ENSEIRB-MATMECA

TROISIÈME ANNÉE, OPTION SE

PROJET AVANCÉ

Système FPro : FPGA Protyping

Auteurs :
MOUHAGIR Ayoub
FERIAT Abdessamad
SABBARI Fakhar-eddine
OUANNAS Anas

Professeur :
Patrice KADIONIK

29 Janvier 2020

Table des matières

Table des figures

Résumé

Ce rapport reprend l'ensemble des travaux réalisés dans le cadre du projet avancé système embarqué du semestre 9 à ENSEIRB-MATMECA proposé par M. Patrice Kadionik, responsable de l'option Système Embarqué. L'objectif principal du projet est de mettre en oeuvre une approche codesign pour la construction d'un système SoPC sur une carte Xilinx Nexys 4 DDR appelé FPro (Fun and Professional) jeu de mots pour : FPGA Prototyping, également appelé system-on-chip (SoC) prototyping. Ce système est basé sur l'usage du microcontrôleur MicroBlaze MCS.

On réalise un premier système appelé FPRO vanilla. Il présente les aspects matériels VHDL (MicroBlaze MCS, bridge, sous-système MMIO d'E/S, GPIO, timer et UART) et logiciels (drivers et applications de test). Ce système permet d'intégrer jusqu'à 64 blocs IP ayant au plus 32 registres de contrôle. Ensuite, on met en œuvre un noyau embarqué sur Microblaze (Portage µC/OS-II), ce qui nous permet d'appliquer les notions de Temps Réel et de noyau µC/OS-II.
A la fin, nous obtenons une plateforme Vanilla-MicroBlaze fonctionnelle intégrant le noyau µC/OS-II.

Remerciements

Nous tenons à remercier notre encadrant Mr. Patrice KADIONIK, responsable de l'option Systèmes Embarqués, qui s'est toujours démené pour trouver des solutions pour nos problèmes que ce soient techniques ou logiciels.

Nous tenons à remercier également Mme pour tous ses conseils qui nous ont permis d'améliorer notre soutenance en Anglais.

1 Introduction

Un système embarqué est généralement doté d'un processeur et de périphériques d'entrée et de sortie simples pour effectuer des tâches générales d'interface utilisateur et d'entretien, ainsi que d'accélérateurs matériels spéciaux pour traiter les opérations à forte intensité de calcul. Ces composants peuvent être intégrés dans un seul circuit intégré, communément appelé SoC (System On Chip). Dans cette optique, le projet avancé s9 porte sur le prototypage d'un SoC basé sur le soft-processeur Xilinx Microblaze MCS sur cible FPGA (Carte Nexys 4) et sur le portage du système d'exploitation en temps réel µC/OS2.

2 Présentation du sujet

Notre projet consiste à développer une plateforme SoC FPro intégrée présentant les caractéristiques suivantes :

— Elle définit un protocole de bus synchrone simple et une structure de pilote de périphérique simple. Une fois qu'un composent matériel est développé, il peut être converti en un bloc IP en ajoutant une interface simple et un pilote de périphérique.

— Elle fournit une variété de périphériques d'entrée/sortie et d'interfaces de communication en série couramment utilisées (UART, SPI et I2C).

— Les blocs IP sont développés à partir du langage VHDL et n'utilisent pas de composants propres à un fournisseur, de sorte que les blocs IP et les codes logiciels sont portables et peuvent être réutilisés pour différentes cibles FPGA.

Vu que la technologie FPGA se développe, la conception d'un SoC peut être réalisée dans une carte FPGA. Elle permet en fait de personnaliser le processeur, de sélectionner les périphériques d'E/S nécessaires, de créer une interface d'E/S personnalisée et de développer des accélérateurs matériels spéciaux. Les systèmes embarqués développés sur FPGA offrent une grande flexibilité car le matériel et le logiciel peuvent être personnalisés pour répondre à des besoins spécifiques, ce qui aboutit à une conception *Codesign*.

La plate-forme FPro est composée de :

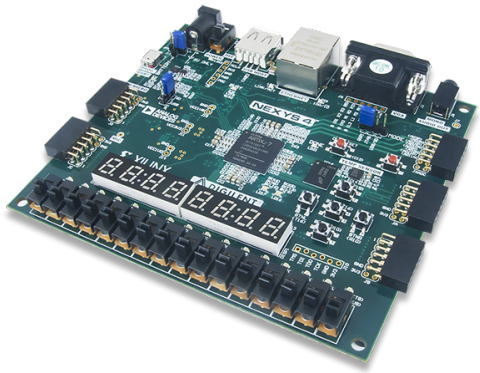

FIGURE 1 – La carte FPGA cible Nexys 4

— Module processeur : un processeur de chemin de données 32 bits avec un espace d'adressage mémoire de 32 bits et un schéma d'entrées/sorties en mémoire pour l'accès aux entrées/sorties.

— Bus FPro : lien entre le processeur et les autres cœurs. Il s'agit d'un bus synchrone.

— MMIO : la mémoire et les registres des périphériques d'entrée/sortie sont mappés sur le même espace d'adressage.

— Schéma logiciel en métal nu : ne contient pas de système d'exploitation. Le processeur démarre directement dans une boucle infinie qui contient des fonctions de test.

3 Système FPro Vanilla

Le système FPro se compose de deux parties, une partie matérielle et une partie logicielle. Le matériel comprend le CPU et la MMIO. Le CPU est un processeur soft-core développé par Xilinx sous le nom de MicroBlaze. La MMIO se compose d'un contrôleur et de cœurs IP pouvant être connectés à des périphériques externes. La programmation logicielle du système utilise le concept appelé Bare Metal. Bare Metal offre des possibilités limités au développeur et nécessite que le programme d'application interagit directement avec le hardware par le biais des pilotes de périphérique. Dans ce système les pilotes sont développés en C++ et représentés par des classes. Dans cette partie, nous allons exposer les étapes suivies lors de l'implémentation du système FPro Vanilla. Après création d'un projet Xilinix Vivado, nous avons importé le design du processeur MicroBlaze à partir du catalogue IP. Nous avons ensuite ajouté au projet les fichiers VHDL disponible sur le site de Mr.CHU. La génération de Bitstream marque la fin de cette partie. La génération du logiciel du système se fait à l'aide de l'outil Xilinx SDK, qui permet à partir des fichiers C/C++ de créer un fichier .elf qui sera utilisé pour la programmation de la carte.

Après que nous avons implémenté le système sur la carte Nexys 4, nous avons procédé à la réalisation d'un protocole de test qui vérifie des le bon fonctionnement des E/S (Timer, GPO, GPI, UART). Ce protocole a confirmé le bon fonctionnement du système. La figure ci-dessous montre le bon fonctionnement des LEDs lors du test du GPO :

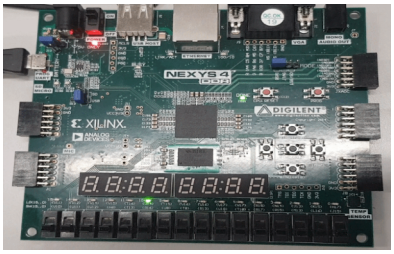

FIGURE 2 – Test du GPO

4 Intégration des périphériques d'entrée/sorties dans le système

Un périphérique d'E/S contient généralement un ensemble de registres de commande, d'état et de données. Dans le cadre du modèle du MMIO, le processeur utilise le même espace d'adressage pour accéder à la mémoire et aux registres des périphériques d'entrée/sortie. Ainsi, les instructions de chargement et de stockage utilisées pour accéder à la mémoire peuvent également être utilisées pour accéder aux périphériques d'entrée/sortie.

Le système FPro se compose de deux sous-systèmes et de nombreux cœurs IP. Afin d'assurer la portabilité, ces coeurs sont fusionnés en un seul espace d'adressage. En d'autres termes, du point de vue du processeur, les deux sous-systèmes, ainsi que tous ses cœurs, sont traités comme un seul module d'E/S. Le FPro bridge et le contrôleur interne du MMIO effectuent le décodage et le multiplexage afin d'accéder à un cœur et à ses registres d'E/S.

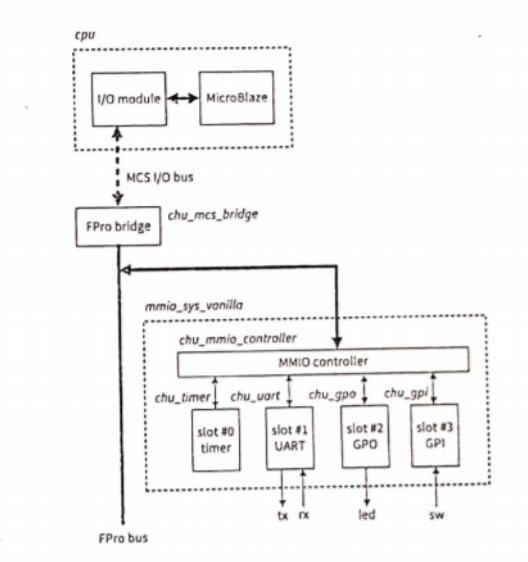

FIGURE 3 – Le système Vanilla FPro

Dans un système bare metal, un registre d'entrée/sortie se voit attribuer une adresse mémoire, ce qui peut être considérée comme la valeur d'un pointeur. La valeur explicite de l'adresse de ce pointeur est connue et doit être utilisée pour accéder au registre. Par exemple, dans le système FPro Vanilla, un noyau GPO est instancié pour les LED de la carte DDR Nexys 4 et connecté au slot 2 du sous-système MMIO. Un noyau GPI est instancié pour les switchs et connecté à l'emplacement 3. Leurs adresses de base sont Oxc0000100 et Oxc0000180, respectivement, avec un offset d'adresse pour les registres de données égal à 0. Dans le système FPro, tous les registres du noyau d'E/S sont traités comme des mots de 32 bits

4.1 Intégration d'un compteur 64bits

L'interface IP du compteur 64 bits utilise trois registres, ses schémas de registres d'E/S sont présentés à la figure ci-dessous.

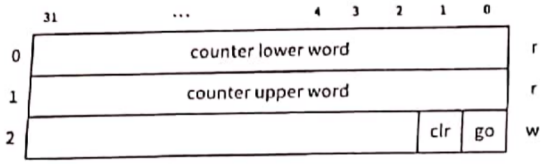

FIGURE 4 – Registres d'E/S du compteur 64 bits

Ce compteur tourne en continu et il peut être mis en pause ou remis à zéro par les signaux de commande. Les registres 0 et 1 du noyau sont reliés aux 32 LSB et 32 MSB du compteur. Les bits 0 et 1 du registre 3 sont connectés aux signaux de validation et de remise à zéro. L'écriture d'un 0 sur le bit 0 met le comptage en pause et l'écriture d'un 1 le reprend. La définition de la classe représentant le compteur 64 bits est indiquée ci-dessous :

```
1  #include "chu_io_rw.h"
2  #include "chu_io_map.h"
3
4  class CmptCore {
5  public:
6
7      enum {
8          COUNTER_LOWER_REG = 0, /**< lower 32 bits of counter */
9          COUNTER_UPPER_REG = 1, /**< upper 32 bits of counter */
10         CTRL_REG = 2           /**< control register */
11     };
12     enum {
13         GO_FIELD = 0x00000001, /**< bit 0 of ctrl_reg; enable bit  */
14         CLR_FIELD = 0x00000002 /**< bit 1 of ctrl_reg; clear bit */
15     };
16
17      CmptCore(uint32_t core_base_addr);
18      ~CmptCore();
19
20          void pause();
21          void go();
22          void clear();
23          uint64_t read_tick();
24          uint64_t read_counter();
25
26 private:
27          uint32_t base_addr;
28          uint32_t ctrl;
29 };
```

La structure de base est similaire à celle du Timer. La première énumération utilise des noms symboliques pour les trois registres de l'offset. Une autre énumération permet de spécifier les masques nécessaires à la récupération des bits d'activation et de réinitialisation. Les méthodes de la classe servent à contrôler le compteur, et à récupérer la valeur courante du compteur. L'adresse de base et la variable *ctrl*, qui maintient une copie des données identique au contenu du registre de contrôle du noyau du compteur, sont déclarées comme des attributs privés de la classe. Le constructeur sauvegarde l'adresse de base et déclenche le compteur. La méthode *pause()* réinitialise le bit de validation pour mettre le comptage en pause. La méthode *resume()* remet à zéro le bit de validation pour reprendre le comptage. La méthode *clear()* permet d'écrire le bit de remise à zéro du registre de contrôle. La méthode *read_ tick()* récupère le contenu de deux registres et les conditionne pour obtenir le nombre de ticks d'horloge écoulés depuis le dernier effacement. En se basant sur les informations de fréquence d'horloge du système, la méthode *read_ time()* traduit les ticks d'horloge écoulés en temps écoulé en microsecondes.

4.2 Intégration d'un port I²C

Dans cette partie, nous allons détailler le développement d'un module I²C. Le bus I2C (Inter Integrated Circuit) a été développé au début des années 80 par Philips semiconductors pour permettre de relier facilement à un microprocesseur les différents circuits d'un téléviseur moderne. Le module I²C développé adopte une configuration à maître unique. Le module de commande I²C est réalisé à partir d'une machine d'états fini et d'une logique de contrôle de sortie spéciale pour gérer le flux de données et l'acquittement. Le description VHDL de ce module se compose de deux parties :

— Une partie qui décrit la machin d'états et la logique des sortie

— Une partie de wrapper qui permet d'adapter le module I²C avec le module MMIO

Le pilote du port I²C contient un ensemble d'instructions permettant de fixer la fréquence de l'horloge, et des opération de lecture et écriture. L'extrait ci-dessous présente la définition de la classe de ce pilote :

```
1  class I2cCore {
2  public:
3      enum {
4          DVSR_REG = 0,
5          WR_REG = 1,
6          RD_REG = 0
7      };
8      enum {
9          I2C_START_CMD = 0x00 << 8,
10         I2C_WR_CMD = 0x01 << 8,
11         I2C_RD_CMD = 0x02 << 8,
12         I2C_STOP_CMD = 0x03 << 8,
13         I2C_RESTART_CMD = 0x04 << 8
14     };
```

```
15      I2cCore(uint32_t core_base_addr);
16      ~I2cCore();
17
18      void set_freq(int freq);
19      int ready();
20      void start();
21      void restart();
22      void stop();
23      int write_byte(uint8_t data);
24      int read_byte(int last);
25      int read_transaction(uint8_t dev, uint8_t *bytes, int num,int restart);
26      int write_transaction(uint8_t dev, uint8_t *bytes, int num, int restart);
27  private:
28      uint32_t base_addr;};
```

La première énumération définit des noms symboliques pour les registres de l'offset. La deuxième énumération permet de spécifier des commandes dans les bits adéquates. Le constructeur *I2cCore()* sauvegarde l'adresse de base et fixe la fréquence scl à 100K Hz. La méthode *set_freq()* fixe la fréquence d'horloge du bus I2C. La méthode *ready()* lit le registre et extrait le bit *Ready*. Les méthodes *start()*, *restart()* et *stop ()* attendent que le contrôleur soit prêt et émettent la commande associée. La méthode *write_ byte()* écrit un octet de données et renvoie son état. La méthode *read_ byte ( )* récupère un octet de données de l'esclave. Le paramètre : *last*, dont la valeur doit être soit 0 soit 1, est utilisé pour indiquer si l'opération de lecture a été la dernière opération effectuée lors d'une transaction.

Vérification à l'aide du capteur ADT7420

Afin de tester le fonctionnement du module I²C, nous avons utilisé le capteur de température ADT7420 intégré dans la carte nexys 4 DDR. La figure ci-dessous montre le bon fonctionnement du module et que nous avons réussi à récupérer la valeur de la température.

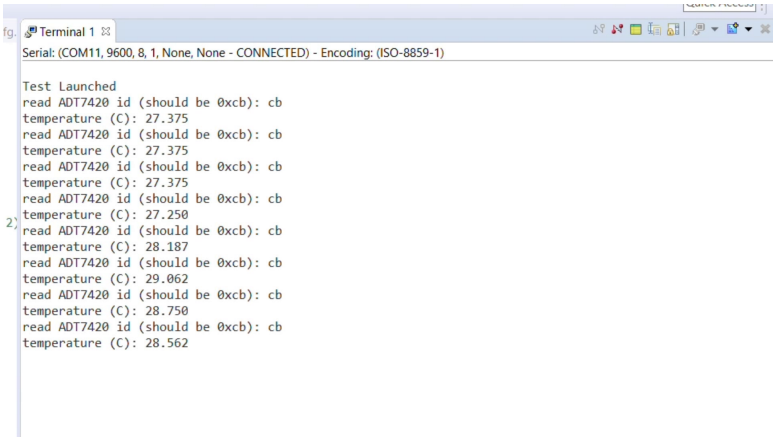

FIGURE 5 – Mesure de la trmpérature par le capteur ADT7420

Vérification à l'aide du PMODTMP3

La deuxième vérification est effectué à laide du PMODTMP3. Ce Pmod permet de mesurer la température en se basant sur le TCN75AVUA. Les données de température mesurées par l'appareil sont formatées en complément à deux et peuvent être configuré pour une résolution de 9 à 12 bits par le biais du registre de configuration. Après modification du design afin de connecter l'interface I^2C avec les broches du PMOD, nous avons réussi à récupérer la valeur mesurée par le capteur. Ci-dessous et le code de la fonction réalisant ce test :

```
void adt7420_check(I2cCore *adt7420_p, GpoCore *led_p) {
   const uint8_t DEV_ADDR = 0x4b;
   uint8_t wbytes[2], bytes[2];
   //int ack;
   uint16_t tmp;
   float tmpC;
   // read adt7420 id register to verify device existence
   // ack = adt7420_p->read_dev_reg_byte(DEV_ADDR, 0x0b, &id);
   wbytes[0] = 0x0b;
   adt7420_p->write_transaction(DEV_ADDR, wbytes, 1, 1);
   adt7420_p->read_transaction(DEV_ADDR, bytes, 1, 0);
   uart.disp("read ADT7420 id (should be 0xcb): ");
   uart.disp(bytes[0], 16);
   uart.disp("\n\r");
   //debug("ADT check ack/id: ", ack, bytes[0]);
   // read 2 bytes
   //ack = adt7420_p->read_dev_reg_bytes(DEV_ADDR, 0x0, bytes, 2);
   wbytes[0] = 0x00;
   adt7420_p->write_transaction(DEV_ADDR, wbytes, 1, 1);
   adt7420_p->read_transaction(DEV_ADDR, bytes, 2, 0);
   // conversion
   tmp = (uint16_t) bytes[0];
   tmp = (tmp << 8) + (uint16_t) bytes[1];
   if (tmp & 0x8000) {
      tmp = tmp >> 3;
      tmpC = (float) ((int) tmp - 8192) / 16;
   } else {
      tmp = tmp >> 3;
      tmpC = (float) tmp / 16;
   }
   uart.disp("temperature (C): ");
   uart.disp(tmpC);
   uart.disp("\n\r");
}
```

5 Design du soft-processeur Xilinx MicroBlaze MCS sur la cible Nexys 4

Pour faire le design du MicroBlaze MCS nous avons suivi le tutoriel mis à disposition par Xilinx. La première étape était de reprendre le design donné par Xilinx et puis implémenter les différentes fonctionnalités à savoir les deux blocs GPIO pour les LEDs et les switches, en plus que le bloc pour axi-timer.
Tout d'abord MicroBlaze MCS design *"**M**icro **C**ontroller **S**ystem"* qui est un système de processeur hautement intégré destiné aux applications de contrôleur. Les données et le programme sont stockés dans une mémoire locale, le débogage est facilité par le module de débogage MicroBlaze (MDM). Un ensemble standard de périphériques est également inclus, offrant des fonctionnalités de base telles que le contrôleur d'interruption, l'UART, les minuteries et les entrées et sorties à usage général. Le noyau du processeur intégré MicroBlaze est un ordinateur à jeu d'instructions réduit (RISC) optimisé pour la mise en œuvre dans les appareils Xilinx. Le design basic de Xilinx contient :

— Le processeur MicroBlaze qui utilise une architecture RISC (Reduced Instruction Set Computer) qui utilise une ISA (Instruction Set Architecture) simple et destinée au système à faible consommation en énergie.

— Mémoire : la mémoire où les données et les programmes sont stockées est implémentée à l'aide du bloc RAM. La taille de la mémoire varie entre 4ko et 128ko. La connexion ou le transfert de données entre la mémoire et le processeur MicroBlaze se fait à l'aide du Bus du mémoire local, du LMB ou à l'aide des cœurs du contrôleur d'interface LMB BRAM.

— Module de Debug : Le cœur MDM (**M**icroBlaze **D**ebug **M**odule) connecte la logique de débogage MicroBlaze au débogueur système Xilinx (XSDB). XSDB peut être utilisé pour télécharger des logiciels, pour définir des points d'arrêt, afficher le registre et le contenu de la mémoire.

— Module I/O : IL constitue les périphériques d'entrée sortie comme le UART, les GPIO (**G**eneral **P**urpuse **I**nput **O**utput), ainsi que les I/O bus qui servent au transfert des données.

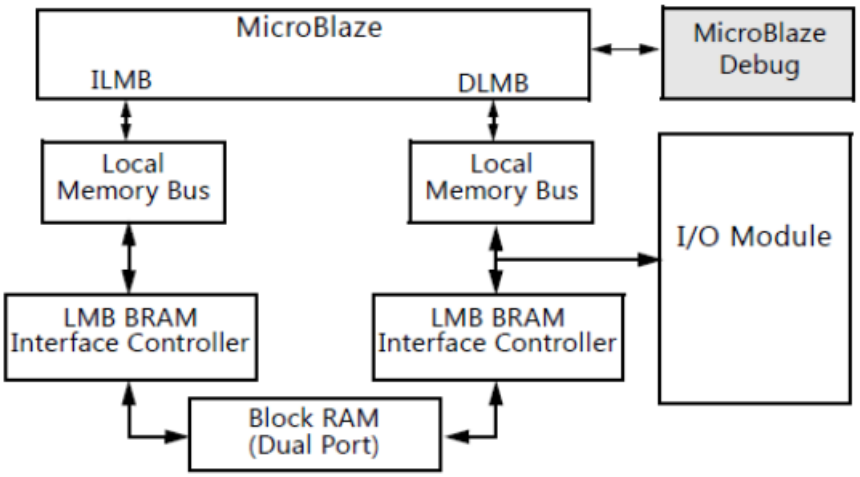

FIGURE 6 – MicroBlaze MCS design (basic)

Ensuite, de nouvelles fonctionnalités (bloc dans le design) peuvent être ajoutées, comme les GPIO responsable au contrôle des LEDs et des switches, ainsi qu'un bloc pour l'axi-timer.

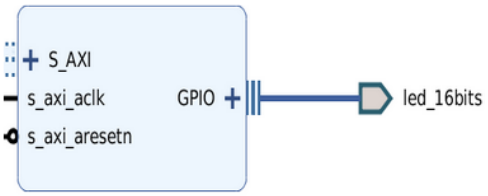

FIGURE 7 – Bloc GPIO pour les LEDs

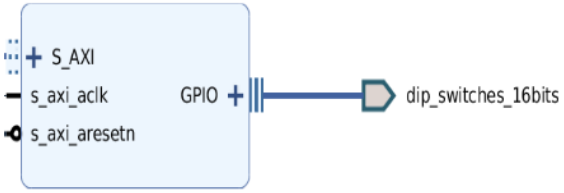

FIGURE 8 – Bloc GPIO pour les switches

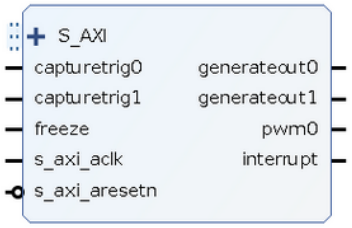

FIGURE 9 – Bloc AXI-timer

Le design complet est celui qui est fourni par Xilinx sur le quel on rajoute nos blocs.

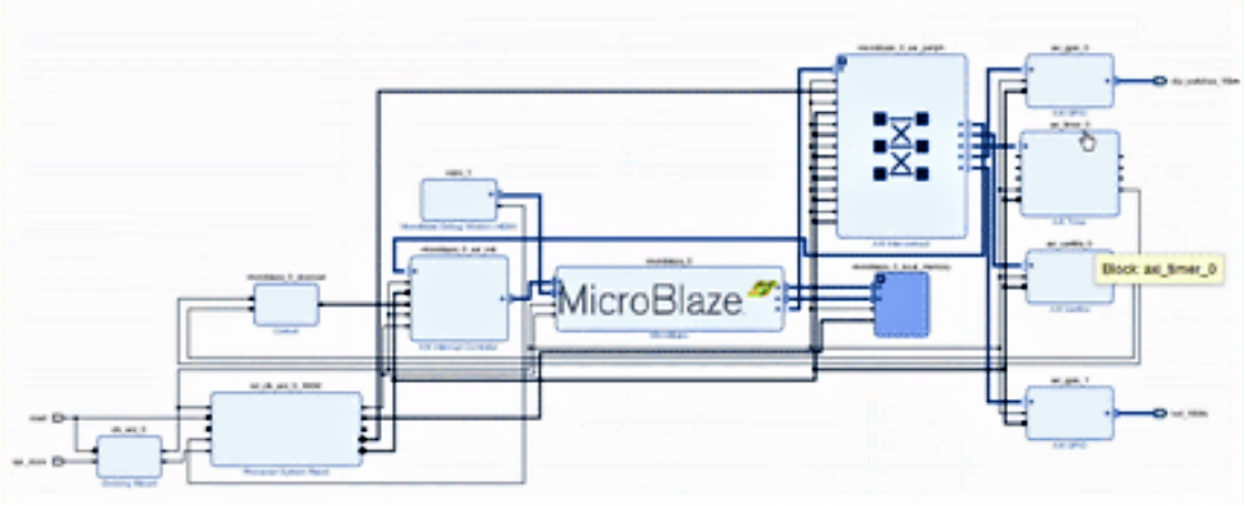

FIGURE 10 – MicroBlaze design (avec les nouveaux blocs)

Après avoir intégrer les nouveaux blocs, il faut tester leur bon fonctionnement. Pour cela, nous effectuions des tests. Le premier test consiste à contrôler les LEDs à l'aide des switches et permettra de vérifier si les deux GPIOs fonctionnent bien. De même, pour l'intégration du µC/OS-II dans le processeur en prenant l'AXI Timer comme le tick source, on a essayé de faire un *print* dans le terminal toutes les secondes pour confirmer le bon fonctionnement de ce dernier.

6 Portage μC/OS II sur Microblaze

µC/OS II est un noyau Temps Réel qui permet d'effectuer une exécution de plusieurs tâches sur un processeur. L'adaptation de µC / OS-III à un microprocesseur ou à un microcontrôleur s'appelle le portage. La majeure partie de µC / OS-II est écrite en C pour une meilleure portabilité. Cependant, il est toujours nécessaire de modifier quelques fichiers spécifiques au processeur en C et en langage assembleur (Figure 11). Le portage d'un noyau On peut trouver sur le site Internet de µC/OS II les différentes versions de µC/OS II portées sur les différents processeurs.

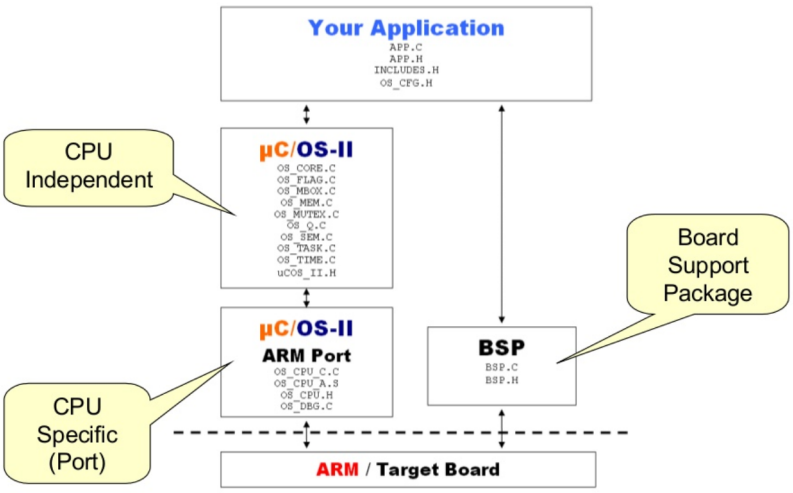

FIGURE 11 – Fichiers µC/OS II

7 Application de test μC/OS II : Interpréteur de commande

Afin de tester et confirmer le bon fonctionnement du portage du noyau μC/OS II, nous avons réalisé un interpréteur de commandes qui ressemble à l'interpréteur shell de UNIX. Cet interpréteur doit permettre de récupérer des informations sur les tâches du noyau et doit accepter les commandes suivantes :

La commande `uname`

Cette commande permet de récupérer et afficher la version du noyauμC/OS II en utilisant la primitive *OSVersion()*.

La commande `cpu`

Cette commande permet de récupérer et afficher le pourcentage du CPU consommé, le nombre de tâches en cours d'exécution et le nombre moyen de changements de contextes. la figure ci-dessous montre le résultat de cette commande :

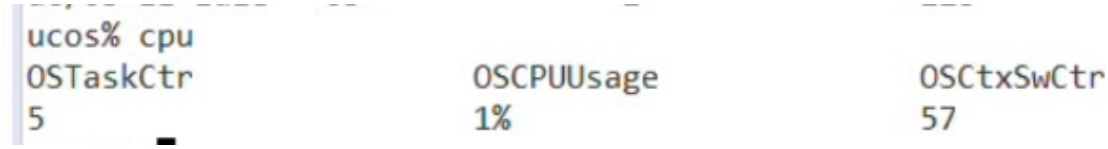

FIGURE 12 – Réponse de l'interpréteur à la commande `cpu`

La commande `ps`

Cette commande permet de récupérer et afficher la liste des tâches en cours et des informations sur ces tâches :

— la priorité de la tache

— le nombre de changements de contextes pour la tâche considérée

— La quantité de mémoire totale, libre et consommée de la pile de la tâche

la figure ci-dessous montre le résultat de cette commande :

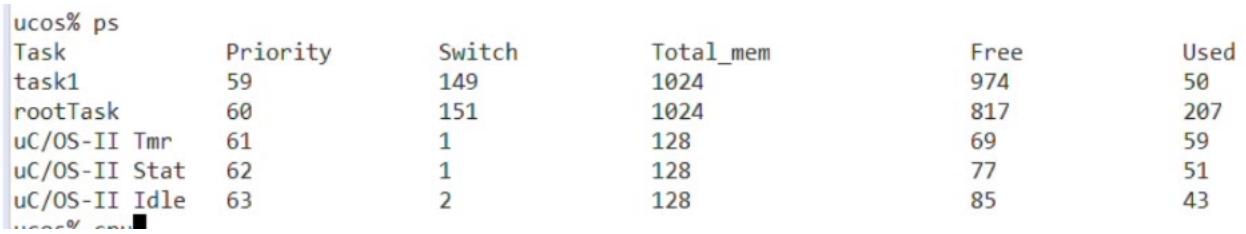

FIGURE 13 – Réponse de l'interpréteur à la commande `ps`

La commande `spawn i`

Cette commande permet de créer une tâche de priorité i (comprise entre 0 et 9 inclus) la figure ci-dessous montre le résultat de cette commande :

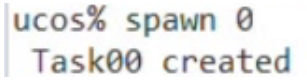

FIGURE 14 – Réponse de l'interpréteur à la commande `spawn`

La commande `kill i`

Cette commande permet de tuer la tâche de priorité i (comprise entre 0 et 9 inclus) : la figure ci-dessous montre le résultat de cette commande :

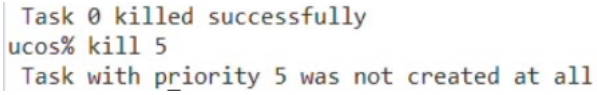

FIGURE 15 – Réponse de l'interpréteur à la commande `kill`

La commande `adt`

Cette commande permet de de récupérer et afficher la température mesuré par le capteur ADT7420 la figure ci-dessous montre le résultat de cette commande :

```
ucos% adt
Temperature on board = 27.437°C
```

FIGURE 16 – Réponse de l'interpréteur à la commande `adt`

Remarque : Il est à noter qu'au cas où l'utilisateur tape une mauvaise commande, l'interpréteur lui affiche la liste des commandes supportées comme le montre la figure ci-dessous :

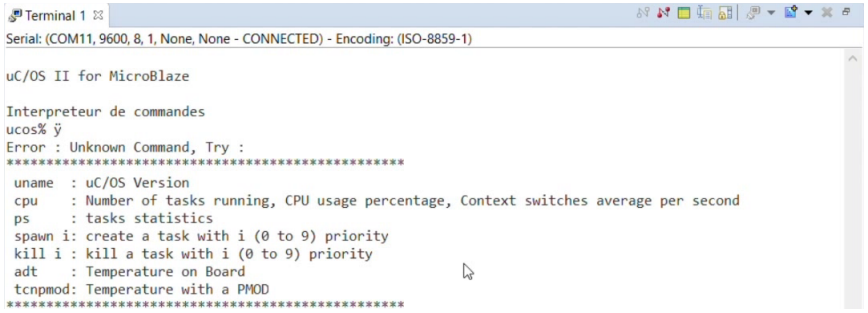

FIGURE 17 – Réponse de l'interpréteur à une commande erronée

8 Conclusion

En conclusion, nous avons réussi à répondre à toutes les spécifications qui nous ont été imposées par le cahier de charges. Un système FPro avec portage noyau µC/OS-II est maintenant fonctionnelle.

Tout au long de ce projet, nous avons découvert plusieurs notions en près, notamment le prototypage FPGA et le portage du noyau µC/OS-II. Ce qui nous a permis d'enrichir notre bagage en systèmes embarqués.

Références

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

[2] Dr. Pong P. Chu
FPGA prototyping by VHDL examples. 2nd edition. Xilinx MicroBlaze MCS SoC. Editions Wiley. 2017
`https://academic.csuohio.edu/chu_p/rtl/fpga_mcs_vhdl.html`

[3] Dr. Jean J. Labrosse
MicroC/OS-II The Real time Kernel. 2nd Edition.

[4] µC/OS-II User manual
`https://doc.micrium.com/pages/viewpage.action?pageId=16879190`

[5] Nexys 4 DDR - Getting Started with Microblaze
`https://reference.digilentinc.com/learn/programmable-logic/tutorials/nexys-4-ddr-getting-started-with-microblaze/start`

[6] Datasheet des capteurs de température : ADT7420 Analog Device et Pmod TMP3 Digilent
`https://www.analog.com/en/products/adt7420.html#product-overview`
`https://reference.digilentinc.com/reference/pmod/pmodtmp3`