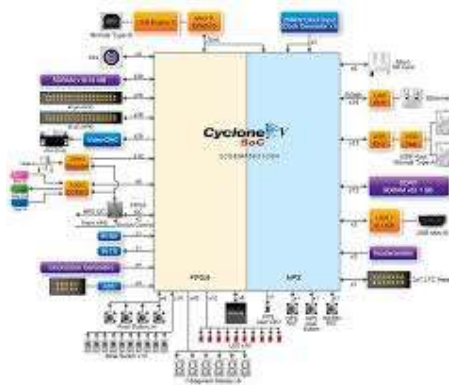


# Porter Linux sur la DE-1

## SYSTÈME SOC INTÉGRÉ AVEC FPGA (SOCF)



Auteur : Spinelli Isaia  
 Prof : Etienne Messerli  
 Ing : Sébastien Masle  
 Date : 15.05.2020  
 Salle : A09 (maison) – HEIG-VD  
 Classe : SOCF

## Table des matières

Introduction.....	- 2 -
Génération de la carte SD .....	- 2 -
Préambule .....	- 2 -
Manipulations.....	- 3 -
Vérification .....	- 3 -
Compilation de Linux.....	- 4 -
Préambule .....	- 4 -
Manipulations.....	- 4 -
Vérification .....	- 4 -
Création du Device Tree.....	- 5 -
Préambule .....	- 5 -
Manipulations.....	- 5 -
Vérification .....	- 6 -
Test du bridge HPS <-> FPGA.....	- 7 -
Préambule .....	- 7 -
Manipulations.....	- 7 -
Vérification .....	- 8 -
Annexes .....	- 9 -
Conclusion .....	- 9 -
Difficultés rencontrées .....	- 9 -
Compétences acquises .....	- 9 -
Résultats obtenus .....	- 9 -

## Introduction

Ce laboratoire a pour but d'apprendre comment partitionner une carte SD, compiler et démarrer Linux. Un bitstream « DE1\_SoC\_top.sof » nous a été fourni.

## Génération de la carte SD

Le but de cette étape est de récupérer une image SD officiel pour la DE1-SoC « Linux console » de Teraisc afin de la copier sur une carte SD et de démarrer correctement la DE1 avec cette carte SD.

### Préambule

Le firmware du HPS est capable de démarrer depuis une carte SD. Il s'attend à trouver le SPL sur une partition contenant l'identificateur 0xA2. Cette partition peut se trouver n'importe où. Le reste peut être partitionné n'importe comment. L'exemple suivant comprend

- Une partition 0xA2 contenant le SPL et U-Boot.
- Une partition FAT32 contient le kernel Linux (zImage), le device tree (dtb) et le bitstream (rbf).
- Une partition ext4 contient le rootfs utilisé par Linux.

```
# fdisk -lu /dev/sdX
```

```
Disk /dev/sdX: 7948 MB, 7948206080 bytes
```

```
245 heads, 62 sectors/track, 1021 cylinders, total 15523840 sectors
```

```
Units = sectors of 1 * 512 = 512 bytes
```

Device	Boot	Start	End	Blocks	Id	System
/dev/sdX1		62	167089	83514	b	W95 FAT32
/dev/sdX2		167090	182279	7595	a2	Unknown
/dev/sdX3		182280	15508989	7663355	83	Linux

Figure 0-1 : Exemple de partition

Il existe de la documentation et un référence design pour la carte DE-1. Il sera possible de récupérer une image de carte SD. Ceci permet d'avoir une base de travail stable et nous épargnera la création des partitions avec fdisk, la compilation de U-Boot et la création du rootfs avec Buildroot. Ce référence design étant vieux (Linux 3.12), il faudra donc recompiler et mettre à jour une version récente du kernel.

## Manipulations

Pour commencer, il faut télécharger l'image de la carte SD « Linux console » de Teraisc que j'ai trouvé directement sur le site de Teraisc : <https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=836&PartNo=4>

Une fois que le fichier ZIP est téléchargé, il faut l'extraire.

Ensuite, il faut brancher la carte SD sur laquelle on souhaite mettre l'image « Linux console ». Il faut trouver quel device correspond à la carte avec l'aide de la commande « `sudo fdisk -l` ». Comme par exemple `/dev/sdd`.

Une fois que ceci est fait, il faut copier l'image binaire précédemment extrait du fichier ZIP sur le périphérique préalablement trouvé (carte SD). Il est possible de faire ceci avec la commande « `dd` ». Voici par exemple la commande que j'ai effectuée :

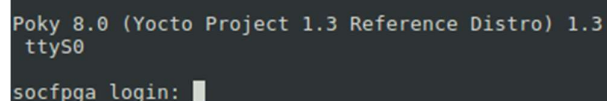
**`dd bs=4M if=DE1_SoC_SD.img of=/dev/sdd conv=fsync`**

## Vérification

Maintenant que la carte SD contient l'image « Linux console » de Teraisc, nous pouvons vérifier que la carte DE1-SoC démarre correctement. Pour ce faire :

1. Insérer la carte SD dans le support de la carte DE1
2. Brancher une câble USB entre la board et l'ordinateur
3. Utiliser la commande « `picocom` » (baudrate de 115200) sur le device USB afin d'observer les logs. Exemple : **`sudo picocom -b 115200 /dev/ttyUSB0`**
4. Allumer la carte DE1-SoC

On peut observer que la carte démarre bien et nous propose de se logger :



```
Poky 8.0 (Yocto Project 1.3 Reference Distro) 1.3
ttyS0
socfpga login: █
```

Figure 0-2 : affichage du login

On est donc sûr que le démarrage s'effectue correctement.

## Compilation de Linux

Le but de cette étape est de récupérer linux mainline, de le compiler et d'utiliser la nouvelle image linux sur la carte SD.

### Préambule

Depuis 2018, les FPGA-SoC sont supporté dans le linux mainline, contrairement à avant où Altera maintenait sa propre version pour ses FPGA-SoC.

Il est possible de trouver les sources sur Github : <https://github.com/torvalds/linux/tree/master>.

Il existe une configuration du noyau pour tous les SoC FPGA. Le kernel n'est donc pas dépendant de la board utilisée ou de la famille du SoC (ceci est géré dans le device tree).

### Manipulations

Il faut commencer par récupérer linux mainline en faisant un clone du repo git ou en téléchargeant directement le repo.

Une fois dans le dossier de linux, il faut configurer et compiler le noyau pour ARM. Pour ce faire, une toolchain ARM est nécessaire. Dans mon cas, elle se trouve sur la machine dans /opt.

Pour configurer le noyau, il faut commencer par exécuter cette commande « make ARCH=arm CROSS\_COMPILE=<TOOLCHAIN\_DIR>/bin/arm-linux-gnueabi- socfpga\_defconfig ». Pour ma part, voici la commande utilisé :

**make ARCH=arm CROSS\_COMPILE=/opt/toolchain/gcc-linaro-arm-linux-gnueabi-4.7-2013.04-20130415\_linux/bin/arm-linux-gnueabi- socfpga\_defconfig**

Une fois ceci fait, il est possible de compiler le noyau à l'aide de cette commande « make ARCH=arm CROSS\_COMPILE=/bin/arm-linux-gnueabi- -j8 ». Voici la commande que j'ai exécuté :

**make ARCH=arm CROSS\_COMPILE=/opt/toolchain/gcc-linaro-arm-linux-gnueabi-4.7-2013.04-20130415\_linux/bin/arm-linux-gnueabi- -j8**

Maintenant que le noyau est correctement compilé, un nouveau fichier « zImage » a été généré dans « linux/arch/arm/boot ». Il faut mettre à jour ce fichier dans la carte SD. Personnellement, j'ai simplement glissé le zImage dans la bonne partition :



 socfpga.dtb	17.1 kB	Binary	Mon 06 Jan 2014 04:21:38 PM CET
 zImage	5.1 MB	Binary	Fri 15 May 2020 02:23:07 PM CEST

Figure 0-1 : Mise à jour du fichier zImage

### Vérification

Maintenant que l'image du linux a été mise à jour sur la carte SD, une vérification peut être faite. Cette étape consiste aux mêmes opérations que la vérification du chapitre [précédent](#). Cependant, le résultat ne doit plus être une demande de login mais le boot doit échouer pendant l'initialisation du kernel. Il doit être possible de voir des logs qui doit ressembler à ceci :

```
Starting kernel ...
undefined instruction
pc : [0000b924] lr : [00008044]
sp : 3ff4f300 ip : 00000000 fp : 00000000
r10: 00000000 r9 : 200001d3 r8 : 3ff4ff60
r7 : 3ffba5a4 r6 : 00000000 r5 : 00000000 r4 : 3ff4f330
r3 : 00000000 r2 : 03ff8000 r1 : 56944c23 r0 : 00000000
Flags: nzCv IRQs off FIQs off Mode SVC_32
Resetting CPU ...

resetting ...

U-Boot SPL 2013.01.01 (Nov 04 2013 - 19:51:38)
BOARD : Altera SOCFPGA Cyclone V Board
SDRAM: Initializing MMR registers
SDRAM: Calibrating PHY
SEQ.C: Preparing to start memory calibration
SEQ.C: CALIBRATION PASSED
ALTERA DWMAC: 0

U-Boot 2013.01.01 (Oct 24 2013 - 17:40:22)
```

Figure 0-2 : Echec du boot

## Création du Device Tree

Le but de cette étape est de créer un device tree correspondant à la DE1-SoC afin de démarrer correctement jusqu'au login.

### Préambule

Le device tree décrit les composants présents dans le SoC et sur la carte. Le kernel met à disposition des dtsi à inclure dans votre device tree. Ces dtsi décrivent les principales familles de SoC (Cyclone V, Aria V, Aria 10, Stratix 10, etc.).

Il nous reste qu'à activer et configurer les principaux composants présents sur la DE-1. Un conseil que l'on rencontre souvent lorsque l'on désire porter une carte sur Linux ou U-Boot est de partir d'une carte similaire déjà supportée. Un device tree pour la DE-0 est déjà présent dans les sources du kernel, nous allons l'utiliser comme point de départ. Le device tree doit être compilé.

### Manipulations

La première étape est de trouver le device tree pour la DE-0 afin d'avoir un bon point de départ. Voici la commande que j'ai utilisée afin de trouver rapidement le device tree :

```
reduser@red-eda:~/Desktop/linux$ sudo find . -iname *DE0*.dts
./arch/arm/boot/dts/socfpga_cyclone5_de0_nano_soc.dts
```

Figure 0-1 : Commande de recherche

Ensuite, il faut copier ce device tree sous un nouveau nom afin de créer le device tree pour la DE1 :

```
reduser@red-eda:~/Desktop/linux/arch/arm/boot/dts$ cp socfpga_cyclone5_de0_nano_soc.dts socfpga_cyclone5_de1.dts
```

Figure 0-2 : Commande de copie

Après avoir analysé complètement le device tree, j'ai pu enlever les nœuds gpio1, gpio2, gpio3 et le nœud i2c0. Ensuite, il faut corriger l'adresse de base du bridge lightweight dans le fichier source socfpga.dtsi comme ceci :

```
fpga_bridge0: fpga_bridge@ff200000 {
    compatible = "altr,socfpga-lwips2fpga-bridge";
    reg = <0xff200000 0x200000>;
    resets = <&rst LWIPS2FPGA_RESET>;
    clocks = <&l4_main_clk>;
};
```

Figure 0-3 : Nœud du bridge lightweight

Maintenant que le device est compatible avec la board DE1, il faut compiler le device tree avec le nom donné au nouveau fichier :

```
reduser@red-eda:~/Desktop/linux$ make ARCH=arm CROSS_COMPILE=/opt/toolchain/gcc-linaro-arm-linux-gnueabi-4.7-2013.04-20130415_linux/bin/arm-linux-gnueabi- socfpga_cyclone5_de1.dtb
UPD include/config/kernel.release
DTB arch/arm/boot/dts/socfpga_cyclone5_de1.dtb
```

Figure 0-4 : Compilation du device tree

Une fois le device tree compilé, un fichier « arch/arm/boot/dts/<Nom\_du\_new\_fichier>.dts » est créé. Il faut le copier sur la carte SD dans la partition FAT32 en le renommant « **socfpga.dtb** » afin d'écraser l'ancien device tree.

## Vérification

Maintenant que le device tree a été mise à jour sur la carte SD, une vérification peut être faite. Cette étape consiste aux mêmes opérations que la vérification du premier [chapitre](#). Le résultat devrait être un démarrage correcte de la carte avec un login.

## Test du bridge HPS <-> FPGA

Le but de cette étape consiste à s'assurer que le bridge (lw\_hps2fpga) entre le HPS et la FPGA fonctionne correctement.

### Préambule

Un mapping de la mémoire physique est accessible par le fichier /dev/mem. Il existe un utilitaire nommé devmem2 qui permet de lire et d'écrire dans ce fichier facilement.

Le code source de cet utilitaire est disponible sur Github :

<https://github.com/hackndev/tools/blob/master/devmem2.c>

### Manipulations

Pour commencer, il est recommandé de s'assurer que le portage c'est correctement effectué. Pour ce faire, il faut observer les logs de démarrage du kernel grâce à la commande « dmesg ». Par exemple, les bridge doivent être correctement initialisés. Il doit être possible de voir ces messages :

```
1.540657] usbhid: USB HID core driver
1.544778] fpga manager fpga0: Altera SOCFPGA FPGA Manager registered
1.551859] altera_hps2fpga_bridge ff200000.fpga_bridge: fpga bridge [lw_hps2fpga] registered
1.560550] altera_hps2fpga_bridge ff500000.fpga_bridge: fpga bridge [hps2fpga] registered
```

Figure 0-1 : Dmesg - bridges

Ensuite, il faut charger le bitstream qu'a été fourni à l'aide de Quatus. Une fois le paramétrage correcte et le fichier de bitstream sélectionné, on peut commencer le chargement.

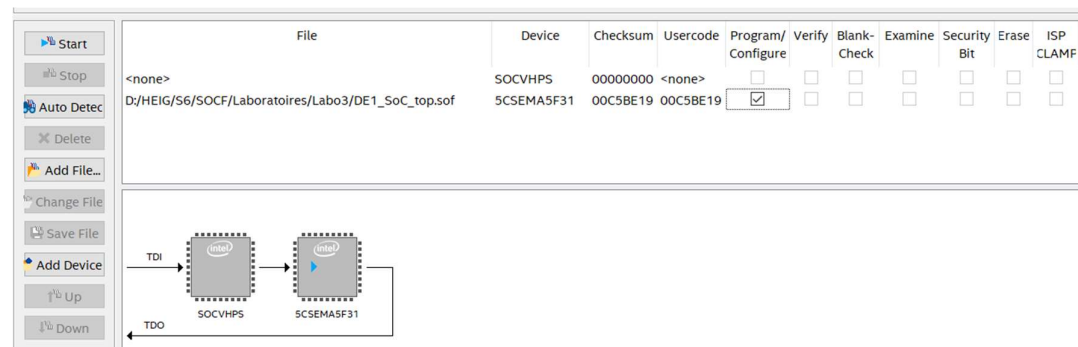


Figure 0-2 : Chargement du bitstream

Une fois le chargement terminé, il doit être possible de voir deux des 7 segments s'allumer comme ci-dessous :



Figure 0-3 : Validation du chargement

On peut maintenant avoir la confirmation que le chargement c'est correctement effectué.

Afin de profiter de l'utilitaire « devmem2 », il faut commencer par le récupérer via le git précédemment indiqué. Ensuite, il faut le cross-compiler afin qu'il soit exécutable sur la board DE1.



Pour ce faire, il est nécessaire d'utiliser la toolchain précédemment évoquée. Voici la commande que j'ai effectué :

```
reduser@reds-eda:~/Desktop/SOCF/Labo3/devmem2$ /opt/toolchain/gcc-linaro-arm-linux-gnueabi-4.7-2013.04-20130415_linux/bin/arm-linux-gnueabi-gcc devmem2.c -o devmem2
reduser@reds-eda:~/Desktop/SOCF/Labo3/devmem2$ ls
devmem2 devmem2.c
```

Figure 0-4 : Cross-compilation

Maintenant que l'exécutable est prêt, il faut le placer dans le rootfs de la carte SD afin d'y avoir accès dans la DE1. Par exemple, on peut le mettre dans le dossier « /home/root/ ». Comme ceci :

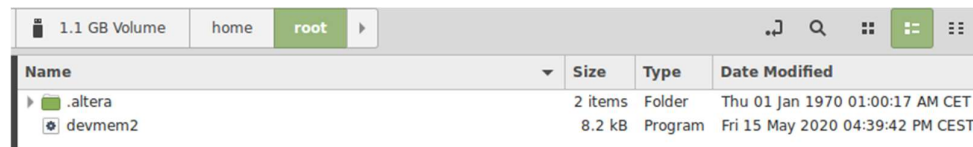


Figure 0-5 : devmem2 sur la carte SD

On peut voir que l'exécutable a été placé dans le dossier « /home/root/ ».

## Vérification

Afin de vérifier le bon fonctionnement du bridge avec le nouvel utilitaire, on peut placer la carte SD, avec l'exécutable, dans la DE1 et la démarrer. Il faut ensuite se logger avec « root ». Il est maintenant possible de tester l'utilitaire afin de lire l'état des boutons poussoirs et des switches. Les boutons sont mappés sur le lwjps2fpga bridge à l'offset 0x30 et les switches à l'offset 0x20.

J'ai commencé par lire l'état des Keys en cliquant sur les 4 boutons :

```
root@socfpga:~# ./devmem2 0xff200030
/dev/mem opened.
Memory mapped at address 0xb6f7a000.
Value at address 0xFF200030 (0xb6f7a030): 0xF
root@socfpga:~#
```

Figure 0-6 : Lecture des keys

On peut voir que la valeur lue est bien 0xF.

Ensuite, j'ai lu l'état des switches avec un switch sur deux activés :

```
root@socfpga:~# ./devmem2 0xff200020
/dev/mem opened.
Memory mapped at address 0xb6f6c000.
Value at address 0xFF200020 (0xb6f6c020): 0x155
root@socfpga:~#
```

Figure 0-7 : Lecture des switches

On peut voir que la lecture c'est correctement effectuée.

## Annexes

Voici la liste dans l'ordre des annexes :

1. socfpga\_cyclone5\_de1.dts
2. socfpga\_cyclone5.dtsi
3. socfpga.dtsi
4. kernel.log

## Conclusion

### Difficultés rencontrées

- Pendant ce laboratoire, j'ai eu des problèmes avec ma carte SD. Elle est passée en mode « read only » après plusieurs changements de fichier. Pour résoudre le problème, j'ai dû recopier l'image binaire sur la carte SD.

### Compétences acquises

- Amélioration de compréhension sur le fonctionnement global d'un système HPS FPGA avec Linux et les devices tree.

### Résultats obtenus

J'ai réussi à mettre en place toutes les étapes qui m'étaient demandées dans ce laboratoire.

Date : 29.05.20

Nom de l'étudiant : Spinelli Isaia