Laboratoire 1 : Introduction

Auteur

- Urs Behrmann

Table des matières

Préparation de la carte SD

dispositif associé avec le lecteur de cartes : /dev/sdb

Commande de copie:

```
sudo dd if=drv-2024-sdcard.img of=/dev/sdb bs=4M conv=fsync status=progress
```

Connexion console à la carte

Connexion à la carte via USB

Utilisez l'outil picocom (ou minicom, si vous le préférez) afin d'ouvrir une session de terminal avec la carte.

```
picocom -b 115200 /dev/ttyUSB0
```

Pour quitter Picocom, appuyez 'Ctrl+A' suivi de 'Ctrl+X'.

On devra donc mettre dans /home/reds/tftpboot l'image du noyau (zImage) et le fichier du Device Tree (socfpga.dtb) contenus dans l'archive drv-2025-boot.tar.gz. Ensuite, donnez les droits de lecture à tout le monde à ces fichiers avec

```
cd /home/reds/tftpboot
chmod 777 *
```

Configuration et utilisation de U-Boot

Pourriez-vous expliquer les lignes ci-dessus ?

```
setenv ethaddr "12:34:56:78:90:12"
setenv serverip "192.168.0.1"
setenv ipaddr "192.168.0.2"
setenv gatewayip "192.168.0.1"
setenv bootdelay "2"
setenv loadaddr "0xF000"
setenv bootcmd "mmc rescan; tftpboot ${loadaddr} zImage; tftpboot ${fdtaddr} ${fdtimage};
setenv mmcboot "setenv bootargs console=ttyS0,115200 root=${mmcroot} rw rootwait ip=${ipa
saveenv
```

1. Configurer la mac address de la carte
    i. 'ethaddr "12:34:56:78:90:12"'
2. Configurer le réseau
    i. 'serverip "192.168.0.1"'

    ii. 'ipaddr "192.168.0.2"'
    iii. 'gatewayip "192.168.0.1"'
3. Configurer le délai de boot
    i. 'bootdelay "2"'
4. Configuration des adresses mémoire
    i. 'loadaddr "0xF000"'
5. Commande de démarrage automatique
    i. 'bootcmd "mmc rescan; tftpboot $loadaddr zImage; tftpboot$ {fdtaddr} ${fdtimage}; run fpgaload; run bridge_enable_handoff; run mmcboot"'
6. Séquence de démarrage Linux
    i. 'mmcboot "setenv bootargs console=ttyS0,115200 root=$mmcrootrwrootwaitip =$ {ipaddr}:$serverip$:{serverip}:255.255.255.0:de1soclinux:eth0:on; bootz $loadaddr-$ {fdtaddr}"'
7. Sauvegarde de la configuration
    i. 'saveenv'

Vérification des toolchains

Pourriez-vous expliquer la signification des différents mots composant le nom de la toolchain? Pourquoi on a spécifié un numéro de version ? Qu'est-ce qui se passe si l'on prend tout simplement la toute dernière version de la toolchain?

La toolchain arm-linux-gnueabihf-6.4.1 est une chaîne de compilation croisée destinée aux processeurs ARM, fonctionnant sous Linux avec une ABI (Application Binary Interface) EABI Hard Float. Le suffixe gnueabihf indique que cette toolchain utilise la glibc GNU et prend en charge les calculs en virgule flottante matérielle (hard float) pour de meilleures performances sur les processeurs ARM compatibles. La version 6.4.1 fait référence au compilateur GCC 6.4.1, garantissant une compatibilité et une stabilité éprouvées pour les systèmes embarqués, tout en bénéficiant des optimisations et corrections de cette version.

Pourriez-vous expliquer ce que la ligne ci-dessus fait?

La ligne compile un programme C minimal en un exécutable ARM et affiche ses informations via la commande file.

```
echo "int main(){}" | arm-linux-gnueabihf-gcc-6.4.1 -x c - -o /dev/stdout | file -
```

Réponse :

```
/dev/stdin: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked, i
```

Que signifie "not stripped" ?

L'exécutable contient encore les symboles de débogage et autres métadonnées, ce qui augmente sa taille.

Compilation sur la cible (native)

Quelle version de GCC est installée sur la carte ?

```
root@de1soclinux:~# arm-linux-gnueabihf-gcc --version
arm-linux-gnueabihf-gcc (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3
Copyright (C) 2011 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

La version de GCC installée sur la carte est 4.6.3 (Ubuntu/Linaro 4.6.3-1ubuntu5).

Compilation croisée sous DE1-SoC

```
$ arm-linux-gnueabihf-gcc-6.4.1 hello_cross.c -o hello_cross
$ ls -al hello_cross
-rwxrwx--- 1 root vboxsf 11888 Mar  5 10:44 hello_cross
$ file hello_cross
hello_cross: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked,
```

Qu'est-ce qui se passerait si vous essayiez d'exécuter ce fichier sur votre PC ? Pourquoi ?

```
./hello_cross
bash: ./hello_cross: cannot execute binary file: Exec format error
```

L'exécution échoue avec "Exec format error" car hello_cross est un binaire compilé pour ARM (ELF 32-bit LSB ARM), alors que mon PC utilise une architecture x86_64. Le processeur ne peut pas exécuter un binaire destiné à une autre architecture.

Compilation statique

```
$ arm-linux-gnueabihf-gcc-6.4.1 hello_cross.c -o hello_static -static
$ ls -al hello_static
-rwxrwx--- 1 root vboxsf 2789828 Mar  5 10:46 hello_static
$ file hello_static
hello_static: ELF 32-bit LSB executable, ARM, EABI5 version 1 (GNU/Linux), statically lir
```

Run on DE1-SoC

```
root@de1soclinux:~# ~/drv/hello_cross
Hello, cross-compiled world!
```

Accès aux périphériques du CycloneV-SoC depuis U-Boot

Exercice 1

Expliquez les différences entre les lignes ci-dessous.

```
md.b 0x80008000 0x1
80008000: 46    F
md.w 0x80008000 0x1
80008000: 4c46    FL
md.l 0x80008000 0x1
80008000: eb004c46    FL..
md.b 0x80008000 0x4
80008000: 46 4c 00 eb    FL..
md.w 0x80008000 0x4
80008000: 4c46 eb00 9000 e10f    FL......
md.l 0x80008000 0x4
80008000: eb004c46 e10f9000 e229901a e319001f    FL........).....
```

Les différences entre ces commandes viennent de la taille des données lues en mémoire :

- md.b (memory display - byte) : Affiche 1 octet
- md.w (memory display - word) : Affiche 2 octets
- md.l (memory display - long) : Affiche 4 octets

Utilisez la commande md pour lire la valeur binaire écrite avec les switches et écrivez-la sur les LEDs.

Base Address	End Address	I/O Peripheral
0xFF200000	0xFF20000F	Red LEDs
0xFF200040	0xFF20004F	Slider Switches

```
md.b 0xFF200040 0x1
ff200040: 03  .
mw.b 0xFF200000 0x3 0x1
```

Qu'est-ce qui se passe si vous essayez d'accéder à une adresse qui n'est pas alignée (par exemple 0x01010101) et pourquoi ?

```
SOCFPGA_CYCLONE5 # md.b 0x01010101 0x1
01010101: ea    .
```

L'accès à une adresse non alignée (0x01010101) est possible, mais les données lues ne sont pas significatives. Les données lues sont arbitraires et dépendent de l'état de la mémoire à cet

endroit.

On peut le lire, car c'est une adresse mémoire valide, mais les données lues ne peuvent pas être interprétées sans connaissance du contexte.

Exercice 2

Écrivez un script (à l'aide d'une variable d'environnement) U-Boot qui va alterner, chaque seconde, entre afficher □□□□□□ et □□□□□□ sur les affichages à 7-segments.

Adresse des affichages à 7-segments

Base Address	End Address	I/O Peripheral
0xFF200020	0xFF20002F	7-segment HEX3−HEX0 Displays
0xFF200030	0xFF20003F	7-segment HEX5−HEX4 Displays

000000

```
mw.b 0xFF200020 0x3f 0x4
mw.b 0xFF200030 0x3f 0x2
```

555555

```
mw.b 0xFF200020 0x6d 0x4
mw.b 0xFF200030 0x6d 0x2
```

sleep N

Attendez N secondes.

```
sleep 1
```

Solution

```
setenv led_loop 'while true; do mw.b 0xFF200020 0x3f 0x4; mw.b 0xFF200030 0x3f 0x2; sleep
saveenv
run led_loop
```

Accès aux périphériques du DE1-SoC depuis Linux

Pouvez-vous identifier au moins deux gros problèmes de cette approche ?

L'accès direct aux périphériques matériels depuis l'espace utilisateur est dangereux et non recommandé. Cela peut entraîner des plantages du système, des fuites de mémoire, des corruptions de données et des problèmes de sécurité.

Exercice 3

Écrivez un logiciel user-space en C qui utilise /dev/mem pour accéder aux périphériques. Au démarrage, le logiciel commence par afficher A sur l'affichage HEX0. Ensuite, l'utilisateur peut contrôler la lettre affichée sur HEX0 de la façon suivante:

- En appuyant sur KEY0, le caractère actuellement affiché est décrémenté
  - Z devient Y, ... , B devient A
  - Si A est affiché, on recommence à Z
- En appuyant sur KEY1, le caractère actuellement affiché est incrémenté
  - A devient B, ... , Y devient Z
  - Si Z est déjà affiché, on recommence à A

De plus, le code ASCII de la lettre affichée doit être représenté en binaire sur les LEDs.

- Exemple: si A est affiché (soit 0x41 / 0b1000001 en ASCII), LED0 et LED6 doivent être allumée

Note: vous pouvez trouver un exemple de police de caractère pour afficheurs 7-segments sur cette page Wikipedia. Libre à vous de vous en inspirer.

Bonus 1: éteignez les afficheurs 7-segments et les LEDs en partant (quand le programme est arrêté avec Ctrl+C) pour éviter de faire fondre la banquise inutilement.

Compilation

```
arm-linux-gnueabihf-gcc-6.4.1 led_alpha.c -o led_alpha
```

Run

```
~/drv/led_alpha
```

Quel est le souci principal dans l'écriture de ce logiciel ?