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)

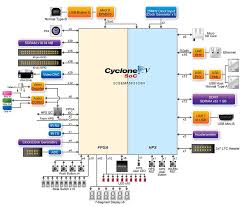


Table des matières

[Introduction - 2 -](#_Toc39243026)

[Première partie : sans interruption - 2 -](#_Toc39243027)

[Plan d’adressage - 2 -](#_Toc39243028)

[Conception - 3 -](#_Toc39243029)

[Ecriture - 3 -](#_Toc39243030)

[Lecture - 3 -](#_Toc39243031)

[Description VHDL - 3 -](#_Toc39243032)

[Test et validation de l’IP - 3 -](#_Toc39243033)

[Création du composant - 3 -](#_Toc39243034)

[Ajout du composant - 3 -](#_Toc39243035)

[Génération des fichiers HDL - 3 -](#_Toc39243036)

[Modification du top - 3 -](#_Toc39243037)

[Synthétiser et faire le placement-routage du projet - 3 -](#_Toc39243038)

[Réalisation de la spécification - 3 -](#_Toc39243039)

[Deuxième partie : avec interruption - 4 -](#_Toc39243040)

[Plan d’adressage - 4 -](#_Toc39243041)

[Conception - 4 -](#_Toc39243042)

[Test de l’IP - 4 -](#_Toc39243043)

[Mise à jour du composant (Qsys) - 4 -](#_Toc39243044)

[Génération des fichiers HDL - 4 -](#_Toc39243045)

[Synthétiser et faire le placement-routage du projet - 4 -](#_Toc39243046)

[Réalisation de la spécification - 4 -](#_Toc39243047)

[Compléter le code C - 4 -](#_Toc39243048)

[Modifier la configuration mémoire - 4 -](#_Toc39243049)

[Compiler et tester - 4 -](#_Toc39243050)

[Conclusion - 5 -](#_Toc39243051)

[Difficultés rencontrées - 5 -](#_Toc39243052)

[Compétences acquises - 5 -](#_Toc39243053)

[Résultats obtenus - 5 -](#_Toc39243054)

# 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.

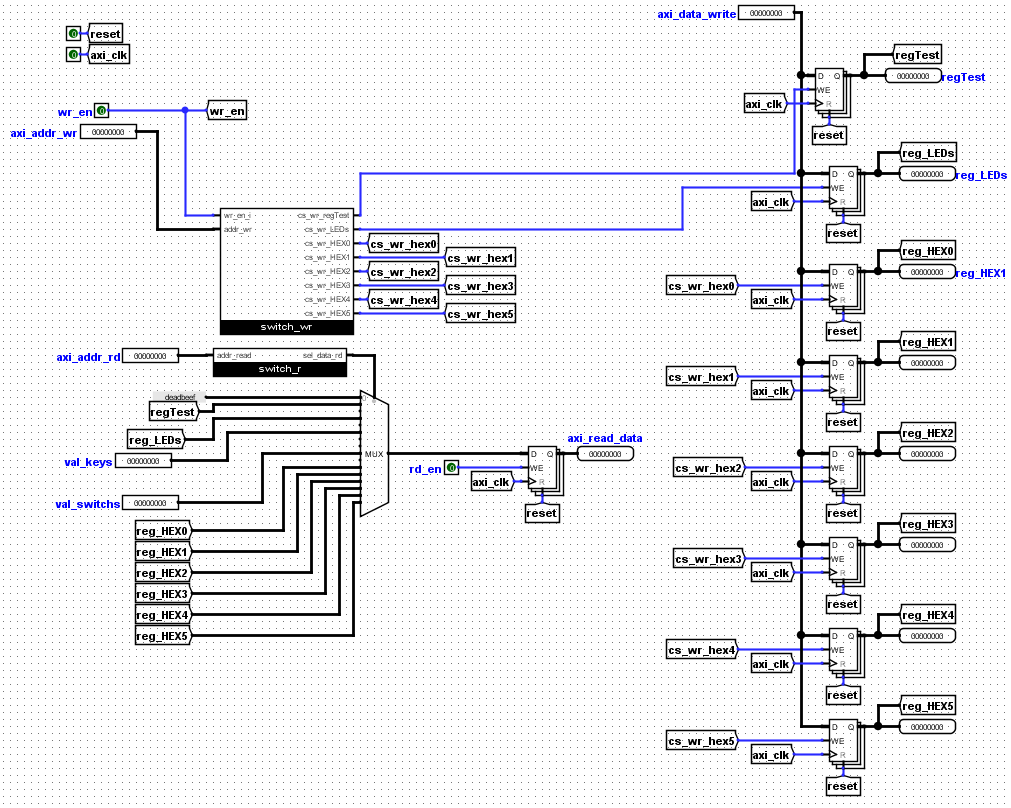
À l’offset 0x300 il y a les 10 switches accessibles uniquement en lecture comme les Keys. En cas d’écriture à cette adresse, il n’y aura aucun effet.

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.

## Conception

Je dois dire qu’au début de ce laboratoire j’était 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 :

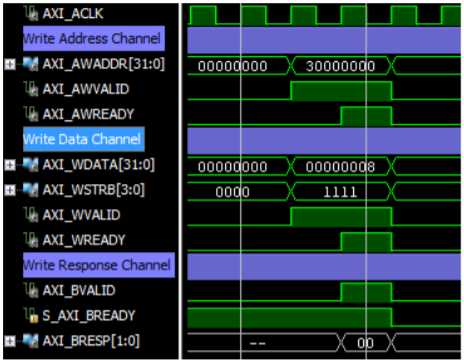


Figure 0‑1 : Chronogramme d'écriture sur un bus AXI light

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 :

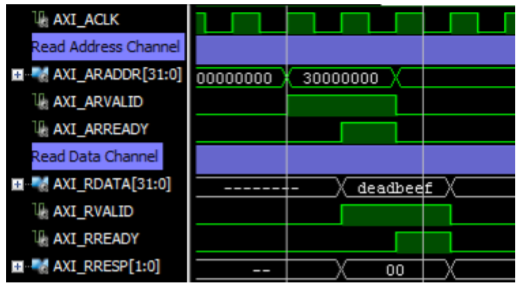


Figure 0‑2: Chronogramme de lecture sur un bus AXI light

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

Change valeur defaut 7 seg (inverse)

## Description VHDL

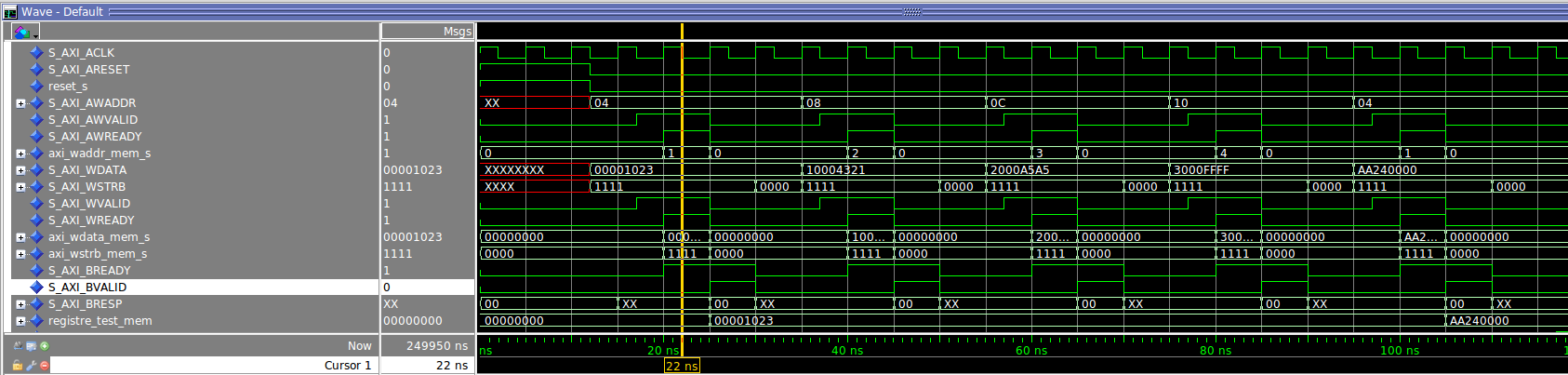
La description VHDL de l’interface du bus AXI4-lite est en annexe.

## 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

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 à 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. S

## Création du composant

## Ajout du composant

## Génération des fichiers HDL

## Modification du top

## Synthétiser et faire le placement-routage du projet

## Réalisation de la spécification

Créer un nouveau projet

Coder

Compiler et tester

# 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