Auteur : Spinelli Isaia

Prof : [Etienne](https://cyberlearn.hes-so.ch/user/view.php?id=104149&course=1) Messerli

Ing : Sébastien Masle

Date : 03.04.2020

Salle : A09 (maison) – HEIG-VD

Classe : SOCF

IP AXI4-lite avec I/O de la FPGA

[Système Soc intégré avec fpga (socf)](https://cyberlearn.hes-so.ch/course/view.php?id=14116#section-1)

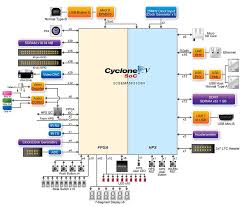


Table des matières

[Introduction - 2 -](#_Toc39331315)

[Première partie : sans interruption - 2 -](#_Toc39331316)

[Plan d’adressage - 2 -](#_Toc39331317)

[Conception - 3 -](#_Toc39331318)

[Ecriture - 4 -](#_Toc39331319)

[Lecture - 5 -](#_Toc39331320)

[Description VHDL - 6 -](#_Toc39331321)

[Test et validation de l’IP - 7 -](#_Toc39331322)

[Écriture - 7 -](#_Toc39331323)

[Lecture - 8 -](#_Toc39331324)

[Création du composant - 8 -](#_Toc39331325)

[Ajout du composant - 9 -](#_Toc39331326)

[Modification du top - 10 -](#_Toc39331327)

[Validation pratique - 10 -](#_Toc39331328)

[Réalisation de la spécification - 11 -](#_Toc39331329)

[Deuxième partie : avec interruption - 11 -](#_Toc39331330)

[Plan d’adressage - 11 -](#_Toc39331331)

[Conception - 11 -](#_Toc39331332)

[Test de l’IP - 11 -](#_Toc39331333)

[Mise à jour du composant (Qsys) - 11 -](#_Toc39331334)

[Génération des fichiers HDL - 11 -](#_Toc39331335)

[Synthétiser et faire le placement-routage du projet - 11 -](#_Toc39331336)

[Réalisation de la spécification - 11 -](#_Toc39331337)

[Compléter le code C - 11 -](#_Toc39331338)

[Modifier la configuration mémoire - 11 -](#_Toc39331339)

[Compiler et tester - 11 -](#_Toc39331340)

[Conclusion - 12 -](#_Toc39331341)

[Difficultés rencontrées - 12 -](#_Toc39331342)

[Compétences acquises - 12 -](#_Toc39331343)

[Résultats obtenus - 12 -](#_Toc39331344)

# Introduction

Ce laboratoire a pour but de réaliser une IP avec une interface AXI4-lite et connectée sur le bus Lightweight HPS-to-FPGA. Cette IP doit permettre d’accéder à des I/O câblées sur la partie FPGA via des registres. Je dois analyser le fonctionnement du bus AXI4-lite afin de concevoir une IP personnalisée pour les besoins du laboratoire

# Première partie : sans interruption

L'objectif est d'interfacer à l'aide d'une IP AXI4-lite tous les I/O disponibles sur la FPGA, sans utiliser des composants PIO, soit les boutons (KEYs), les switchs (SW), les LEDs et les afficheurs 7 segments.

Mon IP AXI4-lite comprend une constante 32 bits à l’offset 0x0 ainsi qu'un registre de test R/W à l’offset 0x4. Les offsets sont relatifs à l’adresse de base donnée à l'instance de l’IP dans Qsys.

## Plan d’adressage

Pour commencer, j’ai conçu un plain d’adressage afin mettre au claire les différents aspects de mon interface.



L’interface dispose de 12 bits adressables ce qui représente 4Ko avec un bus de 32bits d’adresse et de donnée.

On peut voir à l’offset 0 une constante d’une valeur de 0xDEADBEEF afin quel la valeur soit facilement reconnaissable. Cette constant sera disponible seulement en lecture et non pas en écriture.

À l’offset 0x4 il y a un registre de test accessible en écriture et lecture afin de tester facilement l’interface. Étant donné que je dispose d’une grande plage d’adresse, je me suis permis afin de facilité le décodage d’adresse de laisser un offset de 0x100 entre chaque I/O de mon interface.

Comme on peut le voir, à l’offset 0x100, il y a les leds accessible en écriture ainsi qu’en lecture. Comme il y a que 10 leds, uniquement les 10 premiers bits sont utilisés et les autres (31 à 10) sont réservés en cas d’écriture et une valeur de 0 sera retourné en cas de lecture. **Cela signifie qu’une écriture sur ces bits réservés n’aura aucun n’effet.**

Ensuite, à l’offset 0x200, il y a les inputs des 4 Keys qui sont accessible uniquement en lecture. En cas d’écriture à cette adresse, il n’y aura aucun effet. On peut remarquer une ligne noire en dessous car il est demandé plus tard de gérer les interruptions et donc une ou plusieurs adresses sera nécessaires pour la gestion de ces interruptions.

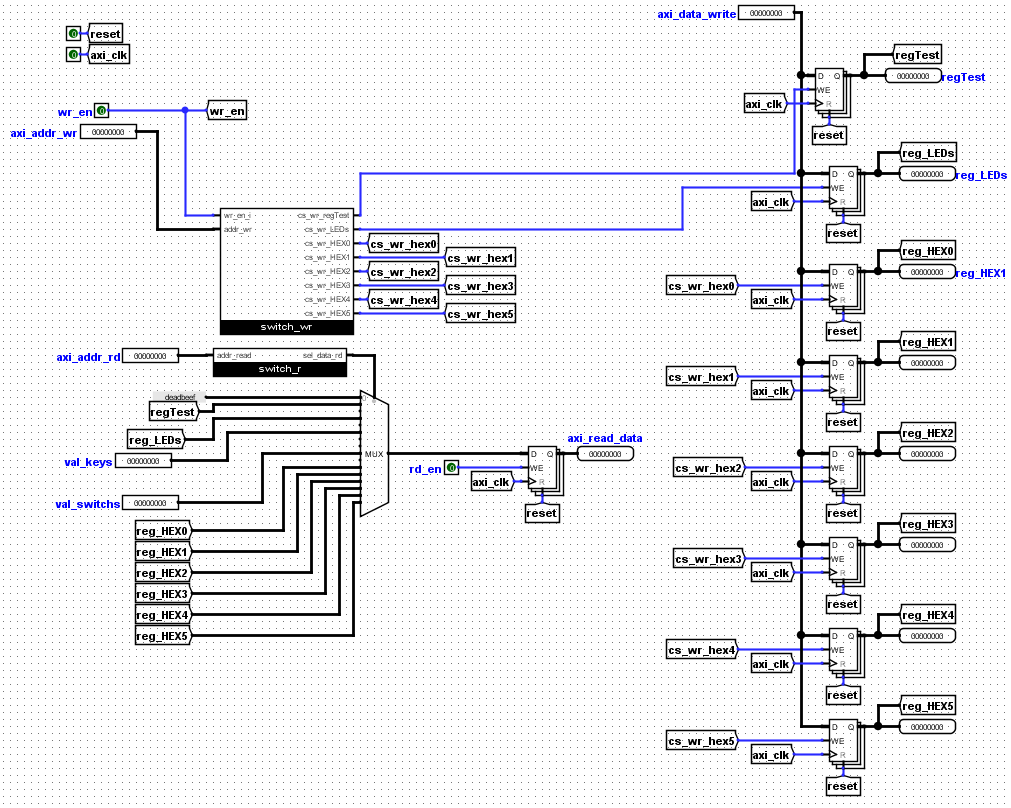
À l’offset 0x300 il y a les 10 switches accessibles uniquement en lecture comme les Keys. En cas d’écriture à cette adresse, il n’y aura aucun effet.

A partir de l’offset 0x400, il y a les 6 afficheurs 7 seg décaler avec un offset de 0x10. Par exemple, le premier afficheur est à l’offset 0x400 et le seconde à 0x410. Ces différents afficheurs sont accessibles en lecture ainsi qu’en écriture. Uniquement les 7 premiers bits sont utilisés pour les 7 segments étant donné que le point n’est pas branché.

La plage d’adresse s’étend jusqu’à un offset de 0xfff car l’interface dispose de 12bits. Toutes les adresses non utilisées sont pour l’instant réservées et sera peut-être utilisées plus tard. J’ai décidé qu’en cas de lecture à une adresse non utilisée, cela n’aura aucun effet.

## Conception

Je dois dire qu’au début de ce laboratoire j’étais perdu, je ne savais pas par quoi commencer. De ce fait, comme cela me faisait penser à IFS, j’ai commencé faire un petit schéma pour représenter grossièrement mon interface :



Au lieu d’avoir des décodeurs, des simples switches seront utilisé pour décoder l’adresse pour une lecture et une écriture. Cette étape m’a éclairé afin de mieux percevoir le système général.

Une fois que le concept m’est paru plus claire, j’ai commencé par lire le document fourni « designing\_a\_custom\_axi\_slave\_rev1.pdf ». Ce document m’as fait comprendre plus exactement les étapes à réaliser.

Ensuite, j’ai analysé le code fourni, ce qui m’a encore aidé pour commencer à implémenter l’interface AXI4-lite.

### Ecriture

Une bonne partie du code VHDL pour l’écriture d’une adresse et des données été déjà écrite. J’ai pu compléter le reste en m’inspirant du code déjà fourni et surtout en analysant le document donné qui explique comment designer un bus axi slave.

Après avoir lus entièrement le chapitre sur la transition d’une écriture, j’ai pris connaissance de chaque signal du bus et des différents canaux. Ce qui m’a le plus aidé à finir l’implémentation est le chronogramme dans le document fourni :

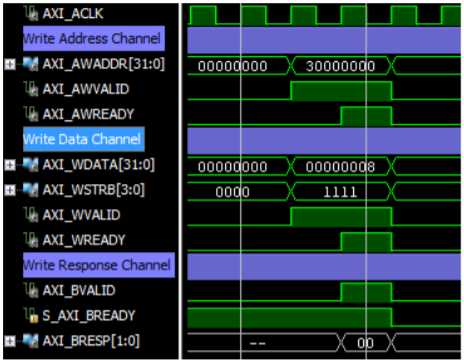


Figure ‑ : Chronogramme d'écriture sur un bus AXI light

Dans ce chronogramme, on peut voir tous les signaux utiles pour une transaction d’écriture du master au slave. De plus, on peut voir les différents timings ainsi que les 3 canaux utilisés :

1. Le canal d’adresse et de contrôle
2. Le canal des données et de paramètre (strobe)
3. Le canal de réponse

C’est trois canaux sont indiqués par les bandes bleus sur le chronogramme. De plus, on peut voir qu’il est possible d’utiliser deux canaux simultanément. Ici, on écrit l’adresse et les données en même temps.

Le paramètre strobe, envoyé en même temps que les datas, indique quel octet nous souhaitons écrire.

### Lecture

Le canal de l’adresse de lecture était déjà implémenté. Cependant, celui des données ne l’était pas du tout. Comme pour la partie écriture, je me suis grandement aidé du document fourni « designing\_a\_custom\_axi\_slave\_rev1.pdf ». Celui m’a permis de connaitre les deux canaux de lecture et tous les signaux utiles à une lecture.

Un chronogramme pour la lecture est documenté. Celui aussi m’a beaucoup aidé pour les timings de la transaction :

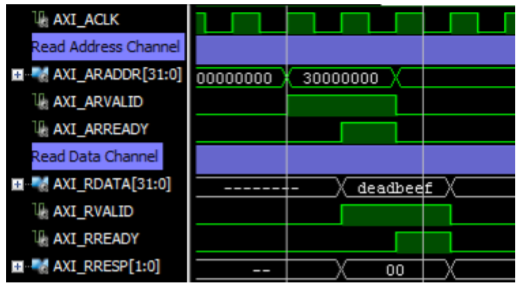


Figure ‑: Chronogramme de lecture sur un bus AXI light

On peut voir les deux différents canaux :

1. Le canal d’adresse et de contrôle
2. Le canal de donnée et de réponse

On indique l’adresse qu’on souhaite lire et au flanc montant suivant, la donnée est prête à être lue.

Finalement, après avoir réalisé un petit schéma avec logisim, étudier les documents fournis, analyser le code déjà écrit et surtout m’inspirer des chronogrammes, j’ai pu concevoir l’IP demandée avec une interface AXI4-lite

## Description VHDL

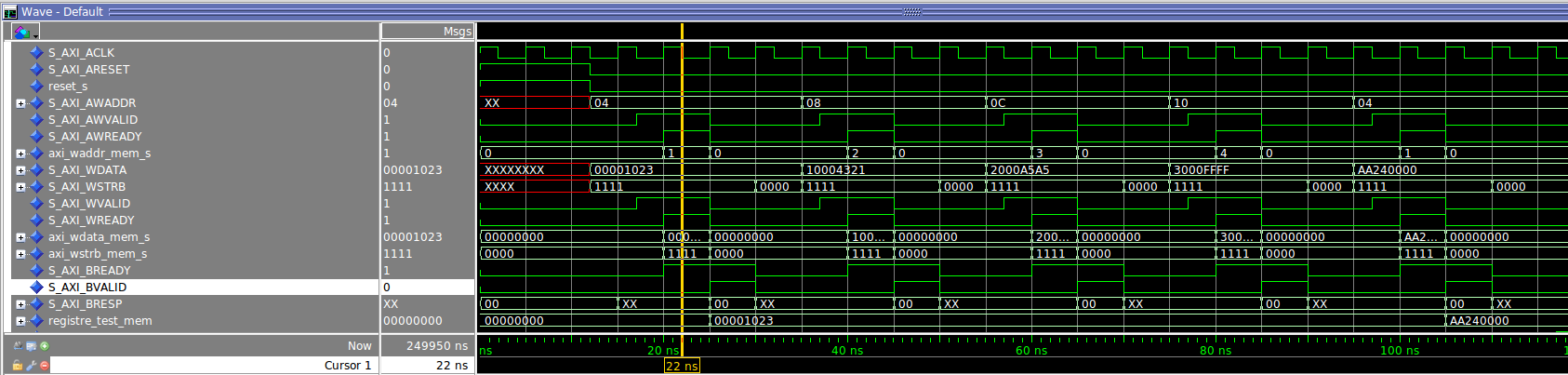
La description VHDL de l’interface du bus AXI4-lite est en annexe.

## Test et validation de l’IP

Pour commencer, j’ai testé l’IP afin de valider son fonctionnement. Pour ce faire, j’ai utilisé le test Bench fourni qui teste la validité des accès en lecture et écriture.

### Écriture

J’ai commencé par tester l’accès en écriture. Au début quelques timing n’était pas respecté, j’ai donc dû modifier un peu mon IP. Après un certain nombre de correction, j’ai obtenu le chronogramme suivant :



1

Figure ‑ : Écriture avec le test Bench

2

3

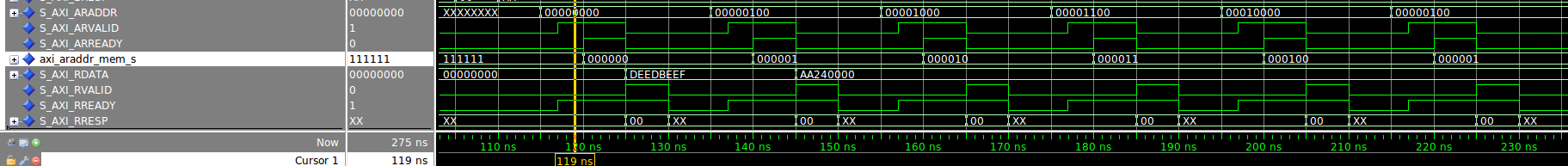
1. Après le reset du début, on peut voir qu’un accès d’écriture vas être effectué à l’adresse 0x04. Une fois que le master a levé le signal AWValid qui indique que l’adresse est valide, et que le slave est prête (AWReady à 1) on voit le signal « axi\_waddr\_mem\_s » qui enregistre la valeur de l’adresse. L’adresse 0x4 correspond à l’adresse 0x1 car les 2 bits de poids faible sont ignorés car nous travaillons avec des mots de 32 bits.
2. Simultanément, les données à écrire ainsi que le paramètre strobe est envoyé. Après que le master est indiqué que les données et le paramètre strobe sont valide (WValid à 1), et que le slave est prêt à les lire (WReady à 1), les données et le paramètre strobe sont enregistrés dans les signaux correspondent (axi\_wdata\_mem\_s et axi\_wstrb\_mem\_s).
3. Lorsque le master est prêt à lire la réponse (BReady à 1) et ensuite que le slave à une réponse valide (BValid à 1), le slave envoie la réponse et effectue l’écriture.

Les trois prochaines écrites se passe correctement mais elles sont faites à des adresses pas prise en compte par mon IP. Cependant la dernière écriture s’effectue aussi à l’adresse 0x4, donc le registre de test est de nouveau affecté par la nouvelle valeur donnée.

Après chaque transaction, j’ai décider de remettre des valeurs par défaut afin de bien voir les transitions. Par exemple, BResp passe à chaque fois à XX après les transactions et il en va de même pour les signaux internes (axi\_waddr\_mem\_s, axi\_wdata\_mems\_s et axi\_wstrb\_mem\_s passe à 0).

### Lecture

Après avoir testé et validé la partie écriture de mon IP, j’ai commencé à tester la partie lecture. Une fois avoir obtenu le chronogramme ci-dessous, j’en ai déduit que la partie lecture était correcte.



3

1

2

4

Figure ‑ : Lecture avec le test Bench

1. Le master commence par indiquer quelle adresse il souhaite lire. Dans le premier l’adresse est 0x0, ce qui correspond à ma constante (0xdeedbeef). Une fois que le master indique que l’adresse est valide et que le slave est ready, l’adresse est lue et enregistrer dans le signal interne axi\_araddr\_mem\_s.
2. Une fois qu’une adresse a été enregistrée par le slave et que le master est prêt à recevoir la réponse, le slave peut envoyer sur le bus de lecteur (S\_AXI\_RDATA) les données à l’adresse souhaité ainsi que le signal de réponse (S\_AXI\_RRESP).
3. L’étape est la même qu’au point 1, mais l’adresse souhaitée est « 100 » (0x4) ce qui correspond à un offset de 1 étant donné que nous travaillons par mot de 32 bits.
4. L’étape est la même qu’au point 2. La valeur 0xAA240000 écrite précédemment dans la partie écriture (Figure ‑) est maintenant relue.

On peut voir que la chaine complète fonctionne, écriture suivis d’une lecture grâce au registre de test à l’offset 0x4. Maintenant que d’après le test bench mon IP fonctionne correctement je souhaits le vérifier à l’aide d’un petit code C qui permettrai d’écrire simplement les switches sur les leds. Pour cela, je dois maintenant créer et ajouter mon IP dans mon projet VHDL.

## Création du composant

Comme indiqué dans la donnée du laboratoire, j’ai créé un composant dans mon projet de Qsys afin d’ajouter mon IP à Qsys. Malheureusement j’ai perdu du temps à cause d’une petite erreur stupide. Je n’ai pas tout de suite cliqué sur le bouton « Analyze Syntheses Files », j’ai donc ajouté les signaux manuellement et les noms correspondait pas.

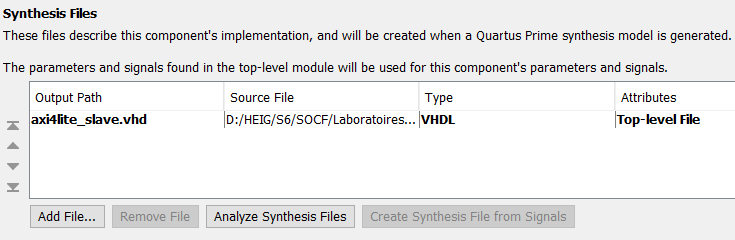


Figure ‑ : Bouton oublié lors de la création de l'IP

Grâce à l’aide de l’assistant M. Masle, j’ai pu résoudre ce problème.

Voici à quoi doit ressembler les signaux et les interfaces :

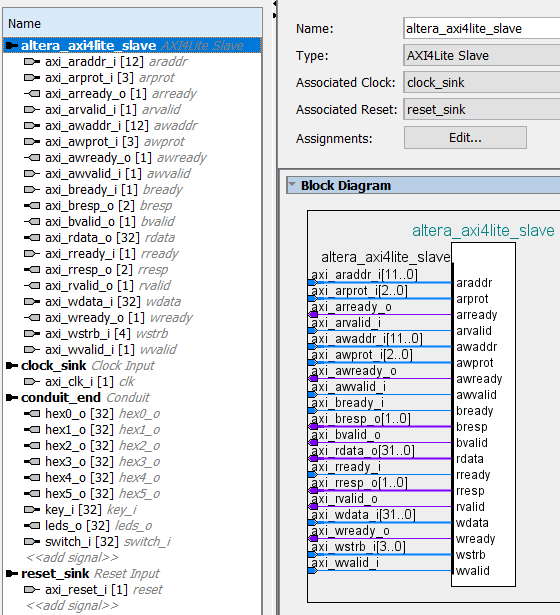


Figure ‑ : Signaux et interfaces

Après avoir ajouté les composants AXI4Lite Slave, Clock Inuput, Reset Input et Conduit, j’ai pu glisser les signaux dans les interfaces correspondante. Il a aussi fallu lier la clock et le reset à l’interface AXI4Lite slave.

## Ajout du composant

Après avoir créer mon nouveau composant, je l’ai ajouté dans le système Qsys. Ensuite, j’ai effectué les connexions ainsi que les exports de memory, hps\_io et du conduit de mon IP. Finalement, j’ai ajouté l’adressage du composant. Voici à quoi cela doit ressembler :

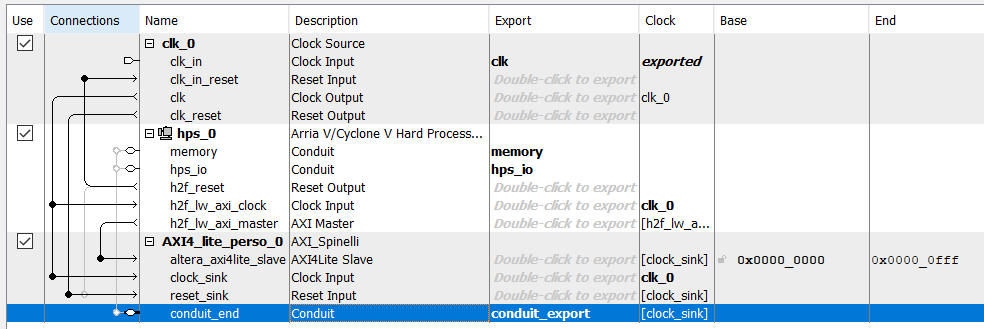


Figure ‑ : Système Qsys

Après avoir fait tout cela, j’ai pu générer les fichiers HDL du projet Qsys.

Remarque : La génération des fichiers HDL doit être refaite à chaque modification de l’IP.

## Modification du top

Grâce au menu Generate -> Show Instantiation Template dans Qsys, j’ai pu apporter les modifications nécessaires au top du projet dans le fichier DE1\_SoC\_top.vhd. J’ai donc ajouté les nouveaux signaux « conduit » dans le composant « qsys\_system ». Ensuite, j’ai mappé les conduits avec les I/Os de la FPGA comme ci-dessous :

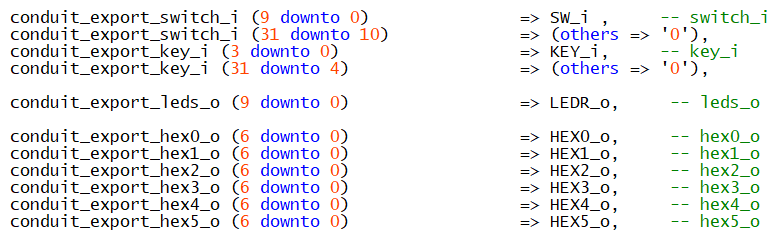


Figure 0‑9 : Mapping du Top

On peut voit ici que j’ai décidé de crée une sortie pour chaque afficheur 7 segments. Il aurait été possible de combiner les afficheurs 0 à 3 et 4 à 5. Cependant, j’ai préféré avoir accès à chaque afficheur indépendamment. De plus, il a fallu mettre à 0 tous les bits non utilisé des entrées keys et switch.

Maintenant que tout est prêt, j’ai pu synthétiser et faire le placement routage du projet.

## Validation pratique

Avant de me lancer dans les spécifications, je souhaitais m’assurer du bon fonctionnement réel de mon interface grâce à un test pratique. J’ai donc écris quelques lignes de code C afin de tester que l’écriture ainsi que la lecture ce déroule correctement. Voici le code que j’ai testé :

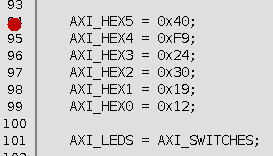


Figure ‑ : Code de test

Ce code affiche de 0 à 5 sur les afficheurs 7 segments dans gauche à droite et copie les valeurs des switches sur les leds. Une fois le projet lancé, voici ce que j’ai pu voir sur ma carte DE1-SoC :

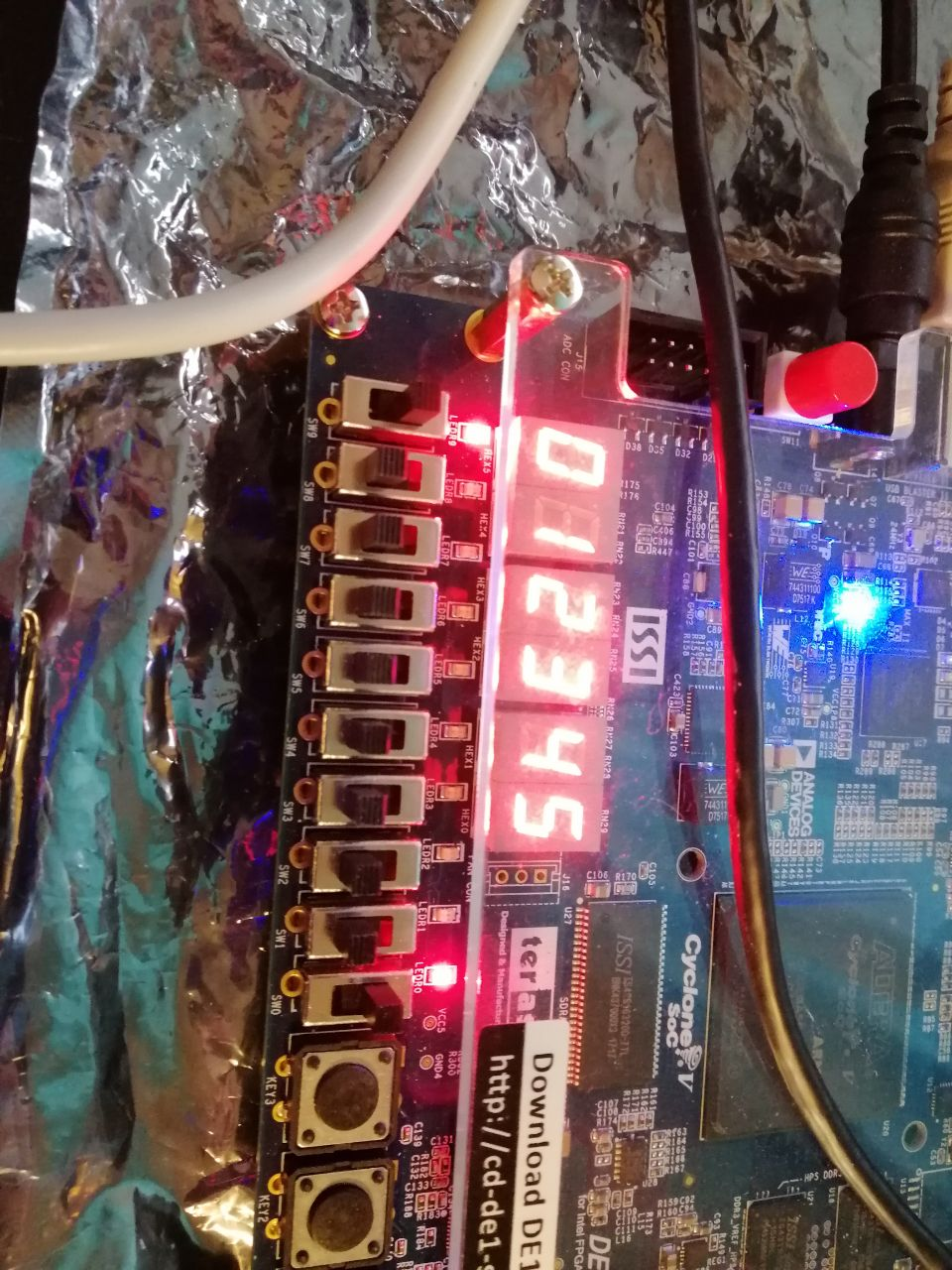


Figure ‑ : Test sur la DE1

Grâce à ce test, j’ai pu m’assurer qu’avec le matériel mon interface fonctionne vraiment.

## Réalisation de la spécification

Maintenant que tout est prêt, j’ai pu réaliser la spécification de la partie 1. J’ai commencé par créer un projet Altera Monitor Program. Ensuite, j’ai repris les fichiers C du laboratoire précèdent et j’ai adapté le code afin de répondre aux spécifications demandées dans ce laboratoire.

Ceci était facile car le 90% de la spécification est la même que le labo précédent.

Vous pouvez voir le code final de la partie 1 dans le répertoire « axi4lite/axi4lite/soft/src/labo5\_partie1.c ».

*Remarque : J’ai perdu énormément de temps car j’avais d’étranges problèmes pour charger mon code C sur la carte DE1-SoC. Heureusement, M. Masle a mis à disposition sur switch tous les softs nécessaires pour Windows. J’ai donc pu télécharger, dézipper et faire les installations des différents programmes sur ma machine native. Grâce à cela, j’ai gagné beaucoup de temps pour chaque action de plus il était maintenant possible de programmer la DE1-SoC.*

# Deuxième partie : avec interruption

## Plan d’adressage



## Conception

## Test de l’IP

## Mise à jour du composant (Qsys)

## Génération des fichiers HDL

## Synthétiser et faire le placement-routage du projet

## Réalisation de la spécification

### Compléter le code C

### Modifier la configuration mémoire

### Compiler et tester

# Conclusion

## Difficultés rencontrées

## Compétences acquises

## Résultats obtenus

Date : 07.05.20

Nom de l’étudiant : Spinelli Isaia