



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 : 8 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	3
2.3	Partie 3	3
2.4	Adress Map finale	4
3	Réalisation et implémentation	5
3.1	Ajout des PIOs	5
3.2	Activation des interruptions dans Qsys	7
3.3	Programme pour les features sans interruption	7
3.4	Activation des registres du GIC et de la FPGA dans le code	8
3.5	Routine d'interruption	9
4	Simulation et tests	10
5	Conclusion	13
6	Signatures	13

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 switchs (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'inverse 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 était principalement une marche à suivre. Il n'y avait pas réellement d'analyse à effectuer sur cette partie du laboratoire. Le but était de suivre la marche à suivre et de comprendre les actions que nous réalisions.

2.2 Partie 2

Cette partie nous demandais d'ajouter la gestion des leds et des afficheurs 7 segments en fonction des switches et sans gérer pour le moment les interruptions (Key2 et Key3). Ce qui est intéressant dans cette partie, c'est l'ajout des Keys et des afficheurs 7 segments en tant que PIO, en se servant de l'expérience reçue lors de la réalisation de la première partie.

2.3 Partie 3

Cette partie demandait un peu plus d'analyses que les deux parties précédentes. Il n'y avait pas de PIO à ajouter, par contre la gestion des interruptions devait être

mise en place pour chacun des différents éléments du système. Ainsi, il fallait prendre en compte le CPU, le GIC et les PIO.

Pour ce qui concerne le CPU, la configuration des vecteurs d'interruptions était fournie, mais la routine d'interruption devait être implémentée.

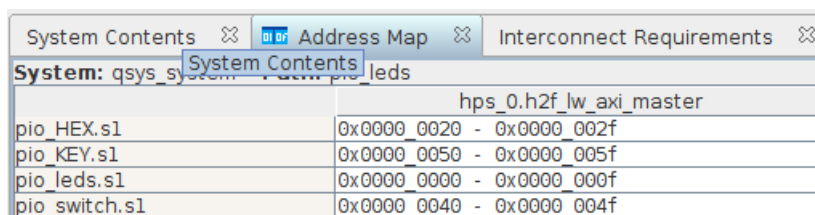
Concernant le GIC, je me suis inspiré des exemples de codes fournis dans le document "Gic_Altera_Manuel_short.pdf", et j'ai dû comprendre comment configurer le GIC. Voici les registres du GIC qui ont dû être configurés ou qui ont été utilisés lors de la routine d'interruption :

- ICCICR : Permet d'activer la transmission depuis le CPU interface jusqu'au CPU lui correspondant. C'est un enable.
- ICCPMR : Permet de set la priorité minimale pour qu'une interruption soit transmise au CPU.
- ICCIAR : Lors d'une interruption, contient l'ID de l'interruption en question.
- ICCEOIR : Permet au processeur de clear l'interrupt en écrivant l'ID correspondant à l'intérieur
- ICDDCR : Permet d'activer le distributor. C'est un enable.
- ICDISER : Permet d'activer une interruption (unmask)
- ICDIPTR : Permet de définir vers quel CPU doit être envoyé l'interruption.

A cela s'ajoutait deux registre , KEYS_INTERRUPT_ENABLE et KEY_INTERRUPT_REGISTER, qui permettait d'activer les interruptions des keys dans la FPGA et de récupérer ainsi que nettoyer lesdites interruptions.

Nous parlerons des valeurs qui ont été attribué à chaque registre dans la partie implémentation. Les adresses des registres sont basés sur les calculs fournis dans la documentation en fonction du numéro d'interruption 72, qui correspond au numéro d'interruption 0 de la FPGA (Nous verrons dans la partie implémentation d'où vient ce numéro).

2.4 Adress Map finale



System Contents		Address Map		Interconnect Requirements	
System: qsys_sys					
hps_0.h2f_lw_axi_master					
pio_HEX.s1	0x0000_0020	-	0x0000_002f		
pio_KEY.s1	0x0000_0050	-	0x0000_005f		
pio_leds.s1	0x0000_0000	-	0x0000_000f		
pio_switch.s1	0x0000_0040	-	0x0000_004f		

FIGURE 1 – Laboratoire 2 : Address map

Le choix des offsets ci-dessus vient à la base d'un problème de compréhension. J'avais choisi ces offsets en fonction des adresses correspondantes dans le memory layout fournit dans le document "DE1-SoC_Computer_ARM.pdf". Si cela n'apporte pas de problème en tant que tel, il faut tout de même noter que j'aurais pu en choisir d'autres, en faisant en sorte qu'ils soient contiguës par exemple.

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 Ajout des PIOs

Voici ce que l'on peut voir quand on ouvre le projet Qsys qui a été créé :

Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Tags	Opcode Name
✓	hps_0	memory	Arria V/Cyclone V Hard Process...	memory						
		h2f_reset	Reset Output	Double-click to	clk_0					
		h2f_w_axi_clock	Clock Input	Double-click to	h2f_w_axi...					
		h2f_w_axi_master	AXI Master	Double-click to						
		f2h_irq0	Interrupt Receiver	Double-click to						
		f2h_irq1	Interrupt Receiver	Double-click to						
✓		pio_leds	PIO (Parallel I/O)	clk	exported					
		clk_0	Clock Source	Double-click to	clk_0					
		clk_in	Clock Input	Double-click to						
		clk_in_reset	Reset Input	Double-click to						
		clk	Clock Output	Double-click to	clk_0					
		clk_reset	Reset Output	Double-click to						
✓		pio_leds	PIO (Parallel I/O)	clk						
		reset	Reset Input	Double-click to	clk_0					
		s1	Avalon Memory Mapped Slave	Double-click to	[clk]	0x0000_0000	0x0000_000f			
✓		pio_switch	PIO (Parallel I/O)	clk						
		reset	Reset Input	Double-click to	clk_0					
		s1	Avalon Memory Mapped Slave	Double-click to	[clk]	0x0000_0040	0x0000_004f			
✓		pio_HEX	PIO (Parallel I/O)	clk						
		reset	Reset Input	Double-click to	clk_0					
		s1	Avalon Memory Mapped Slave	Double-click to	[clk]	0x0000_0020	0x0000_002f			
✓		pio_KEY	PIO (Parallel I/O)	clk						
		reset	Reset Input	Double-click to	clk_0					
		s1	Avalon Memory Mapped Slave	Double-click to	[clk]	0x0000_0050	0x0000_005f			
		external_connec...	Conduit	Double-click to						
		irq	Interrupt Sender	Double-click to						

FIGURE 2 – Laboratoire 2 : PIOs

On peut voir que 4 parallel I/O (pio) ont été ajoutés, un pour chacun des éléments de notre système que l'on souhaitait gérer.

La configuration de ces pio est la suivante :

- pio_leds : une longueur de 10 (Pour les 10 leds) et set en output
- pio_switch : une longueur de 10 (Pour les 10 switches) et set en input
- pio_HEX : une longueur de 32 (8 bits par afficheur, avec 2 bits non utilisés, donc $4 \times 8 = 32$) et set en output
- pio_KEY : un longueur de 4 (Pour les 4 boutons), set en input, avec l'edge capture en rising (Pour récupérer l'appui sur le bouton) et avec les irq actives (irq de type EDGE, afin de se basé sur l'état de l'edge capture pour créer une interruption)

Le câblage avec les autres éléments consistait à brancher toutes les clocks sur la clock du hps, ainsi que les resets (sur le reset du hps). De plus, pour le cas des KEYS, l'interruption est branchée sur la première "banque" d'interruption de la fpga qui va sur le hps (f2h_irq0).

La génération du VHDL s'est passé sans problème, et j'ai dû ajouter les pio que j'avais créés au fichier DE1_SoC_top.vhd. Pour ce faire, j'ai utilisé qsys et "Show instantiation template" :

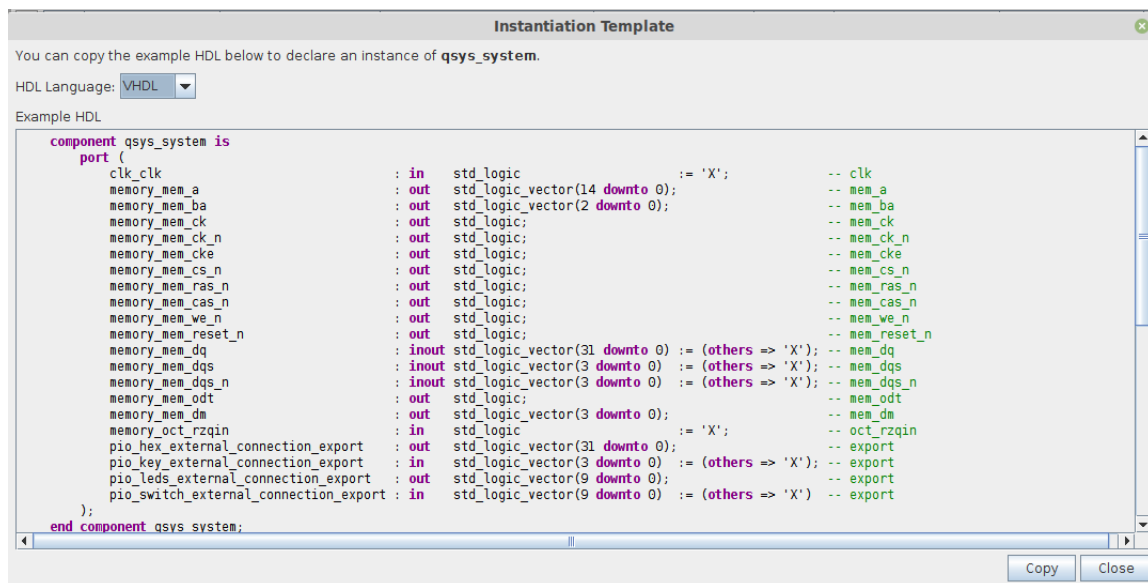


FIGURE 3 – Laboratoire 2 : Instantiation template

et j'ai ensuite modifié le fichier DE1_SoC_top.vhd afin de permettre l'utilisation des pios. J'ai copié les lignes que j'avais trouvées dans l'instanciation template et je les ai mises dans l'import du composant qsys_system dans le fichier DE1_Soc_top. De plus, j'ai dû mapper les entrées et sorties du composant dans le même fichier :

```

1 System : component qsys_system
2 port map (
3 -----
4 -- FPGA Side
5 -----
6
7 -- Global signals
8
9 -----
10 -- HPS Side
11 -----
12 -- DDR3 SDRAM
13 clk_clk => CLOCK_50_i,
14 pio_leds_external_connection_export => LEDR_o,
15 pio_switch_external_connection_export => SW_I,
16 pio_hex_external_connection_export => temp_hex_s,
17 pio_key_external_connection_export => KEY_i,
18 memory_mem_a => HPS_DDR3_ADDR_o,
19 memory_mem_ba => HPS_DDR3_BA_o,
20 memory_mem_ck => HPS_DDR3_CK_P_o,
21 memory_mem_ck_n => HPS_DDR3_CK_N_o,
22 memory_mem_cke => HPS_DDR3_CKE_o,
23 memory_mem_cs_n => HPS_DDR3_CS_N_o,
24 memory_mem_ras_n => HPS_DDR3_RAS_N_o,
25 memory_mem_cas_n => HPS_DDR3_CAS_N_o,
26 memory_mem_we_n => HPS_DDR3_WE_N_o,
27 memory_mem_reset_n => HPS_DDR3_RESET_N_o,
28 memory_mem_dq => HPS_DDR3_DQ_io,
29 memory_mem_dqs => HPS_DDR3_DQS_P_io,

```

```

30 memory_mem_dqs_n    => HPS_DDR3_DQS_N_io,
31 memory_mem_odt      => HPS_DDR3_ODT_o,
32 memory_mem_dm       => HPS_DDR3_DM_o,
33 memory_oct_rzqin    => HPS_DDR3_RZQ_i
34 );
35
36 HEX0_o <= temp_hex_s(6 downto 0);
37 HEX1_o <= temp_hex_s(14 downto 8);
38 HEX2_o <= temp_hex_s(22 downto 16);
39 HEX3_o <= temp_hex_s(30 downto 24);

```

Comme on peut le voir ci-dessus, j'ai de plus dû créer un signal temporaire permettant de dispatcher la sortie du pio HEX entre les différentes sorties HEX qui étaient présentes dans le fichier.

3.2 Activation des interruptions dans Qsys

Dans Qsys, plusieurs configurations devaient être effectuées afin de pouvoir utiliser les interruptions. Voici une liste des démarches effectuées :

- Il faut activer dans la configuration de hps_0 , sous "Fpga inteface" -> "interrupts", l'option "Enable FPGA-to-HPS interrupts"
- le/les PIO qui doit bénéficier des interruptions doit être linké sur l'un des deux champs "f2h_irq0/1".
- Dans la colonne IRQ aussi, l'interruption du pio doit être linkée sur l'un des deux champs vu plus haut. Dans notre cas, l'interruption porte le numéro 0 et est linkée sur "f2h_irq0", ce qui implique que le numéro de l'interruption sera le numéro 0 de la fpga(L'image ci-dessous montre le numéro de l'interruption, qui est donc 72)

72	FPGA	FPGA_IRQ0	—	Level or Edge
----	------	-----------	---	---------------

FIGURE 4 – Laboratoire 2 : Numéro de l'interruption

Voici les modifications qui étaient nécessaire dans Qsys afin d'activer les interruptions (En plus de celle déjà configurée lors de la création des pios)

3.3 Programme pour les features sans interruption

Le fonctionnement du programme pour les parties 1 et 2 du laboratoire a été entièrement commenté directement dans le code.

```

1   while(1)
2   {
3       // Si on appuie sur le Bouton KEY1
4       if(!(KEYS & 0x1))
5       {
6           // On eteind les leds et l'afficheur 7 segments

```

```

7      LEDS = 0x0;
8      HEX3_0 = ~0x0;
9
10     // On reporte l'état des switchs sur les leds
11     LEDS = SWITCHS;
12
13     // On effectue les manipulations demandées pour ...
14     // l'afficheur 7 segments pour KEY1
15     HEX3_0 = ~(0x0 | (temp[(LEDS & 0x200) >> 9] << 24) | ...
16     (temp[(LEDS & 0x100) >> 8] << 16) | (temp[(LEDS & ...
17     0xF0) >> 4] << 8) | (temp[(LEDS & 0xF)]));
18
19 }
20 // Si on appuie sur le bouton KEY2
21 else if(!(KEYS & 0x2))
22 {
23     // On eteind les leds et l'afficheur 7 segments
24     LEDS = 0x0;
25     HEX3_0 = ~0x0;
26
27     // On reporte l'état des switchs sur les leds
28     LEDS = ~SWITCHS;
29
30     // On effectue les manipulations demandées pour ...
31     // l'afficheur 7 segments pour KEY2
32     HEX3_0 = ~(0x0 | (temp[(LEDS & 0x200) >> 9] << 24) | ...
33     (temp[(LEDS & 0x100) >> 8] << 16) | (temp[(LEDS & ...
34     0xF0) >> 4] << 8) | (temp[(LEDS & 0xF)]));
35
36 }
37 }

```

3.4 Activation des registres du GIC et de la FPGA dans le code

Nous allons nous intéresser ici aux valeurs attribuées aux registres dans la partie de code ci-dessous (Les commentaires expliquent le choix des valeurs) :

```

1 // Initialize the banked stack pointer register for IRQ mode
2 set_A9_IRQ_stack();
3
4 // On active l'envoi d'interruption au core via l'interface CPU 0
5 ICCICR = 1;
6
7 // On met le niveau nécessaire pour qu'une interruption soit ...
8 // transmise au cpu au minimum afin que toutes les ...
9 // interruptions passent.
10 ICCPMR = 0xFFFF;
11
12 // On active le distributeur
13 ICDDCR = 1;
14
15

```



```
13 // On calcule la valeur à mettre dans le registre ICDISER ...
    (Interrupt Set Enable Registers) et on remplit le registre
14 // Le calcul pour la valeur peut être trouvé dans le document ...
    sous source dans l'entête.
15 value = 0x1<<(72%32);
16 ICDISER |= value;
17
18 //On renseigne que notre interruption doit être envoyée à ...
    l'interface CPU 0
19 ICDIPTR = 1;
20
21
22 // On active les interruptions pour les boutons KEY3 et KEY2
23 KEYS_INTERRUPT_ENABLE = 0xC;
24
25 // On active les interruptions sur le processeur
26 enable_A9_interrupts();
```

3.5 Routine d'interruption

Voici la routine d'interruption qui a été mise en place pour ce laboratoire :

```
1 // Define the IRQ exception handler
2 void __attribute__((interrupt)) __cs3_isr_irq(void)
3 {
4     // On lit le registre de l'interface CPU pour savoir quel ...
        périphérique a causé l'interruption
5     int interrupt_ID = ICCIAR;
6     int hex_val, press;
7
8     press = KEYS_INTERRUPT_REGISTER; // On récupère le ...
        bouton qui à causer l'interruption
9     KEYS_INTERRUPT_REGISTER = press; // On nettoie ...
        l'interruption dans le registre des interruptions pour ...
        les KEYS
10
11     // Si l'interruption à été causer par un bouton
12     if(interrupt_ID == 72)
13     {
14         if (press & 0x4) // KEY2
15         {
16             // On deplace les leds vers la droite ainsi que les ...
                valeurs des afficheurs 7 segments
17             LEDS = (LEDS >> 1) | ((LEDS & (0x1 ))<<9);
18             hex_val = HEX3_0;
19             HEX3_0 = ~0x0;
20             HEX3_0 = (0x0 | ((hex_val & (0x7F)) << 24) | ...
                ((hex_val & (0x7F << 24)) >> 8) | ((hex_val & ...
                (0x7F << 16)) >> 8) | ((hex_val & (0x7F << 8))>>8));
21         }
22
23         else if (press & 0x8) // KEY3
```

```

24     {
25         // On deplace les leds vers la gauche ainsi que les ...
           valeurs des afficheurs 7 segments
26         LEDS = (LEDS << 1) | ((LEDS & (0x1 << 9))>>9);
27         hex_val = HEX3_0;
28         HEX3_0 = ~0x0;
29         HEX3_0 = (0x0 | ((hex_val & (0x7F << 16)) << 8) | ...
           ((hex_val & (0x7F << 8)) << 8) | ((hex_val & ...
           (0x7F )) << 8) | ((hex_val & (0x7F << 24))>>24));
30     }
31
32 }
33
34 // On nettoie l'interruption dans le registre de ...
           interruptions pour le processeur
35 ICCEOIR = interrupt_ID;
36 return;
37 }

```

4 Simulation et tests

Il n'y avait pas de simulation à faire dans ce laboratoire. Cependant, des tests ont été effectués tout au long du laboratoire. De plus, le contrôle du fonctionnement de l'application avait été validé auprès de l'assistant avant la période de confinement. Je note tout de même ici les tests qui ont été effectués par moi-même afin de m'assurer du bon fonctionnement de l'application :

J'ai donc commencé par le plus simple. Tous les switches down et un appui sur key 0 suivi d'un appui sur key 1 :

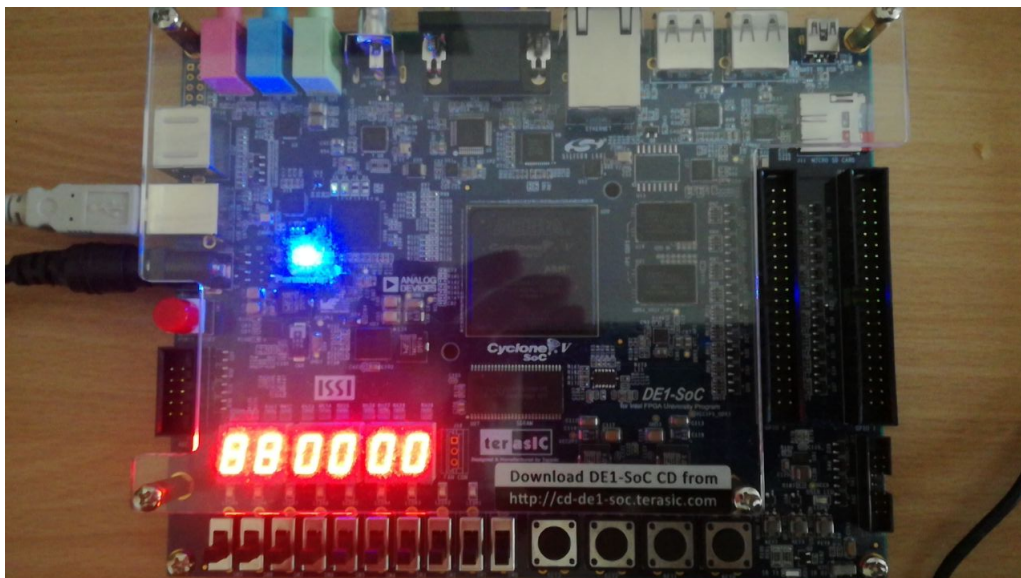


FIGURE 5 – Laboratoire 2 : Appui key 0 avec switch down

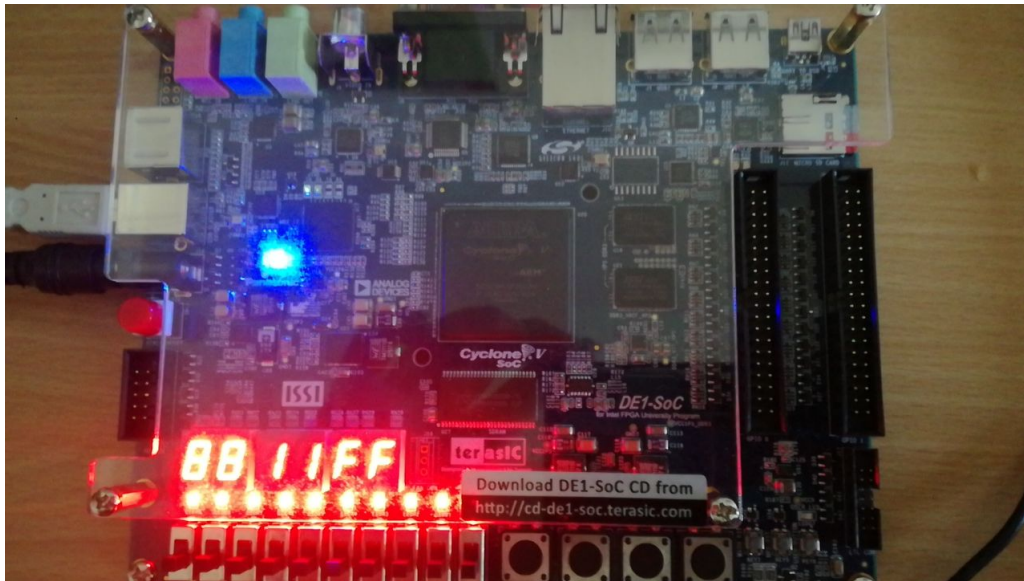


FIGURE 6 – Laboratoire 2 : Appui key 1 avec switch down
J’ai ensuite continué en alternant un switch down et un switch up en commençant par activer le switch 0, et j’ai appuyé sur key 0 puis sur key 1 :

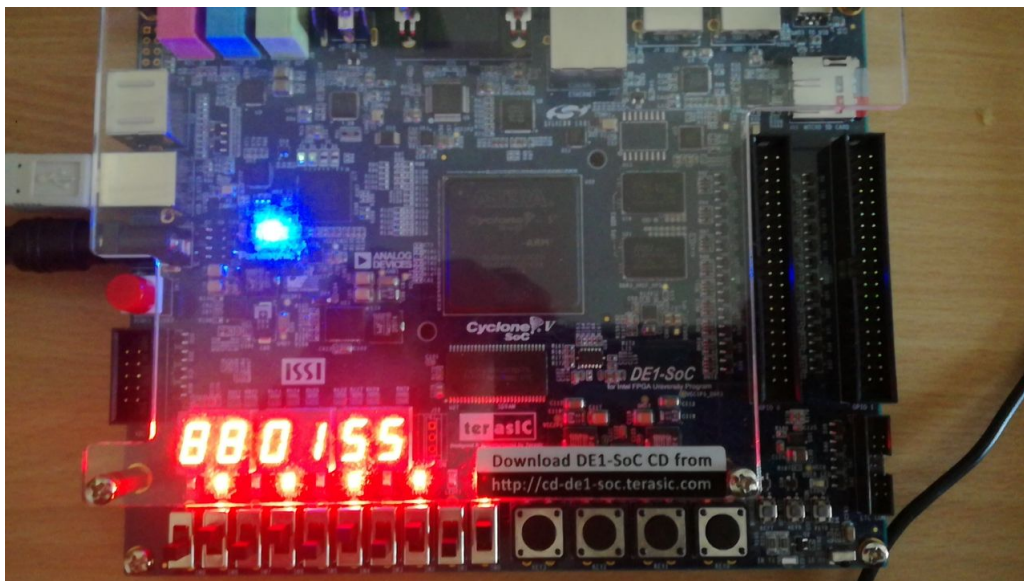


FIGURE 7 – Laboratoire 2 : Appui key 0 avec switch up/down

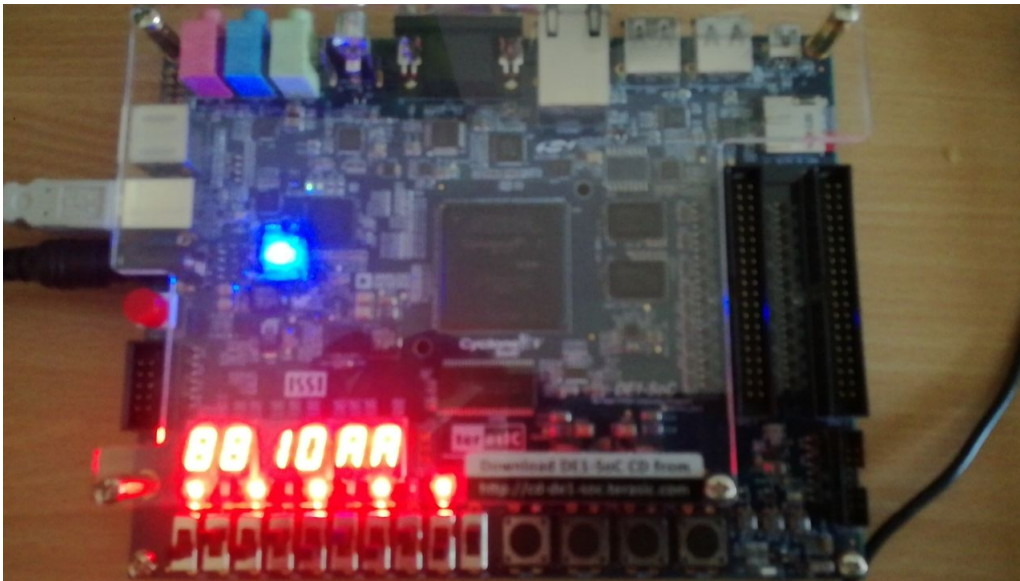


FIGURE 8 – Laboratoire 2 : Appui key 1 avec switch up/down
Les valeurs correspondaient au fonctionnement demandé par la donnée. Toujours avec cet état de switch, j'ai essayé les interruptions en appuyant deux fois sur key 3 puis une fois sur key 2

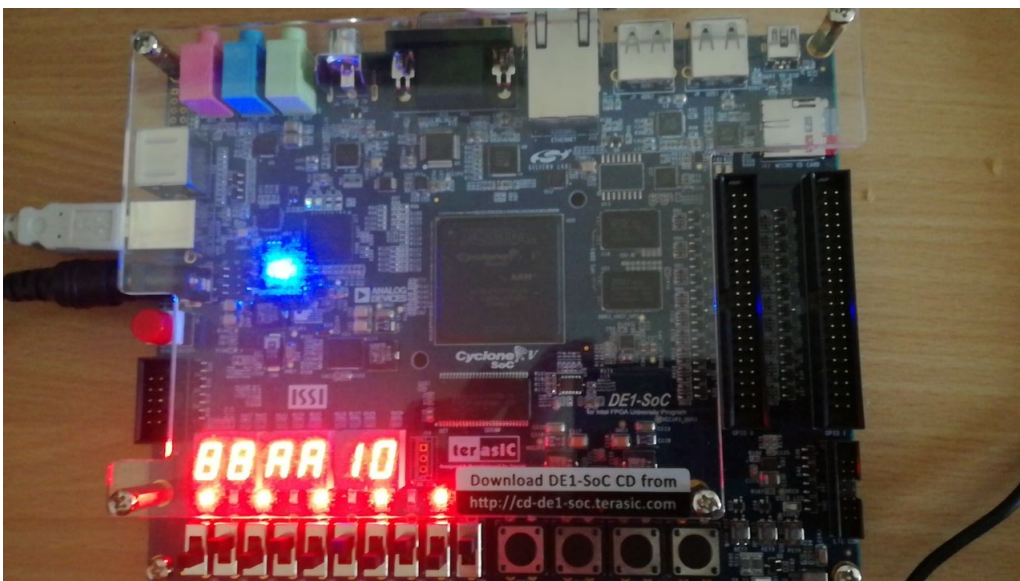


FIGURE 9 – Laboratoire 2 : Appui 2 fois sur key 3

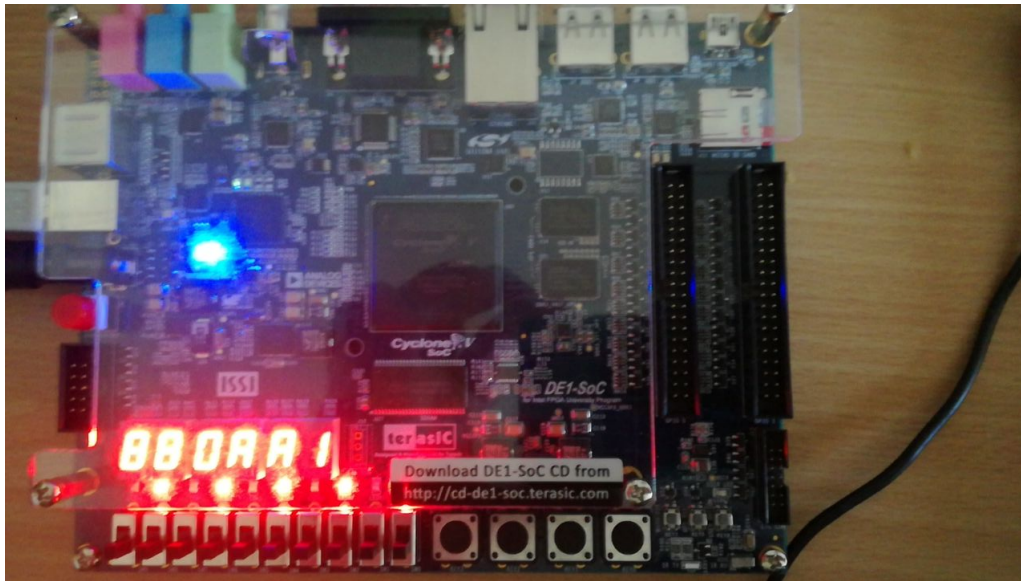


FIGURE 10 – Laboratoire 2 : Appui 1 fois sur key 2

Tout à fonctionner sans problème. J'ai testé la modification des switches lorsque j'appuie sur key 2 ou key 3 et conformément à la donnée, les leds et l'afficheur 7 segment n'était pas impacté.

5 Conclusion

Ce laboratoire nous a permis de nous familiariser avec les principes de PIO et d'interactions entre FPGA et HPS. De plus, il nous a permis de nous assurer que la machine virtuelle mise en place pour le cours SOCF pour toute la durée de la période de travail à la maison fonctionne bien.

6 Signatures

Yverdon-les-Bains le 8 mai 2020

Pierrick Muller

Table des figures

1	Laboratoire 2 : Address map	4
2	Laboratoire 2 : PIOs	5
3	Laboratoire 2 : Instantiation template	6
4	Laboratoire 2 : Numéro de l'interruption	7
5	Laboratoire 2 : Appui key 0 avec switch down	10
6	Laboratoire 2 : Appui key 1 avec switch down	11
7	Laboratoire 2 : Appui key 0 avec switch up/down	11
8	Laboratoire 2 : Appui key 1 avec switch up/down	12

9	Laboratoire 2 : Appui 2 fois sur key 3	12
10	Laboratoire 2 : Appui 1 fois sur key 2	13