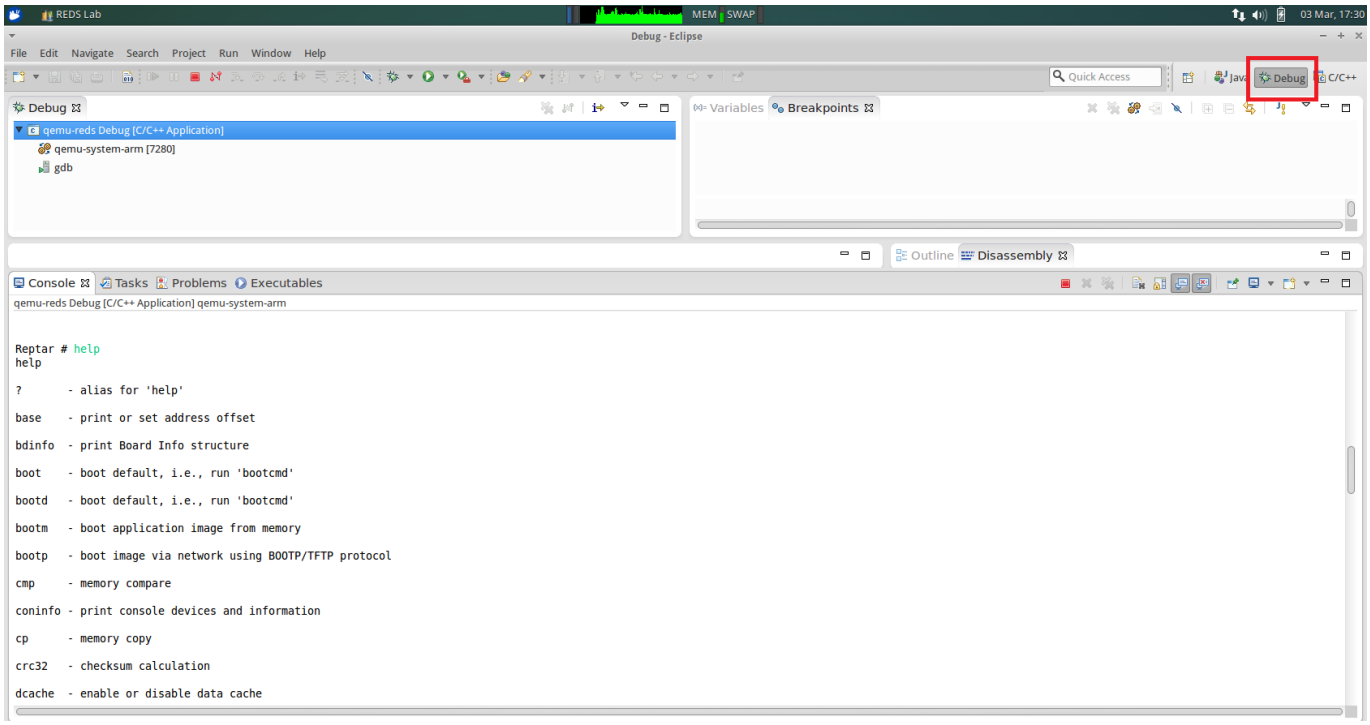

FIGURE 3 – Command help dans l'U-boot

b) Donnée : Interrompez l'exécution du programme en cliquant sur l'icône pause. Identifiez la ligne en cours d'exécution dans le code source.

Travail réalisé : En interrompant le programme avec le bouton *suspend*, on obtient la vue assembleur ci-dessous. L'environnement essaie d'ouvrir le fichier ppoll.c, on est donc en attente d'un événement. Le programme est interrompu après un syscall.

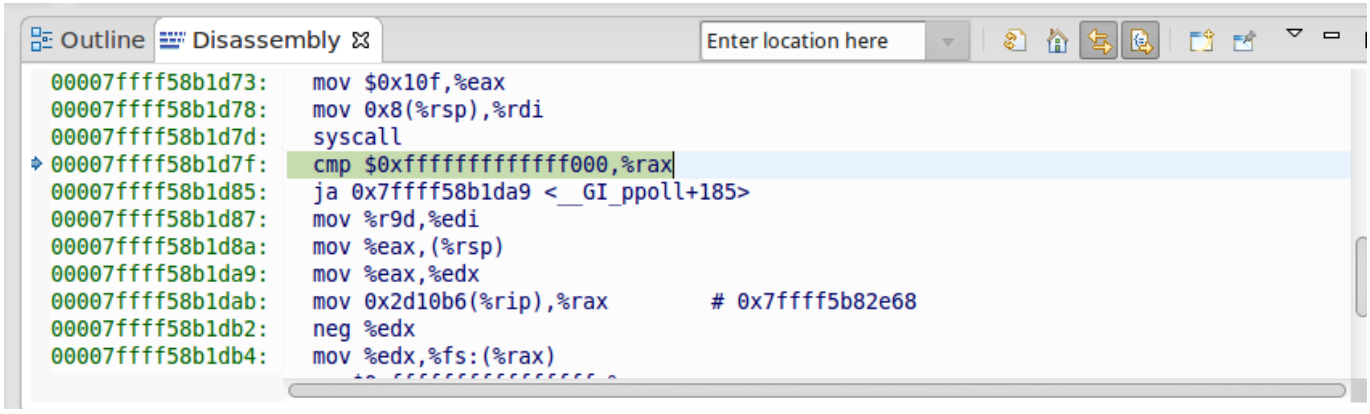

FIGURE 4 – Command help dans l'U-boot

c) Donnée : Stoppez l'exécution, et dans une fenêtre de commande, démarrez qemu à l'aide du script stf (en tapant ./stf) dans le répertoire racine. Vous arrivez dans U-boot.

Travail réalisé : Cette partie n'a plus rien avoir avec Eclipse, on peut le fermer et lancer un terminal. Avec la commande *stf* tapée à la racine du répertoire seee_student, on arrive au même point qu'en lançant le Debug dans Eclipse. On peut également essayer la commande *help*

```
$ cd ~/seee_student
$ ./stf
WARNING: Image format was not specified for 'filesystem/flash' and probing
    guessed raw.
...

Reptar # help
?       - alias for 'help'
base    - print or set address offset
bdinfo  - print Board Info structure
boot    - boot default, i.e., run 'bootcmd'
bootd   - boot default, i.e., run 'bootcmd'
bootm   - boot application image from memory
bootp   - boot image via network usi
...
```

1.3 Tests avec U-boot

a) Donnée : Dans U-boot, listez les variables d'environnement avec la commande printenv. Observez les variables prédéfinies « tftp1, tftp2 et goapp ». Ces variables définissent des commandes U-boot qui peuvent être exécutées à l'aide de la commande run (par exemple run tftp1). La commande go <addr> permet de lancer l'exécution à l'adresse physique <addr>. Vous pouvez définir/modifier vos propres variables et les sauvegarder dans la flash émulée avec la commande saveenv (seulement avec le lancement via stf).

Travail Réalisé : Après être entré dans l'U-boot avec *stf*, nous avons pu lister les variables d'environnement suivantes :

```
$ cd ~/seee_student/
$ ./stf
WARNING: Image format was not specified for 'filesystem/flash' and probing
    guessed raw.
Reptar # printenv
...
goapp=go 0x81600000
...
tftp1=tftp helloworld_u-boot/helloworld.bin
tftp2=tftp gpio_u-boot/gpio_u-boot.bin

Environment size: 930/4092 bytes
Reptar #
```

Les variables *tftp1* et *tftp2* sont des alias permettant de lancer des applications, la variable goapp est un alias permettant de lancer l'exécution de l'adresse physique 0x81600000. Elle définit l'adresse de début des applications. Voici un exemple d'utilisation de ces variables :

```
$ cd ~/seee_student/helloworld_u-boot/
$ make
...
$ cd ../gpio_u-boot/
$ make
...
$ cd ..
$ ./stf
WARNING: Image format was not specified for 'filesystem/flash' and probing
    guessed raw.
...
Reptar # run tftp1
smc911x: detected LAN9118 controller
smc911x: phy initialized
smc911x: MAC e4:af:a1:40:01:fe
Using smc911x-0 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.10
Filename 'helloworld_u-boot/helloworld.bin'.
Load address: 0x81600000
Loading: #
done
Bytes transferred = 776 (308 hex)

Reptar # run goapp
## Starting application at 0x81600000 ...
Example expects ABI version 6
Actual U-Boot ABI version 6
Hello World
argc = 1
argv[0] = "0x81600000"
argv[1] = "<NULL>"
Hit any key to exit ...
## Application terminated, rc = 0x0

Reptar # run tftp2
smc911x: detected LAN9118 controller
smc911x: phy initialized
smc911x: MAC e4:af:a1:40:01:fe
Using smc911x-0 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.10
Filename 'gpio_u-boot/gpio_u-boot.bin'.
Load address: 0x81600000
Loading: #
done
Bytes transferred = 3080 (c08 hex)

Reptar # run goapp
## Starting application at 0x81600000 ...
Start of the GPIO U-boot Standalone Application
Stop of the GPIO U-boot Standalone Application
## Application terminated, rc = 0x0
Reptar #
```

La commande run tftp<x> charge une application à l'adresse 0x81600000, tandis que run

goapp va exécuter l'application depuis cette adresse comme le montre l'exemple ci-dessus.

b) Donnée : La production de l'exécutable helloworld_u-boot s'effectue en tapant la commande make dans le répertoire contenant les sources du programme. Ensuite, vous pouvez transférer le fichier (extension .bin) dans U-boot et exécuter le binaire (aidez-vous des variables d'environnement prédéfinies).

Travail réalisé : Ce point a été fait en même temps que le précédent. Il ne faut rien faire de particulier pour que le .bin soit accessible dans l'U-boot. Il suffit d'utiliser les variables d'environnement prédéfinies.

c) Donnée : Testez le debugger dans Eclipse avec le projet helloworld_u-boot. Mettez un breakpoint dans le code source au démarrage du programme, et lancez le debugger avec la configuration de debug « helloworld_u-boot Debug ».

Travail Réalisé : Il faut que qemu soit lancé dans un terminal externe avec *stf* pour que la manipulation fonctionne avec Eclipse.

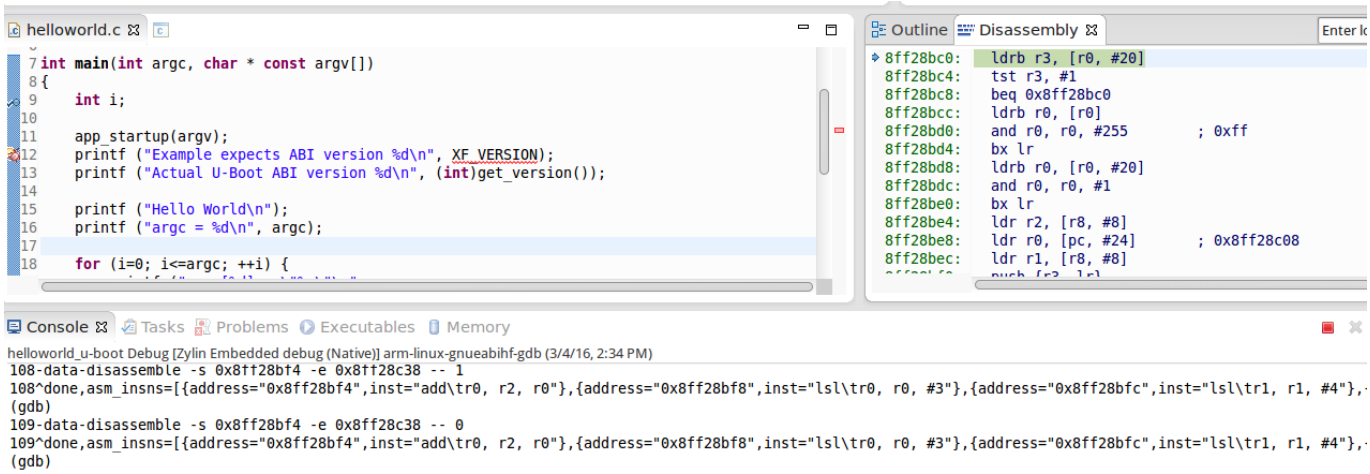

FIGURE 5 – Debug d'hello_world_u-boot

Remarque : Qemu contient un serveur GDB, ce qui permet à Eclipse (avec un plug-in) de communiquer avec lui et de debugger/lancer des applications à distance. On peut ainsi debugger des applications de l'environnement émulé depuis la machine hôte.

1.4 Tests avec Linux

a) Donnée : Lancez le script ./deploy qui permettra de déployer le noyau Linux dans la sdcard virtuelle (ignorez l'erreur due à l'absence de certains fichiers).

Travail réalisé :

```
$ cd ~/seee_student
$ ./deploy
```

```
Deploying into reptar rootfs ...
Mounting filesystem/sd-card.img...
[sudo] password for redsuser:
SD card partitions mounted in 'boot_tmp' and 'filesystem_tmp' directories
cp: cannot stat 'drivers/sp6.ko': No such file or directory
cp: cannot stat 'drivers/usertest': No such file or directory
cp: cannot stat 'drivers/buttons_test': No such file or directory
Unmounting SD card image...
Synchronizing .img file
Unmounting 'boot_tmp' and 'filesystem_tmp'...
Done !
$
```

b) Donnée : Poursuivez ensuite en cross-compilant l'application helloworld pour Linux (via make).

Travail réalisé :

```
$ cd ~/seee_student/helloworld_linux/
$ make
...
$
```

c) Donnée : Copiez l'exécutable dans le rootfs

Travail réalisé :

```
$ cd ~/seee_student
$ ./mount-sd.sh
Mounting filesystem/sd-card.img...
SD card partitions mounted in 'boot_tmp' and 'filesystem_tmp' directories

$ sudo cp helloworld_linux/helloworld filesystem_tmp/root

$ ./umount-sd.sh
Unmounting SD card image...
Synchronizing .img file
Unmounting 'boot_tmp' and 'filesystem_tmp'...
Done !
$
```

Remarque : Pour accéder au rootfs de la sdcard virtuelle, on va d'abord la monter son image sur la machine hôte. On peut ensuite copier l'exécutable de notre application à la racine du système de fichiers de la carte sd. Pour terminer, il faut démonter la carte Sd.

d) Donnée : Lancez le script stq suivi de la commande boot dans U-boot pour amorcer le démarrage de Linux

Travail réalisé : Avec la commande stq, une représentation graphique de la carte se lance en plus de la console U-boot.

```
$ cd ~/seee_student
$ ./stq
libGL error: failed to authenticate magic 1
libGL error: failed to load driver: vboxvideo
Running QEMU
...
Warning: smc911x-0 MAC addresses don't match:
Address in SROM is         52:54:00:12:34:56
Address in environment is  e4:af:a1:40:01:fe
Reptar # boot
reading uImage
...
*** Welcome on REPTAR (HEIG-VD/REDS): use root/root to log in ***
reptar login: root
Password:
#
```

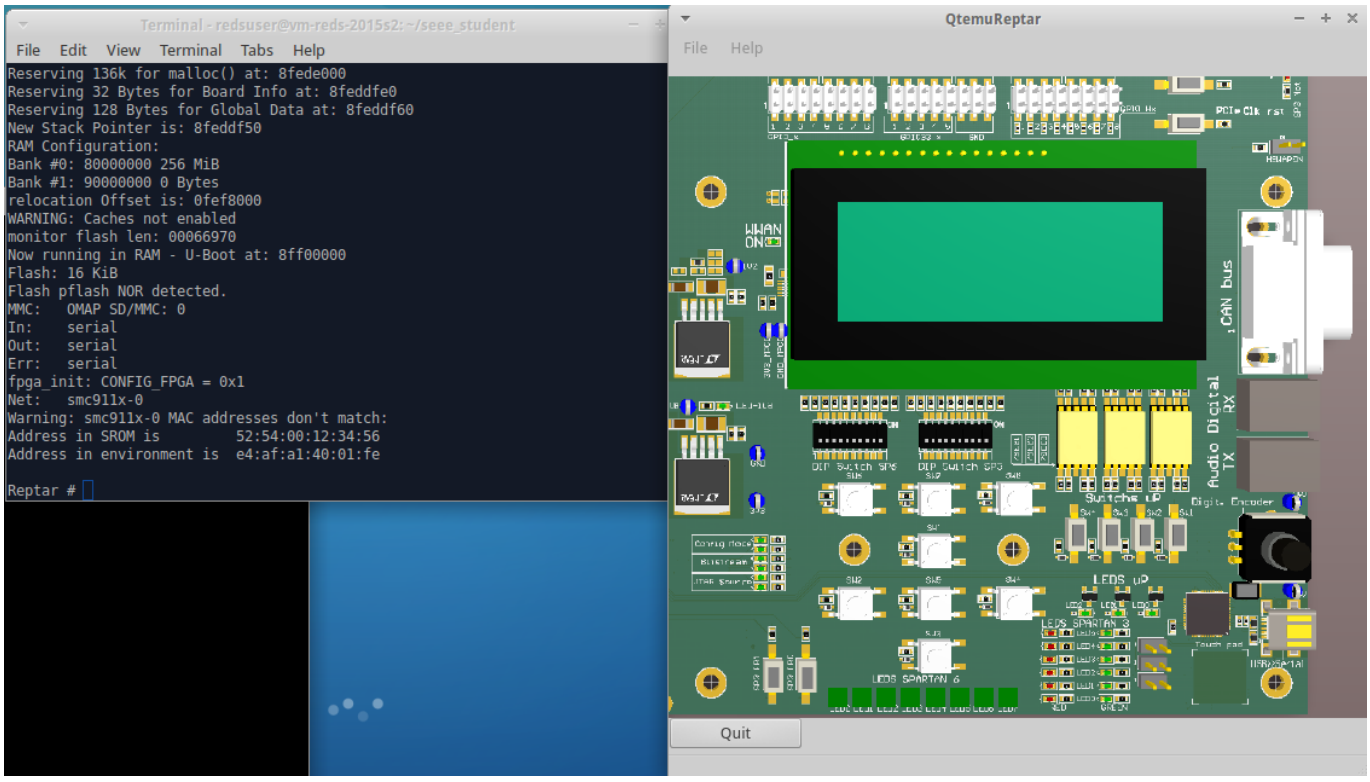

FIGURE 6 – Environnement émulé

e) Donnée : Lancez votre application

Travail réalisé : Avec la commande ls, on voit que l'application que nous avons chargée dans le rootfs est bien présente.

```
# ls
Settings        fs              helloworld      rootfs_domU.img
```

```
# ./helloworld
Hello world within Linux
argv[0] = ./helloworld
#
```

f) Donnée : Dans Linux, tapez la commande suivante :

```
$ /usr/share/qt/examples/effects/lighting/lighting -qws &
```

Travail réalisé : Cette commande permet de lancer une application pré installée de l'émulateur.

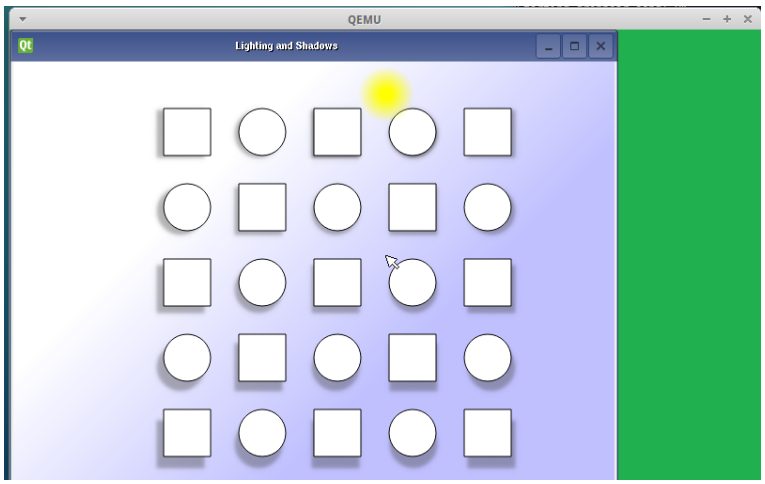

FIGURE 7 – Lancement d'une application

1.5 Tests sur la plate-forme réelle

a) Donnée : Déployez l'application helloworld dans U-boot sur la plate-forme REPTAR avec l'interface réseau. Le transfert peut s'effectuer avec la commande tftp. Il est nécessaire d'exécuter la commande suivante pour mettre à jour les adresses IP et MAC de la plate-forme REPTAR :

```
# run setmac setip
```

Travail réalisé : Avec la commande tftp il faut donner comme paramètre le *.bin* de l'application ainsi que l'adresse physique où charger le programme. Cette adresse est 0x81600000 comme dans les exercices précédents.
Pour charger notre application, il faut placer le .bin dans le répertoire tftpboot de la machine hôte. On va utiliser la connexion série (USB) pour exécuter la commande tftp et transférer l'exécutable sur la plateforme.

```
$ cd ~/seee_student/helloworld_u-boot
$ make
```

```
$ cp helloworld.bin /home/redsuser/tftpboot
$ sudo picocom -b 115200 /dev/ttyUSB0
[sudo] password for redsuser:
picocom v1.7
...
Terminal ready

Reptar # run setmac setip
Reptar # tftp 0x81600000 helloworld.bin
smc911x: detected LAN9220 controller
smc911x: phy initialized
smc911x: MAC e4:af:a1:40:01:fe
Using smc911x-0 device
TFTP from server 192.168.1.1; our IP address is 192.168.1.254
Filename 'helloworld.bin'.
Load address: 0x81600000
Loading: T #
done
Bytes transferred = 776 (308 hex)

Reptar # go 0x81600000
## Starting application at 0x81600000 ...
Example expects ABI version 6
Actual U-Boot ABI version 6
Hello World
argc = 1
argv[0] = "0x81600000"
argv[1] = "<NULL>"
Hit any key to exit ...
```

Remarque : La commande tftp ne fonctionnera pas tant que la configuration réseau n'est pas correcte. Il faut impérativement que l'adresse IP de la connexion par pont de la VM soit 192.168.1.1.

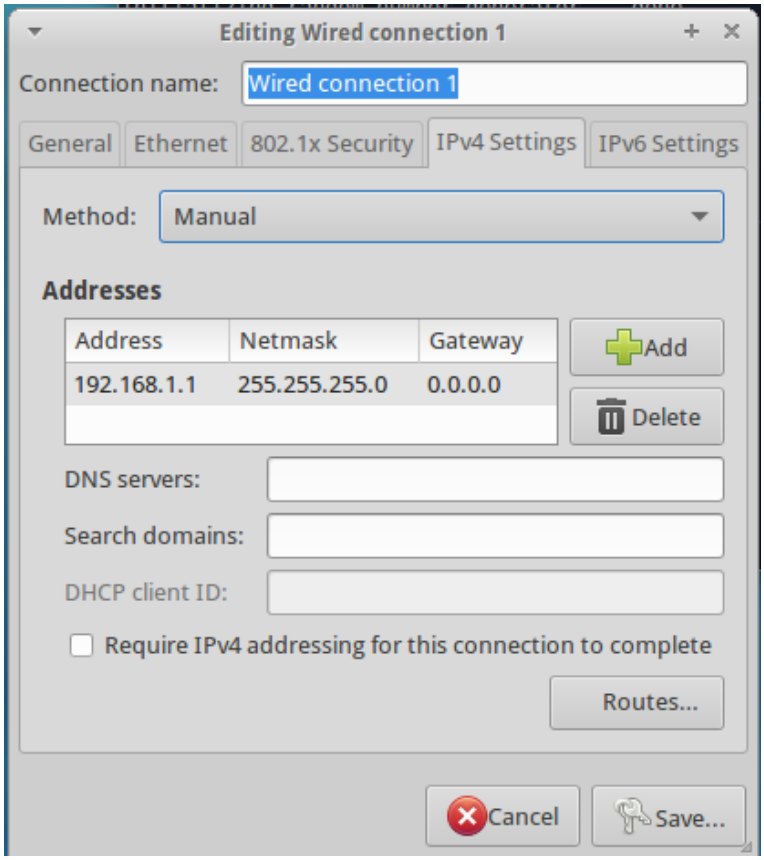

FIGURE 8 – Configuration réseau

b) Donnée : Déployez l'application helloworld dans Linux à l'aide du réseau et de la commande scp.

Travail réalisé : Nous avons découvert que l'adresse IP de la carte n'était pas celle attendue, nous avons donc dû adapter scp pour l'adresse IP 192.168.1.254.

```
Reptar # boot
reading uImage
...
*** Welcome on REPTAR (HEIG-VD/REDS): use root/root to log in ***
reptar login: root
Password:
# ifconfig
eth0      Link encap:Ethernet  HWaddr E4:AF:A1:40:01:FE
inet addr:192.168.1.254  Bcast:192.168.1.255  Mask:255.255.255.0
...
```

La commande scp permet de transférer le helloworld_linux sur la carte reptar par l'interface réseau depuis la machine hôte.

```
$ scp helloworld root@192.168.1.254:helloworld

The authenticity of host '192.168.1.254 (192.168.1.254)' can't be established.
```

```
RSA key fingerprint is fb:59:a3:73:97:9d:b7:b9:8a:40:e8:bc:19:ab:ab:70.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.254' (RSA) to the list of known hosts.
root@192.168.1.254's password:
helloworld                                    100% 6877     6.7KB/s   00:00
```

L'application helloworld est maintenant chargée sur la cible, il ne reste plus qu'à l'exécuter.

```
# ls
bitstreams  helloworld     tests
# ./helloworld
Hello world within Linux
argv[0] = ./helloworld
#
```

1.6 Accès aux périphériques REPTAR

a) Donnée : Sur la base de l'exemple gpio_u-boot., vous devez développer une application permettant d'interagir avec les LEDs et les switchs présents sur la carte CPU de la plate-forme REPTAR. Le but de l'application est d'allumer une LED lorsqu'on appuie sur un switch.

1. La LED 0 doit s'allumer lorsqu'on appuie sur le SWITCH 0.
2. La LED 1 s'allume si l'on appuie sur le SWITCH 1.
3. Et ainsi de suite pour les LEDs et switchs 0..3 de la carte CPU.

Le switch numéro 4 sert à quitter l'application. Aidez-vous des fichiers d'en-tête (#include) déjà présents dans le chablon fourni.
L'application gpio_u-boot est à déployer dans U-boot via la commande tftp.

Emplacement du code : /gpio_u-boot.c

Ce code est très basique, mais implémente correctement les points exigés par la donnée. Les commandes suivantes ont permis de lancer l'application sur la cible réelle dans l'U-boot. Une pression sur le switch numéro 4 permet de terminer l'application.

```
$ cd ~/seee_student/gpio_u-boot
$ make
...
$ cp gpio_u-boot.bin /home/redsuser/tftpboot
$ sudo picocom -b 115200 /dev/ttyUSB0
[sudo] password for redsuser:
picocom v1.7
...
Terminal ready

Reptar # tftp 0x81600000 gpio_u-boot.bin
smc911x: detected LAN9220 controller
smc911x: phy initialized
smc911x: MAC e4:af:a1:40:01:fe
```

```
15 Using smc911x-0 device
16 TFTP from server 192.168.1.1; our IP address is 192.168.1.254
17 Filename 'gpio_u-boot.bin'.
18 Load address: 0x81600000
19 Loading: T #
20 done
21 Bytes transferred = 776 (308 hex)
22
23 Reptar # go 0x81600000
24 ...
25 Stop of the GPIO U-boot Standalone Application
26 Reptar #
```

2 Émulation de périphériques

2.1 Environnement Qemu et machine Reptar

Cette étape vous permet de vous familiariser avec l'environnement que nous utiliserons pour l'émulation de périphériques. Dans cette étape, il est nécessaire de travailler avec l'application graphique Qtemu, qui constitue le frontend graphique de Qemu. L'application est développée en C++ et utilise la librairie Qt.

a) Donnée : A partir du répertoire seee_student, lancez le frontend graphique avec le script stq

Travail réalisé :

```
$ cd ~/seee_student/
$ ./stq
...
Reptar #
```

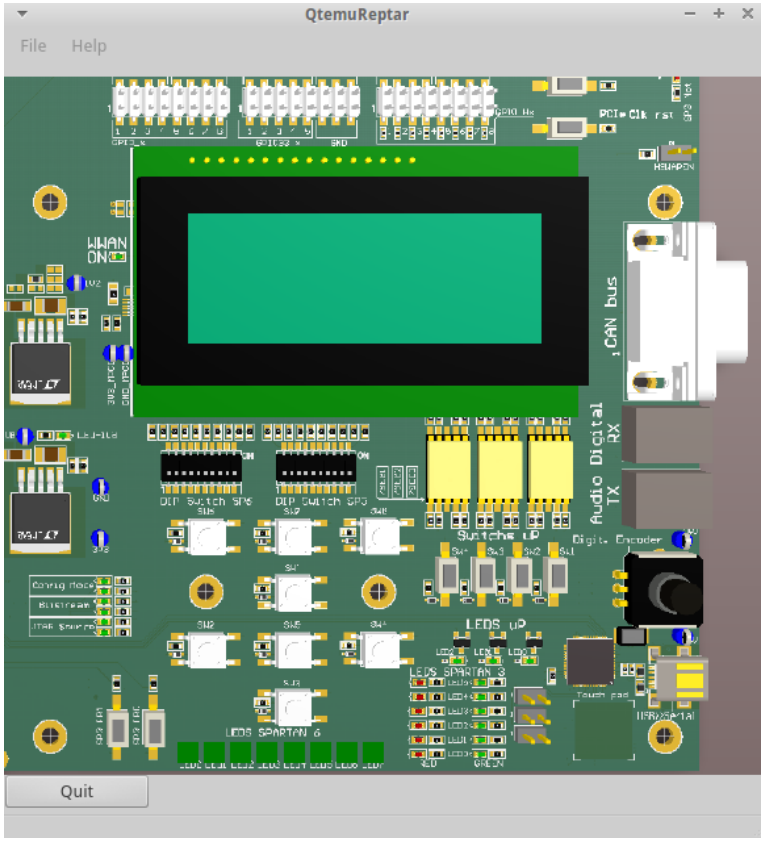

FIGURE 9 – Frontend graphique de Qemu

b) Donnée : Les fichiers sources de Qemu se trouvent dans le répertoire qemu-reds. Examinez les fichiers suivants :

1. hw/arm/reptar/reptar.c Emulation plate-forme REPTAR
2. hw/reptar_sp6.c Emulation de la FPGA
3. hw/reptar_sp6_clcd.c Emulation gestion du LCD4x20
4. hw/reptar_sp6_buttons.c Emulation gestion des boutons
5. hw/reptar_sp6_emul.c Gateway entre Qemu et Qtemu

Vous trouverez également toute la documentation nécessaire sur la plate-forme Reptar dans le répertoire doc.

Remarque : Ces différents fichiers implémentent ce qui ressemble à des modules noyaux.

c) Donnée : La compilation de Qemu pourra s'effectuer dans le répertoire qemu-reds directement, avec la commande make (utilisez make -j4 ou -j8 pour aller plus vite !).

Travail réalisé : Par la suite, seule la commande *make* sera nécessaire pour recompiler l'émulateur.
En lançant *./qtemu* et Eclipse, on pourra debugger l'émulateur Qemu.

```
$ cd ~/seee_student/qemu-reds/
$ ./configure --target-list=arm-softmmu --enable-debug --disable-attr --disable
    -docs
...
$ make -j8
...
$
```

2.2 Émulation de la FPGA Spartan6

Dans cette étape, il s'agit de mettre en place la structure nécessaire à l'émulation de la FPGA intégrée à la plate-forme. La FPGA implémente des registres associés aux périphériques externes. Dans cette étape, il s'agit de s'assurer que l'accès aux adresses I/O en lecture et écriture fonctionne.

a) Donnée : Complétez l'émulation de la FPGA afin de tester l'écriture et la lecture à l'une ou l'autre adresse dédiée à la FPGA (affichez simplement un message).

Emplacement du code :
/emulationSpartan6_part2/reptar_sp6.c
/emulationSpartan6_part2/reptar.c

Travail réalisé : Nous avons modifié les fichiers *reptar_sp6.c* et *reptar.c*
Le point crucial de cette partie du labo a été de trouvé l'adresse de base de la FPGA qui est **0x18000000**. Nous avons en effet besoin de cette adresse pour lier/créer le *reptar_sp6* dans la partie reptar.

```
  sysbus_create_simple("reptar_sp6",0x18000000,NULL);
```

Pour le reste de l'implémentation, nous nous somme basé sur le diagramme de séquence du support de cours et avons pris le document *versatilepb.c* comme exemple pour le contenu des méthodes callback.
Le bon fonctionnement du code a été "testé" premièrement en réussissant la compilation sans erreurs, puis le lancement sans crash. Nous avons également ajouté des messages affichés dans la console dans les différentes méthodes callback pour suivre l'initialisation. L'exécution a été faite de la manière suivante :

```
$ cd ~/seee_student/qemu-reds/
$ make
...
$ cd ..
$ ./stq
...
sp6 init
reptar-sp6-emul: sp6_emul_init
sp6 initfn
...
Reptar #
```

b) Donnée : Testez les accès en lecture-écriture avec U-Boot.

Travail réalisé : Pour l'instant, les callback de lecture/écriture que nous avons implémentés contiennent uniquement des messages d'indication qui sont affichés dans la console. À l'aide des commandes suivantes, nous avons pu tester leur bon fonctionnement. Les commandes suivantes tentent de lire, puis écrire à l'adresse de base de la FPGA.

```
Reptar # md.l 0x18000000 1
18000000:sp6 read
 00000000     ....
Reptar # mw.l 0x18000000 1
sp6 write
Reptar #
```

2.3 Émulation des devices de type LED (output)

Donnée : La FPGA est connectée à des LEDs qui sont visibles sur l'interface graphique. Cette étape consiste à implémenter le code d'émulation précédent afin de gérer l'accès aux LEDs reliées à la FPGA. Les interactions entre la FPGA et l'interface graphique doivent être gérées proprement.

Emplacement du code :*/emulationSpartan6_part3/reptar_sp6.c*

Travail réalisé : Pour cette partie, il a fallu rechercher l'offset du registre des LEDs qui est *0x003A*. Il faut donc ajouter cet offset à l'adresse de base de la FPGA.
Nous avons défini une variable qui garde la valeur écrite dans le registre des LEDs pour permettre la relecture de la valeur.

Pour la lecture et l'écriture, on teste si l'offset correspond et si l'on lit/écrit des données de la bonne taille, soit 16bits. Des messages ont été implémentés pour indiquer si la lecture/écriture est faite correctement.
La valeur lue est simplement affichée dans la console. Pour l'écriture, la valeur écrite est transmise à l'aide d'un message JSON conformément aux directives du *Guide d'utilisation de l'infrastructure*.
Nous avons testé le bon fonctionnement du code avec le test 10 de itbok [1]. Le test montre que l'on arrive à lire et écrire correctement sur les leds.

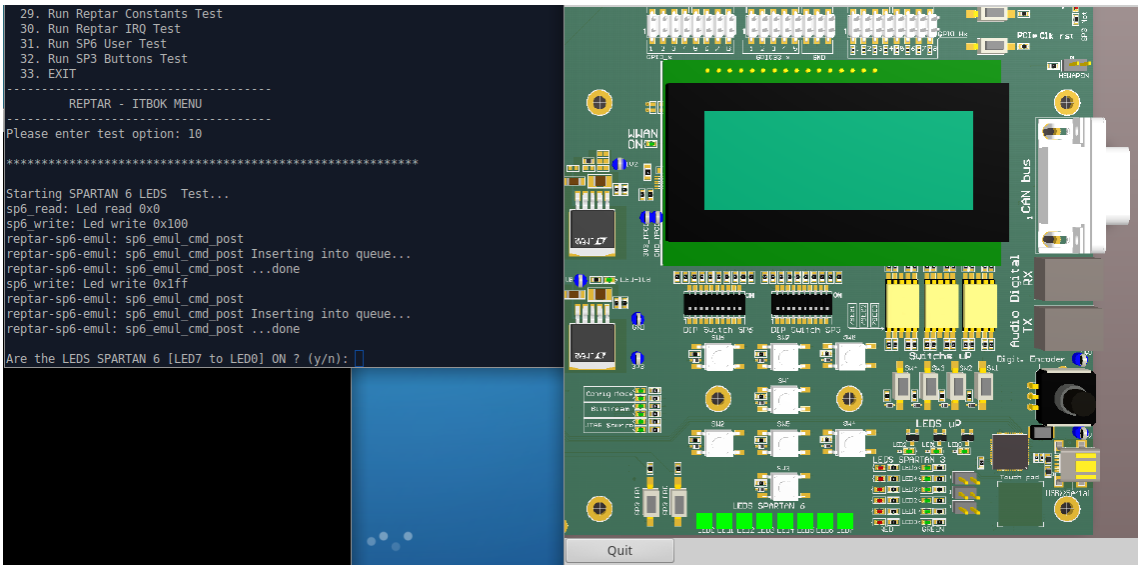

FIGURE 10 – Test d'allumage des LEDs

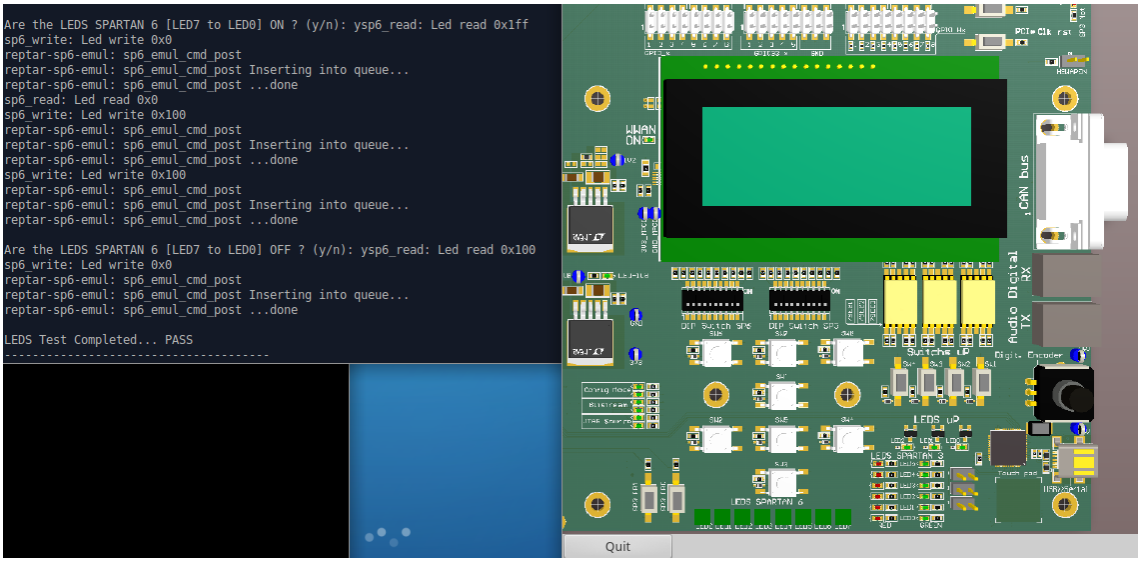

FIGURE 11 – Test d'extinction des LEDs

1. itbok, signifie Is The Board OK

2.4 Émulation de type boutons (input)

La FPGA est connectée à une série de boutons (switches) sur la plate-forme Reptar. Cette étape consiste à mettre en place la structure nécessaire à la gestion de ces boutons.

a) Donnée : Adaptez les fichiers nécessaires afin que l'émulation de votre périphérique (FPGA) puisse détecter la pression d'une touche, sans vous préoccuper pour le moment des interruptions.

Emplacement du code :
/emulationSpartan6_part4/reptar_sp6.c
/emulationSpartan6_part4/reptar_sp6_buttons.c

Travail réalisé : Le code de cette partie est inspiré du *Guide d'utilisation de l'infrastructure*. L'offset pour lire la valeur des boutons est *0x0012*. Lorsqu'un bouton est pressé, le handler du fichier sp6_button est appelé et la valeur du registre est mémorisée. La valeur des boutons peut également être récupérée lors d'une lecture du registre correspondant.

b) Donnée : Le projet sp6_buttons_u-boot contient une application permettant de tester vos boutons (en mode polling). Compilez l'application et effectuez quelques tests.

Travail réalisé : L'application a été compilée avec make. Il faut ensuite lancer l'émulateur avec la commande *stq*. Une fois dans l'U-boot, on peut lancer l'application testant les boutons. Elle est enregistrée dans les variables d'environnements sous le nom *tftp3*. Les lignes ci-dessous démontrent le bon fonctionnement des boutons. Le bouton exit arrête l'application.

```
$ cd ~/seee_student
$ ./stq
...
Reptar # run tftp3
smc911x: detected LAN9118 controller
smc911x: phy initialized
smc911x: MAC e4:af:a1:40:01:fe
Using smc911x-0 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.10
Filename 'sp6_buttons_u-boot/sp6_buttons.bin'.
Load address: 0x81600000
Loading: ########
done
Bytes transferred = 34512 (86d0 hex)

Reptar # run goapp
## Starting application at 0x81600000 ...
Start of the SP6 buttons standalone test application.
Button ONE pressed
...
Button ONE pressed
Button ONE pressed
Button ONE pressed
reptar-sp6-emul: sp6_emul_event_handle: read 29
reptar-sp6-emul: sp6_emul_event_handle: cJSON_Parse done
Button status : 0x0
```

```
27
28 Button LEFT pressed
29 Button LEFT pressed
30 Button LEFT pressed
31 reptar-sp6-emul: sp6_emul_event_handle: read 29
32 reptar-sp6-emul: sp6_emul_event_handle: cJSON_Parse done
33 Button status : 0x0
34
35 reptar-sp6-emul: sp6_emul_event_handle: read 30
36 reptar-sp6-emul: sp6_emul_event_handle: cJSON_Parse done
37 Button status : 0x10
38 Button EXIT pressed
39 SP6 buttons standalone test application exit.
40 ## Application terminated, rc = 0x0
41 Reptar # reptar-sp6-emul: sp6_emul_event_handle: read 29
42 reptar-sp6-emul: sp6_emul_event_handle: cJSON_Parse done
43 Button status : 0x0
```

2.5 Gestion des interruptions (IRQ) avec les boutons

Complétez votre émulateur avec le code nécessaire à la gestion d'une interruption à niveau émise par la FPGA lorsqu'un bouton est pressé. L'interruption est censée être acquittée par le driver. Il faut donc gérer l'état interne associé à cette interruption.

a) Donnée : Commencez par adapter le code d'initialisation de la plate-forme (reptar.c) afin d'instancier une interruption en provenance de la FPGA ; l'interruption sera de type niveau.

Emplacement du code :
/emulationSpartan6_part5/reptar_sp6.c
/emulationSpartan6_part5/reptar.c
/emulationSpartan6_part5/reptar_sp6_buttons.c

Travail réalisé : La première étape a consisté à assigner le reptar_sp6 sur la pin GPIO 10 dans le fichier *reptar.c*.
Il a ensuite fallu faire en sorte de générer une interruption de type niveau lors d'une pression sur un bouton. Cela a été fait dans le fichier *reptar_sp6_buttons.c*. Notre code génère l'interruption uniquement si les IRQ ont été préalablement autorisées en configurant les registres de contrôle. Nous avons choisi de ne pas générer d'interruptions lors de la relâche du bouton (0x0).
Finalement, le fichier *reptar_sp6.c* a été adapté pour permettre la lecture et l'écriture du registre d'IRQ des boutons. Lorsque le registre est lu, on va lire la valeur des bouton et l'ajouter dans le registre de status de l'IRQ, puis retourner le tout. Pour l'écriture, un masquage est fait afin de savoir si l'on veut activer et/ou quittancer les interruptions et dans ce cas repasser la GPIO10 à l'état bas.

b) Donnée : Testez que l'interruption fonctionne en configurant le contrôleur GPIO et en

interrogeant le registre d'état, dans U-Boot. Les registres du microcontrôleur à utiliser sont les suivants : GPIO_RISINGDETECT, GPIO_IRQENABLE1 et GPIO_IRQSTATUS1 De plus, l'interruption doit aussi être activée au niveau de la FPGA (cf documentation).

Travail réalisé : Notre code a dans un premier temps été testé à l'aide d'*itbok* avec le test numéro 30. Cela a permis de valider le bon fonctionnement des interruptions avec tous les boutons. On peut voir que le message *IRQ RAISE* ne s'affiche plus si l'on désactive les interruptions.

```
Press on SW7...
reptar-sp6-emul: sp6_emul_event_handle: read 29
reptar-sp6-emul: sp6_emul_event_handle: cJSON_Parse done
Button status : 0x0
reptar-sp6-emul: sp6_emul_event_handle: read 30
reptar-sp6-emul: sp6_emul_event_handle: cJSON_Parse done
Button status : 0x40
IRQ RAISE
sp6_read: Button irq status read 0x8d (button value 0x7)
sp6_write: Button irq status write 0x81
Enable IRQ
Clear IRQ
IRQ detected:
- button: 7  ......... Test PASSED
Press on SW8...
reptar-sp6-emul: sp6_emul_event_handle: read 29
reptar-sp6-emul: sp6_emul_event_handle: cJSON_Parse done
Button status : 0x0
reptar-sp6-emul: sp6_emul_event_handle: read 31
reptar-sp6-emul: sp6_emul_event_handle: cJSON_Parse done
Button status : 0x80
IRQ RAISE
sp6_read: Button irq status read 0x8f (button value 0x8)
sp6_write: Button irq status write 0x81
Enable IRQ
Clear IRQ
IRQ detected:
- button: 8  ......... Test PASSED
sp6_write: Button irq status write 0x1
Disable IRQ
Clear IRQ

IRQ test complete. Press Enter to exit
reptar-sp6-emul: sp6_emul_event_handle: read 29
reptar-sp6-emul: sp6_emul_event_handle: cJSON_Parse done
Button status : 0x0
reptar-sp6-emul: sp6_emul_event_handle: read 31
reptar-sp6-emul: sp6_emul_event_handle: cJSON_Parse done
Button status : 0x80
reptar-sp6-emul: sp6_emul_event_handle: read 29
reptar-sp6-emul: sp6_emul_event_handle: cJSON_Parse done
Button status : 0x0
```

Notre code a ensuite été testé avec les registres GPIO du microcontrôleur (*GPIO_RISINGDETECT,*

GPIO_IRQENABLE1, GPIO_IRQSTATUS1). Il faut utiliser pour ce faire le banque GPIO1, ce qui donne les adresses de registre correspondantes : 0x48310048, 0x4831001C et 0x48310018. Comme les boutons sont sur la GPIO 10, il faut aller autoriser l'interruption en activant le bit 10 des registres GPIO_RISINGDETECT et GPIO_IRQENABLE1. Il ne faut pas oublier d'aller ensuite autoriser les interruptions sur la FPGA. La capture ci-dessous présente cette configuration. Après chaque interruption, il faut aller quittancer le bit 10 du GPIO_IRQSTATUS1 et également intervenir au niveau de la FPGA.

```
#Enable interrupts in GPIO
Reptar # mw.l 0x4831001C 0x00000400
Reptar # mw.l 0x48310048 0x00000400

#Enable interrupts in FPGA
Reptar # mw.w 0x18000018 0x0081
sp6_write: Button irq status write 0x81
Enable IRQ
Clear IRQ

#The IRQ are now available, the status register doesn't have detect interrupt
    yet
md.l 0x48310018 1
48310018: 00000000    ....

#Generate an interrupt
Reptar # reptar-sp6-emul: sp6_emul_event_handle: read 30
reptar-sp6-emul: sp6_emul_event_handle: cJSON_Parse done
Button status : 0x40
IRQ RAISE
reptar-sp6-emul: sp6_emul_event_handle: read 29
reptar-sp6-emul: sp6_emul_event_handle: cJSON_Parse done
Button status : 0x0

#Check status register, interrupt has been detected
md.l 0x48310018 1
48310018: 00000400    ....

#Need to ack the interrupt
Reptar # mw.l 0x48310018 0x00000400
Reptar # mw.w 0x18000018 0x0081
sp6_write: Button irq status write 0x81
Enable IRQ
Clear IRQ

#...Repeat operations
```

2.6 Émulation de l'afficheur 7 segments

La FPGA est connectée à un afficheur 7 segments, visible sur l'émulateur. Cette étape consiste à mettre en place la gestion de cet afficheur 7 segments.

a) Donnée : Adaptez les fichiers nécessaires afin que l'émulation de votre périphérique

(FPGA) puisse gérer les trois digits de l'afficheur 7 segments.

Emplacement du code : */emulationSpartan6_part6/reptar_sp6.c*

Travail réalisé : Pour cette partie, nous avons simplement ajouté le code pour écrire et lire dans le registre de chacun des trois digits de l'affichage 7 segments. La valeur de chaque affichage est stocké dans une variable. Pour l'écriture, il faut spécifier dans le message Json le nom du périphérique, le numéro du digit ainsi que la valeur à afficher.

b) Donnée : Le dossier 7seg_u-boot contient une application permettant de tester l'afficheur 7 segments : les chiffres de 0 à 9 doivent défiler progressivement : 012, puis 123, 234, 456, 567, ..., 901, 012, etc. Compilez l'application et effectuez quelques tests.

Travail réalisé : Une fois l'application compilée, il a fallu la lancer dans l'U-boot. Comme elle n'est pas définie dans les variables d'environnement, il faut utiliser la commande complète. Une fois lancée, l'application va incrémenter la valeur des digits. L'image ci-dessous démontre le bon fonctionnement.

```
$ cd ~/seee_student
$ ./stq
...
Reptar # tftp 7seg_u-boot/7seg_u-boot.bin
smc911x: detected LAN9118 controller
smc911x: phy initialized
smc911x: MAC e4:af:a1:40:01:fe
Using smc911x-0 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.10
Filename '7seg_u-boot/7seg_u-boot.bin'.
Load address: 0x81600000
Loading: ########
done
Bytes transferred = 34932 (8874 hex)
Reptar # run goapp
```

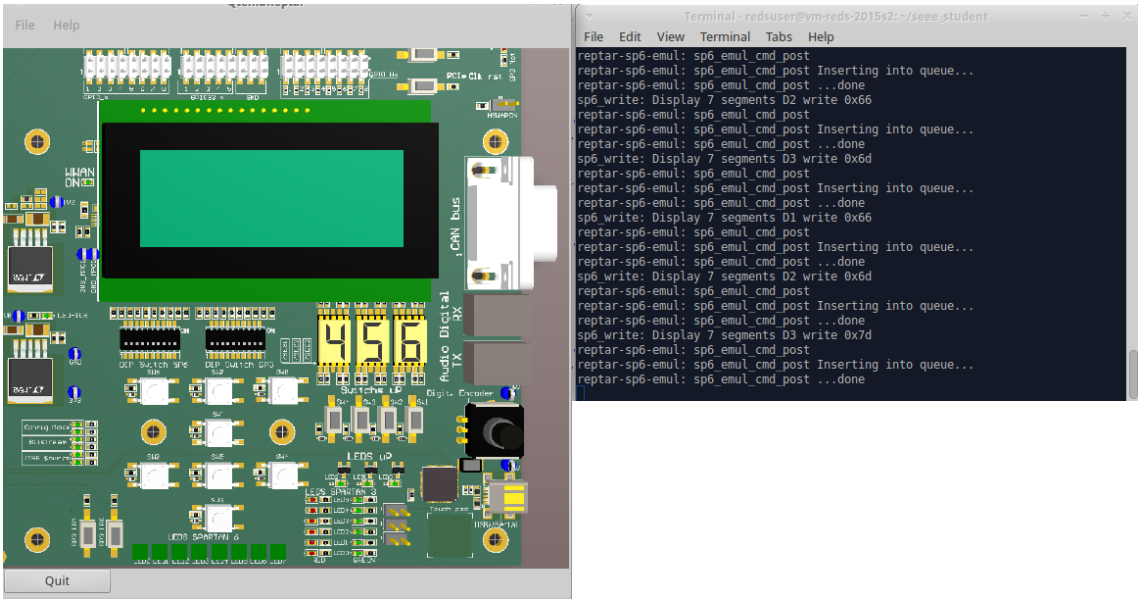

FIGURE 12 – Frontend graphique de Qemu avec affichage 7 segments

2.7 Mini-application utilisant les boutons et l'afficheur 7 segments

a) Donnée : Le dossier miniapp_u-boot contient un chablon. Complétez-le afin de créer une application qui utilise les boutons SW2, SW5, SW4 et SW3, ainsi que l'afficheur 7 segments.

1. Lors d'un appui sur SW2, SW5 ou SW4, le digit respectivement à gauche, au centre ou au milieu est incrémenté de 1, modulo 10. Si un digit atteint 9, il reviendra à 0.
2. La valeur initiale de chaque digit, au démarrage de l'application, est 0 (on affichera 000).
3. Un appui sur SW3 quitte l'application.
4. Vous devrez gérer l'anti-rebond : le digit ne devra être incrémenté que si le bouton est pressé puis relâché (comme un appui sur une touche de sonnette par exemple).

Emplacement du code : */miniapp_test_emulation/miniapp_u-boot.c*

Travail réalisé : L'application a été codée avec une boucle *while* exécutant de manière répétitive les étapes suivantes :

- On lit le registre d'état des boutons et on stocke sa valeur dans une première variable.
- On attend un petit moment.
- On relit le registre d'état des boutons et on stocke cette nouvelle valeur dans une seconde variable.
- Ensuite, on teste les deux variables avec des masques pour savoir si :
  - Dans la première variable, le masquage retourne une valeur supérieure à 0 et indique que le bouton testé est enfoncé.
  - Dans la seconde variable, le masquage retourne une valeur égale à 0 et indique que le bouton a été relâché.

Ces tests sont effectués pour les trois boutons d'incrémentation des 7 segments et pour le bouton d'arrêt de l'application.

— Si un des tests passe :
  — Si c'est un des boutons d'incrémentation, on appelle la fonction *incr_7seg* avec en paramètre l'index du 7 segments à incrémenter.
  — Si c'est le bouton d'arrêt, on quitte la boucle avec un *break*.

Nous avons remarqué une chose lors de la compilation. Il s'avère que le compilateur prend comme point d'entrée la première instruction du programme, avec le Makefile fourni. Lors des premiers essais, la fonction *incr_7seg* était implémentée avant le *main*, ce qui en faisait le point d'entrée du programme. Après correction, c'est à dire, définition du prototype de la fonction avant le *main* et implémentation après, tout fonctionne correctement.

b) Donnée : Testez votre application sur l'émulateur

Travail réalisé :
La figure 13 montre l'application qui vient de démarrer. Tous les 7-segments sont initialisés à zero.

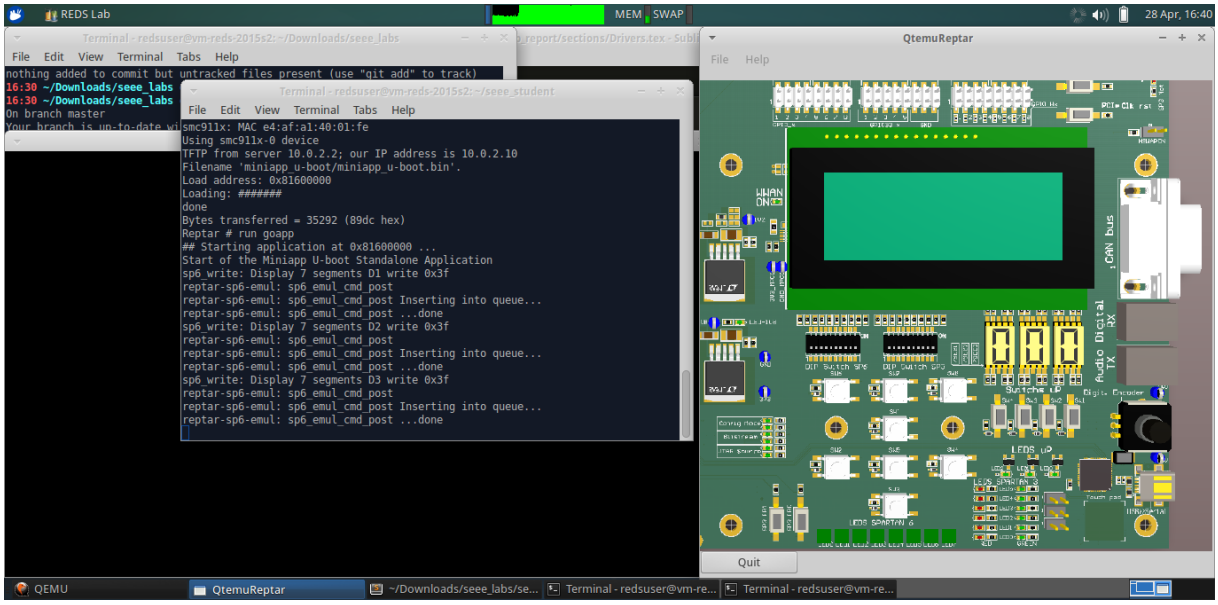

FIGURE 13 – Initialisation des 7-segments durant le lancement de l'application sur le frontend d'émulation

La figure 14 montre l'application avec plusieurs 7-segments incrémentés.

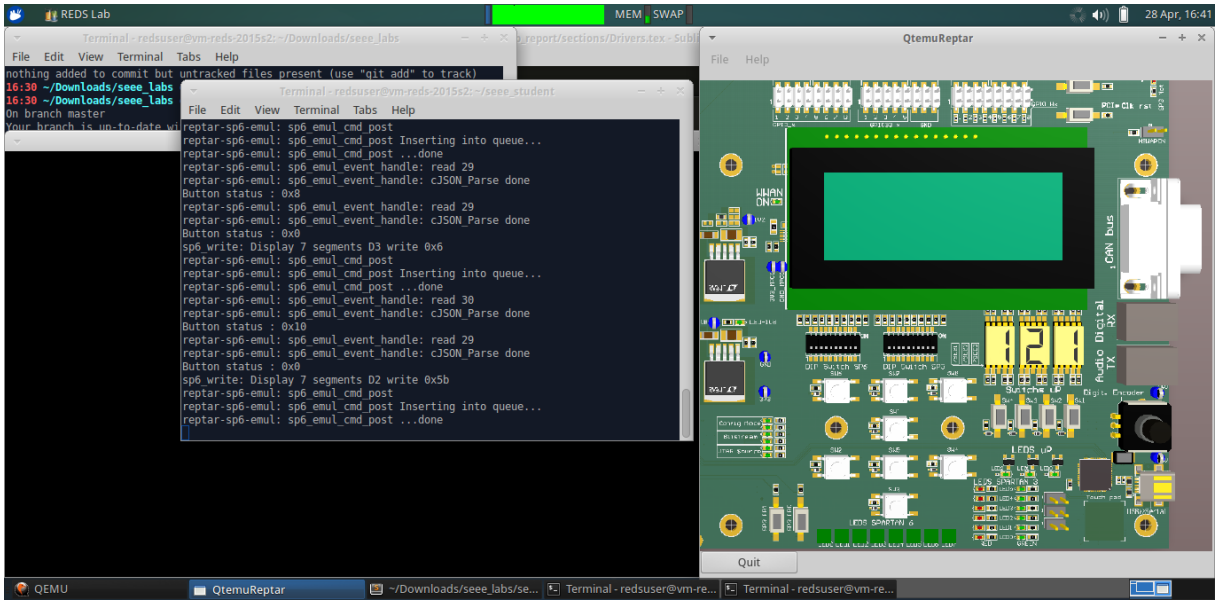

FIGURE 14 – Incrémentation des 7-segments du frontend d'émulation

La figure 15 montre la fin de l'application. Le bouton tout en bas de la croix a été pressé pour l'arrêter.

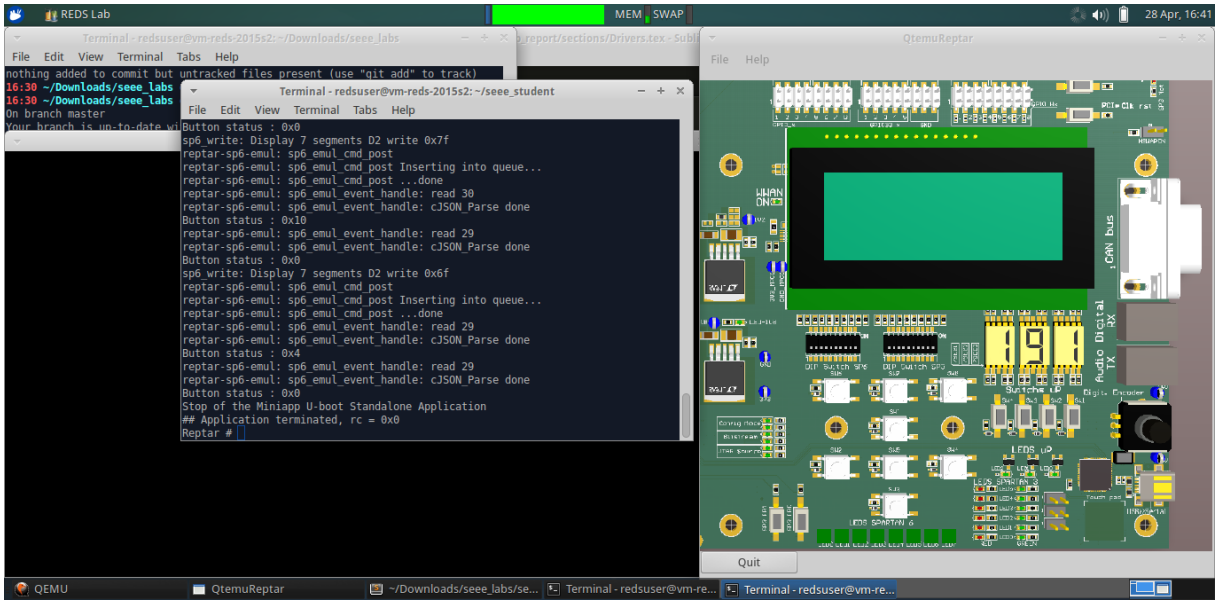

FIGURE 15 – Frontend graphique de Qemu avec affichage 7 segments

c) Donnée : Déployez et testez votre application sur la plate-forme réelle

Travail réalisé : Une fois l'application compilée, il a fallu copier le *.bin* dans le dossier *tftpboot*. On peut ensuite lancer l'application depuis l'U-boot en accédant la carte Reptar avec *picocom*. La pression sur le switch 3 réussit ici aussi à terminer l'application.

```
$ cd ~/seee_student
$ sudo picocom -b 115200 /dev/ttyUSB0
[sudo] password for redsuser:
...
Terminal ready

Reptar # tftp 0x81600000 miniapp_u-boot.bin
smc911x: detected LAN9220 controller
smc911x: phy initialized
smc911x: MAC e4:af:a1:40:01:0a
Using smc911x-0 device
TFTP from server 192.168.1.1; our IP address is 192.168.1.10
Filename 'miniapp_u-boot.bin'.
Load address: 0x81600000
Loading: T ###
done
Bytes transferred = 35292 (89dc hex)

Reptar # go 0x81600000
## Starting application at 0x81600000 ...
Start of the Miniapp U-boot Standalone Application
Stop of the Miniapp U-boot Standalone Application
## Application terminated, rc = 0x0
Reptar #
```

Et ici, une photo de l'application en train de tourner sur la plateforme :

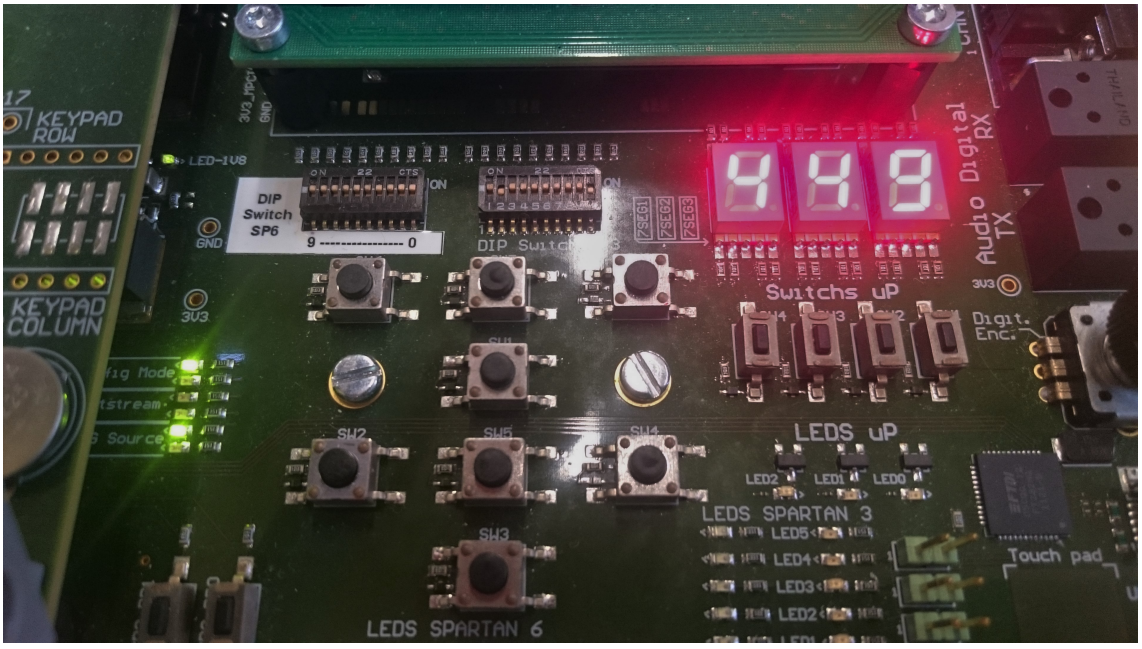

FIGURE 16 – Programme miniapp en fonctionnement sur la plateforme physique

3 Drivers

3.1 Environnement Qemu et plate-forme Reptar

Lors de cette étape, nous allons déployer l'environnement de la cible à partir d'une image de carte MMC (sdcard) et nous nous familiariserons avec l'insertion dynamique de module. Le projet concerné est le projet drivers (répertoire du même nom à la racine du workspace).

a) Donnée : Lancez make dans le répertoire drivers/

Travail réalisé :

```
$ cd ~/seee_student/drivers/
$ make
...
make[1]: Leaving directory '/home/redsuser/seee_student/linux-3.0-reptar'
arm-linux-gnueabihf-gcc -marm -I../linux-3.0-reptar -static buttons_test.c -o
    buttons_test
$
```

b) Donnée : A la racine du workspace, lancez les scripts suivants, puis la commande boot dans U-boot :

```
$ ./deploy
$ ./stf
Reptar # boot
```

Travail réalisé :

```
$ cd ~/seee_student/drivers
$ ./deploy
Deploying into reptar rootfs ...
Mounting filesystem/sd-card.img...
[sudo] password for redsuser:
SD card partitions mounted in 'boot_tmp' and 'filesystem_tmp' directories
Unmounting SD card image...
Synchronizing .img file
Unmounting 'boot_tmp' and 'filesystem_tmp'...
Done !
$ ./stf
...
Reptar # boot
reading uImage
...
*** Welcome on REPTAR (HEIG-VD/REDS): use root/root to log in ***
reptar login: root
Password:
#
```

c) Donnée : A la racine du rootfs (cd /), insérez le module avec la commande suivante :

```
# insmod sp6.ko
```

Vérifiez qu'il n'y ait aucun message d'erreur. La liste des modules chargés dynamiquement est obtenue avec la commande lsmod et le retrait du module avec la commande rmmod

```
# lsmod
# rmmod sp6
reptar_sp6: bye bye!
#
```

Travail réalisé : Avec la commande lsmod, on peut vérifier que notre module est correctement chargé. Si on le retire, il n'apparaît plus dans la liste.

```
# cd /
# pwd
/
# insmod sp6.ko
reptar_sp6: module starting...
Probing FPGA driver (device: fpga)
input: reptar_sp6_buttons as /devices/platform/fpga/reptar_sp6_buttons/input/
    input1
reptar_sp6: done.
# lsmod
Module                  Size  Used by    Not tainted
sp6                     4606  0
# rmmod sp6
reptar_sp6: bye bye!
# lsmod
Module                  Size  Used by    Not tainted
#
```

3.2 Driver de type caractère

Cette étape consiste à travailler sur un driver de type caractère au niveau FPGA. Le code de cette partie se trouve dans les fichiers reptar_sp6.h et reptar_sp6.c. Sur la base du code existant, on souhaite pouvoir écrire et lire une chaîne de caractères contenant la version du bitstream (hypothétique) dans la FPGA, stockée dans la variable globale bitstream_version.

a) Donnée : Complétez les callbacks read() et write() afin qu'une application utilisateur puisse lire et écrire une chaîne de (80 max.) caractères.

Travail réalisé : Les callbacks read et write ont été implémentés très sommairement dans le module avec la fonction *copy_from/to_user*. Il faut faire attention à bien retourner le nombre de bytes qui ont été lus ou écrits, pour que l'application puisse être au fait et détecter le cas échéant, si une erreur s'est produite.

```
# cd /
# insmod sp6.ko
reptar_sp6: module starting...
Probing FPGA driver (device: fpga)
input: reptar_sp6_buttons as /devices/platform/fpga/reptar_sp6_buttons/input/
    input1
reptar_sp6: done.
# ./usertest
Device ID : 0
Inode number : 5
Protection mode : 0
Num of hard links : 680
User ID of owner : 8624
Group ID of owner : 1
Device ID (spec files only): 0
Total size [bytes] : 0
Setting bitstream version to : mais coucou mon petit
Device ID : 0
Inode number : 5
Protection mode : 0
Num of hard links : 680
User ID of owner : 8624
Group ID of owner : 1
Device ID (spec files only): 0
Total size [bytes] : 0
Normally read buffer
Bitstream version : mais coucou mon petit
```

b) Donnée : Pour identifier le nom de l'entrée dans /dev/ qui sera créée automatiquement, examinez la fonction probe( ) du driver.

Réponse aux questions :

1. Comment l'entrée dans /dev est-elle générée ?
2. Quel sera le nom de l'entrée dans /dev ?

Travail réalisé :

c) Donnée : Afin de tester votre driver, écrivez une application usertest (fichier usertest.c) qui écrira puis relira la chaîne de version en utilisant l'entrée dans /dev évoquée ci-dessus. Une application a aussi été codée pour tester les callbacks. Celle-ci affiche les stats du fichier créé dans le dossier */dev*, écrit une version de bitstream totalement fabuliste, relit les stats et enfin lit le bitstream pour confirmer le succès de l'écriture.

Réponse aux questions :

1. Recherchez les valeurs du major et minor attribuées à ce driver. Expliquez votre démarche

Travail réalisé :

3.3 Pilotage des LEDs

Le code de pilotage des LEDs se trouve dans le fichier reptar_sp6_leds.c. La réalisation du driver des LEDs.

1. L'application graphique qtemu sera utilisée pour l'environnement émulé.
2. Le driver devra être également testé sur la plate-forme réelle.

Pour le pilotage des LEDs, on souhaite utiliser le sous-système leds présent dans le noyau Linux. Effectuez un mappage du registre des LEDs à l'aide de la fonction ioremap( ), en vous servant de la structure fpga_resource.

a) Donnée : Enregistrez le device comme un device de type leds à l'aide de la fonction led_classdev_register( ).

Travail réalisé :

b) Donnée : Cherchez et implémentez le(s) callback(s) gérant l'enclenchement/déclenchement des LEDs.

Travail réalisé :

Réponse aux questions :

1. Combien y a-t-il de devices de type LED gérés par notre driver ?

Il y a 6 LEDs selon la déclaration dans reptar_sp6.h et il y a bien 6 LEDs dans la structure reptar_sp6_leds_pdata[] du fichier reptar_sp6.h

```
1 /* Only LEDS 0 to 5 are under CPU control. 6 and 7 are used by the FPGA itself
    */
2 #define SP6_NUM_LEDS 6
```

c) Donnée : Testez le driver LED dans l'environnement qtemu (application graphique). Lancez le script ledstest.sh.

Travail réalisé :

d) Donnée : Testez votre driver sur la plate-forme (réelle) Reptar.

Travail réalisé :

3.4 Pilotage des boutons

Lors de cette étape, nous travaillerons sur le driver gérant la pression des boutons. L'objectif est de contrôler une application dans l'espace utilisateur à l'aide des boutons.

a) Donnée : Complétez la fonction probe( ) pour l'enregistrement des deux callbacks d'interruption (traitement immédiat + traitement différé) à l'aide de la fonction request_threaded_irq().

Travail réalisé :

b) Donnée : Implémentez le traitement immédiat lié à l'interruption. Il devra :

1. stocker la valeur du registre bouton dans le champ current_button de la structure privée (l'adresse du registre contenant cette information est également disponible dans la structure privée)
2. acquitter l'interruption.

Travail réalisé :

c) Donnée : Testez votre driver boutons à l'aide de l'application buttons_test. Dans l'environnement émulé, l'application devra ouvrir le fichier /dev/input/event1. Il faudra taper la commande suivante :

```
# ./buttons_test -e1
```

Testez les boutons un par un.

Travail réalisé :