



HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD

www.heig-vd.ch

Hes·so

Haute Ecole Spécialisée
de Suisse occidentale

Fachhochschule Westschweiz

University of Applied Sciences and Arts
Western Switzerland

Laboratoire 5 : IP AXI4-lite avec I/O de la FPGA

Département : Technologies de l'Information et de la Communication (TIC)
Unité d'enseignement : System on chip on FPGA (SOCF)

Auteurs : Pierrick Muller
Professeur : Alberto Dassati, Etienne Messerli
Assistant : Sébastien Masle, Sydney Hauke
Classe : SOCF-1-A
Date : 12 mai 2020

Table des matières

1 Introduction	2
1.1 Objectifs	2
1.2 Spécifications sans les interruptions	2
2 Analyse	3
2.1 Partie 1	3
2.2 Partie 2	4
2.3 Adress Map finale	5
3 Réalisation et implémentation	5
3.1 Création de l'IP axi lite	5
3.2 Ajout de l'IP dans Qsys	6
3.3 Actions dans le projet Qsys	6
3.4 Implémentation dans le fichier top du projet	7
3.5 Code C	8
4 Simulation et tests	9
4.1 Contrôle des timings	9
4.2 Test du strobe	10
4.3 Problèmes rencontrés	10
4.4 Tests sur board	11
5 Conclusion	14
6 Signatures	14

1 Introduction

1.1 Objectifs

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. Vous devrez analyser le fonctionnement du bus AXI4-lite afin de concevoir une IP personnalisée pour les besoins du laboratoire.

1.2 Spécifications sans les interruptions

L'objectif est d'interfacer à l'aide d'une IPAXI4-lite tous les I/O disponibles sur la FPGA, sans utiliser des composants PIO, soit les boutons (KEYs), les switches (SW), les LEDs et les afficheurs 7 segments.

Votre IP AXI4-lite comprendra 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.

Spécifications du programme :

Le but est d'allumer les LEDs selon l'état des boutons (KEYs) et interrupteurs (switch) disponibles. Les afficheurs 7 segments sont dépendants de la constante définie dans l'IP. La spécification du fonctionnement est la suivante :

- Appui sur KEY0 : l'états des switches est copiés sur les LEDs. Les afficheurs HEX5 à HEX0 affichent en hexadécimal les bits 23 à 0 de la constante définie dans l'IP.
- Appui sur KEY1 : l'états inverses des switches est copiés sur les LEDs. Les afficheurs HEX5 à HEX0 affichent en hexadécimal l'inverses des bits 23 à 0 de la constante définie dans l'IP
- Appui sur KEY2 : l'affichage des LEDs et des afficheurs 7 segments subit une rotation à droite. Rotation d'un bit pour les LEDs, rotation d'un afficheur complet pour les afficheurs 7 segments.
- Appui sur KEY3 : l'affichage des LEDs et des afficheurs 7 segments subit une rotation à gauche. Rotation d'un bit pour les LEDs, rotation d'un afficheur complet pour les afficheurs 7 segments.

Dans une seconde partie, l'appui sur les boutons KEY2 ou KEY3 devra être géré à l'aide d'interruption vers le HPS (2ème partie).

- Votre design doit permettre de générer une interruption lors de l'activation (détection de flanc) d'un des 4 boutons. Vous devez prévoir les accès et les flags nécessaires pour gérer l'interruption. Il doit être possible d'activer/masquer l'interruption pour chaque bouton.

2 Analyse

2.1 Partie 1

Cette partie nous demandait dans un premier temps d'établir un plan d'adressage en incluant certains registres spécifiques, soit un registre contenant une constante à l'adresse 0x000 et un registre de test pouvant être écrit et lu à l'adresse 0x004. La taille du bus d'adresse étant de 12 bits, nous avons un espace de 4 Ko à notre disposition pour l'adressage. Le plan d'adressage final est disponible à la fin de la partie analyse. Je précise ici que quand je note un champ de bits comme "reserved", cela indique que ces champs de bits sont réservés pour des utilisations futures, et que l'écriture sur ces bits précis n'aura pas d'impact, et ces bits seront toujours considérés comme ayant la valeur '0'.

Concernant la réflexion sur l'implémentation du bus axi lite, je me suis basé sur le document qui était fourni avec ce laboratoire, "Designing a AXI4-lite Slave Peripheral, Griffin, Xilinx". J'ai identifié deux points importants dès le début de

l'analyse :

- l'"Assert and wait" rule est un point important à respecter dans l'implémentation du bus, et doit être gardée à l'esprit lors de l'écriture des différents process de l'IP
- Les timings de chaque canal doivent être respectés, faute de quoi le bus ne fonctionnera pas. Je me suis rendu compte par la suite que si c'était particulièrement vrai pour les timings concernant les canaux de lecture, certains timings pouvaient être décalés sans poser de problèmes dans le cadre des canaux d'écriture (Je pense au canal de réponse de l'écriture, dans mon cas)

En gardant ces deux points à l'esprit, j'ai pu commencer le laboratoire. J'ai d'abord fonctionné en utilisant vsim et le test-bench fournit afin de pouvoir contrôler les timings et les résultats stockés ou lus du bus axi lite.

2.2 Partie 2

Nous devons ajouter une gestion des interruptions dans cette partie. J'ai commencé par regarder le laboratoire précédent pour voir quels registres devaient être implémentés en plus. Suite à cela, le plan d'adressage a reçu deux nouvelles entrées, concernant le registre keys_IM pour interruption mask et keys_EC pour edge capture. Ces deux registres servent le but suivant :

- Le registre d'interruption mask permet de définir quels keys génèrent une interruption lorsqu'on les pressent. les bits 3 à 0 correspondent aux keys 3 à 0 et la présence d'un bit actif dans l'une de ces positions signifie que le key correspondant génère une interruption lors de son appui.
- Le registre d'edge capture permet de savoir quel key a été appuyé lorsqu'une interruption a lieu. L'accès en écriture à ce registre remplit le registre de bit '0' et clean l'interruption qui a été générée.

Ces deux registres seront utilisés dans le code VHDL de l'IP axi lite afin de pouvoir lever une exception. Une chose intéressante à garder en tête c'est que ces registres ne sont pas juste des registres permettant de gérer des I/O ou des registres n'ayant comme fonction que d'être lus ou écrit. Le contenu du registre IM sera utilisé dans la logique VHDL pour déclencher une irq et le registre EC sera peuplé dans la logique VHDL en fonction des Keys appuyés.

2.3 Adress Map finale

Offset	Read D31...0	Write D31...0
0x000	[31...0] const (0xDEADBEEF)	not used
0x004	[31...0] reg_test_rw	[31...0] reg_test_rw
0x008	[31...10] '0...0' [9...0] Leds9...0	[31...10] reserved [9...0] Leds9...0
0x00C	[31] '0' [30...24] Hex3 [23] '0' [22...16] Hex2 [15] '0' [14...8] Hex1 [7] '0' [6...0] Hex0	[31] reserved [30...24] Hex3 [23] reserved [22...16] Hex2 [15] reserved [14...8] Hex1 [7] reserved [6...0] Hex0
0x010	[31...15] '0' [14...8] Hex5 [7] '0' [6...0] Hex4	[31...15] reserved [14...8] Hex5 [7] reserved [6...0] Hex4
0x014	[31...10] '0...0' [9...0] Switchs9...0	not used
0x018	[31...4] '0...0' [3...0] Keys3...0	not used
0x01C	[31...4] '0...0' [3...0] Keys_IM3...0	[31...4] reserved [3...0] Keys_IM3...0
0x020	[31...4] '0...0' [3...0] Keys_EC3...0	[31...4] reserved [3...0] Keys_EC3...0

J'ai essayé d'avoir un plan d'adressage le plus compact possible. Dans le plan de base, j'avais prévu de mettre le registre gérant les keys à la fin afin d'avoir une certaine logique dans l'adressage des registres permettant la gestion des interruptions par la suite. L'adresse de base à laquelle les offsets viennent s'ajouter est décidée dans QSys, et l'on peut raisonnablement prédire qu'elle sera la même que celle du laboratoire 2, soit l'adresse du bus h2f_lw, 0xFF200000.

3 Réalisation et implémentation

J'ai choisi de ne pas présenter l'implémentation faite partie par partie, mais de présenter l'implémentation finale car elle contient les éléments correspondant à chacune des parties.

3.1 Création de l'IP axi lite

Une base de code nous était fournie dans ce laboratoire. J'ai commenté le code afin d'expliquer son fonctionnement, j'ai mis en annexe le résultat obtenu. Dans la

partie simulation, nous parlerons des tests effectués avec vsim, mais ici je n'ai pas grand-chose à ajouter, le code ayant été commenté dans son entier.

3.2 Ajout de l'IP dans Qsys

Certaines informations nous étaient fournies dans cette partie. J'ai donc respecté les consignes données, tout en rajoutant certaines choses :

- Ajout du composant dans QSys
- Ajout du fichier vhdl correspondant à l'IP dans le nouveau composant
- Appui sur le bouton "Analyze Synthesis Files" afin de récupérer les signaux
- Ajout des interfaces (altera_axi4lite_slave, clock_sink, conduit_end, interrupt_sender, reset_sender)
- Assignment et configuration des signaux de l'IP aux interfaces

Le résultat final dans l'onglet "Signals & Interfaces" est le suivant :

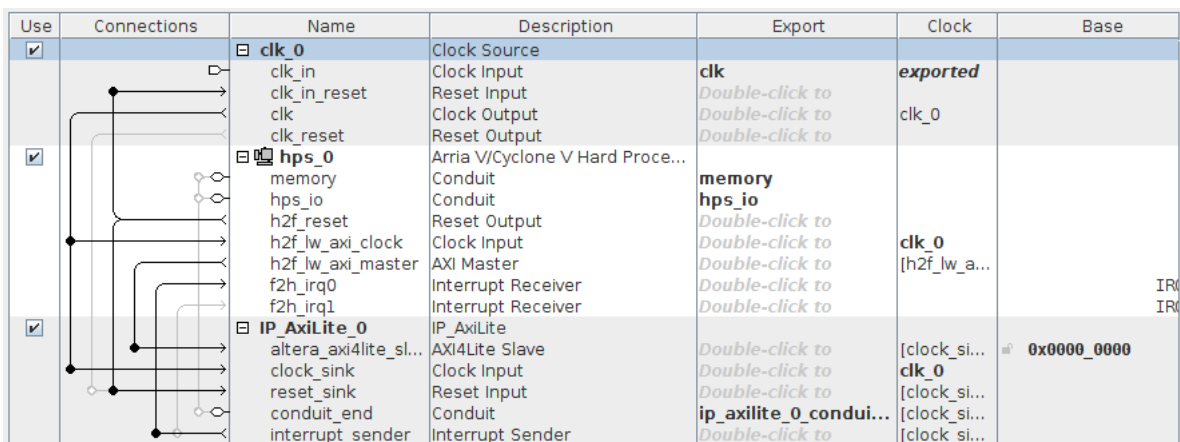
Name
▶ altera_axi4lite_slave AXI4Lite Slave
▶ axi_araddr_i [12] araddr
▶ axi_arprot_i [3] arprot
▶ axi_arready_o [1] arready
▶ axi_arvalid_i [1] arvalid
▶ axi_awaddr_i [12] awaddr
▶ axi_awprot_i [3] awprot
▶ axi_awready_o [1] awready
▶ axi_awvalid_i [1] awvalid
▶ axi_bready_i [1] bready
▶ axi_bresp_o [2] bresp
▶ axi_bvalid_o [1] bvalid
▶ axi_rdata_o [32] rdata
▶ axi_rready_i [1] rready
▶ axi_rresp_o [2] rresp
▶ axi_rvalid_o [1] rvalid
▶ axi_wdata_i [32] wdata
▶ axi_wready_o [1] wready
▶ axi_wstrb_i [4] wstrb
▶ axi_wvalid_i [1] wvalid
▶ clock_sink Clock Input
▶ axi_clk_i [1] clk
▶ conduit_end Conduit
▶ hex03_o [32] hex03_output
▶ hex54_o [32] hex54_output
▶ keys_i [32] keys_input
▶ leds_o [32] leds_output
▶ switch_i [32] switch_input
<<add signal>>
▶ interrupt_sender Interrupt Sender
▶ irq_o [1] irq
▶ reset_sink Reset Input
▶ axi_reset_i [1] reset
<<add signal>>
<<add interface>>

FIGURE 1 – Laboratoire 5 : Signals et Interfaces

3.3 Actions dans le projet Qsys

Comme dans le laboratoire 2, certaines actions devaient être effectuées dans Qsys afin de permettre le fonctionnement du programme :

- Il faut activer dans la configuration de hps_0 , sous "Fpga interface" -> "interrupts", l'option "Enable FPGA-to-HPS interrupts"
- L'IP qui doit bénéficier des interruptions doit être linké sur l'un des deux champs "f2h_irq0/1".
- Dans la colonne IRQ aussi, l'interruption de l'interrupt sender de l'IP doit être liée sur l'un des deux champs vu plus haut. Dans notre cas, l'interruption porte le numéro 0 et est liée sur "f2h_irq0", ce qui implique que le numéro de l'interruption sera le numéro 0 de la fpga(numéro d'interruption 72)
- Les autres interfaces de l'IP doivent être câblé sur les bons éléments dans le projet Qsys comme le présente l'image ci-dessous.



Use	Connections	Name	Description	Export	Clock	Base
<input checked="" type="checkbox"/>		clk_0	Clock Source			
		clk_in	Clock Input	clk	exported	
		clk_in_reset	Reset Input	Double-click to		
		clk	Clock Output	Double-click to	clk_0	
		clk_reset	Reset Output	Double-click to		
<input checked="" type="checkbox"/>		hps_0	Arria V/Cyclone V Hard Proce...			
		memory	Conduit	memory		
		hps_io	Conduit	hps_io		
		h2f_reset	Reset Output	Double-click to		
		h2f_lw_axi_clock	Clock Input	Double-click to	clk_0	
		h2f_lw_axi_master	AXI Master	Double-click to	[h2f_lw_a...	
		f2h_irq0	Interrupt Receiver	Double-click to		IRQ
		f2h_irq1	Interrupt Receiver	Double-click to		IRQ
<input checked="" type="checkbox"/>		IP_Axilite_0	IP_Axilite			
		altera_axi4lite_sl...	AXI4Lite Slave	Double-click to	[clock_si...	0x0000_0000
		clock_sink	Clock Input	Double-click to	clk_0	
		reset_sink	Reset Input	Double-click to	[clock_si...	
		conduit_end	Conduit	Double-click to	[clock_si...	
		interrupt_sender	Interrupt Sender	Double-click to	[clock_si...	

FIGURE 2 – Laboratoire 5 : Projet Qsys

Comme on peut le voir, l'interface Axilite est câblé sur le bus h2f_lw_axi_master avec comme base 0x00000000, ce qui veut dire que l'adresse de base à laquelle il faut ajouter les offsets du plan d'adressage sera l'adresse du début du bus h2f_lw, soit 0xFF200000.

3.4 Implémentation dans le fichier top du projet

Après avoir généré le code VHDL du projet Qsys, je devais ajouter le composant au fichier top du projet Quartus. J'ai copié le résultat obtenu avec Qsys pour l'attribution des sorties et entrées du composant créé, et j'ai dû ajouter des traitements supplémentaires, voici lesdites modifications :

```

1      ...
2
3      signal temp_hex03_s : std_logic_vector(31 downto 0);
4      signal temp_hex45_s : std_logic_vector(31 downto 0);
5      signal temp_leds_s : std_logic_vector(31 downto 0);
6      signal temp_key : std_logic_vector(31 downto 0);
7      signal temp_sw : std_logic_vector(31 downto 0);
8      signal temp_zero_key : std_logic_vector(31-4 downto 0);
9      signal temp_zero_sw : std_logic_vector(31-10 downto 0);
10

```

```

11 begin
12
13     -----
14     -- HPS mapping
15     -----
16     temp_zero_key <= (others => '0');
17     temp_zero_sw <= (others => '0');
18     temp_key <= temp_zero_key & KEY_i ;
19     temp_sw <= temp_zero_sw & SW_i;
20
21     System : component qsys_system
22     ...
23     -- User input-output
24     ip_axilite_0_conduit_end_hex03_output : out    ...
25         std_logic_vector(31 downto 0);           -- ...
26         hex03_output
27     ip_axilite_0_conduit_end_hex54_output : out    ...
28         std_logic_vector(31 downto 0);           -- ...
29         hex54_output
30     ip_axilite_0_conduit_end_keys_input  : in      ...
31         std_logic_vector(31 downto 0) := (others => 'X'); -- ...
32         keys_input
33     ip_axilite_0_conduit_end_leds_output : out      ...
34         std_logic_vector(31 downto 0);           -- ...
35         leds_output
36     ip_axilite_0_conduit_end_switch_input : in      ...
37         std_logic_vector(31 downto 0) := (others => 'X'); -- ...
38         switch_input
39     ...
40 );
41
42     HEX0_o <= temp_hex03_s(6 downto 0);
43     HEX1_o <= temp_hex03_s(14 downto 8);
44     HEX2_o <= temp_hex03_s(22 downto 16);
45     HEX3_o <= temp_hex03_s(30 downto 24);
46     HEX4_o <= temp_hex45_s(6 downto 0);
47     HEX5_o <= temp_hex45_s(14 downto 8);
48     LEDR_o <= temp_leds_s(9 downto 0);
49 end top;

```

Le premier traitement, celui concernant les keys et les switches, permet de formater les inputs des keys et des switches reçus afin de nous assurer que le plan d'adressage soit respecté et que les bits devant être lu comme '0' le soit bien. Le traitement concernant les afficheurs 7 segments et les leds permet de réduire la taille des valeurs reçues depuis le composant afin de pouvoir les adapter à la taille des sorties.

3.5 Code C

Le code C ressemble beaucoup au code C du laboratoire précédent, le laboratoire 2. La seule différence résidant au final dans la gestion de deux afficheurs 7 segments en plus, mais la méthode utilisée reste la même. J'ai tout de même modifié légèrement le code afin de réduire le nombre de lectures effectuées, suite aux remarques concernant le laboratoire précédent. Le code C peut être trouvé dans le projet, je ne vais pas plus

en parler ici, la gestion des interruptions se fait de la même manière. Les adresses utilisées ont dû être légèrement modifiée afin de coller avec le plan d'adressage.

4 Simulation et tests

Cette partie contient les différents tests et simulations effectuées au cours du projet

4.1 Contrôle des timings

J'ai utilisé vsim afin de contrôler que mes timings étaient bons en me basant sur les timings montrés dans le document "Designing a AXI4-lite Slave Peripheral, Griffin, Xilinx", qui sont les suivants :

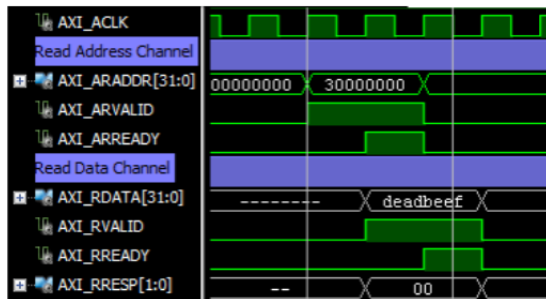


FIGURE 3 – Laboratoire 5 : Timing read

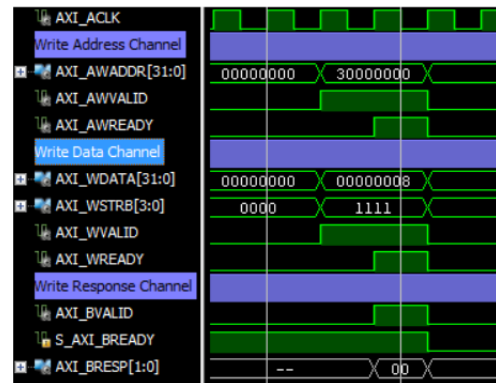


FIGURE 4 – Laboratoire 5 : Timing write

Et maintenant, voici les timings que j'ai obtenus lors de la simulation :

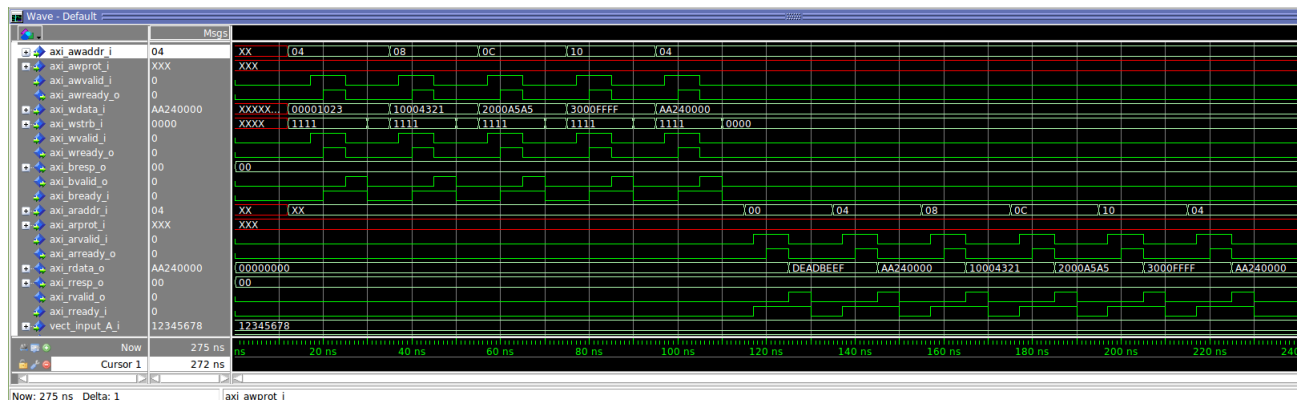


FIGURE 5 – Laboratoire 5 : Timing implémentation bus axi lite

On peut voir que le timing du read correspond au timing voulu à l'exception du fait que dans notre cas la durée d'activation d'axi_rvalid et de axi_bready est inversé. Cela n'impacte pas le fonctionnement du programme, les deux signaux se retrouvant désactivés au même moment. Par contre, le timing du canal de réponse du write est décalé d'un coup d'horloge. Cela n'a pas d'impact sur le fonctionnement du bus, étant donnée que dans le cadre l'écriture, nous pouvons nous permettre de prendre

un peu plus de temps, car on nous fournit une donnée que nous devons stocker, ce n'est pas nous qui devons fournir la donnée sur le bus de lecture.

L'écriture et la lecture fonctionnent correctement dans cette simulation, les registres et bus étant mis à jour et les timings respectés.

4.2 Test du strobe

En modifiant le testbench, il était possible de réaliser des tests sur le fonctionnement du strobe. Voici les résultats obtenus :

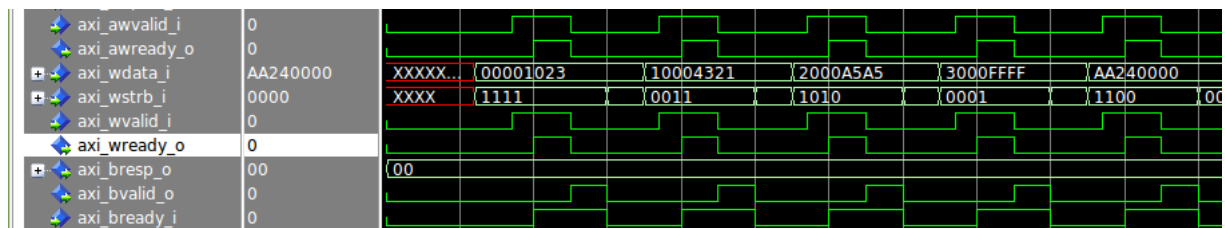


FIGURE 6 – Laboratoire 5 : Write avec différents strobes

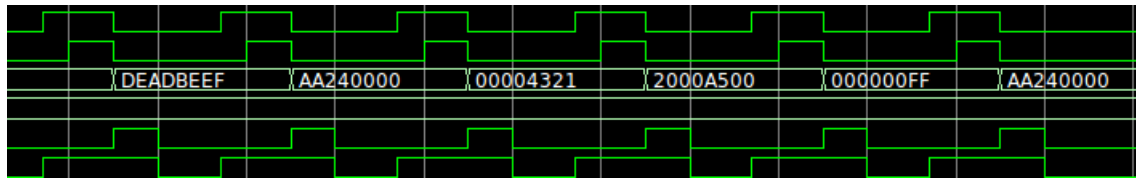


FIGURE 7 – Laboratoire 5 : Read après différents strobes

Je rends attentif ici au fait que le registre à l'offset 04 est écrit deux fois, une fois au tout début de la phase d'écriture et une fois à la fin de la phase d'écriture, ce qui explique la valeur 0xAA240000 lue, valeur qui respecte le strobe qui lui était attribué (1100). On peut voir que les valeurs respectent les strobes qui ont été utilisés lors de l'écriture quand on les lit.

4.3 Problèmes rencontrés

Durant ce laboratoire, j'ai rencontré un problème principal, qui était un problème de timing. Pour imaginer cela, je vais insérer ici une image présentant la phase de debug que j'ai dû effectuer pour m'en rendre compte. J'ai utilisé SignalTap 2 :

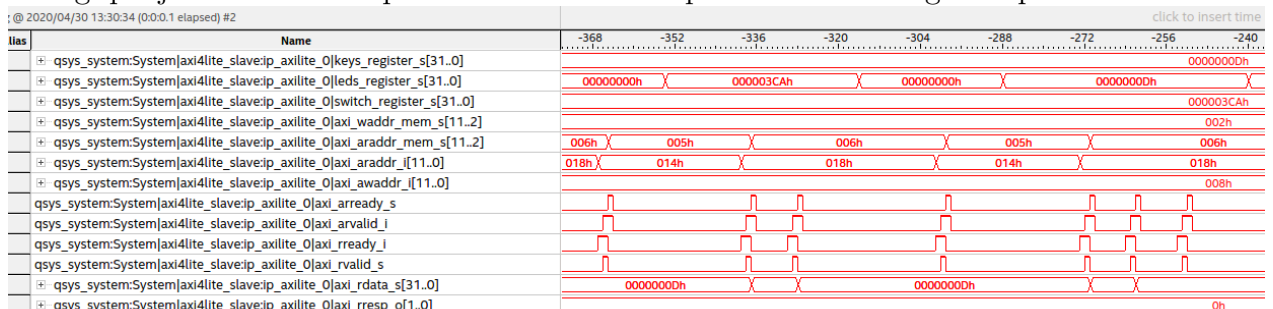


FIGURE 8 – Laboratoire 5 : Read Signal Tap 2

On peut voir que les signaux arready et arvalid sont activés après les signaux

ready et rvalid, ce qui au vu des timings étudiés plus tôt représente une erreur d'implémentation. J'ai pu régler ce problème dans le code VHDL par la suite, pour arriver à l'implémentation finale qui vous a été présentée.

4.4 Tests sur board

Voici les tests effectués sur la board :

Comme dans le dernier laboratoire, j'ai commencer par effectuer un appui sur key 0 et un appui sur key 1 en gardant les switches down :

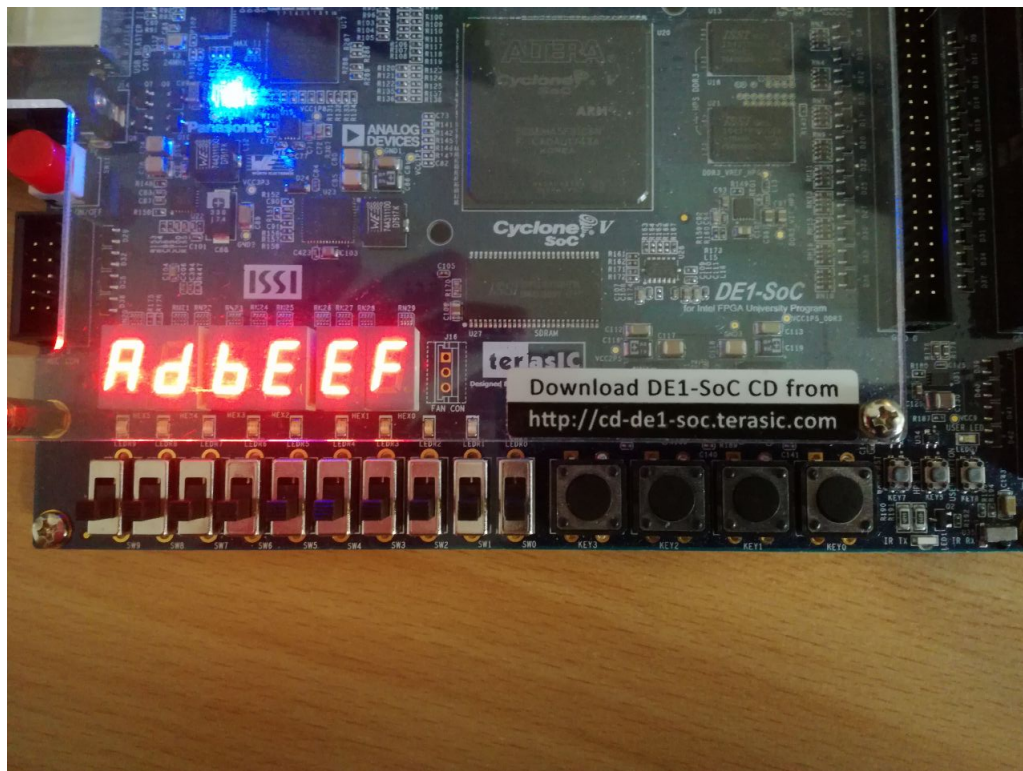


FIGURE 9 – Laboratoire 5 : Appuis sur key 0, switch down

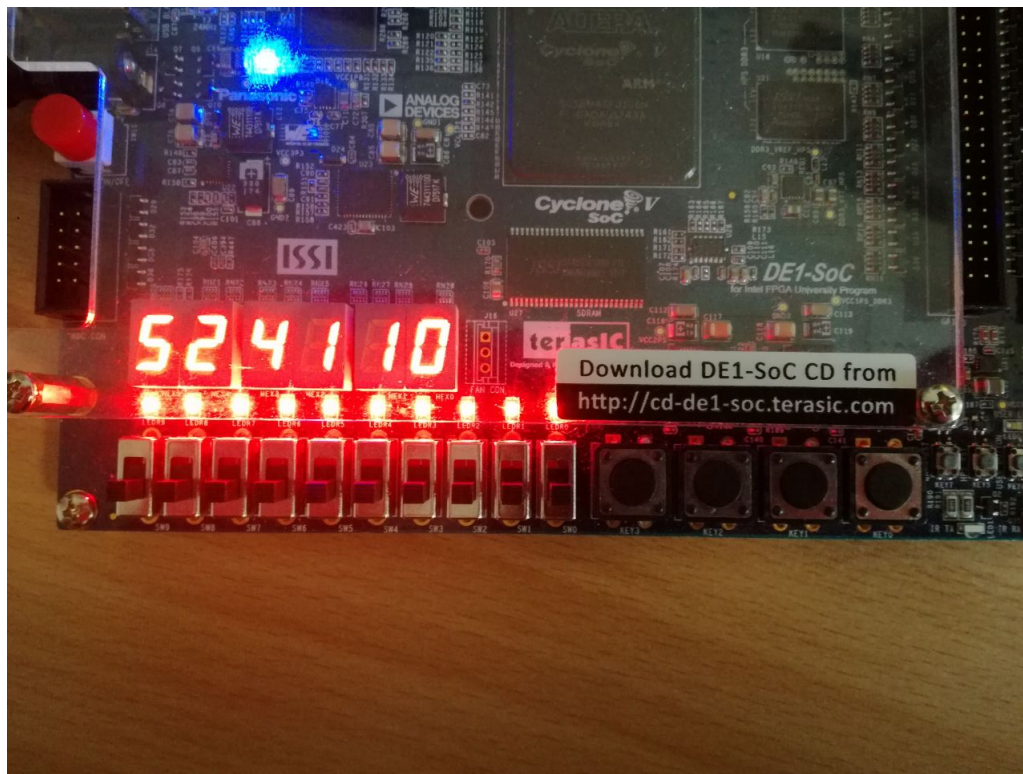


FIGURE 10 – Laboratoire 5 : Appuis sur key 1, switch down
La spécification des leds est respectée, ainsi que la spécification des afficheurs 7 segments. la constante étant (0xDEADBEEF), on voit apparaître l'état des bits 23 à 0 dans le cas d'un appui sur key 0 (ADBEEF), et l'état inverse dans le cas d'un appui sur key 1. J'ai par la suite alterné les switch en en laissant un up et un down, en commençant avec le switch 9 down, et j'ai répéter la même action qu'auparavant :

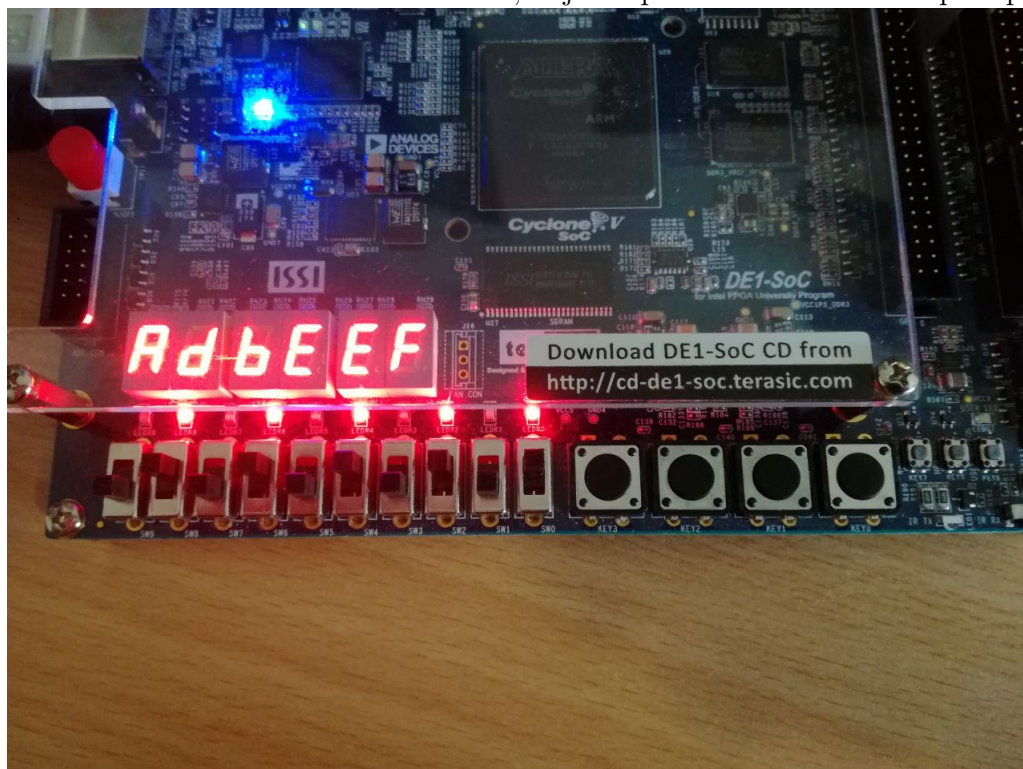


FIGURE 11 – Laboratoire 5 : Appuis sur key 0, switch alternés

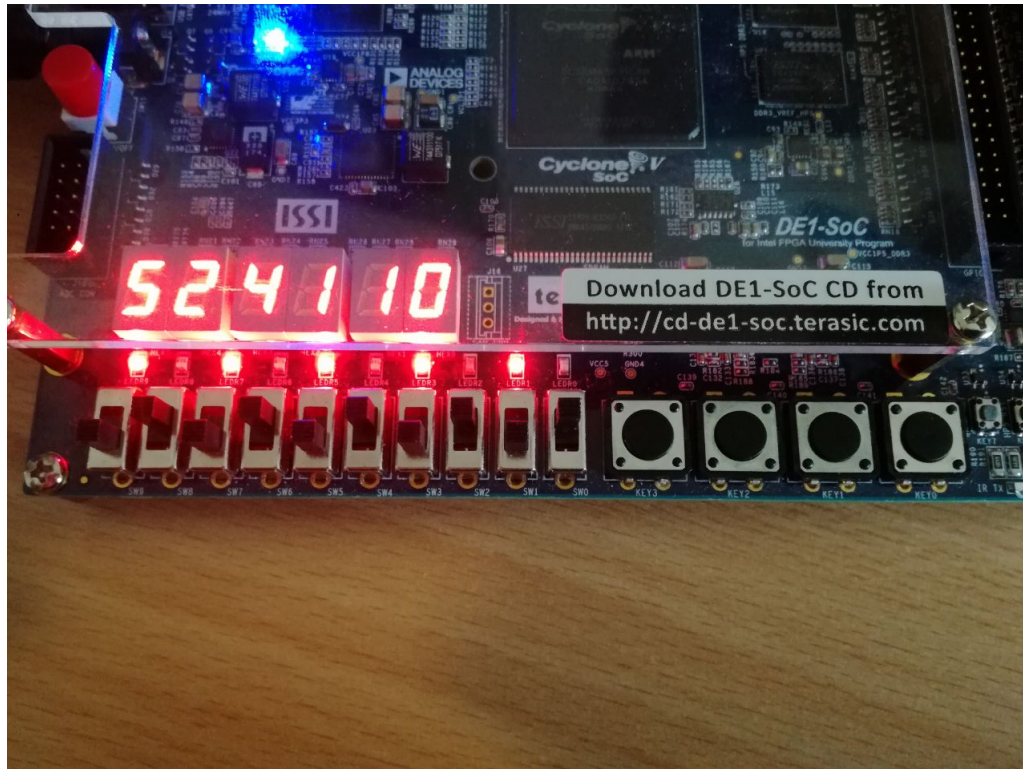


FIGURE 12 – Laboratoire 5 : Appuis sur key 1, switch alternés

On peut voir ici que le fonctionnement est correct. J'ai finalement testé les interruptions en appuyant 2 fois sur le bouton key 3 puis 1 fois sur le bouton key 2 :

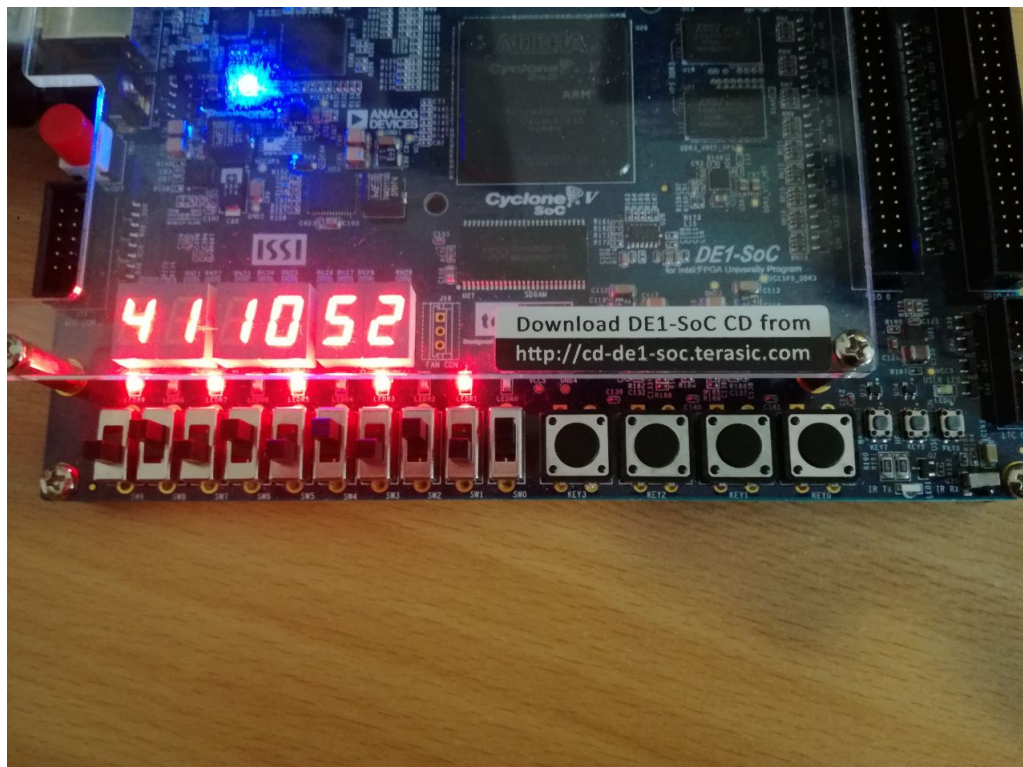


FIGURE 13 – Laboratoire 5 : 2 Appuis sur key 3

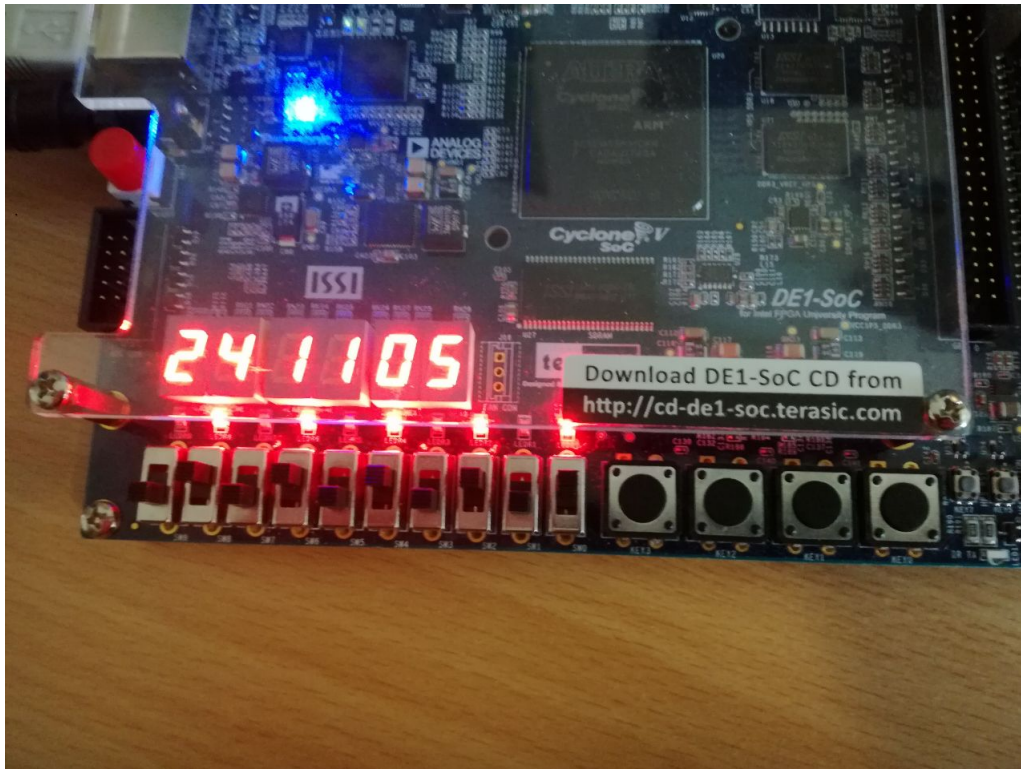


FIGURE 14 – Laboratoire 5 : 1 Appuis sur key 2

Ici à nouveau, le fonctionnement est correct et le résultat correspond à la spécification fournie.

5 Conclusion

Ce laboratoire nous a permis de comprendre le principe d'ajout d'IP à un projet existant. De plus, il nous a permis de comprendre les concepts et les timings qui se retrouvent impliqués dans un bus Axi lite. La partie la plus dure a été pour moi la gestion des timings, du moins c'est la partie qui m'a pris le plus de temps afin d'obtenir un résultat correct. Ce laboratoire m'a de plus permis de me familiariser avec l'outil Signal Tap 2, ce qui est un plus pour la suite.

6 Signatures

Yverdon-les-Bains le 12 mai 2020

Pierrick Muller

Table des figures

1	Laboratoire 5 : Signals et Interfaces	6
---	---------------------------------------	---

2	Laboratoire 5 : Projet Qsys	7
3	Laboratoire 5 : Timing read	9
4	Laboratoire 5 : Timing write	9
5	Laboratoire 5 : Timing implémentation bus axi lite	9
6	Laboratoire 5 : Write avec différents strobes	10
7	Laboratoire 5 : Read après different strobes	10
8	Laboratoire 5 : Read Signal Tap 2	10
9	Laboratoire 5 : Appuis sur key 0, switch down	11
10	Laboratoire 5 : Appuis sur key 1, switch down	12
11	Laboratoire 5 : Appuis sur key 0, switch alternés	12
12	Laboratoire 5 : Appuis sur key 1, switch alternés	13
13	Laboratoire 5 : 2 Appuis sur key 3	13
14	Laboratoire 5 : 1 Appuis sur key 2	14

Annexe

```
-----
-- HEIG-VD, Haute Ecole d'Ingenierie et de Gestion du canton de Vaud
-- Institut REDS, Reconfigurable & Embedded Digital Systems
--
-- File      : axi4lite_slave.vhd
-- Author    : E. Messerli    27.07.2017
-- Description : slave interface AXI (without burst)
-- used for   : SOCF lab
--| Modifications |-----
-- Ver  Date      Auteur  Description
-- 1.0  26.03.2019  EMI     Adaptation du chablon pour les etudiants
-- 1.1  08.05.2029  PMR     Complétion du laboratoire
-----

library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

entity axi4lite_slave is
    generic (
        -- Users to add parameters here

        -- User parameters ends

        -- Width of S_AXI data bus
        AXI_DATA_WIDTH : integer := 32; -- 32 or 64 bits
        -- Width of S_AXI address bus
        AXI_ADDR_WIDTH  : integer := 12
    );
    port (
        axi_clk_i      : in  std_logic;
        axi_reset_i     : in  std_logic;
        -- AXI4-Lite
        axi_awaddr_i    : in  std_logic_vector(AXI_ADDR_WIDTH-1 downto 0);
        axi_awprot_i    : in  std_logic_vector( 2 downto 0); -- Pas utilisés
        axi_awvalid_i   : in  std_logic;
        axi_awready_o    : out std_logic;
        axi_wdata_i     : in  std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
        axi_wstrb_i     : in  std_logic_vector((AXI_DATA_WIDTH/8)-1 downto 0);
        axi_wvalid_i    : in  std_logic;
        axi_wready_o    : out std_logic;
        axi_bresp_o     : out std_logic_vector(1 downto 0);
        axi_bvalid_o    : out std_logic;
        axi_bready_i    : in  std_logic;
        axi_araddr_i    : in  std_logic_vector(AXI_ADDR_WIDTH-1 downto 0);
        axi_arprot_i    : in  std_logic_vector( 2 downto 0); -- Pas utilisés
        axi_arvalid_i   : in  std_logic;
        axi_arready_o    : out std_logic;
        axi_rdata_o     : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
        axi_rresp_o     : out std_logic_vector(1 downto 0);
```



```

    axi_rvalid_o      : out std_logic;
    axi_rready_i      : in  std_logic;

    -- User input-output

    --TEST POUR TESTBENCH
    --      vect_input_A_i : in std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
    --      vect_input_B_i : in std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
    --      vect_input_C_i : in std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
    --      vect_input_D_i : in std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
    --
    --      output_reg_A_o : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
    --      output_reg_B_o : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
    --      output_reg_C_o : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
    --      output_reg_D_o : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0)

    switch_i          : in std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
    keys_i            : in std_logic_vector(AXI_DATA_WIDTH-1 downto 0);

    leds_o            : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
    hex03_o           : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
    hex54_o           : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
    -- Interruptions

    irq_o             : out std_logic

);
end entity axi4lite_slave;

architecture rtl of axi4lite_slave is

    signal reset_s : std_logic;

    -- local parameter for addressing 32 bit / 64 bits, cst: AXI_DATA_WIDTH
    -- ADDR_LSB is used for addressing word 32/64 bits registers/memories
    -- ADDR_LSB = 2 for 32 bits (n-1 downto 2)
    -- ADDR_LSB = 3 for 64 bits (n-1 downto 3)
    constant ADDR_LSB : integer := (AXI_DATA_WIDTH/32)+ 1;

    --signal for the AXI slave
    --intern signal for output
    signal axi_awready_s      : std_logic;
    signal axi_wready_s       : std_logic;
    signal axi_arready_s      : std_logic;
    signal axi_rvalid_s       : std_logic;
    signal axi_bvalid_s       : std_logic;
    signal axi_bresp_s        : std_logic_vector(1 downto 0);
    signal axi_rdata_s        : std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
    signal axi_wdata_s        : std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
    --intern signal for the axi interface
    signal axi_waddr_mem_s    : std_logic_vector(AXI_ADDR_WIDTH-1 downto
ADDR_LSB);
    signal axi_araddr_mem_s   : std_logic_vector(AXI_ADDR_WIDTH-1 downto
ADDR_LSB);
    signal axi_data_wren_s    : std_logic;
    signal axi_data_reen_s    : std_logic;

```

```

-- signal representing registers
signal const_register_s      : std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
signal test_register_s      : std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
signal leds_register_s      : std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
signal hex03_register_s     : std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
signal hex54_register_s     : std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
signal switch_register_s    : std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
signal keys_register_s      : std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
signal keys_im_register_s   : std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
signal keys_ec_register_s   : std_logic_vector(AXI_DATA_WIDTH-1 downto 0);

-- signaux permettant la gestion du strobe
signal temp_Vect_Strb_1_s : std_logic_vector(AXI_DATA_WIDTH/4 -1 downto 0);
signal temp_Vect_Strb_2_s : std_logic_vector(AXI_DATA_WIDTH/4 -1 downto 0);
signal temp_Vect_Strb_3_s : std_logic_vector(AXI_DATA_WIDTH/4 -1 downto 0);
signal temp_Vect_Strb_4_s : std_logic_vector(AXI_DATA_WIDTH/4 -1 downto 0);

-- signal d'interruption
signal irq_s      : std_logic;

-- signaux de masquage pour la lecture des registres
signal KEY_MASK_s      : std_logic_vector(AXI_DATA_WIDTH - 1 downto 0);
signal SWITCH_LEDS_MASK_s : std_logic_vector(AXI_DATA_WIDTH - 1 downto 0);
signal HEX54_MASK_s    : std_logic_vector(AXI_DATA_WIDTH - 1 downto 0);
signal HEX03_MASK_s    : std_logic_vector(AXI_DATA_WIDTH - 1 downto 0);

--intern signal for adress decoding (Prévu dans la première version)

--signal local_address_write_s      : integer;
--signal local_address_read_s       : integer;
--signal const_register_address_valid_write_s : std_logic;
--signal test_register_address_valid_write_s : std_logic;
--signal leds_register_address_valid_write_s : std_logic;
--signal hex03_register_address_valid_write_s : std_logic;
--signal hex45_register_address_valid_write_s : std_logic;
--signal switch_register_address_valid_write_s : std_logic;
--signal keys_register_address_valid_write_s : std_logic;
--signal const_register_address_valid_read_s : std_logic;
--signal test_register_address_valid_read_s : std_logic;
--signal leds_register_address_valid_read_s : std_logic;
--signal hex03_register_address_valid_read_s : std_logic;
--signal hex45_register_address_valid_read_s : std_logic;
--signal switch_register_address_valid_read_s : std_logic;
--signal keys_register_address_valid_read_s : std_logic;

begin

    reset_s <= axi_reset_i;

-- Ce code ci-dessous présente le premier essai basé sur le document présentant
l'implémentation d'une IP axi_lite

```

```

-----
-- address decoding

-- On récupère l'offset dans l'adresse. On utilise pas les deux bits de poids
faible
--local_address_write_s <= to_integer(unsigned(aci_waddr_mem_s(AXI_ADDR_WIDTH -
1 downto 0)));
--local_address_read_s <= to_integer(unsigned(aci_araddr_mem_s(AXI_ADDR_WIDTH -
1 downto 0)));

--address_range_analysis : process (local_address_write_s, local_address_read_s)
--begin
--  -- write valid
--  const_register_address_valid_write_s <= '0';
--  test_register_address_valid_write_s <= '0';
--  leds_register_address_valid_write_s <= '0';
--  hex03_register_address_valid_write_s <= '0';
--  hex45_register_address_valid_write_s <= '0';
--  switch_register_address_valid_write_s <= '0';
--  keys_register_address_valid_write_s <= '0';
--
--  -- read valid
--  const_register_address_valid_read_s <= '0';
--  test_register_address_valid_read_s <= '0';
--  leds_register_address_valid_read_s <= '0';
--  hex03_register_address_valid_read_s <= '0';
--  hex45_register_address_valid_read_s <= '0';
--  switch_register_address_valid_read_s <= '0';
--  keys_register_address_valid_read_s <= '0';
--
--  case (local_address_write_s) is
--    when 0 => const_register_address_valid_write_s <= '1';
--    when 1 => test_register_address_valid_write_s <= '1';
--    when 2 => leds_register_address_valid_write_s <= '1';
--    when 3 => hex03_register_address_valid_write_s <= '1';
--    when 4 => hex45_register_address_valid_write_s <= '1';
--    when 5 => switch_register_address_valid_write_s <= '1';
--    when 6 => keys_register_address_valid_write_s <= '1';
--    when others => NULL;
--  end case;
--
--  case (local_address_read_s) is
--    when 0 => const_register_address_valid_read_s <= '1';
--    when 1 => test_register_address_valid_read_s <= '1';
--    when 2 => leds_register_address_valid_read_s <= '1';
--    when 3 => hex03_register_address_valid_read_s <= '1';
--    when 4 => hex45_register_address_valid_read_s <= '1';
--    when 5 => switch_register_address_valid_read_s <= '1';
--    when 6 => keys_register_address_valid_read_s <= '1';
--    when others => NULL;
--  end case;
--end process;
--
-----
-- Write address channel

-- GESTION DU STROBE

```

```

-- On récupère les bits voulus en fonction du strobe et on assemble les
résultats dans un signal temporaire (axi_wdata_s)
temp_Vect_Strb_1_s <= axi_wdata_i(7 downto 0) when axi_wstrb_i(0) = '1' else
(others => '0');
temp_Vect_Strb_2_s <= axi_wdata_i(15 downto 8) when axi_wstrb_i(1) = '1'
else (others => '0');
temp_Vect_Strb_3_s <= axi_wdata_i(23 downto 16) when axi_wstrb_i(2) = '1'
else (others => '0');
temp_Vect_Strb_4_s <= axi_wdata_i(31 downto 24) when axi_wstrb_i(3) = '1'
else (others => '0');
axi_wdata_s <= temp_Vect_Strb_4_s & temp_Vect_Strb_3_s & temp_Vect_Strb_2_s
& temp_Vect_Strb_1_s;

process (reset_s, axi_clk_i)
begin
    if reset_s = '1' then
        axi_awready_s <= '0';
        axi_waddr_mem_s <= (others => '0');
    elsif rising_edge(axi_clk_i) then
        if (axi_awready_s = '0' and axi_awvalid_i = '1') then --and
axi_wvalid_i = '1') then  modif EMI 10juil2018
            -- slave is ready to accept write address when
            -- there is a valid write address
            axi_awready_s <= '1';
            -- Write Address memorizing
            axi_waddr_mem_s <= axi_awaddr_i(AXI_ADDR_WIDTH-1 downto
ADDR_LSB);
        else
            axi_awready_s <= '0';
        end if;
    end if;
end process;
axi_awready_o <= axi_awready_s;

-----
-- Write data channel

-- Implement axi_wready generation
process (reset_s, axi_clk_i)
begin
    if reset_s = '1' then
        axi_wready_s <= '0';
    elsif rising_edge(axi_clk_i) then
        if(axi_wready_s = '0' and axi_wvalid_i = '1') then
            axi_wready_s <= '1';
        else
            axi_wready_s <= '0';
        end if;
    end if;
end process;

-- Assignment de la sortie axi_wready_o
axi_wready_o <= axi_wready_s;

```

```

--condition to write data. On peut écrire des données quand axi_wready_s et
axi_wvalid_i sont actif (Selon documentation axi_lite)
axi_data_wren_s <= '0' when (axi_wready_s = '1' and axi_wvalid_i = '1') else
    '1';

process (reset_s, axi_clk_i)
    --number address to access 32 or 64 bits data
    variable int_waddr_v : natural;
begin
    if reset_s = '1' then
        --Assignment des valeurs de base aux registres et au signal
d'interruption lors du reset
        test_register_s <= (others => '0');--vect_input_A_i;
        const_register_s <= x"DEADBEEF";
        leds_register_s <= x"00000000";
        hex03_register_s <= x"00000000";
        hex54_register_s <= x"00000000";
        switch_register_s <= switch_i;--vect_input_B_i;
        keys_register_s <= keys_i;--vect_input_C_i;
        keys_im_register_s <= (others => '0');
        keys_ec_register_s <= (others => '0');
        irq_s <= '0';

        -- création des masques
        KEY_MASK_s <= (others => '0');
        SWITCH_LEDS_MASK_s <= (others => '0');
        HEX03_MASK_s <= (others => '1');
        HEX54_MASK_s <= (others => '1');

        KEY_MASK_s(3 downto 0) <= (others => '1');
        SWITCH_LEDS_MASK_s(9 downto 0) <= (others => '1');
        HEX03_MASK_s(31) <= '0';
        HEX03_MASK_s(23) <= '0';
        HEX03_MASK_s(15) <= '0';
        HEX03_MASK_s(7) <= '0';
        HEX54_MASK_s(31 downto 15) <= (others => '0');
        HEX54_MASK_s(7) <= '0';
    elsif rising_edge(axi_clk_i) then
        -- Récupération de la valeur des switchs dans le registre du bus
axi_lite correspondant
        switch_register_s <= switch_i;

        -- GESTION DE L'INTERRUPTION
        -- Si un bouton à été appuyé, on regarde quel bouton est actif, on
s'assure que le registre de masque d'interruption est
        -- actif pour le bit correspondant au bouton actif, on remplit le
bit du registre d'edge capture correspondant et on active
        -- le signal générant l'irq
        if(keys_register_s /= keys_i) then
            if(keys_register_s(0) = '0' and keys_i(0) = '1' and
(keys_im_register_s(0) = '1')) then
                keys_ec_register_s(0) <= '1';
                irq_s <= '1';
            end if;
            if(keys_register_s(1) = '0' and keys_i(1) = '1' and
(keys_im_register_s(1) = '1')) then
                keys_ec_register_s(1) <= '1';

```

```

        irq_s <= '1';
    end if;
    if(keys_register_s(2) = '0' and keys_i(2) = '1' and
(keys_im_register_s(2) = '1')) then
        keys_ec_register_s(2) <= '1';
        irq_s <= '1';
    end if;
    if(keys_register_s(3) = '0' and keys_i(3) = '1' and
(keys_im_register_s(3) = '1')) then
        keys_ec_register_s(3) <= '1';
        irq_s <= '1';
    end if;
end if;

-- Récupération de la valeur des boutons dans le registre du bus
axi_lite correspondant
keys_register_s <= keys_i;

-- Si on peut écrire
if axi_data_wren_s = '0' then

    -- On transforme l'adresse mémorisée en entier et on effectue
l'écriture souhaitée en fonction de l'adresse.
    -- J'ai choisit d'appliquer ici un masque assurant que la valeur
écrite suive le plan d'adressage et permettant
    -- de contenir dans les registres les valeurs souhaitées lors
par la suite de la lecture du registre selon le plan
    -- d'adressage. Ce traitement pourrait être fait dans la partie
lecture, mais je préfère avoir le contrôle sur ce
    -- qui est écrit dans mes registres plutôt que de renvoyer lors
de la lecture un contenu modifié du registre afin
    -- de coller au plan d'adressage (Lecture de '0' pour certains
bits). De plus, la gestion des lectures avec les
    -- bits à '0' pour les keys et les switchs est gérée directement
dans le top de la DE1-SOC
    int_waddr_v := to_integer(unsigned(axi_waddr_mem_s));
    case int_waddr_v is
        when 0 => null; -- constante, on écrit pas dedans
        when 1 => test_register_s <= axi_wdata_s; -- Test register
        when 2 => leds_register_s <= axi_wdata_s and
SWITCH_LEDS_MASK_s; -- Leds register
        when 3 => hex03_register_s <= axi_wdata_s and
HEX03_MASK_s; -- HEX3..0 register
        when 4 => hex54_register_s <= axi_wdata_s and
HEX54_MASK_s; -- HEX5..4 register
        when 7 => keys_im_register_s <= axi_wdata_s and
KEY_MASK_s;
        when 8 => keys_ec_register_s <= (others => '0');irq_s <=
'0';
        when others => null; --on écrit pas dedans
    end case;
end if;
end if;
end process;

-- Assignation de l'interruption
irq_o <= irq_s;
-----

```

```

-- Write response channel

process (reset_s , axi_clk_i)
begin
    if reset_s = '1' then
        -- Assignation des valeurs de base pour les signaux du canal réponse de
write
        axi_bvalid_s <= '0';
        axi_bresp_s <= "00";
    elsif rising_edge(axi_clk_i) then
        -- Si on peut écrire, on met le signal bvalid_s actif. La réponse sera
        toujours 00 dans notre cas. Je laisse l'assignation de
        -- la réponse ici dans le cas d'une future modification.
        if axi_data_wren_s = '0' then
            axi_bvalid_s <= '1';
            axi_bresp_s <= "00";
        else
            axi_bvalid_s <= '0';
        end if;
    end if;
end process;

-- Assignation des signaux de réponses
axi_bvalid_o <= axi_bvalid_s;
axi_bresp_o <= axi_bresp_s;

```

```

-- Read address channel

```

```

process (reset_s, axi_clk_i)
begin
    if reset_s = '1' then
        axi_arready_s <= '0';
        axi_araddr_mem_s <= (others => '1');
    elsif rising_edge(axi_clk_i) then
        if axi_arready_s = '0' and axi_arvalid_i = '1' then
            -- indicates that the slave has accepted the valid read address
            axi_arready_s <= '1';
            -- Read Address memorizing
            axi_araddr_mem_s <= axi_araddr_i(AXI_ADDR_WIDTH-1 downto
ADDR_LSB);
        else
            axi_arready_s <= '0';
        end if;
    end if;
end process;

axi_arready_o <= axi_arready_s;

```

```

-- Read data channel

```

```

process (reset_s, axi_clk_i)
    --number address to access 32 or 64 bits data
    variable int_raddr_v : natural;

```

```

begin
    if reset_s = '1' then

        axi_rdata_s <= (others => '0');
        axi_rvalid_s <= '0';
    elsif rising_edge(axi_clk_i) then
        -- Afin de respecter les timings, pour pouvoir remplir le signal
        axi_rdata_s, on doit s'assurer que axi_arready_s et axi_arvalid_i soit à 1
        -- si c'est le cas, on met axi_rvalid_s actif, on récupère l'adresse
        -- recu pour la lecture et on remplit le axi_rdata_s avec le
        -- contenu du registre correspondant.
        if axi_arready_s = '1' and axi_arvalid_i = '1' then
            axi_rvalid_s <= '1';
            int_raddr_v := to_integer(unsigned(axi_araddr_mem_s));
            case int_raddr_v is
                when 0 => axi_rdata_s <= const_register_s; -- constante,
on écrit pas dedans
                when 1 => axi_rdata_s <= test_register_s; -- Test register
                when 2 => axi_rdata_s <= leds_register_s; -- Leds register
                when 3 => axi_rdata_s <= hex03_register_s; -- HEX3..0
register
                when 4 => axi_rdata_s <= hex54_register_s; -- HEX5..4
register
                when 5 => axi_rdata_s <= switch_register_s; -- switch
register
                when 6 => axi_rdata_s <= keys_register_s; -- key
register
                when 7 => axi_rdata_s <= keys_im_register_s; -- interrupt
mask register
                when 8 => axi_rdata_s <= keys_ec_register_s; -- edge
capture register
                when others => null; -- Dans les autres cas, on ne fait
rien
            end case;
        else
            axi_rvalid_s <= '0';
        end if;
    end if;
end process;

-- Assignment des sorties des canaux de lectures
axi_rvalid_o <= axi_rvalid_s;
axi_rdata_o <= axi_rdata_s;
axi_rresp_o <= "00";

-- Assignment pour test-bench
-- output_reg_A_o <= const_register_s; --const_register_s;
-- output_reg_B_o <= test_register_s;
-- output_reg_C_o <= leds_register_s;
-- output_reg_D_o <= hex03_register_s;

-- Assignment des I/O
leds_o <= leds_register_s;
hex03_o <= hex03_register_s;
hex54_o <= hex54_register_s;

```



```
end rtl;
```