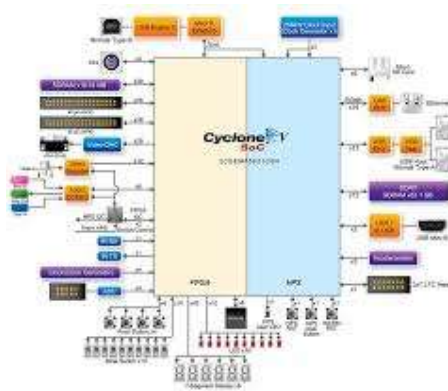


IP AXI4-lite avec I/O de la FPGA

SYSTÈME SOC INTÉGRÉ AVEC FPGA (SOCF)



Auteur : Spinelli Isaia
Prof : Etienne Messerli
Ing : Sébastien Masle
Date : 03.04.2020
Salle : A09 (maison) – HEIG-VD
Classe : SOCF

Table des matières

Introduction.....	- 3 -
Première partie : sans interruption.....	- 3 -
Plan d'adressage.....	- 3 -
Conception	- 4 -
Ecriture	- 5 -
Lecture.....	- 6 -
Description VHDL.....	- 7 -
Test et validation de l'IP	- 8 -
Écriture	- 8 -
Lecture.....	- 9 -
Création du composant.....	- 9 -
Ajout du composant	- 10 -
Modification du top.....	- 11 -
Validation pratique.....	- 11 -
Réalisation de la spécification	- 12 -
Deuxième partie : avec interruption	- 13 -
Plan d'adressage.....	- 13 -
Conception	- 14 -
Test de l'IP	- 15 -
Mise à jour dans Qsys.....	- 15 -
Test de l'IP avec le code C	- 16 -
Réalisation de la spécification	- 16 -
Compléter le code C	- 16 -
Modifier la configuration mémoire.....	- 17 -
Compiler et tester.....	- 17 -
Fonctionnalité de strobe	- 18 -
Description VHDL.....	- 19 -
Test de la fonctionnalité strobe (test bench).....	- 19 -
Test de la fonctionnalité strobe (code)	- 20 -
Supplémentaire : Gestion Edge	- 21 -
Plan d'adressage.....	- 21 -
Description VHDL.....	- 21 -
Mise à jour du projet	- 22 -
Test de fonctionnalité	- 22 -
Annexes	- 23 -

Conclusion	- 23 -
Difficultés rencontrées	- 23 -
Compétences acquises	- 23 -
Résultats obtenus	- 23 -

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 plan d'adressage afin mettre au claire les différents aspects de mon interface.

<i>N</i>	<i>Offset</i>	<i>D32</i> <i>Read</i> <i>0</i>	<i>D32</i> <i>Write</i> <i>0</i>	<i>I / O</i>
0	0x0000 0000	Constante (0xDEADBEEF)	not used	Test
1	0x0000 0004	[31..0] regTest	[31..0] regTest	
2	0x0000 0008	Reserved	Reserved	
3	0x0000 000C	Reserved	Reserved	
4	0x0000 0010	Reserved	Reserved	
5	0x0000 0014	Reserved	Reserved	
...	...	Reserved	Reserved	
64	0x0000 0100	[31..10] '0..0' - [9..0] dataLEDs (9..0)	[31..10] reserved - [9..0] dataLEDs (9..0)	Leds
...	...	Reserved	Reserved	
128	0x0000 0200	[31..4] '0..0' - [3..0] dataKeys (3..0)	not used	Keys
129	0x0000 0204			
...	...	Reserved	Reserved	
192	0x0000 0300	[31..10] '0..0' - [9..0] dataSwitchs (9..0)	not used	Switchs
...	...	Reserved	Reserved	
256	0x0000 0400	[31..7] '0..0' - [6..0] dataHEX0 (6..0)	[31..7] reserved - [6..0] dataHEX0 (6..0)	7seg (Le point n'est pas connecté)
260	0x0000 0410	[31..7] '0..0' - [6..0] dataHEX1 (6..0)	[31..7] reserved - [6..0] dataHEX1 (6..0)	
264	0x0000 0420	[31..7] '0..0' - [6..0] dataHEX2 (6..0)	[31..7] reserved - [6..0] dataHEX2 (6..0)	
268	0x0000 0430	[31..7] '0..0' - [6..0] dataHEX3 (6..0)	[31..7] reserved - [6..0] dataHEX3 (6..0)	
272	0x0000 0440	[31..7] '0..0' - [6..0] dataHEX4 (6..0)	[31..7] reserved - [6..0] dataHEX4 (6..0)	
276	0x0000 0450	[31..7] '0..0' - [6..0] dataHEX5 (6..0)	[31..7] reserved - [6..0] dataHEX5 (6..0)	
...	...	Reserved	Reserved	
1023	0x0000 0FFF	Reserved	Reserved	

Figure 0-1 : Plan d'adressage (partie 1)

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 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 0xffff 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 :

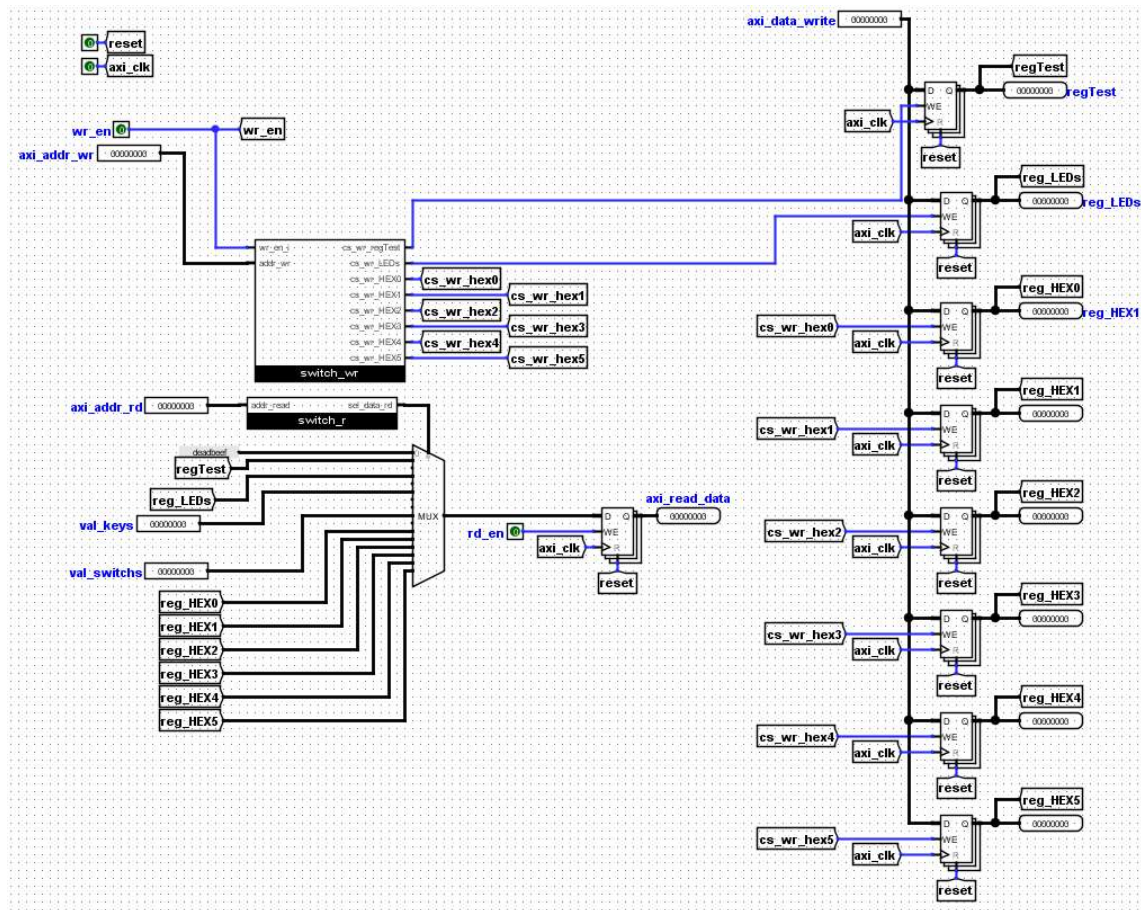


Figure 0-2 : Schéma grossier de l'interface

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 0-3 : 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 connaître 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 0-4: 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 :

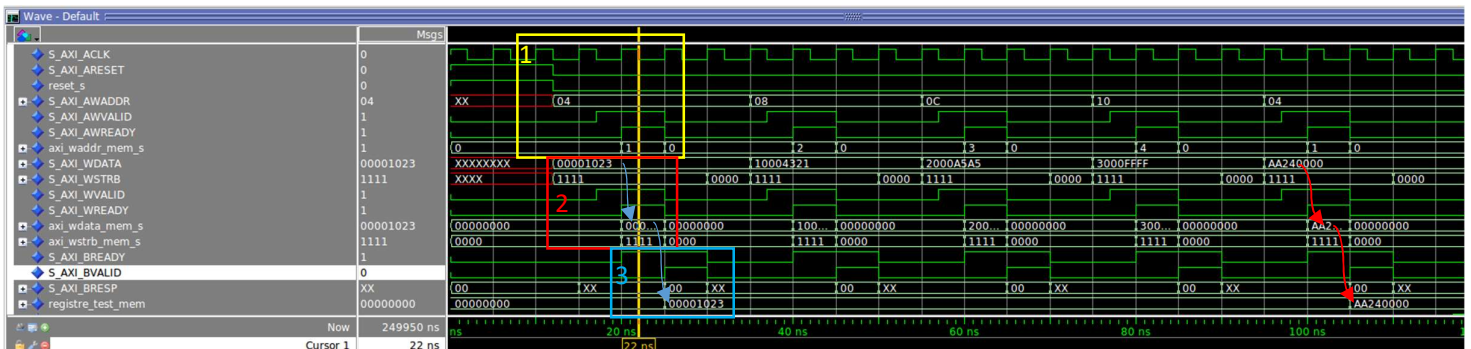


Figure 0-5 : Écriture avec le test Bench

1. Après le reset du début, on peut voir qu'un accès d'écriture va 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 prêt (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 correspondant (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 écritures se passent correctement mais elles sont faites à des adresses pas prises 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 décidé 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_mem_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.

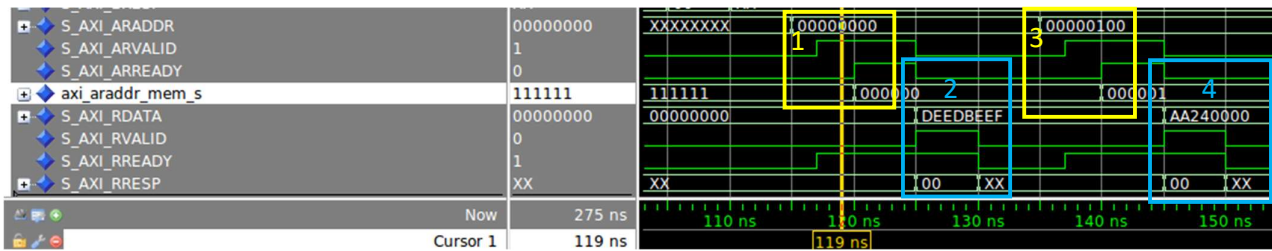


Figure 0-6 : Lecture avec le test Bench

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 enregistré 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 0-7) est maintenant relue.

On peut voir que la chaîne complète fonctionne, écriture suivie d'une lecture grâce au registre de test à l'offset 0x4. Maintenant que d'après le test bench mon IP fonctionne correctement je souhaite le vérifier à l'aide d'un petit code C qui permettra 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 correspondaient pas.

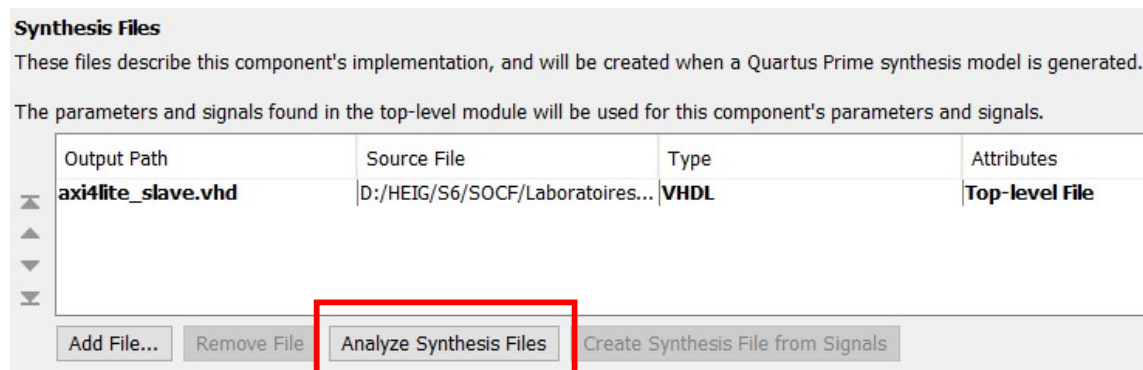


Figure 0-8 : Bouton oublié lors de la création de l'IP

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 :

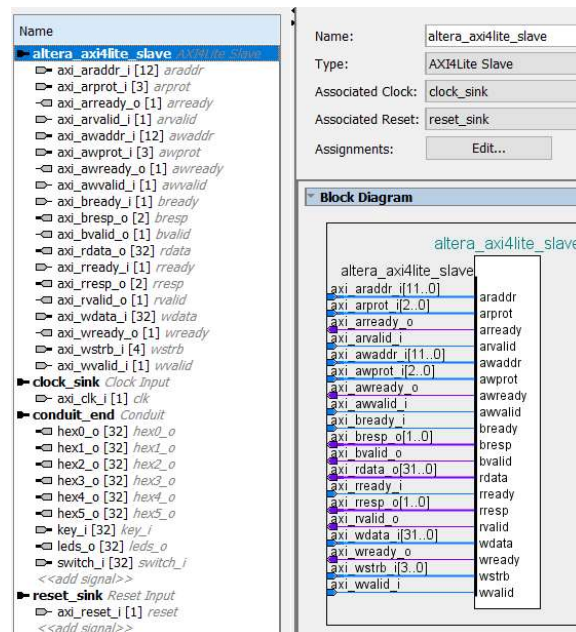


Figure 0-9 : Signaux et interfaces

Après avoir ajouté les composants AXI4Lite Slave, Clock Input, 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 :

Use	Connections	Name	Description	Export	Clock	Base	End
<input checked="" type="checkbox"/>		clk_0	Clock Source Clock Input Reset Input Clock Output Reset Output	clk <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	exported clk_0		
<input checked="" type="checkbox"/>		hps_0	Arria V/Cyclone V Hard Process...	memory hps_io <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>			
<input checked="" type="checkbox"/>		AXI4_lite_perso_0	AXI Spinelli AXI4Lite Slave Reset Input Conduit	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> conduit_export	[clock_sink] clk_0 [clock_sink]	0x0000_0000	0x0000_0fff

Figure 0-10 : 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 :

conduit_export_switch_i (9 downto 0)	=> SW_i ,	-- switch_i
conduit_export_switch_i (31 downto 10)	=> (others => '0'),	
conduit_export_key_i (3 downto 0)	=> KEY_i ,	-- key_i
conduit_export_key_i (31 downto 4)	=> (others => '0'),	
conduit_export_leds_o (9 downto 0)	=> LEDR_o ,	-- leds_o
conduit_export_hex0_o (6 downto 0)	=> HEX0_o ,	-- hex0_o
conduit_export_hex1_o (6 downto 0)	=> HEX1_o ,	-- hex1_o
conduit_export_hex2_o (6 downto 0)	=> HEX2_o ,	-- hex2_o
conduit_export_hex3_o (6 downto 0)	=> HEX3_o ,	-- hex3_o
conduit_export_hex4_o (6 downto 0)	=> HEX4_o ,	-- hex4_o
conduit_export_hex5_o (6 downto 0)	=> HEX5_o ,	-- hex5_o

Figure 0-11 : Mapping du Top

On peut voir ici que j'ai décidé de créer 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 utilisés 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 écrit quelques lignes de code C afin de tester que l'écriture ainsi que la lecture se déroulent correctement. Voici le code que j'ai testé :

```

93
94  AXI_HEX5 = 0x40;
95  AXI_HEX4 = 0xF9;
96  AXI_HEX3 = 0x24;
97  AXI_HEX2 = 0x30;
98  AXI_HEX1 = 0x19;
99  AXI_HEX0 = 0x12;
100
101  AXI_LEDS = AXI_SWITCHES;
102

```

Figure 0-12 : 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 :

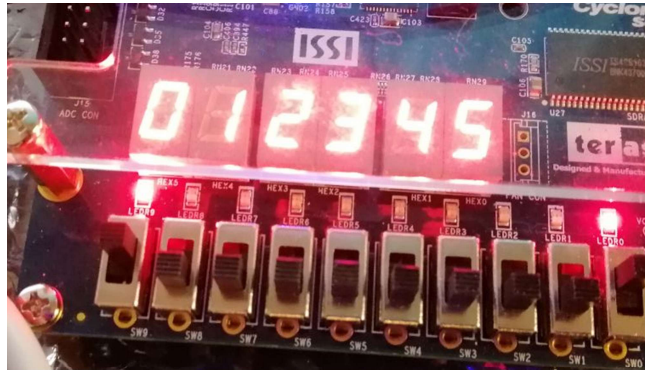


Figure 0-13 : Test sur la DE1

Grâce à ce test, j'ai pu m'assurer que mon interface fonctionne aussi dans la pratique.

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 précédent 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/lab05_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

Pour la deuxième partie, il est demandé de gérer l'appui sur les boutons KEY 2 et 3 à l'aide d'interruption vers le HPS. Le design doit générer une interruption lors d'une détection d'un flanc d'un des 4 boutons. Il doit être possible de masquer/démasquer l'interruption pour chaque bouton.

Plan d'adressage

Afin de répondre à la deuxième partie, j'ai complété mon plan d'adressage afin de gérer les interruptions.

N	Offset	D32	Read	0	D32	Write	0	I/O
0	0x0000 0000		Constante (0xDEADBEEF)			not used		
1	0x0000 0004		[31..0] regTest			[31..0] regTest		Test
2	0x0000 0008		Reserved			Reserved		
3	0x0000 000C		Reserved			Reserved		
4	0x0000 0010		Reserved			Reserved		
5	0x0000 0014		Reserved			Reserved		
...	...		Reserved			Reserved		
64	0x0000 0100		[31..10] '0..0' - [9..0] dataLEDs (9..0)			[31..10] reserved - [9..0] dataLEDs (9..0)		Leds
...	...		Reserved			Reserved		
128	0x0000 0200		[31..4] '0..0' - [3..0] dataKeys (3..0)			not used		
129	0x0000 0204		[31..4] '0..0' - [3..0] sourceIRQ (3..0)			not used		Keys
130	0x0000 0208		[31..4] '0..0' - [3..0] maskIRQ (3..0)			[31..4] reserved - [3..0] maskIRQ (3..0)		
...	...		Reserved			Reserved		
192	0x0000 0300		[31..10] '0..0' - [9..0] dataSwitchs (9..0)			not used		Switchs
...	...		Reserved			Reserved		
256	0x0000 0400		[31..7] '0..0' - [6..0] dataHEX0 (6..0)			[31..7] reserved - [6..0] dataHEX0 (6..0)		7seg (Le point n'est pas connecté)
260	0x0000 0410		[31..7] '0..0' - [6..0] dataHEX1 (6..0)			[31..7] reserved - [6..0] dataHEX1 (6..0)		
264	0x0000 0420		[31..7] '0..0' - [6..0] dataHEX2 (6..0)			[31..7] reserved - [6..0] dataHEX2 (6..0)		
268	0x0000 0430		[31..7] '0..0' - [6..0] dataHEX3 (6..0)			[31..7] reserved - [6..0] dataHEX3 (6..0)		
272	0x0000 0440		[31..7] '0..0' - [6..0] dataHEX4 (6..0)			[31..7] reserved - [6..0] dataHEX4 (6..0)		
276	0x0000 0450		[31..7] '0..0' - [6..0] dataHEX5 (6..0)			[31..7] reserved - [6..0] dataHEX5 (6..0)		
...	...		Reserved			Reserved		
1023	0x0000 0FFF		Reserved			Reserved		

Figure 0-1 : Plan d'adressage (Partie 2)

Mon plan d'adressage est resté globalement identique mais j'ai rajouté 2 I/Os. Pour commencer, à l'offset 0x204, j'ai ajouté un champ afin de lire la source d'interruption. Chaque bit correspond à chaque bouton. Par exemple, Si le bit 0 du champs « sourceIRQ » est à 1, cela signifie qu'il y a eu une interruption sur la KEY0. J'ai décidé de faire un acquittement lors de la lecture de la source comme ça cela est fait automatiquement.

Le deuxième champ est « maskIRQ » qui est accessible en lecture et écriture. Il permet, comme son nom l'indique, de masquer ou pas une interruption. Par défaut, les 4 bits sont à '0' ce qui signifie que les quatre interruptions sont actives (non masquée).

Conception

Afin de gérer les interruptions, j'ai commencé par ajouté une sortie à mon interface qui sera directement connecté sur une ligne d'interruption du HPS.

```
-- Interruption
irq_o      : out std_logic
```

Figure 0-2 : Déclaration de la sortie irq

Ensuite, j'ai ajouté quelques nouveaux signaux afin de gérer les interruptions :

```
----- SIGNAUX GESTION IRQ -----
signal irq_s      : std_logic;
signal irq_source  : std_logic_vector(3 downto 0) := (others => '0');
signal key_val_save : std_logic_vector(3 downto 0) := (others => '1');
-- par défaut, toutes les irq actives
signal key_irq_mask : std_logic_vector(3 downto 0) := (others => '0');
```

Figure 0-3 : Signaux pour la gestion des interruptions

- Le signal « irq_s » est simplement le signal lié à la sortie irq_o.
- Le signal « irq_source » représente le champ « sourceIRQ » dans mon plan d'adressage. Il permet d'indiquer la source de l'interruption. Par défaut, l'état des bits est à '0', signifiant qu'il n'y a pas eu d'interruption.
- Le signal « key_val_save » permet d'enregistrer la valeur des KEYS afin de pouvoir le comparer avec la valeur réelle pour détecter un flanc. Par défaut, l'état des bits est à '1', car les boutons sont actifs bas.
- Le signal « key_irq_mask » représente le champ « maskIRQ » dans mon plan d'adressage. Il permet de gérer le masquage/démasquage de l'interruption de chaque bouton.

Afin de gérer les interruptions, je suis vite parti sur une solution de créer un process et d'utiliser la fonction « rising_edge » sur chaque bit des entrées « key_i ». Malheureusement, ce n'était pas aussi facile. En effet, il m'était impossible d'utiliser la fonction « rising_edge » sur l'entrée « key_i ». De plus, il est impossible de changer l'état d'un signal dans deux process différents. Étant donné que je devais gérer l'acquittement lors d'une lecture, il était plus simple de tout faire dans le process de lecture. Cependant, il aurait été possible de faire un signal de synchronisation entre les deux process. Voici mon process de lecture de données dans lequel j'ai ajouté la gestion des interruptions :

```

-- Read data channel
-- Implement axi_wready generation
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_waddr_done_s <= '0';
        axi_rvalid_s <= '0';
        axi_rdata_mem_s <= (others => '0');
        axi_rresp_s <= "00";

        1 irq_source <= "0000";
        irq_s <= '0';

    elsif rising_edge(axi_clk_i) then
        -- Gestion des interruptions
        2 if (key_val_save(0) /= registre_key_mem(0) and registre_key_mem(0) = '0' and key_irq_mask(0) = '0') then
            irq_source(0) <= '1';
            irq_s <= '1';
        elsif (key_val_save(1) /= registre_key_mem(1) and registre_key_mem(1) = '0' and key_irq_mask(1) = '0') then
            irq_source(1) <= '1';
            irq_s <= '1';
        elsif (key_val_save(2) /= registre_key_mem(2) and registre_key_mem(2) = '0' and key_irq_mask(2) = '0') then
            irq_source(2) <= '1';
            irq_s <= '1';
        elsif (key_val_save(3) /= registre_key_mem(3) and registre_key_mem(3) = '0' and key_irq_mask(3) = '0') then
            irq_source(3) <= '1';
            irq_s <= '1';
        end if;
        3 -- Met à jour l'ancienne valeur des keys
        key_val_save <= registre_key_mem;
    end if;
end process;

```

Figure 0-4 : Code pour la gestion des interruptions

1. Remise à '0' des signaux en cas de reset
2. Détection de flanc et test du masque. Si oui, mise à '1' de la source et de l'interruption.
3. Mise à jour des valeurs des boutons dans le signal de sauvegarde.

J'ai décidé de faire une détection sur flanc descendant car les boutons sont actifs bas.

Comme l'indique le plan d'adressage, j'ai ajouté le signal « irq_source » en lecture et « key_irq_mask » en lecture et écriture.

Remarque : Le code complet est en annexe.

Test de l'IP

Étant donné que j'ai trouvé cette partie relativement simple et que le code C du laboratoire précédent permet déjà de tester si une interruption est générée, je n'ai pas voulu perdre du temps à modifier le test Bench afin de tester la fonctionnalité d'interruption.

Mise à jour dans Qsys

Il est maintenant nécessaire de modifier mon composant dans Qsys afin d'ajouter une ligne d'interruption. Pour ce faire, j'ai ajouté une interface « Interrupt Sender » à mon composant en y ajoutant le signal de sortie « irq_o » correspondant :

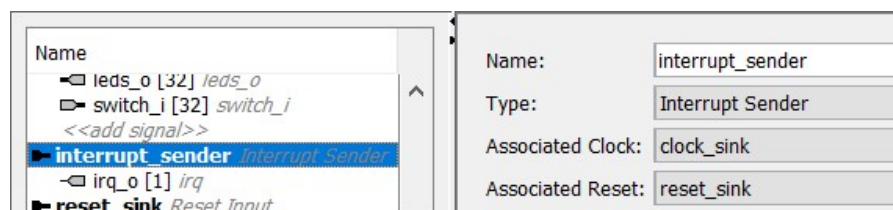


Figure 0-5 : Interface de l'interruption

Ensuite, comme pour le laboratoire précédent, j'ai activé les interruptions FPGA to HPS dans le composant HPS. Puis, j'ai connecté la ligne d'interruption sur le composant HPS sur la même ligne que le laboratoire précédent afin de garder le même numéro d'interruption (72).



Figure 0-6 : Connexion dans Qsys

Postérieurement, j'ai pu générer les fichiers HDL. Et finalement, synthétiser et faire le placement routage du projet.

Test de l'IP avec le code C

Maintenant que tout est prêt, j'ai pu reprendre le code C afin d'activer les interruptions du laboratoire précédent. Ensuite, afin de m'assurer que cela fonctionne, j'ai mis du code C qui affiche sur des afficheurs 7 segments des informations dans la routine d'interruption :

```
void pushbutton_ISR(void) {
    static int i = 0;
    int src_irq = AXI_INT_SRC;

    AXI_HEX0 = src_irq;
    AXI_HEX1 = i++;
}
```

Figure 0-7 : Code de test des interruptions

J'ai facilement pu constater grâce aux afficheurs 7 segments que les interruptions étaient bien générées et acquittées.

Réalisation de la spécification

Maintenant que j'ai testé le bon fonctionnement de mon interface, j'ai commencé par réaliser la spécification de la partie 2 du code qui consiste à utiliser une interruption pour les actions sur les boutons KEY2 et KEY3.

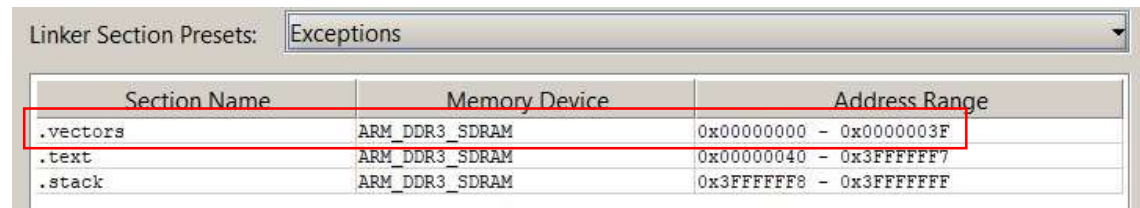
Compléter le code C

Cette étape était simple car le changement dans le code nécessitait de reproduire le code du laboratoire précédent afin de traiter des interruptions. Cependant, j'ai tout de même apporté une modification. Tous les traitements sont faits dans le « main » et non plus dans la routine d'interruption qui ne devrait contenir uniquement le strict minimum de traitement.

Remarque : Le code complet du laboratoire est en annexe.

Modifier la configuration mémoire

Afin d'allouer une portion mémoire pour les vecteurs d'interruptions, il a fallu modifier la configuration mémoire du projet « Altera Monitor Program » :



Section Name	Memory Device	Address Range
.vectors	ARM_DDR3_SDRAM	0x00000000 - 0x0000003F
.text	ARM_DDR3_SDRAM	0x00000040 - 0x3FFFFFF7
.stack	ARM_DDR3_SDRAM	0x3FFFFFF8 - 0x3FFFFFFF

Figure 0-8 : Modification de la configuration mémoire

On peut voir qu'une section a été ajoutée pour les vecteurs.

Compiler et tester

Après avoir compiler et charger mon programme dans la DE1-SoC, j'ai pu tester le bon fonctionnement des spécifications demandées.

Test des spécifications

Pour ce faire, j'ai appuyé plusieurs fois sur KEY0 avec des valeurs de switch différents afin de m'assurer que les leds aient toujours la même valeur des switches après l'appuis. J'ai aussi vérifié que les afficheurs HEX5 à HEX0 affichent en hexadécimal les bits 23 à 0 de la constante définie dans l'IP.

Ensuite, j'ai fait les mêmes tests pour KEY1. Je me suis assuré que l'états inverses des switches est copiés sur les LEDs et que les afficheurs HEX5 à HEX0 affichent en hexadécimal l'inverse des bits 23 à 0 de la constante définie dans l'IP.

J'ai aussi testé le bon fonctionnement de ces deux boutons en les testant consécutivement l'un après l'autre.

Ultérieurement, j'ai testé KEY2 plusieurs fois et l'effet sur les bords. L'affichage des LEDs et des afficheurs 7 segments ont bien subi une rotation à droite. Rotation d'un bit pour les LEDs, rotation d'un afficheur complet pour les afficheurs 7 segments.

Finalement, j'ai testé KEY3 plusieurs fois et l'effet sur les bords. L'affichage des LEDs et des afficheurs 7 segments ont bien subi une rotation à gauche. Rotation d'un bit pour les LEDs, rotation d'un afficheur complet pour les afficheurs 7 segments.

Test du masquage

Afin de tester le masquage des interruptions, j'ai initialisé la valeur du masque à 0x08 afin de masque le bouton KEY3. Puis, j'ai ajouté ce code dans la routine d'interruption :

```
// Tous les 3 interruptions de KEY0 et KEY1, change le masque de key 2 et 3
if (src_irq & KEY0 || src_irq & KEY1) {
    cpt_int++;

    if (cpt_int % 3 == 0)
        AXI_INT_MASK = AXI_INT_MASK ^ (KEY3 | KEY2);
}
```

Figure 0-9 : Code de test du masquage

Il permet d'intervertir le masque de KEY2 et KEY3 après 3 interruptions sur KEY0 ou KEY1. Donc, au démarrage l'appui sur la KEY3 n'avait aucun effet contrairement à KEY2. Après 3 appuis sur KEY1/KEY0, c'était le contraire comme attendu. L'appui sur KEY2 n'avait aucun effet contrairement à KEY3.

Fonctionnalité de strobe

Cette fonctionnalité permet de choisir quelle partie des bits du bus AXI_WDATA vont être pris en compte. Voici un schéma dans la documentation qui l'explique bien :

S_AXI_WSTRB signals		
S_AXI_WSTRB [3:0]	S_AXI_WDATA active bits [31:0]	Description
1111	11111111111111111111111111111111	All bits active
0011	00000000000000001111111111111111	Least significant 16 bits active
0001	00000000000000000000000011111111	Least significant byte (8 bits) active.
1100	11111111111111111000000000000000	Most significant 16 bits active

Figure 0-10 : Tableau du la fonction strobe

Je pensais ne pas avoir assez de temps pour réaliser ce laboratoire donc j'ai décidé par ne pas gérer cette fonctionnalité au début. Étant donné que du temps supplémentaire nous a été donné, j'en ai profité pour réaliser cette fonctionnalité.

J'ai commencé par modifier le test Bench afin de tester ce paramètre :

```
constant TAB_STI_AXI_WRITE : Type_Tab_Stimuli_AXI_WRITE :=
--REM: table fixe pour 32 bits de data
(
    ( 4, x"01234567", "1111"),    --écriture adresse 0x04
    ( 8, x"01234567", "0111"),    --écriture adresse 0x08
    (12, x"01234567", "0011"),    --écriture adresse 0x0C
    (16, x"01234567", "0001"),    --écriture adresse 0x10
    ( 4, x"01234567", "1100")    --écriture adresse 0x04
);
```

Figure 0-11 : Modification du test Bench

Grâce aux nouvelles valeurs de ce tableau d'écriture, il sera facile de voir le bon fonctionnement du paramètre strobe.

J'ai relancé le nouveau test bench pour voir que la fonctionnalité n'est pas réalisée :



Figure 0-12 : Test 1 du paramètre strobe

On peut voir que la donnée à écrire est bien 0x01234567 et que le paramètre strobe est à « 0011 ». On peut donc s'attendre à ce que les deux octets de poids fort de la donnée à écrire ne soient pas actifs. Ce qui donnerait 0x00004567. Cependant, les données enregistrées restent 0x01234567 dans le signal « axi_wdata_mem_s ». Le but maintenant est de réaliser cette fonctionnalité.

Description VHDL

Afin de prendre en compte le paramètre strobe, j'ai modifié le process qui s'occupe du canal des données d'écriture afin d'enregistrer uniquement les octets souhaités par le paramètre strobe. Voici à quoi ressemble le code :

```

1 axi_wdata_mem_s <= (others => '0');
2
3 if (axi_wstrb_i(0) = '1') then
4     axi_wdata_mem_s(7 downto 0) <= axi_wdata_i(7 downto 0);
5 end if;
6 if (axi_wstrb_i(1) = '1') then
7     axi_wdata_mem_s(15 downto 8) <= axi_wdata_i(15 downto 8);
8 end if;
9 if (axi_wstrb_i(2) = '1') then
10    axi_wdata_mem_s(23 downto 16) <= axi_wdata_i(23 downto 16);
11 end if;
12 if (axi_wstrb_i(3) = '1') then
13    axi_wdata_mem_s(31 downto 24) <= axi_wdata_i(31 downto 24);
14 end if;

```

Figure 0-13 : Code gestion de strobe

1. J'ai commencé par mettre tous les bits à '0' comme si aucun octet était actif.
2. Ensuite, j'ai testé chaque bit du paramètre strobe afin d'assigner strictement les octets actifs.

Test de la fonctionnalité strobe (test bench)

Maintenant que le test bench est déjà prêt, il suffit de compiler le nouveau code de mon IP et de lancer le test bench.

Remarque : Afin de voir plus facilement les valeurs dans le signal « axi_wdata_mem_s » j'ai commenté se remise à 0 à chaque fin de lecture des données.



Figure 0-14 : Chronogramme de test de strobe

Comme indiqué par les flèches orange, les données d'écriture enregistrées correspondent bien aux paramètre strobe (S_AXI_WSTRB). Par exemple, pour la troisième flèche orange, le paramètre strobe vaut « 0011 » et on peut voir comme attendu que les données enregistrées sont bien « 0x00004567 ». On peut aussi bien voir dans le dernier cas, le paramètre vaut « 1100 », donc on souhaite activer seulement les 2 octets de poids fort. Comme attendu, les données enregistrées sont bien « 0x01230000 ».

Test de la fonctionnalité strobe (code)

Afin de m'assurer du bon fonctionnement de l'implémentation du paramètre strobe, j'ai voulu le tester en situation réel. J'ai donc écrit du code pour tester ce paramètre. Puis grâce au débogueur j'ai avancé pas à pas dans le code en assembleur afin de m'assurer du bon fonctionnement.

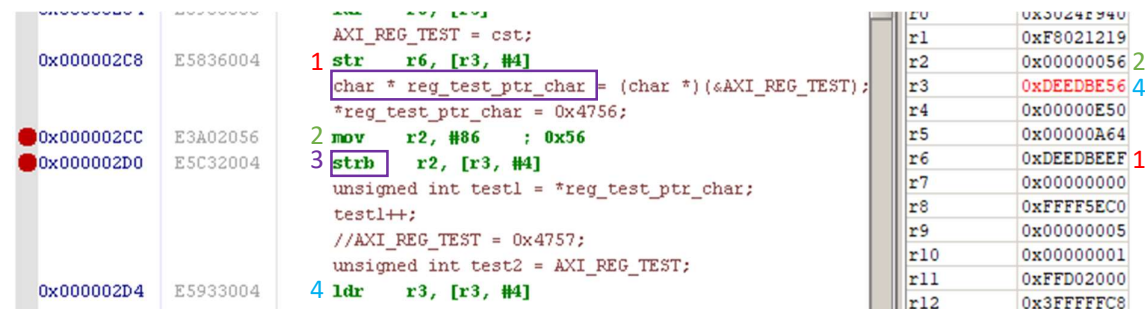


Figure 0-15 : Débogage pour tester le paramètre strobe

On début du programme, on lit l'adresse de la constante (0xDEADBEEF) et la place dans le registre de test qui est à l'offset 0x4. A la première instruction, le registre r3 comprend l'adresse de base (0xFF200000) donc l'adresse de la constante. De ce fait, « r3, #4 » indique l'adresse du registre de test.

1. **Str** : la première instruction permet d'écrire la valeur du registre r6 (qui contient la valeur de la constante) dans le registre de test à l'offset 0x4.
2. **Mov** : Place la valeur de 8 bits (0x56) dans le registre r2.
3. **Strb** : Cette instruction permet de placer un byte de la valeur du registre r2 (0x56) à l'adresse du registre de test. Un pointeur de char est utilisé afin de forcer le programme à utiliser l'instruction « strb » pour s'assurer qu'une écriture d'uniquement 8 bits se fait et pas 32 bits. Ce qui permet de tester la fonctionnalité strobe implémenté.
4. **Ldr** : Cette instruction permet de lire 32 bits à l'adresse du registre de test et la place dans le registre r3. On peut voir que la valeur vaut 0xDEADBEE56. Ce qui confirme que mon IP n'est pas écrasé 0xDEADBE par 0x000000 grâce à l'implémentation du strobe.

Finalement, j'ai pu confirmer que mon implémentation du paramètre strobe dans mon IP est bien fonctionnel même sur le matériel.

Supplémentaire : Gestion Edge

Le paramètre Edge permet de choisir sur quel flanc l'interruption sera levée. Par défaut, j'ai fixé ce paramètre au flanc descendant afin d'avoir une interruption dès qu'un bouton est pressé, car ils sont actifs bas.

Plan d'adressage

Pour gérer ce nouveau paramètre, j'ai ajouté un champ dans mon plan d'adressage. Voici la modification que j'ai effectuée :

128	0x0000 0200	[31..4] '0..0' - [3..0] dataKeys (3..0)	not used	Keys
129	0x0000 0204	[31..4] '0..0' - [3..0] sourceIRQ (3..0)	not used	
130	0x0000 0208	[31..4] '0..0' - [3..0] maskIRQ (3..0)	[31..4] reserved - [3..0] maskIRQ (3..0)	
131	0x0000 020C	[31..4] '0..0' - [3..0] edgeIRQ (3..0)	[31..4] reserved - [3..0] edgeIRQ (3..0)	

Figure 0-1 : Modification du plan d'adressage

Toute la tables reste identiques mise à part le nouveau champ « edgeIRQ » à l'offset 0x20C (131). Ce paramètre est évidemment accessible en lecture ainsi qu'en écriture. Il permettra de choisir sur quel flanc générée l'interruption.

Description VHDL

Afin de réaliser cette nouvelle fonctionnalité, j'ai ajouté un signal « key_irq_edge » de 4 bits initialisé à 0 afin d'activer l'interruption sur un flanc descendant par défaut.

```
-- par défaut, toutes les irq sur flanc descendant
signal key_irq_edge          : std_logic_vector(3 downto 0) := (others => '0');
```

Figure 0-2 : Déclaration du signal edge

Ensuite, dans les « switch » des canaux de lecture et d'écriture, j'ai ajouté le « case » pour ce nouveau champ afin d'y accéder en lecture et en écriture.

Finalement, j'ai modifié la gestion des interruptions afin de détecter une interruption sur un flanc en fonction du signal « key_irq_edge » :

```
elsif rising_edge(axi_clk_i) then
-- Gestion des interruptions
if (key_val_save(0) /= registre_key_mem(0) and registre_key_mem(0) = key_irq_edge(0) and key_irq_mask(0) = '0') then
    irq_source(0) <= '1';
    irq_s <= '1';
elsif (key_val_save(1) /= registre_key_mem(1) and registre_key_mem(1) = key_irq_edge(1) and key_irq_mask(1) = '0') then
    irq_source(1) <= '1';
    irq_s <= '1';
elsif (key_val_save(2) /= registre_key_mem(2) and registre_key_mem(2) = key_irq_edge(2) and key_irq_mask(2) = '0') then
    irq_source(2) <= '1';
    irq_s <= '1';
elsif (key_val_save(3) /= registre_key_mem(3) and registre_key_mem(3) = key_irq_edge(3) and key_irq_mask(3) = '0') then
    irq_source(3) <= '1';
    irq_s <= '1';
end if;
-- Met à jour l'ancienne valeur des keys
key_val_save <= registre_key_mem;
```

Figure 0-3 : Gestion des interruptions avec le paramètre edge

Avant, je testais l'égalité des valeurs des boutons « registre_key_mem » avec la constante '0' afin de détecter un flanc descendant. Maintenant, je test l'égalité avec le paramètre edge des boutons « key_irq_edge ».

Mise à jour du projet

Une fois toutes ces modifications apportées, j'ai compilé ma nouvelle description VHDL de mon IP. Ensuite j'ai mise à jour mon projet en mettant à jour mon composant dans Qsys, puis, j'ai généré les fichiers HDL, ensuite, j'ai synthétisé et fait le placement routage du projet.

Test de fonctionnalité

Afin de tester cette nouvelle fonctionnalité, j'ai modifié le code C. J'ai commencé par ajouter une définition qui permet de lire et écrire facilement dans le champ Edge des keys :

```
// 0 = interruption flanc descendant (défaut)
#define AXI_INT_EDGE          *(vuint *) (AXI_LIGHT_BASE_ADDR + 0x20C)
```

Figure 0-4 : Définition du paramètre edge

Ensuite, j'ai modifié la valeur par défaut de ce champ afin de tester si un flanc montant était bien géré. J'ai donc mis à 1 le bit correspondant à la KEY3 afin de lever une interruption uniquement au flanc montant de la KEY3 :

```
// KEY 3 sur flanc montant
AXI_INT_EDGE = KEY3;
```

Figure 0-5 : Initialisation du paramètre edge

Après ces modifications, j'ai pu tester cette nouvelle fonctionnalité directement sur le matériel. J'ai commencé par télécharger le nouveau système sur la carte DE1 grâce au programme « Altera Monitor Program ». Ensuite, j'ai compilé et loadé le nouveau programme C. Finalement, en lançant le nouveau programme, j'ai pu constater que la rotation à gauche était bien effectuée seulement lors du relâchement du bouton KEY3.

J'ai décidé de laisser cette modification au programme final rendu car il respecte toujours les spécifications demandées et permet de prouver, sans changement de code, cette fonctionnalité.

Annexes

Voici la liste dans l'ordre des annexes :

1. Code VHDL de mon IP (axi4lite_slave.vhd)
2. Code VHDL du top (DE1_SoC_top.vhd)
3. Code du programme principal (labo5.c)
4. Définitions du code (defines.h)
5. Point H du fichier exception (exceptions.h)
6. Code de fonctions utiles (exceptions.c)
7. Définitions d'adresse (address_map_arm.h)

Conclusion

Je dois avouer qu'au début du laboratoire j'étais perdu et avais peur de tout le travail demandé. Finalement, sans prendre en compte les temps de compilations extrêmement long, j'ai beaucoup apprécié ce laboratoire. C'est pourquoi, c'est avec plaisir que j'ai ajouté la fonctionnalité edge sur les interruptions.

Difficultés rencontrées

- Le bon fonctionnement de tous les programmes fut difficile. En effet, une mise en place sur Windows fût nécessaire.
- La compréhension du fonctionnement complet du bus AXI 4 Lite.

Compétences acquises

- Installation complète de l'environnement sur Windows
- Perfectionnement de la méthodologie
- Perfectionnement des logiciels (Quartus Prime, Qsys, altera monitor program et Questasim)

Résultats obtenus

J'ai réussi à mettre en place toutes les étapes qui m'était demandé dans ce laboratoire. Les description VHDL sont synthétisable et intégrable. Je suis particulièrement fier d'avoir réussi à faire complètement le travail demandé. Je me dois remercier les professeurs d'avoir repoussé la date du rendu et surtout l'assistant M. Masle qui a passé environ 2 heures afin de m'aider en partie pour résoudre ces gros problèmes de logiciel.

Date : 08.05.20

Nom de l'étudiant : Spinelli Isaia


```

1  -----
2  -- HEIG-VD, Haute Ecole d'Ingenierie et de Gestion du canton de Vaud
3  -- Institut REDS, Reconfigurable & Embedded Digital Systems
4  --
5  -- File      : axi4lite_slave.vhd
6  -- Author    : E. Messerli    27.07.2017
7  -- Description : slave interface AXI (without burst)
8  -- used for   : SOCF lab
9  --| Modifications |-----
10 -- Ver  Date      Auteur  Description
11 -- 1.0  26.04.2019  EMI    Adaptation du chablon pour les etudiants
12 -- 1.1  03.05.2020  ISS    Complète le chablon pour le laboratoire 5 Partie 2
13 -- 1.2  08.05.2020  ISS    Ajout de la fonctionnalité edge pour les irq
14 -----
15
16 library ieee;
17     use ieee.std_logic_1164.all;
18     use ieee.numeric_std.all;
19
20 entity axi4lite_slave is
21     generic (
22         -- Users to add parameters here
23
24         -- User parameters ends
25
26         -- Width of S_AXI data bus
27         AXI_DATA_WIDTH  : integer    := 32; -- 32 or 64 bits
28         -- Width of S_AXI address bus
29         AXI_ADDR_WIDTH  : integer    := 12
30     );
31     port (
32         -- AXI4-Lite
33         axi_clk_i      : in  std_logic;
34         axi_reset_i    : in  std_logic;
35
36         -- Write Address Channel
37         axi_awaddr_i    : in  std_logic_vector(AXI_ADDR_WIDTH-1 downto 0);
38         axi_awprot_i    : in  std_logic_vector( 2 downto 0); -- not used
39         axi_awvalid_i   : in  std_logic;
40         axi_awready_o   : out std_logic;
41
42         -- Write Data Channel
43         axi_wdata_i     : in  std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
44         axi_wstrb_i     : in  std_logic_vector((AXI_DATA_WIDTH/8)-1 downto 0);
45         axi_wvalid_i    : in  std_logic;
46         axi_wready_o    : out std_logic;
47
48         -- Write Response Channel
49         axi_bresp_o     : out std_logic_vector(1 downto 0);
50         axi_bvalid_o    : out std_logic;
51         axi_bready_i    : in  std_logic;
52
53         -- Read Address Channel
54         axi_araddr_i    : in  std_logic_vector(AXI_ADDR_WIDTH-1 downto 0);
55         axi_arprot_i    : in  std_logic_vector( 2 downto 0); -- not used
56         axi_arvalid_i   : in  std_logic;
57         axi_arready_o   : out std_logic;
58
59         -- Read Data Channel
60         axi_rdata_o     : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
61         axi_rresp_o     : out std_logic_vector(1 downto 0);
62         axi_rvalid_o    : out std_logic;
63         axi_rready_i    : in  std_logic;
64
65         -- User input-output
66         switch_i       : in  std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
67         key_i          : in  std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
68
69         leds_o         : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);

```

```

70
71     hex0_o           : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
72     hex1_o           : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
73     hex2_o           : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
74     hex3_o           : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
75     hex4_o           : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
76     hex5_o           : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
77
78
79     -- Interruption
80     irq_o             : out std_logic
81 );
82 end entity axi4lite_slave;
83
84 architecture rtl of axi4lite_slave is
85
86     signal reset_s : std_logic;
87
88     -- local parameter for addressing 32 bit / 64 bits, cst: AXI_DATA_WIDTH
89     -- ADDR_LSB is used for addressing word 32/64 bits registers/memories
90     -- ADDR_LSB = 2 for 32 bits (n-1 downto 2)
91     -- ADDR_LSB = 3 for 64 bits (n-1 downto 3)
92     constant ADDR_LSB : integer := (AXI_DATA_WIDTH/32)+ 1;
93
94     ----- SIGNAUX AXI 4 LIGHT -----
95
96     --signal for the AXI slave
97     --intern signal for output
98     signal axi_awready_s : std_logic;
99     signal axi_arready_s : std_logic;
100
101     signal axi_wready_s : std_logic;
102     signal axi_rready_s : std_logic;
103
104     signal axi_rvalid_s : std_logic;
105     signal axi_rresp_s : std_logic_vector(1 downto 0);
106     signal axi_rdata_mem_s : std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
107
108     -- write enable
109     signal axi_data_wren_s : std_logic;
110
111     --intern signal for the axi interface
112     signal axi_waddr_mem_s : std_logic_vector(AXI_ADDR_WIDTH-1 downto ADDR_LSB);
113     signal axi_araddr_mem_s : std_logic_vector(AXI_ADDR_WIDTH-1 downto ADDR_LSB);
114
115     signal axi_wdata_mem_s : std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
116     signal axi_wstrb_mem_s : std_logic_vector((AXI_DATA_WIDTH/8)-1 downto 0);
117     -- signal axi_araddr_mem_s : std_logic_vector(AXI_ADDR_WIDTH-1 downto ADDR_LSB);
118
119     signal axi_bresp_s : std_logic_vector(1 downto 0);
120     signal axi_bvalid_s : std_logic;
121
122
123     ----- SIGNAUX ENTREES / SORTIES -----
124
125     constant registre_cst_mem : std_logic_vector(AXI_DATA_WIDTH-1 downto 0) :=
126     x"deedbeef";
127
128     signal registre_test_mem : std_logic_vector(AXI_DATA_WIDTH-1 downto 0) :=
129     x"12345678";
130
131
132     -- signal for registre input (switch / key)
133     signal registre_switch_mem : std_logic_vector(9 downto 0) := (others => 'X');
134     signal registre_key_mem : std_logic_vector(3 downto 0) := (others => 'X');
135
136     -- signal for registre leds
137     signal registre_led_mem : std_logic_vector(9 downto 0) := (others => 'X');
138
139     -- signal for registre 7 seg
140     signal registre_hex0_mem : std_logic_vector(6 downto 0) := (others => 'X');

```

```

137     signal registre_hex1_mem      : std_logic_vector(6 downto 0) := (others => 'X');
138     signal registre_hex2_mem      : std_logic_vector(6 downto 0) := (others => 'X');
139     signal registre_hex3_mem      : std_logic_vector(6 downto 0) := (others => 'X');
140     signal registre_hex4_mem      : std_logic_vector(6 downto 0) := (others => 'X');
141     signal registre_hex5_mem      : std_logic_vector(6 downto 0) := (others => 'X');
142
143     ----- SIGNAUX GESTION IRQ -----
144     signal irq_s                   : std_logic;
145     signal irq_source              : std_logic_vector(3 downto 0) := (others => '0');
146     signal key_val_save            : std_logic_vector(3 downto 0) := (others => '1');
147     -- par défaut, toutes les irq actives
148     signal key_irq_mask            : std_logic_vector(3 downto 0) := (others => '0');
149     -- par défaut, toutes les irq sur flanc descendant
150     signal key_irq_edge            : std_logic_vector(3 downto 0) := (others => '0');
151
152 begin
153
154     -- mise à jour des entrées
155     reset_s <= axi_reset_i;
156
157     registre_switch_mem <= switch_i(9 downto 0);
158     registre_key_mem    <= key_i(3 downto 0);
159
160
161
162     -----
163     -- Write address channel
164
165     process (reset_s, axi_clk_i)
166     begin
167         -- En cas de reset
168         if reset_s = '1' then
169             -- Valeur par défaut
170             axi_awready_s <= '0';
171             axi_waddr_mem_s <= (others => '0');
172         elsif rising_edge(axi_clk_i) then
173             -- Si une adresse d'écriture est valide
174             if (axi_awready_s = '0' and axi_awvalid_i = '1') then --and axi_wvalid_i =
175                 '1') then modif EMI 10juil2018
176                     -- slave is ready to accept write address when
177                     -- there is a valid write address
178                     axi_awready_s <= '1';
179                     -- Write Address memorizing
180                     axi_waddr_mem_s <= axi_awaddr_i (AXI_ADDR_WIDTH-1 downto ADDR_LSB);
181                 else
182                     axi_awready_s <= '0';
183                     axi_waddr_mem_s <= (others => '0');
184                 end if;
185             end if;
186         end process;
187         axi_awready_o <= axi_awready_s;
188
189     -----
190     -- Write data channel
191
192     -- Implement axi_wready generation
193     process (reset_s, axi_clk_i)
194     begin
195         -- En cas de reset
196         if reset_s = '1' then
197             -- Valeur par défaut
198             axi_wready_s <= '0';
199             axi_wdata_mem_s <= (others => '0');
200             axi_wstrb_mem_s <= (others => '0');
201         elsif rising_edge(axi_clk_i) then
202             -- Si les données d'écriture est valide
203             if (axi_wready_s = '0' and axi_wvalid_i = '1') then
204                 -- slave is ready to accept write data when

```

```

205         -- there is a valid write data
206         axi_wready_s <= '1';
207
208         -- Read axi_wstrb_i
209         axi_wstrb_mem_s <= axi_wstrb_i((AXI_DATA_WIDTH/8)-1 downto 0);
210
211
212         -- Mémorisation des données à écrire en fonction du paramètre strobe
213         axi_wdata_mem_s <= (others => '0');
214
215         if (axi_wstrb_i(0) = '1') then
216             axi_wdata_mem_s(7 downto 0) <= axi_wdata_i(7 downto 0);
217         end if;
218         if (axi_wstrb_i(1) = '1') then
219             axi_wdata_mem_s(15 downto 8) <= axi_wdata_i(15 downto 8);
220         end if;
221         if (axi_wstrb_i(2) = '1') then
222             axi_wdata_mem_s(23 downto 16) <= axi_wdata_i(23 downto 16);
223         end if;
224         if (axi_wstrb_i(3) = '1') then
225             axi_wdata_mem_s(31 downto 24) <= axi_wdata_i(31 downto 24);
226         end if;
227
228         -- Test sans la fonctionnalité strobe
229         -- axi_wdata_mem_s <= axi_wdata_i;
230
231     else
232         axi_wready_s <= '0';
233         axi_wdata_mem_s <= (others => '0');
234         axi_wstrb_mem_s <= (others => '0');
235
236     end if;
237 end if;
238 end process;
239
240 -- Met à jour la sortie
241 axi_wready_o <= axi_wready_s;
242
243
244 -- condition to write data : si on est prêt à écrire
245 axi_data_wren_s <= '1' when axi_wready_s = '1' else
246     '0';
247
248
249 process (reset_s, axi_clk_i)
250     --number address to access 32 or 64 bits data
251     variable int_waddr_v : natural;
252 begin
253     if reset_s = '1' then
254         -- Valeur par défaut : RESET
255         registre_test_mem <= x"12345678";
256         registre_led_mem <= "0101010101";
257         registre_hex0_mem <= "1000000" ;
258         registre_hex1_mem <= "1111001";
259         registre_hex2_mem <= "0100100";
260         registre_hex3_mem <= "0110000";
261         registre_hex4_mem <= "0011001";
262         registre_hex5_mem <= "0010010";
263
264         key_irq_mask <= "0000";
265         key_irq_edge <= "0000";
266
267     elsif rising_edge(axi_clk_i) then
268         -- Si une écriture est active
269         if axi_data_wren_s = '1' then
270             -- convertie l'adresse d'écriture en integer
271             int_waddr_v := to_integer(unsigned(axi_waddr_mem_s));
272             case int_waddr_v is
273                 -- offset 0 : constante

```

```

274         when 0 =>
275             -- offset 4 : registre de test
276         when 1 =>
277             registre_test_mem <= axi_wdata_mem_s;
278
279             -- offset 64 : leds
280         when 64 =>
281             registre_led_mem <= axi_wdata_mem_s(9 downto 0);
282
283             -- offset 130 : mask irq key
284         when 130 =>
285             key_irq_mask <= axi_wdata_mem_s(3 downto 0);
286             -- offset 130 : mask irq key
287         when 131 =>
288             key_irq_edge <= axi_wdata_mem_s(3 downto 0);
289
290             -- offset 256 - 276 : afficheur 7 seg
291         when 256 =>
292             registre_hex0_mem <= axi_wdata_mem_s(6 downto 0);
293         when 260 =>
294             registre_hex1_mem <= axi_wdata_mem_s(6 downto 0);
295         when 264 =>
296             registre_hex2_mem <= axi_wdata_mem_s(6 downto 0);
297         when 268 =>
298             registre_hex3_mem <= axi_wdata_mem_s(6 downto 0);
299         when 272 =>
300             registre_hex4_mem <= axi_wdata_mem_s(6 downto 0);
301         when 276 =>
302             registre_hex5_mem <= axi_wdata_mem_s(6 downto 0);
303
304
305             when others => null;
306         end case;
307     end if;
308 end if;
309 end process;
310
311
312 -----
313 -- Write response channel
314
315 process (reset_s, axi_clk_i)
316 begin
317     -- En cas de reset
318     if reset_s = '1' then
319         -- Valeur par défaut
320         axi_bresp_s <= "00";
321         axi_bvalid_s <= '0';
322     elsif rising_edge(axi_clk_i) then
323         -- Si le master est pret à lire la réponse
324         if (axi_bvalid_s = '0' and axi_bready_i = '1') then
325             -- slave is ready to accept write data when
326             -- there is a valid write data
327             axi_bvalid_s <= '1';
328             -- Write response
329             axi_bresp_s <= "00";
330         else
331             axi_bvalid_s <= '0';
332             axi_bresp_s <= "--";
333
334         end if;
335     end if;
336 end process;
337 -- Met à jours les sorties
338 axi_bresp_o <= axi_bresp_s;
339 axi_bvalid_o <= axi_bvalid_s;
340
341
342

```

```

343 -----
344 -- Read address channel
345
346 process (reset_s, axi_clk_i)
347 begin
348     -- en cas de reset
349     if reset_s = '1' then
350         -- valeur par défaut
351         axi_arready_s <= '0';
352         axi_araddr_mem_s <= (others => '1');
353     elsif rising_edge(axi_clk_i) then
354         -- Si une adresse de lecture est valide
355         if axi_arready_s = '0' and axi_arvalid_i = '1' then
356             -- indicates that the slave has accepted the valid read address
357             axi_arready_s <= '1';
358             -- Read Address memorizing
359             axi_araddr_mem_s <= axi_araddr_i (AXI_ADDR_WIDTH-1 downto ADDR_LSB);
360         else
361             axi_arready_s <= '0';
362         end if;
363     end if;
364 end process;
365 -- Met à jour la sortie
366 axi_arready_o <= axi_arready_s;
367
368 -----
369 -- Read data channel
370
371 -- Implement axi_wready generation
372 process (reset_s, axi_clk_i)
373 --number address to access 32 or 64 bits data
374     variable int_raddr_v : natural;
375 begin
376
377     -- En cas de reset
378     if reset_s = '1' then
379         -- valeur par défaut
380         axi_rvalid_s <= '0';
381         axi_rdata_mem_s <= (others => '0');
382         axi_rresp_s <= "00";
383
384         irq_source <= "0000";
385         irq_s <= '0';
386
387     elsif rising_edge(axi_clk_i) then
388         -- Gestion des interruptions
389         if (key_val_save(0) /= registre_key_mem(0) and registre_key_mem(0) =
390             key_irq_edge(0) and key_irq_mask(0) = '0') then
391             irq_source(0) <= '1';
392             irq_s <= '1';
393         elsif (key_val_save(1) /= registre_key_mem(1) and registre_key_mem(1) =
394             key_irq_edge(1) and key_irq_mask(1) = '0') then
395             irq_source(1) <= '1';
396             irq_s <= '1';
397         elsif (key_val_save(2) /= registre_key_mem(2) and registre_key_mem(2) =
398             key_irq_edge(2) and key_irq_mask(2) = '0') then
399             irq_source(2) <= '1';
400             irq_s <= '1';
401         elsif (key_val_save(3) /= registre_key_mem(3) and registre_key_mem(3) =
402             key_irq_edge(3) and key_irq_mask(3) = '0') then
403             irq_source(3) <= '1';
404             irq_s <= '1';
405         end if;
406         -- Met à jour l'ancienne valeur des keys
407         key_val_save <= registre_key_mem;
408
409         -- Si une lecture est faite
410         if (axi_arready_s = '1' and axi_rvalid_s = '0') then

```

```

408         -- Pré-charge une lecture à 0
409         axi_rdata_mem_s <= (others => '0');
410
411         -- slave is ready to accept write data when
412         -- there is a valid write data
413         axi_rvalid_s <= '1';
414
415         -- read Data go
416         int_raddr_v := to_integer(unsigned(axi_araddr_mem_s));
417         axi_rresp_s  <= "00";
418
419         -- En fonction de l'adresse qu'on souhaite lire
420         case int_raddr_v is
421             -- Lecture de la constante
422             when 0 =>
423                 axi_rdata_mem_s <= registre_cst_mem;
424             -- Lecture du registre de test
425             when 1 =>
426                 axi_rdata_mem_s <= registre_test_mem;
427             -- Lecture des leds
428             when 64 =>
429                 axi_rdata_mem_s(9 downto 0) <= registre_led_mem;
430             -- Lecture des keys
431             when 128 =>
432                 axi_rdata_mem_s(3 downto 0) <= registre_key_mem;
433             -- lecture de la source d'interruption et acquittement
434             when 129 =>
435                 axi_rdata_mem_s(3 downto 0) <= irq_source;
436                 irq_s <= '0';
437                 irq_source <= "0000";
438
439             -- lecture des masque des irq
440             when 130 =>
441                 axi_rdata_mem_s(3 downto 0) <= key_irq_mask;
442             -- lecture des masque des irq
443             when 131 =>
444                 axi_rdata_mem_s(3 downto 0) <= key_irq_edge;
445
446             -- Lecture des switches
447             when 192 =>
448                 axi_rdata_mem_s(9 downto 0) <= registre_switch_mem;
449
450             -- Lecture d'un afficheur 7 seg (256 - 276)
451             when 256 =>
452                 axi_rdata_mem_s(6 downto 0) <= registre_hex0_mem;
453             when 260 =>
454                 axi_rdata_mem_s(6 downto 0) <= registre_hex1_mem;
455             when 264 =>
456                 axi_rdata_mem_s(6 downto 0) <= registre_hex2_mem;
457             when 268 =>
458                 axi_rdata_mem_s(6 downto 0) <= registre_hex3_mem;
459             when 272 =>
460                 axi_rdata_mem_s(6 downto 0) <= registre_hex4_mem;
461             when 276 =>
462                 axi_rdata_mem_s(6 downto 0) <= registre_hex5_mem;
463
464
465             when others =>
466                 axi_rresp_s <= "00";
467         end case;
468
469     else
470         axi_rvalid_s <= '0';
471         axi_rresp_s  <= "--";
472
473     end if;
474 end if;
475 end process;
476

```

```
477      -- Mise à jour de la ligne l'interruption
478      irq_o <= irq_s;
479
480      -- Mise à jour de la validité de lecture
481      axi_rvalid_o <= axi_rvalid_s;
482
483      -- Mise à jour des données lues
484      axi_rdata_o <= axi_rdata_mem_s;
485
486      -- Mise à jour de la réponse de lecture
487      axi_rresp_o <= axi_rresp_s;
488
489
490      -- Mise à jour des sorties
491      leds_o(9 downto 0)      <= registre_led_mem;
492
493      hex0_o(6 downto 0)      <= registre_hex0_mem;
494      hex1_o(6 downto 0)      <= registre_hex1_mem;
495      hex2_o(6 downto 0)      <= registre_hex2_mem;
496      hex3_o(6 downto 0)      <= registre_hex3_mem;
497      hex4_o(6 downto 0)      <= registre_hex4_mem;
498      hex5_o(6 downto 0)      <= registre_hex5_mem;
499
500
501 end rtl;
502
```



```

1  -----
2  --
3  -- HEIG-VD
4  -- Haute Ecole d'Ingenierie et de Gestion du Canton de Vaud
5  -- School of Business and Engineering in Canton de Vaud
6  -----
7  --
8  -- REDS Institute
9  -- Reconfigurable Embedded Digital Systems
10 -----
11 --
12 -- File           : DE1_SoC_top.vhd
13 -- Author          : Sébastien Masle
14 -- Date            : 17.01.2018
15 --
16 -- Context         : HPA
17 -----
18 --
19 -- Description : top design for DE1-SoC board
20 -----
21 --
22 -- Dependencies :
23 -----
24 --
25 -- Modifications :
26 -- Ver    Date      Engineer    Comments
27 -- 0.0    17.01.2018 SMS         Initial version.
28 -----
29
30
31 library ieee;
32 use ieee.std_logic_1164.all;
33
34 entity DE1_SoC_top is
35     port ( -- clock pins
36         CLOCK_50_i : in  std_logic;
37         CLOCK2_50_i : in  std_logic;
38         CLOCK3_50_i : in  std_logic;
39         CLOCK4_50_i : in  std_logic;
40
41         -- ADC
42         ADC_CS_N_o : out std_logic;
43         ADC_DIN_o  : out std_logic;
44         ADC_DOUT_i : in  std_logic;
45         ADC_SCLK_o : out std_logic;
46
47         -- Audio
48         AUD_ADCLRCK_io : inout std_logic;
49         AUD_ADCDATA_i  : in  std_logic;
50         AUD_DACLCK_io  : inout std_logic;
51         AUD_DACDATA_o  : out std_logic;
52         AUD_XCK_o       : out std_logic;
53         AUD_BCLK_io     : inout std_logic;
54
55         -- SDRAM
56         DRAM_ADDR_o : out std_logic_vector(12 downto 0);
57         DRAM_BA_o   : out std_logic_vector(1  downto 0);
58         DRAM_CAS_N_o : out std_logic;
59         DRAM_CKE_o   : out std_logic;
60         DRAM_CLK_o   : out std_logic;
61         DRAM_CS_N_o  : out std_logic;
62         DRAM_DQ_io   : inout std_logic_vector(15 downto 0);
63         DRAM_LDQM_o  : out std_logic;
64         DRAM_RAS_N_o : out std_logic;
65         DRAM_UDQM_o  : out std_logic;

```

```

63     DRAM_WE_N_o    : out std_logic;
64
65     --I2C Bus for Configuration of the Audio and Video-In Chips
66     FPGA_I2C_SCLK_o : out std_logic;
67     FPGA_I2C_SDAT_io : inout std_logic;
68
69     -- 40-pin headers
70     GPIO_0_io       : inout std_logic_vector(35 downto 0);
71     GPIO_1_io       : inout std_logic_vector(35 downto 0);
72
73     -- Seven Segment Displays
74     HEX0_o          : out std_logic_vector(6 downto 0);
75     HEX1_o          : out std_logic_vector(6 downto 0);
76     HEX2_o          : out std_logic_vector(6 downto 0);
77     HEX3_o          : out std_logic_vector(6 downto 0);
78     HEX4_o          : out std_logic_vector(6 downto 0);
79     HEX5_o          : out std_logic_vector(6 downto 0);
80
81     -- IR
82     IRDA_RXD_i      : in std_logic;
83     IRDA_TXD_o      : out std_logic;
84
85     -- Pushbuttons
86     KEY_i           : in std_logic_vector(3 downto 0);
87
88     -- LEDs
89     LEDR_o          : out std_logic_vector(9 downto 0);
90
91     -- PS2 Ports
92     PS2_CLK_io      : inout std_logic;
93     PS2_DAT_io      : inout std_logic;
94     PS2_CLK2_io     : inout std_logic;
95     PS2_DAT2_io     : inout std_logic;
96
97     -- Slider Switches
98     SW_i            : in std_logic_vector(9 downto 0);
99
100    -- Video-In
101    TD_CLK27_i       : in std_logic;
102    TD_DATA_i        : in std_logic_vector(7 downto 0);
103    TD_HS_i          : in std_logic;
104    TD_RESET_N_o     : out std_logic;
105    TD_VS_i          : in std_logic;
106
107    -- VGA
108    VGA_R_o          : out std_logic_vector(7 downto 0);
109    VGA_G_o          : out std_logic_vector(7 downto 0);
110    VGA_B_o          : out std_logic_vector(7 downto 0);
111    VGA_CLK_o        : out std_logic;
112    VGA_SYNC_N_o     : out std_logic;
113    VGA_BLANK_N_o    : out std_logic;
114    VGA_HS_o         : out std_logic;
115    VGA_VS_o         : out std_logic;
116
117    -- DDR3 SDRAM
118    HPS_DDR3_ADDR_o   : out std_logic_vector(14 downto 0);
119    HPS_DDR3_BA_o     : out std_logic_vector(2 downto 0);
120    HPS_DDR3_CAS_N_o  : out std_logic;
121    HPS_DDR3_CKE_o    : out std_logic;
122    HPS_DDR3_CK_N_o   : out std_logic;
123    HPS_DDR3_CK_P_o   : out std_logic;
124    HPS_DDR3_CS_N_o   : out std_logic;
125    HPS_DDR3_DM_o     : out std_logic_vector(3 downto 0);
126    HPS_DDR3_DQ_io    : inout std_logic_vector(31 downto 0);
127    HPS_DDR3_DQS_N_io : inout std_logic_vector(3 downto 0);
128    HPS_DDR3_DQS_P_io : inout std_logic_vector(3 downto 0);
129    HPS_DDR3_ODT_o    : out std_logic;
130    HPS_DDR3_RAS_N_o  : out std_logic;
131    HPS_DDR3_RESET_N_o : out std_logic;

```

```

132     HPS_DDR3_RZQ_i           : in std_logic;
133     HPS_DDR3_WE_N_o          : out std_logic;
134
135     -- Ethernet
136     --HPS_ENET_GTX_CLK_o      : out std_logic;
137     --HPS_ENET_INT_N_io      : inout std_logic;
138     --HPS_ENET_MDC_o          : out std_logic;
139     --HPS_ENET_MDIO_io       : inout std_logic;
140     --HPS_ENET_RX_CLK_i       : in std_logic;
141     --HPS_ENET_RX_DATA_i      : in std_logic_vector(3 downto 0);
142     --HPS_ENET_RX_DV_i        : in std_logic;
143     --HPS_ENET_TX_DATA_o      : out std_logic_vector(3 downto 0);
144     --HPS_ENET_TX_EN_o        : out std_logic;
145
146     -- Flash
147     --HPS_FLASH_DATA_io       : inout std_logic_vector(3 downto 0);
148     --HPS_FLASH_DCLK_o        : out std_logic;
149     --HPS_FLASH_NCSO_o        : out std_logic;
150
151     -- Accelerometer
152     --HPS_GSENSOR_INT_io      : inout std_logic;
153
154     -- General Purpose I/O
155     --HPS_GPIO_io             : inout std_logic_vector(1 downto 0);
156
157     -- I2C
158     --HPS_I2C_CONTROL_io      : inout std_logic;
159     --HPS_I2C1_SCLK_io        : inout std_logic;
160     --HPS_I2C1_SDAT_io        : inout std_logic;
161     --HPS_I2C2_SCLK_io        : inout std_logic;
162     --HPS_I2C2_SDAT_io        : inout std_logic;
163
164     -- Pushbutton
165     HPS_KEY_io                : inout std_logic;
166
167     -- LED
168     HPS_LED_io                : inout std_logic;
169
170     -- SD Card
171     --HPS_SD_CLK_o            : out std_logic;
172     --HPS_SD_CMD_io           : inout std_logic;
173     --HPS_SD_DATA_io          : inout std_logic_vector(3 downto 0);
174
175     -- SPI
176     --HPS_SPIM_CLK_o          : out std_logic;
177     --HPS_SPIM_MISO_i          : in std_logic;
178     --HPS_SPIM_MOSI_o         : out std_logic;
179     --HPS_SPIM_SS_io          : inout std_logic;
180
181     -- UART
182     --HPS_UART_RX_i           : in std_logic;
183     --HPS_UART_TX_o           : out std_logic;
184
185     -- USB
186     --HPS_CONV_USB_N_io       : inout std_logic;
187     --HPS_USB_CLKOUT_i         : in std_logic;
188     --HPS_USB_DATA_io         : inout std_logic_vector(7 downto 0);
189     --HPS_USB_DIR_i           : in std_logic;
190     --HPS_USB_NXT_i           : in std_logic;
191     --HPS_USB_STP_o           : out std_logic;
192
193     -- LTC connector
194     --HPS_LTC_GPIO_io         : inout std_logic;
195
196     -- FAN
197     FAN_CTRL_o                : out std_logic
198 );
199 end DE1_SoC_top;
200

```

```

201 architecture top of DE1_SoC_top is
202
203     component qsys_system is
204     port (
205         -----
206         -- FPGA Side
207         -----
208
209         -- Global signals
210         clk_clk          : in    std_logic          :=
                'X';          -- clk
211
212         -----
213         -- HPS Side
214         -----
215         -- DDR3 SDRAM
216         memory_mem_a      : out    std_logic_vector(14 downto
                0);          -- mem_a
217         memory_mem_ba      : out    std_logic_vector(2 downto
                0);          -- mem_ba
218         memory_mem_ck      : out    std_logic          -- mem_ck
219         memory_mem_ck_n    : out    std_logic          -- mem_ck_n
220         memory_mem_cke      : out    std_logic          -- mem_cke
221         memory_mem_cs_n    : out    std_logic          -- mem_cs_n
222         memory_mem_ras_n    : out    std_logic          -- mem_ras_n
223         memory_mem_cas_n    : out    std_logic          -- mem_cas_n
224         memory_mem_we_n    : out    std_logic          -- mem_we_n
225         memory_mem_reset_n : out    std_logic          -- mem_reset_n
226         memory_mem_dq       : inout  std_logic_vector(31 downto 0) :=
                (others => 'X'); -- mem_dq
227         memory_mem_dqs       : inout  std_logic_vector(3 downto 0) :=
                (others => 'X'); -- mem_dqs
228         memory_mem_dqs_n     : inout  std_logic_vector(3 downto 0) :=
                (others => 'X'); -- mem_dqs_n
229         memory_mem_odt       : out    std_logic          -- mem_odt
230         memory_mem_dm        : out    std_logic_vector(3 downto
                0);          -- mem_dm
231         memory_oct_rzqin     : in     std_logic          :=
                'X';          -- oct_rzqin
232
233         conduit_export_switch_i : in    std_logic_vector(31 downto
                0) := (others => 'X'); -- switch_i
234         conduit_export_key_i     : in    std_logic_vector(31 downto
                0) := (others => 'X'); -- key_i
235
236         conduit_export_leds_o    : out    std_logic_vector(31 downto
                0);          -- leds_o
237
238         conduit_export_hex0_o    : out    std_logic_vector(31 downto
                0);          -- hex0_o
239         conduit_export_hex1_o    : out    std_logic_vector(31 downto
                0);          -- hex1_o
240         conduit_export_hex2_o    : out    std_logic_vector(31 downto
                0);          -- hex2_o
241         conduit_export_hex3_o    : out    std_logic_vector(31 downto
                0);          -- hex3_o
242         conduit_export_hex4_o    : out    std_logic_vector(31 downto
                0);          -- hex4_o
243         conduit_export_hex5_o    : out    std_logic_vector(31 downto
                0);          -- hex5_o

```

```

244
245         -- Pushbutton
246         hps_io_hps_io_gpio_inst_GPIO54 : inout std_logic           :=
        'X';           -- hps_io_gpio_inst_GPIO54

247
248         -- LED
249         hps_io_hps_io_gpio_inst_GPIO53 : inout std_logic           :=
        'X';           -- hps_io_gpio_inst_GPIO53

250     );
251     end component qsys_system;
252
253 begin
254
255     -----
256     -- HPS mapping
257     -----
258
259     System : component qsys_system
260     port map (
261         -----
262         -- FPGA Side
263         -----
264
265         -- Global signals
266         clk_clk          => CLOCK_50_i,
267
268         -----
269         -- HPS Side
270         -----
271         -- DDR3 SDRAM
272         memory_mem_a      => HPS_DDR3_ADDR_o,
273         memory_mem_ba     => HPS_DDR3_BA_o,
274         memory_mem_ck     => HPS_DDR3_CK_P_o,
275         memory_mem_ck_n   => HPS_DDR3_CK_N_o,
276         memory_mem_cke    => HPS_DDR3_CKE_o,
277         memory_mem_cs_n   => HPS_DDR3_CS_N_o,
278         memory_mem_ras_n  => HPS_DDR3_RAS_N_o,
279         memory_mem_cas_n  => HPS_DDR3_CAS_N_o,
280         memory_mem_we_n   => HPS_DDR3_WE_N_o,
281         memory_mem_reset_n => HPS_DDR3_RESET_N_o,
282         memory_mem_dq     => HPS_DDR3_DQ_io,
283         memory_mem_dqs    => HPS_DDR3_DQS_P_io,
284         memory_mem_dqs_n  => HPS_DDR3_DQS_N_io,
285         memory_mem_odt    => HPS_DDR3_ODT_o,
286         memory_mem_dm     => HPS_DDR3_DM_o,
287         memory_oct_rzqin  => HPS_DDR3_RZQ_i,
288
289         conduit_export_switch_i (9 downto 0)           => SW_i ,           -- switch_i
290         conduit_export_switch_i (31 downto 10)         => (others => '0') ,
291         conduit_export_key_i   (3 downto 0)            => KEY_i ,           --
        key_i
292         conduit_export_key_i   (31 downto 4)           => (others => '0') ,
293
294         conduit_export_leds_o  (9 downto 0)            => LEDR_o ,           --
        leds_o
295
296         conduit_export_hex0_o  (6 downto 0)            => HEX0_o ,           --
        hex0_o
297         conduit_export_hex1_o  (6 downto 0)            => HEX1_o ,           --
        hex1_o
298         conduit_export_hex2_o  (6 downto 0)            => HEX2_o ,           --
        hex2_o
299         conduit_export_hex3_o  (6 downto 0)            => HEX3_o ,           --
        hex3_o
300         conduit_export_hex4_o  (6 downto 0)            => HEX4_o ,           --
        hex4_o
301         conduit_export_hex5_o  (6 downto 0)            => HEX5_o ,           -- hex5_o
302
303

```

```
304         -- Pushbutton
305         hps_io_hps_io_gpio_inst_GPIO54 => HPS_KEY_io,
306
307         -- LED
308         hps_io_hps_io_gpio_inst_GPIO53 => HPS_LED_io
309     );
310
311 end top;
```

```

1  /*****
2  *
3  * HEIG-VD
4  * Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud
5  * School of Business and Engineering in Canton de Vaud
6  *
7  * REDS Institute
8  * Reconfigurable Embedded Digital Systems
9  *
10 * File           : labo5.c
11 * Author          : Spinelli Isaia
12 * Date            : 01.05.2020
13 *
14 * Context         : SOCF tutorial lab
15 *
16 *****/
17 * Brief: Programme for labo 5 of SOCF, for DE1-SoC board
18 *
19 *
20 *****/
21 * Modifications :
22 * Ver    Date      Student      Comments
23 * 0.1    01.05.20   Isaia Spinelli : Modif pour la partie 1
24 * 1.1    03.05.20   Isaia Spinelli : Ajout de la partie 2
25 * 1.2    08.05.20   Isaia Spinelli : Test du paramètre edge des irq
26 *****/
27 /
28 #include "defines.h"
29
30
31 /* Variable globales */
32
33 int irqKey2 = 0;
34 int irqKey3 = 0;
35
36
37
38
39 int main(void){
40
41     // tableau de conversion
42     8      9      a      b      c      d      e      f
43     char tab_dec_to_hex_7seg[16] = {0x40, 0xF9, 0x24, 0x30, 0x19, 0x12, 0x02, 0xF8,
44     0x00, 0x10, 0x08, 0x03, 0x27, 0x21, 0x06, 0x0e };
45     int led_tmp, Seg_tmp;
46
47     /*----- INTI -----*/
48     AXI_HEX5 = 0x40;
49     AXI_HEX4 = 0xF9;
50     AXI_HEX3 = 0x24;
51     AXI_HEX2 = 0x30;
52     AXI_HEX1 = 0x19;
53     AXI_HEX0 = 0x02;
54
55     AXI_LEDS = AXI_SWITCHES;
56
57     unsigned int cst = AXI_REG_CONST;
58     AXI_REG_TEST = cst;

```

```

58
59 // Masque le bouton key3 (pour tester le masquage des interruptions)
60 // AXI_INT_MASK = KEY3;
61
62 // KEY 3 sur flanc montant
63 AXI_INT_EDGE = KEY3;
64
65 disable_A9_interrupts(); // disable interrupts in the A9 processor
66 set_A9_IRQ_stack(); // initialize the stack pointer for IRQ mode
67 config_GIC(); // configure the general interrupt controller
68 config_KEYS(); // configure KEYS to generate interrupts
69 enable_A9_interrupts(); // enable interrupts in the A9 processor
70
71
72
73
74 while(1){
75     /* Appuie sur KEY 0 */
76     if ((AXI_KEYS & KEY0) == 0) {
77         // l'états des switches est copiés sur les LEDs.
78         AXI_LEDS = AXI_SWITCHES;
79         // Les afficheurs HEX5 à HEX0 affichent en hexadécimal les bits 23 à 0 de
80         // la constante définie dans l'IP.
81         AXI_HEX0 = tab_dec_to_hex_7seg[cst & 0xF];
82         AXI_HEX1 = tab_dec_to_hex_7seg[(cst>>4) & 0xF];
83         AXI_HEX2 = tab_dec_to_hex_7seg[(cst>>8) & 0xF];
84         AXI_HEX3 = tab_dec_to_hex_7seg[(cst>>12) & 0xF];
85         AXI_HEX4 = tab_dec_to_hex_7seg[(cst>>16) & 0xF];
86         AXI_HEX5 = tab_dec_to_hex_7seg[(cst>>20) & 0xF];
87
88         /* Appuie sur KEY 1 */
89     } else if ((AXI_KEYS & KEY1) == 0) {
90         // l'états inverses des switches est copiés sur les LEDs.
91         AXI_LEDS = ~AXI_SWITCHES;
92
93         // Les afficheurs HEX5 à HEX0 affichent en hexadécimal l'inverse des bits
94         // 23 à 0 de la
95         // constante définie dans l'IP.
96         AXI_HEX0 = ~tab_dec_to_hex_7seg[cst & 0xF];
97         AXI_HEX1 = ~tab_dec_to_hex_7seg[(cst>>4) & 0xF];
98         AXI_HEX2 = ~tab_dec_to_hex_7seg[(cst>>8) & 0xF];
99         AXI_HEX3 = ~tab_dec_to_hex_7seg[(cst>>12) & 0xF];
100        AXI_HEX4 = ~tab_dec_to_hex_7seg[(cst>>16) & 0xF];
101        AXI_HEX5 = ~tab_dec_to_hex_7seg[(cst>>20) & 0xF];
102
103        // Si le bouton 2 est pressé (via une interruption)
104    } else if (irqKey2) {
105        irqKey2 = 0;
106
107        /* l'affichage des LEDs et des afficheurs 7 segments subit unerotation à
108        droite */
109        led_tmp = AXI_LEDS & 0x1;
110        AXI_LEDS = ((AXI_LEDS & 0x3fff) >> 1) | (led_tmp << 9);
111
112        Seg_tmp = AXI_HEX0;
113        AXI_HEX0 = AXI_HEX1;
114        AXI_HEX1 = AXI_HEX2;
115        AXI_HEX2 = AXI_HEX3;
116        AXI_HEX3 = AXI_HEX4;
117        AXI_HEX4 = AXI_HEX5;
118        AXI_HEX5 = Seg_tmp;
119
120        // Si le bouton 3 est pressé (via une interruption)
121    } else if (irqKey3) {
122        irqKey3 = 0;
123
124        /* l'affichage des LEDs et des afficheurs 7 segments subit une rotation à

```



```

124     gauche */
125     led_tmp = AXI_LEDS & 0x200;
126     AXI_LEDS = (AXI_LEDS << 1) | (led_tmp >> 9);
127
128     Seg_tmp = AXI_HEX5;
129     AXI_HEX5 = AXI_HEX4;
130     AXI_HEX4 = AXI_HEX3;
131     AXI_HEX3 = AXI_HEX2;
132     AXI_HEX2 = AXI_HEX1;
133     AXI_HEX1 = AXI_HEX0;
134     AXI_HEX0 = Seg_tmp;
135
136     }
137
138     }
139 }
140
141 /* Routine d'interruption */
142 void pushbutton_ISR(void){
143     // Permet de tester le masquage
144     // static int cpt_int = 0;
145
146     /* Lecture et acquittement des interruptions */
147     int src_irq = AXI_INT_SRC;
148
149     // Key2 pressé
150     if (src_irq & KEY2) {
151         irqKey2 = 1;
152     }
153
154     // Key3 pressé
155     if (src_irq & KEY3) {
156         irqKey3 = 1;
157     }
158
159
160     // Tous les 3 interruptions de KEY0 et KEY1, change le masque de key 2 et 3
161     /*
162     if (src_irq & KEY0 || src_irq & KEY1) {
163         cpt_int++;
164
165         if (cpt_int % 3 == 0)
166             AXI_INT_MASK = AXI_INT_MASK ^ (KEY3 | KEY2);
167     }
168     */
169
170 }
171

```

```

1  /*****
2  *
3  * HEIG-VD
4  * Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud
5  * School of Business and Engineering in Canton de Vaud
6
7  *****/
8
9  *
10 * REDS Institute
11 * Reconfigurable Embedded Digital Systems
12
13 *****/
14
15 *
16 * File           : defines.h
17 * Author        : Sébastien Masle
18 * Date         : 16.02.2018
19 *
20 * Context       : SOCF class
21
22 *****/
23
24 * Brief: some definitions
25 *
26
27 *****/
28
29 * Modifications :
30 * Ver    Date      Engineer      Comments
31 * 0.0    16.02.2018 SMS           Initial version.
32 * 1.1    06.05.20  Isaia Spinelli : Refactor
33 * 1.2    08.05.20  Isaia Spinelli : Ajout du paramètre edge des irq
34 *****/
35 /
36
37 #include "exceptions.h"
38
39 // Déclaration de fonction
40 void pushbutton_ISR(void);
41
42 // Defines
43
44 #define EDGE_TRIGGERED      0x1
45 #define LEVEL_SENSITIVE    0x0
46 #define CPU0                0x01 // bit-mask; bit 0 represents cpu0
47 #define ENABLE              0x1
48
49
50 #define USER_MODE          0b10000
51 #define FIQ_MODE            0b10001
52 #define IRQ_MODE            0b10010
53 #define SVC_MODE            0b10011
54 #define ABORT_MODE          0b10111
55 #define UNDEF_MODE          0b11011
56 #define SYS_MODE            0b11111
57
58 #define INT_ENABLE          0b01000000
59 #define INT_DISABLE         0b11000000
60
61 // Valeur des keys
62 #define KEY0 0x01
63 #define KEY1 0x02
64 #define KEY2 0x04
65 #define KEY3 0x08
66
67 // Typedef
68 typedef volatile unsigned char vuint;
69 typedef volatile unsigned short vsint;
70 typedef volatile unsigned int vuint;

```

```
60
61 // Adresses
62 #define FPGA_BASE_ADDR_IO      0xFF200000
63 #define AXI_LIGHT_BASE_ADDR    FPGA_BASE_ADDR_IO
64
65
66 #define AXI_REG_CONST_CHAR      *(vcint *) (AXI_LIGHT_BASE_ADDR + 0x0)
67 #define AXI_REG_CONST_SHORT     *(vsint *) (AXI_LIGHT_BASE_ADDR + 0x0)
68 #define AXI_REG_CONST           *(vuint *) (AXI_LIGHT_BASE_ADDR + 0x0)
69
70 #define AXI_REG_TEST            *(vuint *) (AXI_LIGHT_BASE_ADDR + 0x4)
71
72 #define AXI_LEDS                *(vuint *) (AXI_LIGHT_BASE_ADDR + 0x100)
73
74 #define AXI_KEYS                *(vuint *) (AXI_LIGHT_BASE_ADDR + 0x200)
75 // Lecture de la source d'int. + acquitement
76 #define AXI_INT_SRC             *(vuint *) (AXI_LIGHT_BASE_ADDR + 0x204)
77 // 0 = interruption non masquée (défaut)
78 #define AXI_INT_MASK           *(vuint *) (AXI_LIGHT_BASE_ADDR + 0x208)
79 // 0 = interruption flanc descendant (défaut)
80 #define AXI_INT_EDGE           *(vuint *) (AXI_LIGHT_BASE_ADDR + 0x20C)
81
82
83 #define AXI_SWITCHES            *(vuint *) (AXI_LIGHT_BASE_ADDR + 0x300)
84
85 #define AXI_HEX0                *(vuint *) (AXI_LIGHT_BASE_ADDR + 0x400)
86 #define AXI_HEX1                *(vuint *) (AXI_LIGHT_BASE_ADDR + 0x410)
87 #define AXI_HEX2                *(vuint *) (AXI_LIGHT_BASE_ADDR + 0x420)
88 #define AXI_HEX3                *(vuint *) (AXI_LIGHT_BASE_ADDR + 0x430)
89 #define AXI_HEX4                *(vuint *) (AXI_LIGHT_BASE_ADDR + 0x440)
90 #define AXI_HEX5                *(vuint *) (AXI_LIGHT_BASE_ADDR + 0x450)
91
```

```

1  /*****
2  *
3  * HEIG-VD
4  * Haute Ecole d'Ingenierie et de Gestion du Canton de Vaud
5  * School of Business and Engineering in Canton de Vaud
6  *
7  * REDS Institute
8  * Reconfigurable Embedded Digital Systems
9  *
10 * File           : exceptions.h
11 * Author          : Isaia Spinelli
12 * Date            : 06.05.2020
13 *
14 * Context         : SOCF class
15 *
16 *****/
17 * Modifications :
18 * Ver    Date      Engineer    Comments
19 * 1.1    06.05.20   Isaia Spinelli : Refactor
20 *
21 *****/
22 /
23 void disable_A9_interrupts (void);
24 void set_A9_IRQ_stack (void);
25 void config_GIC (void);
26 void config_KEYS (void);
27 void enable_A9_interrupts (void);
28 void config_interrupt (int, int);

```

```

1  /*****
2  *
3  * HEIG-VD
4  * Haute Ecole d'Ingenierie et de Gestion du Canton de Vaud
5  * School of Business and Engineering in Canton de Vaud
6  *
7  * REDS Institute
8  * Reconfigurable Embedded Digital Systems
9  *
10 *
11 * File           : exceptions.c
12 * Author        : Sébastien Masle
13 * Date          : 16.02.2018
14 * Context       : SOCF class
15 *
16 *
17 * Brief: defines exception vectors for the A9 processor
18 *        provides code that sets the IRQ mode stack, and that dis/enables interrupts
19 *        provides code that initializes the generic interrupt controller
20 *
21 *
22 * Modifications :
23 * Ver    Date      Engineer      Comments
24 * 0.0    16.02.2018 SMS           Initial version.
25 * 1.0    13.03.2020 Spinelli Isaia
26 *
27 *****/
28 #include <stdint.h>
29
30 #include "address_map_arm.h"
31 #include "defines.h"
32
33
34
35
36 // Référence : Exemple dans Using The ARM Generic
37
38 // Define the IRQ exception handler
39 void __attribute__((interrupt)) __cs3_isr_irq(void)
40 {
41     /*****
42      * Attention dans Qsys mettre sur flanc et non level !
43      *****/
44
45     // Read CPU Interface registers to determine which peripheral has caused an
46     // interrupt
47     int interrupt_ID = *((int*) 0xFFFFEC10C);
48
49     // Handle the interrupt if it comes from the KEYS
50     if (interrupt_ID == 72) {
51         pushbutton_ISR();
52     } else {
53         while (1); // if unexpected, then stay here
54     }
55
56     // Clear interrupt from the CPU Interface
57     *((int*) 0xFFFFEC110) = interrupt_ID;
58
59     return;

```

```

59     }
60
61     // Define the remaining exception handlers
62     void __attribute__((interrupt)) __cs3_reset (void)
63     {
64         while(1);
65     }
66
67     void __attribute__((interrupt)) __cs3_isr_undef (void)
68     {
69         while(1);
70     }
71
72     void __attribute__((interrupt)) __cs3_isr_swi (void)
73     {
74         while(1);
75     }
76
77     void __attribute__((interrupt)) __cs3_isr_pabort (void)
78     {
79         while(1);
80     }
81
82     void __attribute__((interrupt)) __cs3_isr_dabort (void)
83     {
84         while(1);
85     }
86
87     void __attribute__((interrupt)) __cs3_isr_fiq (void)
88     {
89         while(1);
90     }
91
92     /*
93      * Initialize the banked stack pointer register for IRQ mode
94      */
95     void set_A9_IRQ_stack(void)
96     {
97         uint32_t stack, mode;
98         stack = A9_ONCHIP_END - 7;          // top of A9 onchip memory, aligned to 8 bytes
99         /* change processor to IRQ mode with interrupts disabled */
100        mode = INT_DISABLE | IRQ_MODE;
101        asm("msr cpsr, %[ps]" : : [ps] "r" (mode));
102        /* set banked stack pointer */
103        asm("mov sp, %[ps]" : : [ps] "r" (stack));
104
105        /* go back to SVC mode before executing subroutine return! */
106        mode = INT_DISABLE | SVC_MODE;
107        asm("msr cpsr, %[ps]" : : [ps] "r" (mode));
108    }
109
110    /*
111     * Turn on interrupts in the ARM processor
112     */
113    void enable_A9_interrupts(void)
114    {
115        uint32_t status = SVC_MODE | INT_ENABLE;
116        asm("msr cpsr, %[ps]" : : [ps] "r" (status));
117    }
118
119    /** Turn off interrupts in the ARM processor*/
120    void disable_A9_interrupts(void) {
121        int status = 0b11010011;
122        asm("msr cpsr, %[ps]" : : [ps] "r" (status));
123    }
124
125    void config_GIC (void) {
126        // configure the FPGA KEYs interrupt (72)
127        config_interrupt (72, 1);

```

```

128
129 // Set Interrupt Priority Mask Register (ICCPMR). Enable all priorities
130 *((int*) 0xFFFFEC104) = 0xFFFF;
131
132 // Set the enable in the CPU Interface Control Register (ICCICR)
133 *((int*) 0xFFFFEC100) = 1;
134
135 // Set the enable in the Distributor Control Register (ICDDCR)
136 *((int*) 0xFFFFED000) = 1;
137 }
138
139 void config_KEYS (void) {
140     volatile int*KEY_ptr = (int*) 0xFF200050; // KEY base address
141
142     *(KEY_ptr + 2) = 0xF; // enable interrupts for all four KEYS
143
144 }
145
146 void config_interrupt (int N, int CPU_target) {
147     int reg_offset, index, value, address;
148
149     /*Configure the Interrupt Set-Enable Registers (ICDISERn).
150     *reg_offset = (integer_div(N / 32)*4; value = 1 << (N mod 32)*/
151
152     reg_offset = (N >> 3) & 0xFFFFFFF0;
153     index = N & 0x1F;
154     value = 0x1 << index;
155     address = 0xFFFFED100 + reg_offset;
156
157     /*Using the address and value, set the appropriate bit*/
158     *(int*)address |= value;
159
160     /*Configure the Interrupt Processor Targets Register (ICDIPTRn)
161     * reg_offset = integer_div(N / 4)*4; index = N mod 4*/
162     reg_offset = (N & 0xFFFFFFF0);
163     index = N & 0x3;
164     address = 0xFFFFED800 + reg_offset + index;
165
166     /*Using the address and value, write to (only) the appropriate byte*/
167     *(char*)address = (char) CPU_target;
168 }
169

```

```

1  /*****
2  *
3  * HEIG-VD
4  * Haute Ecole d'Ingenierie et de Gestion du Canton de Vaud
5  * School of Business and Engineering in Canton de Vaud
6  *
7  * REDS Institute
8  * Reconfigurable Embedded Digital Systems
9  *
10 *
11 * File           : address_map_arm.h
12 * Author        : Sébastien Masle
13 * Date          : 16.02.2018
14 * Context       : SOCF class
15 *
16 *****/
17 * Brief: provides address values that exist in the system
18 *
19 *****/
20 * Modifications :
21 * Ver    Date      Engineer    Comments
22 * 0.0    16.02.2018 SMS         Initial version.
23 *
24 *****/
25 /
26 #define BOARD                "DE1-SoC"
27
28 /* Memory */
29 #define DDR_BASE              0x00000000
30 #define DDR_END               0x3FFFFFFF
31 #define A9_ONCHIP_BASE       0xFFFFF000
32 #define A9_ONCHIP_END        0xFFFFFFFF
33 #define SDRAM_BASE           0xC0000000
34 #define SDRAM_END            0xC3FFFFFF
35 #define FPGA_ONCHIP_BASE     0xC8000000
36 #define FPGA_ONCHIP_END      0xC803FFFF
37 #define FPGA_CHAR_BASE       0xC9000000
38 #define FPGA_CHAR_END        0xC9001FFF
39

```