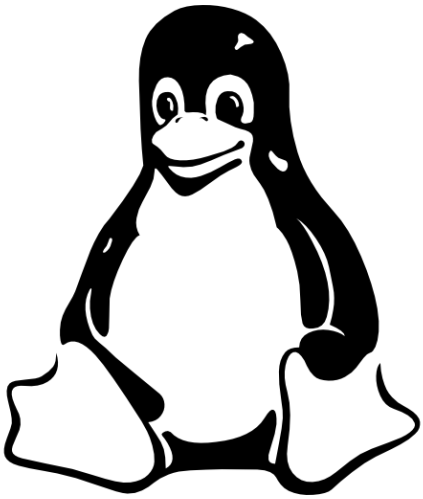
Mise en œuvre d'un Linux Embarqué sur processeur softcore Microblaze

Mégane BERTH

Emilie SOUILLA



Mélanie LUCIAN

Antoine BODIN

Encadrant

Patrice KADIONIK

Bordeaux INP
ENSEIRB
MATMECA

Projet du semestre 9 – Option Systèmes embarqués – Promo 2019

Sommaire

Remerciements

Nous tenons à remercier notre encadrant, également responsable de l'option Systèmes Embarqués, M. Patrice Kadionik, pour son aide lors des séances de projet et pour son expertise dans le domaine.

Merci également à Mme. Suzanne Medina pour ses sages conseils qui nous ont permis d'améliorer la présentation orale de notre projet.

Présentation

Lors du semestre 9 option Systèmes Embarqués, nous avons l'opportunité de choisir un projet commun, afin d'approfondir nos connaissances dans notre spécialité. Nous avons choisi le projet ayant pour but la mise en œuvre d'un Linux embarqué sur un processeur softcore Microblaze, ce qui nous permet d'appliquer les notions de Temps Réel, de logiciels libres pour l'embarqué et de noyau Linux.

1 - Introduction

1 - a - Objectifs du projet

D'abord utilisés dans le domaine militaire, les systèmes embarqués ont su trouver une place dans la vie civile. En effet, à l'heure actuelle, les systèmes embarqués sont présents tout autour de nous, que ce soit dans notre vie quotidienne (MP3, micro-onde, voiture, carte de paiement...) ou professionnelle (imprimante, tablettes, machine à café…). Tous ces systèmes nécessitent une certaine portabilité et donc une miniaturisation de la technologie, mais également un prix raisonnable pour qu'ils soient accessibles au grand public.

Il existe évidemment de nombreux systèmes d'exploitation, mais Linux est l'OS ayant le mieux réussi à s'implanter sur le marché. Ceci s'explique par sa flexibilité, sa robustesse ou encore toute la communauté de développement présente autour du Linux. Il existe plus de 300 distributions différentes de GNU/Linux, mais souvent, les particuliers veulent utiliser et créer leur propre Linux, afin de satisfaire leur curiosité, pour participer au développement d'un projet open source, souvent à des fins pédagogiques ou encore pour adapter la distribution à des besoins spécifiques. Créer sa propre distribution est également le meilleur moyen d'utiliser une distribution sur mesure, dont la taille est parfaitement adaptée aux besoins et donc obtenir la plus petite distribution possible.

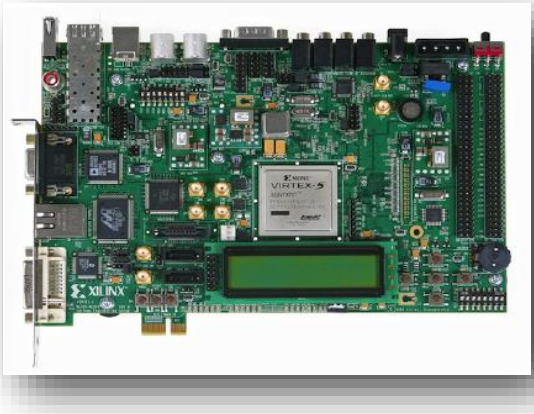
Nous créons donc une distribution à la main, à des fins pédagogiques. Il est tout de même important de savoir qu'il est possible d'utiliser des outils clé en main, comme *YOCTO* ou *Buildroot* qui permettent de créer un Linux très facilement et rapidement. L'outil *Petalinux* par exemple, permet de personnaliser, créer et déployer des Linux embarqués sur les systèmes *Xilinx*.

1 - b - Informations sur les cartes

Deux cartes électroniques sont utilisées pendant le projet. La première carte de développement *Xilinx ML507* nous permet de découvrir le sujet et les notions qui y sont liées. Un TP est suivi afin de mettre une première fois un Linux embarqué sur une carte électronique. Lorsqu'il faut changer de carte afin de pouvoir utiliser des versions plus récentes du noyau Linux, c'est la carte *Nexys4* DDR qui est choisie. La carte *Zedboard* est également une option envisagée, cependant, sa RAM étant dynamique, elle n'est pas accessible directement.

1 - b - 1 - ML507 – Virtex 5

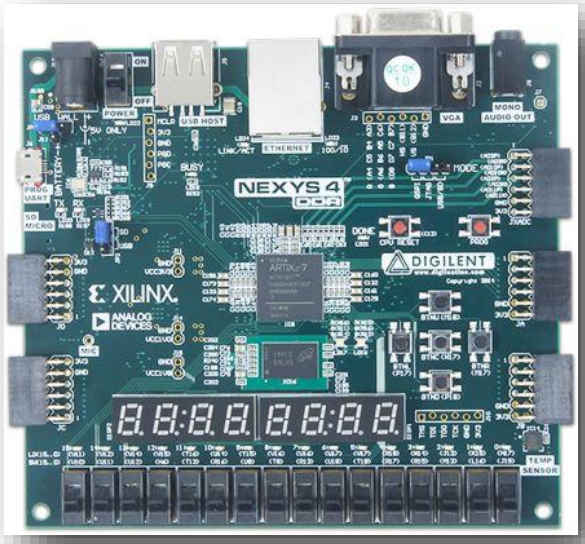
La carte ML507 est une carte de développement utilisant le circuit FPGA *Virtex-5 XC5VFX70T*. Ses caractéristiques principales sont les suivantes :



- Circuit FPGA Xilinx Virtex-5 : XC5VFX70T-1FFG1136.
- 1 Mo de mémoire SRAM.
- 256 Mo de mémoire DDR2 SODIMM.
- 32 Mo de mémoire Flash.
- 2 Mo de mémoire SPI.
- 8 Ko de mémoire I2C.
- 8 interrupteurs - 8 leds.
- 8 boutons poussoirs.
- Afficheur LCD 2x16 caractères.
- Vidéo in (VGA), vidéo output (DVI).
- Interface Ethernet 10/100/1000 Mb/s.
- Liaison série RS232 - JTAG, BDM.
- Connecteur PS/2 pour clavier et souris.
- Entrée/sortie audio AC97.
- Interfaces USB host et device.
- 2 mémoires Flash de 32 Mo Xilinx XCF32P pour la sauvegarde de la configuration.
- 1 système Xilinx System ACE CompactFlash.
- Oscillateur à 100 MHz.

1 - b - 2 - Nexys4 – Artix 7

La carte Nexys4DDR est une carte de développement utilisant le circuit FPGA *Artix-7*. Ses caractéristiques principales sont les suivantes :



- Circuit FPGA Xilinx Artix-7 : XC7A100T-1CSG324C
- Vitesses d'horloge interne supérieures à 450 MHz
- Convertisseur analogique-numérique sur puce (XADC)
- 16 commutateurs utilisateur
- Pont USB-UART
- Sortie VGA 12 bits
- Accéléromètre triaxial
- DDR2 128 MiB
- Pmod pour des signaux XADC
- 16 LEDs utilisateur - Deux LEDs tricolores
- Sortie audio PWM
- Capteur de température
- Mémoire Flash série
- Port USB-JTAG Digilent pour la programmation FPGA et les communications
- Deux affichages à 7 segments, 4 chiffres
- Connecteur de carte Micro SD
- Microphone PDM
- Ethernet PHY 10/100
- Quatre ports Pmod
- Hôte USB HID pour les souris, les claviers et les clés USB

2 - Théorie du Linux embarqué

Un système d'exploitation, ou Operating System (OS), pilote les dispositifs matériels à partir des instructions données par un utilisateur ou d'autres logiciels. Il fournit un environnement graphique à l'utilisateur et lui permet d'installer et utiliser des logiciels. Ces logiciels sont donc écrits pour un système d'exploitation spécifique, ainsi, ils utilisent ses bibliothèques et n'ont pas besoin de prendre en compte le matériel. L'OS fait le lien entre l'ordinateur et les périphériques, tels que le clavier ou l'écran. Il permet aussi de faciliter la gestion de la mémoire. Dans certains cas, le système d'exploitation peut permettre la gestion des différents programmes et utilisateurs afin qu'ils n'interfèrent pas les uns avec les autres, afin de ne pas mettre en péril la sécurité du système. Les systèmes d'exploitation les plus utilisés sont *Windows, MacOS* et *Linux*.

Dans notre cas, c'est un système d'exploitation Linux qui est implémenté sur une carte FPGA. Il s'agit donc d'un système embarqué Linux. Celui-ci est composé du matériel, du noyau Linux et d'un espace utilisateur.

Trois parties principales sont nécessaires pour créer un Linux embarqué (Voir figure ci-contre) :

- La partie matérielle ou hardware
- Le noyau Linux
- L'espace utilisateur

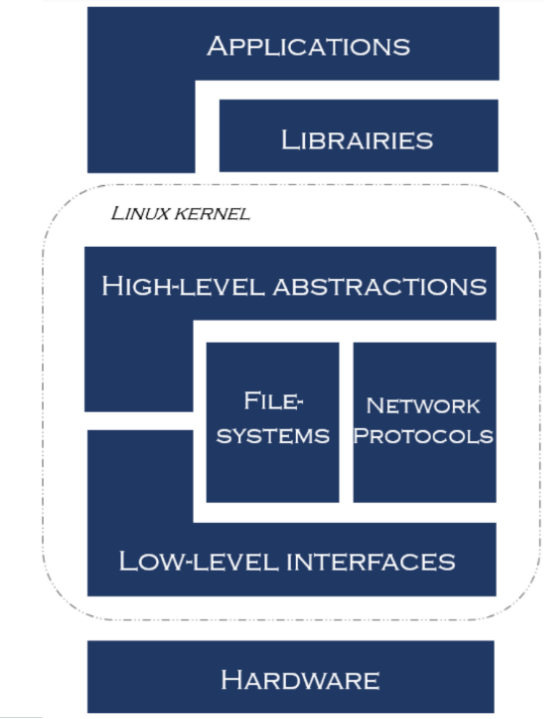


Figure 1 : Organisation d'un système Linux embarqué

2 - a - Le matériel

Dans le cas des systèmes embarqués, le matériel est une contrainte importante. En effet, dans de nombreux domaines, la taille physique de l'objet est un enjeu primordial. Ainsi, la place dédiée au matériel électronique est réduite et cela entraîne de nouvelles contraintes, comme l'autonomie ou la mémoire disponible.

De plus, contrairement au système d'exploitation Linux, le matériel n'est pas gratuit, il est donc possible d'avoir des versions de Linux ayant été développées pour des cartes plus récentes que celles utilisées pour un projet. La compatibilité entre le matériel et le système d'exploitation n'est donc pas toujours assurée, c'est ce qu'on appelle l'obsolescence programmée du matériel.

2 - b - Le noyau Linux

2 - b - 1 - Présentation des blocs

Le noyau Linux, ou *kernel* Linux, est un fichier exécutable permettant d'assurer les fonctions principales du système, comme la gestion des tâches (ordonnancement ou *scheduling*), la gestion de la mémoire, le pilotage des périphériques matériels et virtuels. Le noyau est physiquement représenté par un fichier dans le répertoire */boot*.

Le noyau est une des parties les plus importantes du système d'exploitation, car il gère les ressources de l'ordinateur et permet aux matériels et logiciels de communiquer entre eux.

On retrouve différents niveaux d'abstractions dans un noyau Linux. Le premier contient des interfaces bas niveaux, qui sont spécifiques au matériel. Elles permettent au noyau de fonctionner et de contrôler les ressources matérielles en utilisant des API indépendantes du matériel. Le deuxième niveau d'abstraction est commun à tout système UNIX. Il permet d'utiliser les ressources du système, comme les fichiers, les sockets ou encore les signaux. Enfin, l'élément central du noyau, est le système de fichiers root (ou *rootfs*). Ce système de fichiers est très important, puisqu'il permet l'interprétation de données structurées, provenant de certains périphériques. La structuration des données est effectuée grâce à la mise en place de protocoles de communication entre les fichiers.

2 - b - 2 - Le système de fichier Root

Le système de fichier root, constitue la base des *files system*. Le *rootfs* possède un squelette identique dans chaque système embarqué et chaque dossier présent dans le *rootfs* a un rôle très précis. Les règles pour construire un *rootfs* sont contenues dans le *Filesystem Hierarchy Standard (FHS)*. Les répertoires de la racine sont :

- *bin* : commandes utilisateur binaires essentielles
- *boot :* fichiers statiques utilisés par le bootloader
- *dev : devices* et autre fichiers spéciaux
- *etc :* système de fichier de configuration (inclus les fichiers de lancement)
- *home :* répertoires utilisateurs
- *lib :* librairies essentielles et modules du noyau
- *mnt :* le point d'entrée pour monter le *file system*
- *opt :* logiciels complémentaires
- *proc : file system* virtuel pour le noyau
- *root :* répertoire de base de l'utilisateur racine
- *sbin :* fichiers binaires d'administration essentiels
- *tmp :* fichiers temporaires
- *usr :* hiérarchie secondaire contenant la plupart des applications et documents utiles
- *var :* données variables stockées

2 - b - 3 – Le squelette du noyau

Dans notre environnement de travail, l'ensemble des composants du noyau se trouve dans un répertoire de la forme suivante :

```
$ tar xjf $HOME/Téléchargement/linux-2.6.33.2.tar.bz2
$ cd linux-2.6.33.2
$ ls
arch      crypto         fs       Kbuild       Makefile REPORTING-BUGS sound
block     Documentation  include  kernel       mm       samples        tools
COPYING   drivers        init     lib          net      scripts        usr
CREDITS   firmware       ipc      MAINTAINERS  README   security       virt
```

- *arch :* la couche d'abstraction matérielle (ou HAL pour Hardware Abstraction Layer) virtualise le matériel de la plateforme, de sorte à ce que les différents pilotes puissent être facilement portés sur n'importe quel matériel.
- *mm :* un gestionnaire de mémoire qui contrôle les accès aux ressources mémoires matérielles.
- *kernel :* un ordonnanceur qui fournit au Linux la capacité d'exécuter plusieurs tâches en parallèles.
- *fs :* un système de fichier.
- *drivers :* sous-système IO qui fournit une interface aux périphériques intégrés.
- *net :* sous-systèmes de réseau qui permettent au Linux de supporter plusieurs protocoles réseau.
- *ipc :* (ou Inter Processus Communication) comme les pipes, les sockets, les mémoires partagées ou les sémaphores.

L'ensemble de ces dossiers rassemble les différents blocs présentés dans la partie précédente. On retrouve le système de fichiers, ainsi que les protocoles de communication internes et les outils nécessaires à la mise en place des étages d'abstraction. Le noyau requiert une partie de la mémoire vive physique. Cet espace mémoire est appelé l'espace noyau. La partie restante de la mémoire vive est appelée l'espace utilisateur.

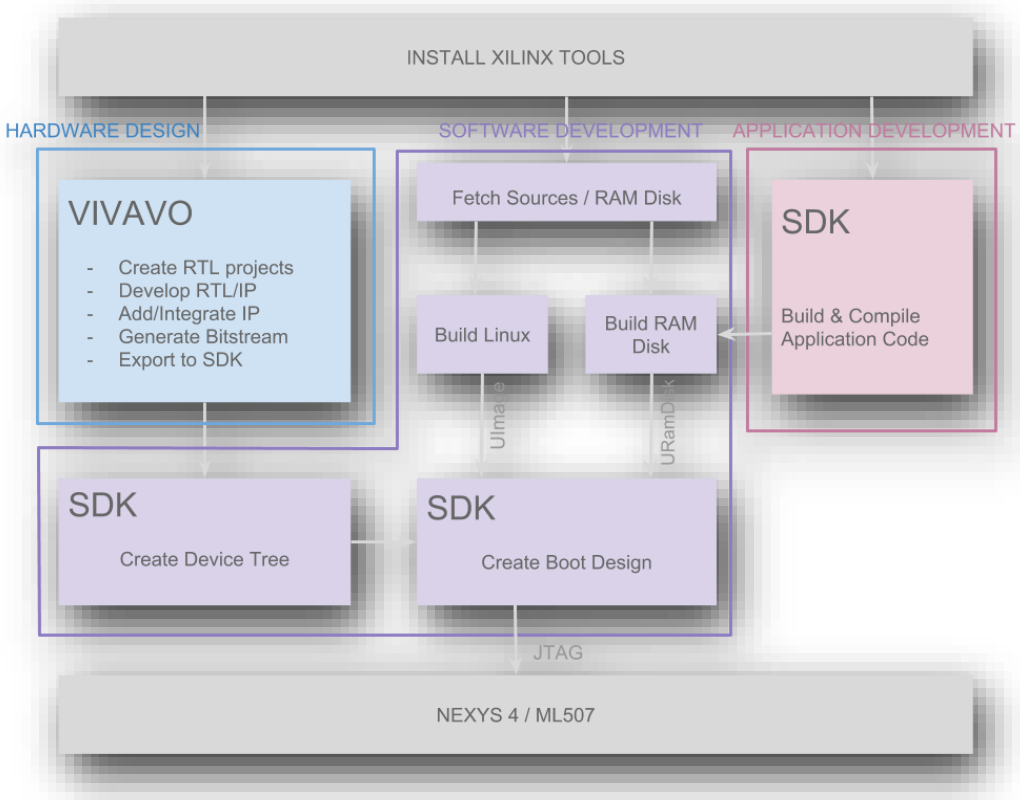
Un noyau étant un composant logiciel très complexe, nous récupérons, pour le projet, les archives de noyau que nous utilisons sur le GitHub de Xilinx.

2 - c - Application et librairies

Une fois le k*ernel* relié à l'espace utilisateur, notamment grâce à la *GNU C Library*, il est possible d'utiliser la mémoire pour stocker des applications et des librairies. Généralement, les librairies sont liées dynamiquement aux applications, ce qui permet de les stocker une seule fois et d'être utilisées par les applications lorsqu'elles sont lancées. C'est le cas pour la librairie C, qui est stockée dans le système de fichier et qui n'est chargée qu'une seule fois sur la RAM. Cette copie est partagée par toutes les applications qui l'utilise.

3 - Mise en pratique

Pour mettre en œuvre notre propre Linux, nous avons décomposé le travail en différentes parties. La première consiste en la création d'un design de référence qui permet à la carte de supporter un Linux de base. En effet, il permet de décrire la configuration de la carte afin que le kernel y ait accès. Nous enrichirons dans un deuxième temps ce design avec des périphériques proposés par la carte FPGA pour pouvoir développer des applications intéressantes. La suite du travail est la construction du noyau qui comporte son système de fichier et la génération d'un *RAMdisk*. Enfin, la dernière partie est l'ajout d'une application dans le *RAMdisk* ce qui nécessite sa re-génération.



3 - a - Design de référence

3 - a - 1 - Création du design de référence avec Vivado

Pour générer le design de référence, nous nous aidons des tutoriels proposés par Digilent, qui fournit également la carte *Nexys4* sur laquelle travaillons. Les grandes lignes sont cependant expliquées dans le paragraphe qui suit.

Nous utilisons d'abord l'outil Vivado afin de créer le design de référence et créons un *RTL Project*. Vivado possède un outil *Board* dans lequel sont déjà enregistrées les cartes les plus utilisées. La *Nexys4* est contenue dans ce *Board*.

Pour construire le design de référence, nous utilisons un processeur Microblaze, cœur de processeur softcore RISC 32 bits de la société *Xilinx*, optimisé pour une implémentation dans un circuit FPGA. Microblaze s'adapte à l'exécution de code *Bare Metal*. Il permet le traitement en temps réel déterministe sur un RTOS et il est compatible avec le Linux embarqué. Son architecture fait de MicroBlaze un processeur configurable. Il est alors possible d'inclure par exemple, des périphériques standards, tels qu'un timer, une liaison UART, une interface Ethernet ou JTAG. Pour un processeur softcore MicroBlaze, il convient d'utiliser la plateforme de développement Xilinx XPS, ainsi que la chaîne d'outils GNU. Nous utilisons les paramètres suivants pour que Microblaze soit compatible avec Linux :

- Local Memory : 32 kB
- Local Memory ECC : None
- Cache configuration : 16 kB
- Debug Module : Debug Only
- Peripheral AXI Port : Enabled
- Clock Connection : New Clocking Wizard (100 MHz)

Nous utilisons ensuite *Block Automation,* qui permet de générer automatiquement les blocs nécessaires à la marche de MicroBlaze. Pour configurer la carte, afin qu'elle reçoive un Linux embarqué, le design de référence doit contenir au minimum une MMU, deux zones de mémoire protégées, deux *timers*, un module UART, un module d'interruption sur lequel seront reliés les *timers* et l'UART. On peut également ajouter des périphériques tels qu'Ethernet, ceux-ci devront aussi être connectés au module d'interruption.

Une fois que tous les modules sont ajoutés, le design doit être validé par Vivado. Enfin, le *HDL System Wrapper* est créé. Le fichier bitstream est généré, ce qui constitue une base du Linux puisqu'il est utilisé dans le noyau. Ceci correspond à la création du hardware.

3 - a - 2 - Création du Device Tree et vérification du design de référence

Le design de référence étant créé, il est exporté sur SDK (Software Development Kit). Il est maintenant possible de développer la partie Software complètement indépendamment du hardware. Le *Device Tree* est alors généré. Celui-ci décrit le système matériel, afin que le kernel puisse se configurer lors du démarrage de Linux. Il est nécessaire de vérifier qu'il est fonctionnel. On dit que l'on teste le design de référence en *bare metal,* c'est-à-dire, effectuer le test d'une application sans OS. Nous choisissons alors la célèbre application « Hello world ! ».

3 - b - Génération du Linux

3 - b - 1 – Construction du système de fichier root

Le système de fichier *root* n'a pas été construit manuellement, car c'est *Busybox* qui a été utilisé. Cet outil permet d'assembler dans un exécutable unique, toutes les commandes, ce qui limite la taille de l'exécutable final. Pour l'installation de *Busybox*, la même configuration que celle du noyau Linux est utilisée. Pour la compilation, l'archive générée doit être extraite. Ce système de fichier *root* sert ensuite à construire un fichier CPIO compressé, qui servira ici de *RAM disk inital* puis de *RAM disk* final.

Généralement, le système de fichier est placé dans un disque dur ou une mémoire non volatile (comme par exemple, une mémoire flash sous forme d'une carte SD). Si ce système de fichier est stocké dans une *RAM,* il est alors appelé un *RAM Disk*. Dans notre cas, on utilise un *RAM Disk Initial* qui sert dans un premier temps à initialiser le noyau, puis devient ensuite lui-même le système de fichier final et constitue donc également le *RAM disk* final. Une fois ce dernier généré, il est copié dans le répertoire Linux qui contient l'ensemble de la distribution. La partie responsable de l'initialisation du noyau est contenue dans le répertoire */init/* du *rootfs*.

3 - b - 2 – Génération du noyau

Une fois la synthèse du design de référence effectuée, il faut :

- Récupérer le fichier *.bit* de programmation du circuit FPGA de la carte cible ML507.
- Récupérer le fichier *.dts* qui décrit le système SoPC (périphériques, adresses de base, interruption...) que l'on copie dans *arch/microblaze/boot/dts*.
- Récupérer le fichier *.cpio* et le copier dans la distribution
- Utiliser le bon compilateur croisé
- Compiler le noyau
- Transférer le noyau sur la carte et observer les traces du boot

Différentes versions du noyau Linux sont testées sur la carte ML507. D'abord la version 2.6.32, puis la version 3.3 et enfin la version 4.14. Lors de ces tests, la compatibilité avec le module *tic timer,* puis avec le module Ethernet est perdue. C'est pourquoi nous utilisons la carte Nexys4DDR pour la suite du projet. Plus la version du noyau est récente, moins les périphériques de la carte sont supportés, d'où le passage à une carte plus récente.

Pour la suite de ce rapport, les scripts et paramètres qui seront donnés, correspondent à la version 4.14 du noyau Linux, implémenté sur la carte Nexys4DDR.

3 - b - 2 – Compilation du noyau

Avant de pouvoir compiler le noyau, il doit être récupéré et extrait de l'archive du noyau, dans notre cas, pris sur Github. Une fois la version souhaitée connue, l'archive sous le format *gzip* peut être récupéré, puis décompressé.
Pour la suite, un compilateur croisé doit être utilisé afin que la compilation soit compatible avec la carte cible. Les variables d'environnement correspondant à l'architecture de la carte cible et au préfixe du compilateur croisé, doivent être spécifiées.

Le script *./go* contenant les informations nécessaires à la compilation croisée ainsi que le nom du fichier contenant le noyau Linux, peut alors être exécuté.

```
make ARCH=microblaze
CROSS_COMPILE=microblazeel-xilinx-linux-gnu-
simpleImage.xilinx -j 4
```

Figure 2 : Script ./go

3 - b - 3 - Téléchargement et lancement du noyau

Une fois tous les éléments réunis dans le répertoire Linux, le noyau Linux et son *RAM disk* sont téléchargés par le JTAG, puis le noyau Linux est lancé grâce à la commande *./gomb* qui utilise la commande *xmd,* soit l'outil *Xilinx Microprocesseur Debugger*.

```
xmd -opt mb.opt
\rm _impactbatch.log 2>/dev/null
```

Figure 3 : Script ./gomb

Matériel	
Processeur cible	MicroBlaze
Fondeur	Xilinx
Carte Cible	1 - ML507 2 2 - Nexys 4
Logiciel	
Version Linux pour MicroBlaze	1 - Linux 2.6.32 2 - Linux 3.3 3 - Linux 4.14
Version du compilateur croisé	microblaze-xilinx-linux-gnu-

3 - c - Ajout de périphériques

Une fois que le Linux embarqué est créé, il est intéressant d'inclure des périphériques à ce système. Les périphériques sont disponibles sur la carte, il faut seulement indiquer au kernel comment y avoir accès. Pour cela, c'est le design de référence qu'il faut modifier.

Notre objectif est d'ajouter le périphérique Ethernet et le capteur de température ADT7420 présent sur la carte.

3 - c - 1 - Ajout d'Ethernet

Pour ajouter le périphérique Ethernet et donc modifier le design de référence, l'outil Vivado est utilisé et plus particulièrement l'intégrateur IP. Comme expliqué précédemment, l'outil Vivado contient la description des cartes. Il est donc aisé de choisir dans l'intégrateur IP le bloc Ethernet et l'ajouter au design de référence.

Nous suivons le tutoriel proposé par Digilent "*Getting Started with Microblaze Servers*". Globalement, pour que le périphérique Ethernet fonctionne, il a besoin des cinq blocs suivants : Memory Interface Generator, Ethernet PHY MII to reduced MII, AXI Uartlite, AXI EthernetLite, AXI Timer. Il faut savoir que les blocs AXI pour Advanced eXtensible Interface sont ajoutés pour connecter le microprocesseur aux périphériques. De plus, une nouvelle horloge doit aussi être générée par le *Clock Wizard* et liée au bloc Ethernet PHY.

L'option *Run Block Automation* permet de relier tous ces blocs automatiquement. Il est surtout nécessaire de vérifier si le bloc Ethernet est lié au bloc interruption. Une fois ces étapes effectuées, il est nécessaire de générer à nouveau le fichier *bitstream* et d'exporter le hardware sur SDK.

3 - c - 2 - Ajout du capteur de température

1 – Le capteur ADT7420

Afin de tester la distribution implémentée sur *Nexys4*, nous voulons utiliser le capteur ADT7420 afin de récupérer la température ambiante. Ce capteur :

- Peut mesurer une température comprise entre -40 et +150°C.
- Doit être alimenté par un voltage compris entre 2,7 et 5,5V.
- Utilise le bus de communication I2C.

De la même manière que précédemment, ajouter le périphérique capteur de température nécessite un changement du design de référence. L'*IP integrator* facilite grandement cette modification. Une fois le bloc *temperature_sensor* ajouté, il doit être relié au bloc interruption.

2 – Le bus I2C

Le bus I2C (Inter-integrated circuit) a été développé par Phillips en 1982. Il permet de connecter plusieurs équipements (maîtres et esclaves) entre eux. Deux lignes sont nécessaires à la connexion : la ligne SDA (Serial Data Line), pour l'échange des données et la ligne SCL (Serial Clock Line), pour l'horloge fournie par le maître. Le schéma de communication entre les composants est le suivant :

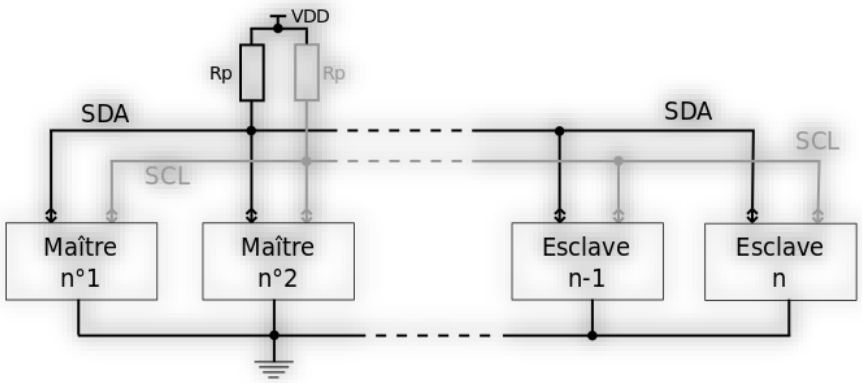


Figure 4: Schéma de communication entre plusieurs équipements

Deux cas sont observables : le maître est émetteur et l'esclave récepteur ou bien, le maître est récepteur et l'esclave émetteur.

4 - Utilisation du Linux

4 - a – Démarrage du système Linux

Afin de démarrer un système Linux, le noyau doit d'abord être chargé et exécuté. Les périphériques matériels sont alors initialisés et leurs pilotes associés, chargés. Le noyau peut alors monter sa partition principale, c'est-à-dire son système de fichier, qui dispose des éléments nécessaires à la suite du démarrage du système. Ainsi, une fois le noyau Linux chargé, le programme d'initialisation est exécuté. Celui-ci analyse alors le fichier de configuration contenant le chemin d'accès au script de démarrage.

4 - b - Ajout d'une application

Nous souhaitons pouvoir récupérer la température à partir du capteur présent sur la carte Nexys4. Pour cela, nous devons procéder en plusieurs étapes :

- Compilation de l'application avec le compilateur croisé
- Rajout de l'exécutable dans le *RAMdisk*
- Re-génération du *RAMdisk*
- Re-compilation du noyau
- Téléchargement du nouveau noyau via JTAG dans la nexys4

Comme pour la génération du design de référence, la configuration matérielle est réalisée sous Vivado. L'outil Vivado SDK nous sert ensuite à avoir les fichiers de configuration matériel qui serviront à générer le *device tree*. La génération de l'application dans le *RAMdisk* se fait ensuite par compilation avec *microblaze-xilinx-linux-gcc* du code source et la copie de l'exécutable dans le répertoire /bin/ du *RAMdisk*.

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include "libi2c.h"

int main(argc, argv)
     int argc;
     char *argv[];
{
  double temp;

  adt7420_init();

  temp = adt7420_read_temp();

  printf("TEMP=%.2f oC\n", temp);

  adt7420_close();

  exit(0);
}
```

Conclusion

Tout au long de ce projet, nous avons eu l'occasion de comprendre ce qu'est un système d'exploitation au travers de la construction d'un Linux embarqué sur deux FPGA. Pour ce faire, nous avons utilisé les outils propriétaires de *Xilinx* tels que *Vivado* ou *ISE*. De plus, nous nous sommes confrontés aux problèmes liés à l'obsolescence programmée, ce qui nous a poussé à trouver des solutions pour y remédier. C'est pour cela que nous avons utilisé deux cartes FPGA. La mise en route de Linux implique une configuration matérielle et logicielle, que nous avons pu réaliser avec les outils *Vivado* et *SDK* respectivement. Pour rendre le Linux utilisable, il est nécessaire que le kernel ait accès aux périphériques présents sur la carte. Nous avons donc enrichi le design de référence avec le bloc Ethernet et le capteur de température, ce qui nous a par la suite servis à réaliser une application.

L'application que nous avons utilisée a été développée par Patrice Kadionik dans des travaux précédents, il aurait été intéressant de programmer notre propre application. Enfin, pour compléter notre projet, nous pourrions mettre en œuvre l'extension Temps Réel dur *Xenomai* après l'implémentation du Linux embarqué. L'ajout d'un noyau temps réel dur nous permettrait de garantir les temps de latence nécessaires à l'utilisation d'applications dont la gestion du temps est importante. Il faudrait également vérifier la compatibilité entre la version du noyau utilisé et l'extension *Xenomai*.

Références

Bibliographie

[1] Pierre Ficheux, Eric Benard, Linux Embarqué Nouvelle étude de cas 4ème édition, 2012

[2] P.Raghavan, Amol Lad, Sriram Neelakandan, Embedded Linux System Design and Development, 2005

Webographie

[1] Site officiel de Patrice Kadionik
http://kadionik.vvv.enseirb-matmeca.fr/

[2] Site écrit par Wikidot, *"MicroBlaze Linux (General)"*
http://xilinx.wikidot.com/microblaze-linux

[3] Tutoriel écrit par Digilent *"Nexys 4 DDR - Getting Started with Microblaze"*
https://reference.digilentinc.com/learn/programmable-logic/tutorials/nexys-4-ddr-getting-started-with-microblaze/start

[4] Livre écrit par eTutorials.org (2008 - 2018), "*Embedded Linux Systems*"
http://etutorials.org/Linux+systems/embedded+linux+systems/

[5] Datasheet d'Analog Devices du capteur de température
https://www.analog.com/en/products/adt7420.html#product-overview