

Laboratoire 2 :
Interface des I/O simples de la FPGA
Spartan 6 de la carte REPTAR via le local
bus

Département : Technologies de l'Information et de la Communication (TIC)
Unité d'enseignement : Interface (IFS)

Auteurs : Pierrick Muller
Professeur : Etienne Messerli
Assistant : Anthony Convers
Classe : IFS-A
Date : 13 octobre 2019

1 Introduction

Dans le cadre de notre cours IFS (Interface), nous étudions les interactions entre le CPU et les entrées/sorties des systèmes embarqués. Nous utilisons la carte REPTAR de la HEIG-VD composé principalement d'une FPGA Spartan 6 et d'un module DM3730 basé sur un microprocesseur Cortex-A8.

Ce laboratoire sert de lien entre les notions vues en cours (écriture et lecture des entrée/sorties et création d'interfaces permettant ces lectures et écritures par le CPU) et la mise en pratique de ces notions.

L'objectif, comme préciser dans la donnée du laboratoire, est de "concevoir une interface pour des I/Os simples, disponible sur la FPGA de la carte REPTAR et de la mettre en œuvre à l'aide d'un programme C". On nous précise d'avance dans la donnée que cela implique deux partie, l'une concernant le développement et l'implémentation matérielle d'une interface (avec la mise en place d'un plan d'adressage et la création du schéma Logisim découlant de notre interface), et l'autre impliquant le développement et l'implémentation d'un programme C permettant de tester et d'exécuter une application utilisant les entrées/sorties de notre interface.

Il reste à noter le découpage du laboratoire en deux parties, l'une appuyant sur la mise en place d'une interface simple et sur son utilisation, l'autre permettant de mettre en place des fonctionnalités plus avancées et nécessitant l'étude des timings du local bus du DM3730.

Ce rapport est basé sur le chablon fournit sur Cyberlearn et chaque section excepté l'introduction et la conclusion est séparé en deux partie, appelées "Partie 1" et "Partie 2" qui correspondent aux deux phase du laboratoire (Implémentation simple et ajout de fonctionnalités).

2 Analyse et conception

Partie 1 :

La première chose à faire dans ce laboratoire était de mettre en place un plan d'adressage permettant de faciliter le décodage des adresses dans notre interface. On nous fournissait une zone dans le plan d'adressage du DM3730 allant de l'adresse 0x1900_0000 à 0x193F_FFFF. Mon choix c'est porté sur le plan ci-dessous pour la zone donnée :

Adresse hex	Lecture(RD actif) 15 ... 0	Écriture(WR actif) 15 ... 0
0x1900_0000	[15...0] const	non utilisé
0x1900_0002	[15...8] '0' [7...0] buttons	non utilisé
0x1900_0004	non utilisé	[15...8] '0' [7...0] leds
0x1900_0006	non utilisé	[15...8] '0' [7...0] aff7seg
.	libre	libre
0x193F_FFFF		

Ce plan d'adressage permet une certaine souplesse dans l'ajout de nouvelles entrée/sorties et permet un décodage des adresses plus aisé dans l'implémentation de l'interface.

Voici ci-dessous le schéma produit de l'interface :

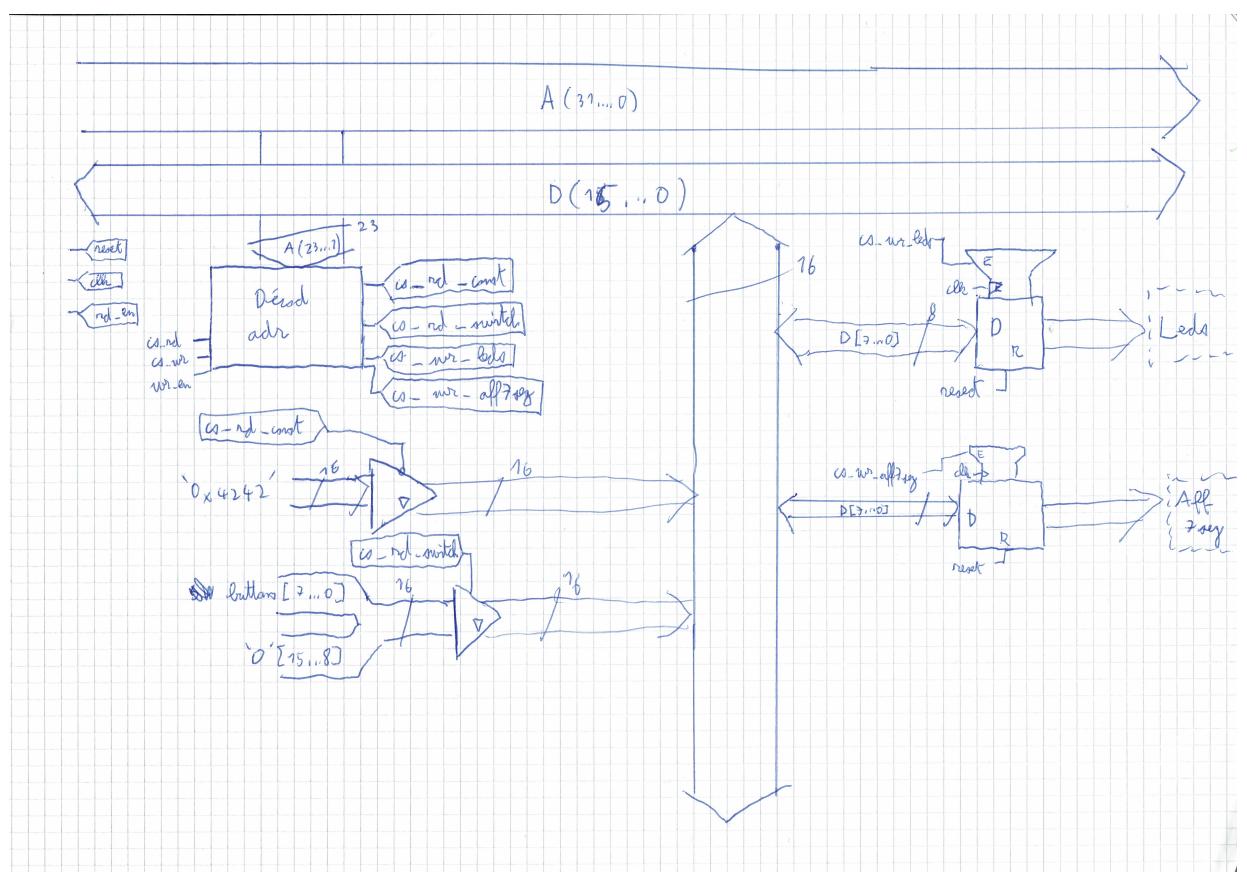


FIGURE 1 – Schéma de l'interface partie 1

Les équations qui accompagnent ce schéma sont les suivantes :

$cs_rd_const \Rightarrow cs_rd \text{ AND } (A == 0x000000)$
 $cs_rd_switch \Rightarrow cs_rd \text{ AND } (A == 0x000001)$
 $cs_wr_leds \Rightarrow cs_wr \text{ AND } wr_en \text{ AND } (A == 0x000002)$
 $cs_wr_aff7seg \Rightarrow cs_wr \text{ AND } wr_en \text{ AND } (A == 0x000003)$

L'implémentation dans logisim se basera donc sur ce schéma et ces équations.

Partie 2 :

Dans la deuxième partie, on nous demandait d'expliquer les fonctions des signaux rd_en_i et wr_en_i. Voici les réponses que j'ai trouvé en analysant le timing du local bus reptar :

- rd_en_i sert à s'assurer que les données chargées soit bien les bonnes. En effet, le chip select qui active le multiplexeur et le bus de sélection des entrées permettent de choisir quelles données lire, et rd_en_i s'active après pendant un coup de clock afin de charger ces données dans le registre de lecture. Cela répond aussi à une autre question qui nous était posé, sur l'utilité du registre et sur pourquoi le signal rd_en (rd_en_i) est connecté sur ce registre. C'est donc pour assurer la validité des données que l'on souhaite lire, et cela indépendamment des possibles aléas qui pourraient survenir si on utilisait juste le chip select. Ce registre permet aussi de garder en mémoire l'état des sorties lors de la lecture.
- wr_en_i permet de préciser que l'on souhaite effectuer une écriture sur une sortie, en complément de cs_wr_i. Cela nous permet d'être bien sûre (en plus du décodage de l'adresse) que c'est l'une des sorties de notre interface que l'on souhaite modifier.

On devait par la suite ajouter la possibilité de relire les leds et réaliser la détection du bouton SW5 de manière matérielle. Cela implique quelques modifications dans notre plan d'adressage :

Adresse hex	Lecture(RD actif) 15 ... 0	Écriture(WR actif) 15 ... 0
0x1900_0000	[15...0] const	non utilisé
0x1900_0002	[15...8] '0' [7...0] buttons	non utilisé
0x1900_0004	[15...8] '0' [7...0] leds	[15...8] '0' [7...0] leds
0x1900_0006	non utilisé	[15...8] '0' [7...0] aff7seg
0x1900_0008	[15...1] '0' [0] detecSW5	[15...1] '0' [0] acquitSW5
.	libre	libre
0x193F_FFFF		

Le point principal ci-dessus vient de l'apparition d'une nouvelle adresse utilisé, 0x1900_0008 qui permettra en lecture de savoir si SW5 a été appuyer et en écriture d'acquitter cette information (remettre le flag interne à 0).

La séquence des actions depuis le CPU-REPTAR pour gérer la détection de l'activation de SW5 est la suivante :

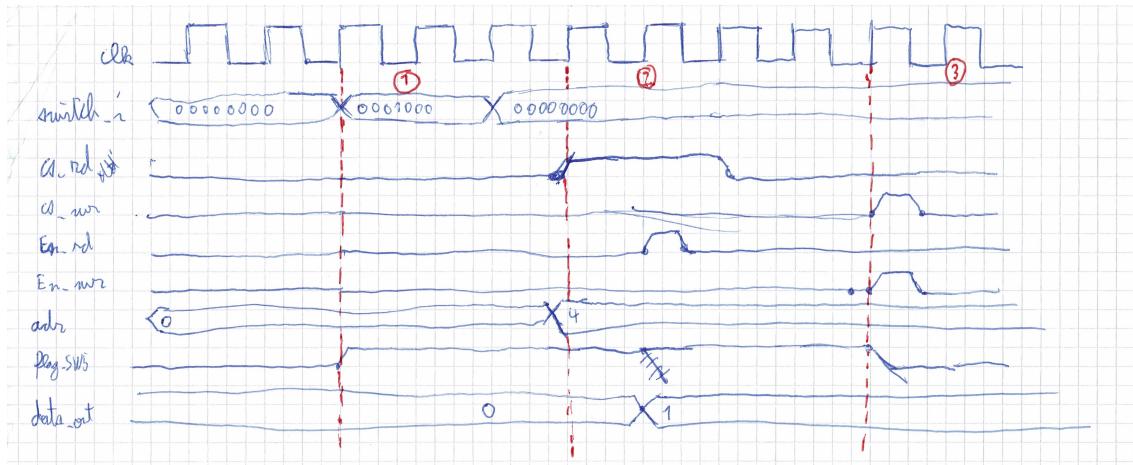


FIGURE 2 – Chronogramme des séquences du CPU

La partie 1 concerne l'activation du flag lors de l'activation de SW5. La partie 2 elle représente une lecture de ce flag, qui est comme une lecture d'une entrée normale. Enfin, la partie 3 représente l'acquittement du flag , comme on peut le voir sur le chronogramme avec le passage du flag de 1 à 0. Pour cette partie, il n'est pas important de connaître la valeur écrite dans le flag, étant donnée qu'elle n'y sera pas vraiment écrite, le write servant de commande permettant d'acquitter le flag

Voici les modifications apportées au schéma de notre interface :

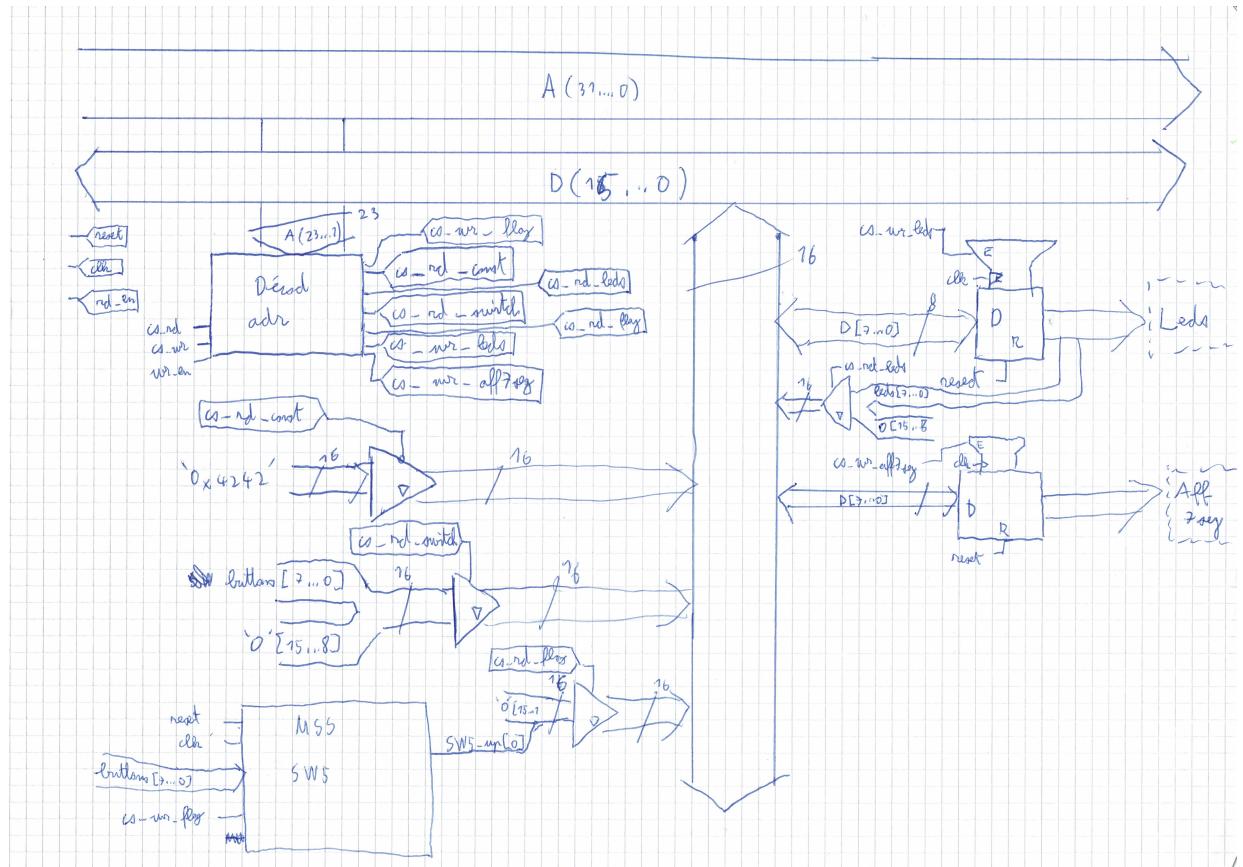


FIGURE 3 – Schéma de l'interface partie 2

Et voici la machine d'état accompagnant ce schéma :

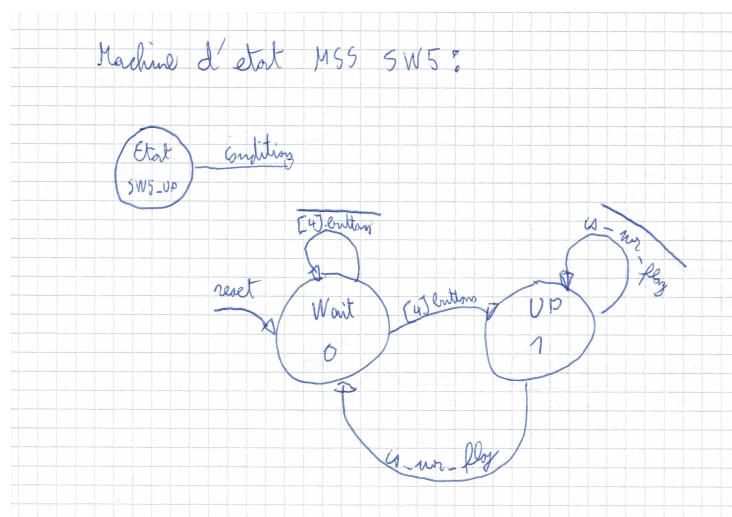


FIGURE 4 – MSS gestion SW5

De plus, trois équations ont été ajoutée à celle de base, soit
 $cs_rd_leds \Rightarrow cs_rd \text{ AND } (A == 0x0000002)$

$cs_rd_flag \Rightarrow cs_rd \text{ AND } (A == 0x0000004)$
 $cs_wr_flag \Rightarrow cs_wr \text{ AND } wr_en \text{ AND } (A == 0x0000004)$

L'implémentation se base sur ces équations et schémas.

3 Réalisation et implémentation

Partie 1 :

Voici l'interface implémenté sur logisim :

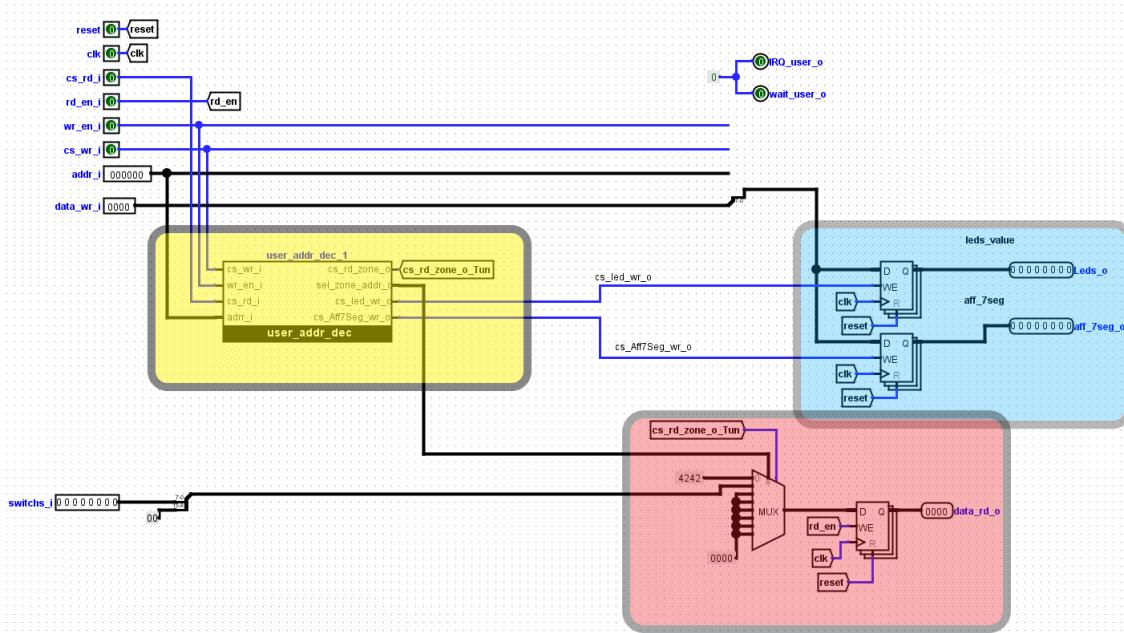


FIGURE 5 – Implémentation complète partie 1

L'interface contient un bloc appelé user_addr_dec (En jaune sur la figure ci-dessus) qui permet de décoder les adresses et de définir les actions que l'interface doit entreprendre. Je vais commencer par préciser les informations relatives à ce bloc :

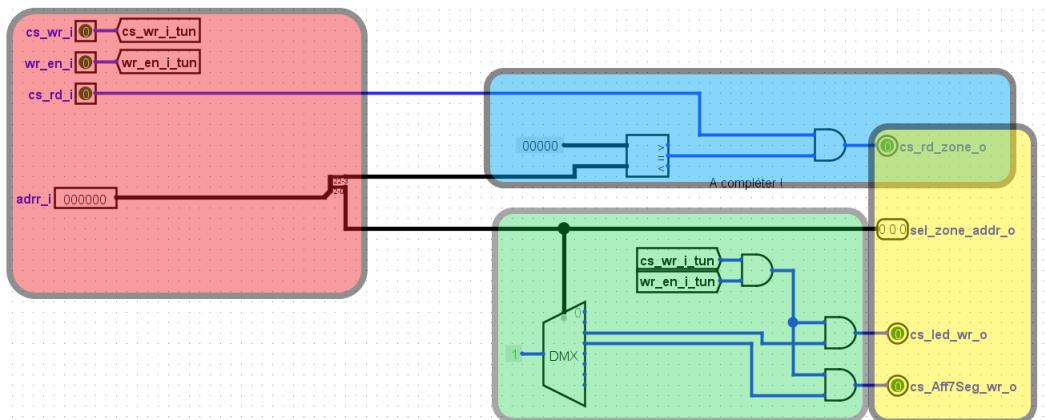


FIGURE 6 – Bloc decod

J'ai séparer le bloc en trois blocs plus petits, indiqués par les carrés de couleurs transparents ci-dessus. Le bloc rouge représente les entrées du bloc, soit :

- cs_wr_i qui représente la demande du cpu d'effectuer une écriture sur une sortie
- wr_en_i qui nous permet de nous assurer de l'intégrité des données à écrire
- cs_rd_i qui représente la demande du cpu d'effectuer une lecture sur une entrée ou une sortie
- adrr_i qui correspond à l'adresse fournit par le CPU

Le bloc jaune représente les sorties du bloc avec notamment les deux chip select pour l'écriture des leds et de l'afficheur 7 segments, les trois bits de sélection de l'entrée à lire et le chip select de lecture qui permet d'activer la lecture d'une entrée.

Le bloc bleu représente la logique permettant d'activer ou non le chip select de lecture des entrées. On regarde si on est bien dans la zone où nous avons définis nos entrées/sortie en comparant les bits de l'adresse avec la valeur d'adresse commune à nos entrée/sortie (0x00000), étant donné que les 9 bits de poids fort côté CPU n'ont pas été transféré à l'interface. En plus d'effectuer cette comparaison, on s'assure que le cpu veux bien lire quelque chose sur l'interface en contrôlant l'état de cs_rd_i. Le contrôle de la comparaison et du signal cité plus haut nous permet d'obtenir un chip select nous permettant d'être sûr qu'une des entrée/sortie veuille être lu (Dans cette première partie, juste les entrée, la relecture des leds n'étant pas encore implémenté).

Finalement, le bloc vert nous permet de gérer l'écriture des sorties. A l'aide d'un démultiplexeur et des bits de sélections de notre interface (Les trois bits de poids faibles, qui correspondent après le passage de l'adresse du CPU à l'interface aux 4 derniers bits de l'adresse divisé par deux ($0 = 0, 2 = 1, 4 = 2, 6 = 3\dots$)), on active les chip select correspondants, en s'assurant à chaque fois que le CPU veut bien écrire sur ces sorties(cs_wr_i et wr_en_i actif).

Voilà pour l'implémentation du décodeur. Les principaux problèmes rencontré dans cette implémentation venait de la compréhension des changements d'état de l'adresse entre le CPU et l'interface.

La gestion des sorties est fait de la manière suivante :

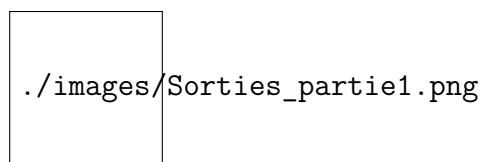


FIGURE 7 – Gestion des sorties

Conformément à notre plan d'adressage, on prend toujours les 8 bits de poids faibles pour l'écriture sur une sortie, les 8 bits de poids fort devant être set à 0. Les

chip selects sont ceux fournis par le décodeur d'adresse et les registres permettent de garder l'état des sorties.

La gestion des entrées est effectué de la manière suivante :

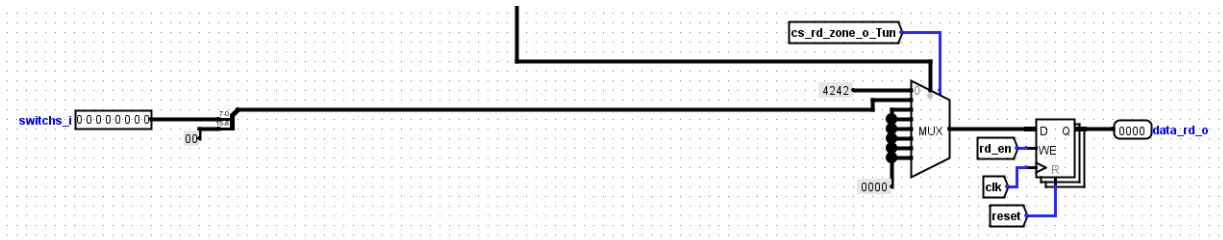


FIGURE 8 – Gestion des entrées

Lors de l'implémentation, au lieu des portes trois états, nous avons un multiplexeur qui permet de définir quel entrée nous voulons lire. Le bus de sélection du multiplexeur vient du bloc decod et nous permet donc de choisir l'entrée à lire, a condition que le chip select de rd soit actif. Derrière, la valeur de l'entrée est stockée dans un registre qui est actif grâce à rd_en, signal sur lequel nous reviendrons plus tard dans ce rapport.

Voila pour l'implémentation du schéma sur logisim. Je vais maintenant passer à l'implémentation du programme C. Je laisse ci-dessous l'implémentation finale du programme C :

```

1  ****
2  * HEIG-VD, Institut REDS
3  *
4  * File      : labo2_io_fpga.c (Adaptation du filchier ...
5  *              rehtar_fpga_std.c)
6  * Author    : Alexandre MALKI
7  * Created on : 09.03.2017
8  *
9  * Description : Programme c pour le labo2 io fpga sp6.
10 *              Realisation d'une application avec les io de ...
11 *              la fpga
12 *
13 * Ver Date      Author      Description
14 * 0.4 09.03.2017 AMX        Adaptation pour le labo2 io fpga
15 *
16 ****
17 * Modifié le : 28.03.2019
18 * Par        : Pierrick Muller
19 *
20 ****
21 */
22

```

```
23 #include <common.h>
24
25 typedef volatile unsigned short vushort;
26
27 /*Les acces a la FPGA sont sur 16 bits. Il faut utiliser un type ...
   short */
28 /*Le type volatile indique que cette donnee ne peut pas etre ...
   mise dans le cash*/
29 /*ADRESSES DU PLAN D'ADDRESSAGE*/
30 #define FPGA_SP6_BASE_ADDR      0x19000000
31 #define FPGA_SP6_CST           *(vushort *) (FPGA_SP6_BASE_ADDR)
32 #define FPGA_SP6_BUTTONS        *(vushort *) (FPGA_SP6_BASE_ADDR ...
   + 0x2)
33 #define FPGA_SP6_LEDS          *(vushort *) (FPGA_SP6_BASE_ADDR ...
   + 0x4)
34 #define FPGA_SP6_AFF7SEG       *(vushort *) (FPGA_SP6_BASE_ADDR ...
   + 0x6)
35
36 /*MASQUE*/
37 #define MASK_8BITS             0xFF
38 #define MASK_SW_3              0x4
39 #define MASK_ROUND             0x20
40 #define MASK_AFF_G             0x40
41
42 /* Adresse pour les acces de SW4 du CPU */
43 #define GPIO_DATAIN_REGISTER_BANK4  *(volatile unsigned int *) ...
   (0x49054038)
44 #define MASK_SW_4               ...
   0x40000000
45
46
47 /* Uniquement declaration des fonctions ici ! */
48 void delay(unsigned int time);
49
50 int lab2_io_fpga(void)
51 {
52     //Variables utilitaires
53     vushort constProg = FPGA_SP6_CST;
54     vushort tempAff = 0;
55     vushort tempAffMid = 0;
56     uchar stateSw3 = 0;
57
58     //Réinitialisation des sorties à 0
59     FPGA_SP6_AFF7SEG &= ~MASK_8BITS;
60     FPGA_SP6_LEDS &= ~MASK_8BITS;
61
62     printf("Uboot: Reptar io fpga sp6\n");
63     //Affichage de la constante
64     printf("Const offset 0x0000 : %d (0x%04x)\n", constProg, ...
   constProg);
65
66     /*Quitte l'application quand le SW4 du CPU est presse*/
67     while (!(GPIO_DATAIN_REGISTER_BANK4 & MASK_SW_4)) {
68
69         //Gestion de la copie de l'état des boutons SW sur les leds
70         if((FPGA_SP6_BUTTONS & MASK_8BITS) != (FPGA_SP6_LEDS & ...
```

```
        MASK_8BITS))
71     {
72         FPGA_SP6_LEDS = (FPGA_SP6_BUTTONS & MASK_8BITS);
73     }
74     else
75     {
76         FPGA_SP6_LEDS = 0x0;
77     }
78
79     //Gestion de l'affichage sur l'afficheur 7 segments
80     if((FPGA_SP6_BUTTONS & MASK_SW_3))
81     {
82         //Permet de ne considérer qu'un appui tant que le ...
83         // bouton n'a pas été relâché
84         if(stateSw3 == 0)
85         {
86             stateSw3 = 1;
87             // Gestion de la transition milieu allumé/milieu éteint
88             if((tempAff & MASK_ROUND) == MASK_ROUND)
89             {
90                 tempAffMid ^= MASK_AFF_G;
91                 tempAff = 1;
92                 FPGA_SP6_AFF7SEG = tempAffMid | tempAff;
93             }
94             else
95             {
96                 //Gestion de l'affichage du tour de l'afficheur 7 ...
97                 // segement
98                 if(tempAff == 0)
99                 {
100                     tempAff = 1;
101                 }
102                 else
103                 {
104                     tempAff = (tempAff << 1) ;
105                 }
106                 FPGA_SP6_AFF7SEG = tempAff | tempAffMid;
107             }
108         }
109         //Gestion des problèmes de bouncings
110         delay(1000000);
111     }
112     else
113     {
114         stateSw3 = 0;
115     }
116
117 }
118
119     return (0);
120 }
121
122 /* Definition des fonctions ici */
123 void delay(unsigned int time)
```

```

124  {
125      unsigned int i = 0;
126      while(i < time)
127      {
128          i++;
129      }
130  }

```

La différence entre le code de base et le code avec solution logicielle permettant d'éviter les sauts de l'afficheur 7 segments vient de l'utilisation de la fonction delay dans la solution finale qui permet d'éviter les rebonds en s'assurant que l'appui à bien été effectuer.

Voila qui clôture l'implémentation de la partie 1. Si peu de difficultés ont été rencontrée dans la partie "Implémentation Logisim", l'écriture du code à apporté plus de problèmes. La gestion de l'appui du bouton avec une variable permettant de s'assurer que le bouton n'exécute pas plusieurs fois l'action d'afficher une un segment sur l'afficheur 7 segment à demander du temps de réflexion pour comprendre d'où venait le problème. Finalement, le programme est fonctionnel et a été contrôlé en séance de laboratoire.

Partie 2 :

Voici l'interface modifiée pour ajouter les nouvelles fonctionnalités :

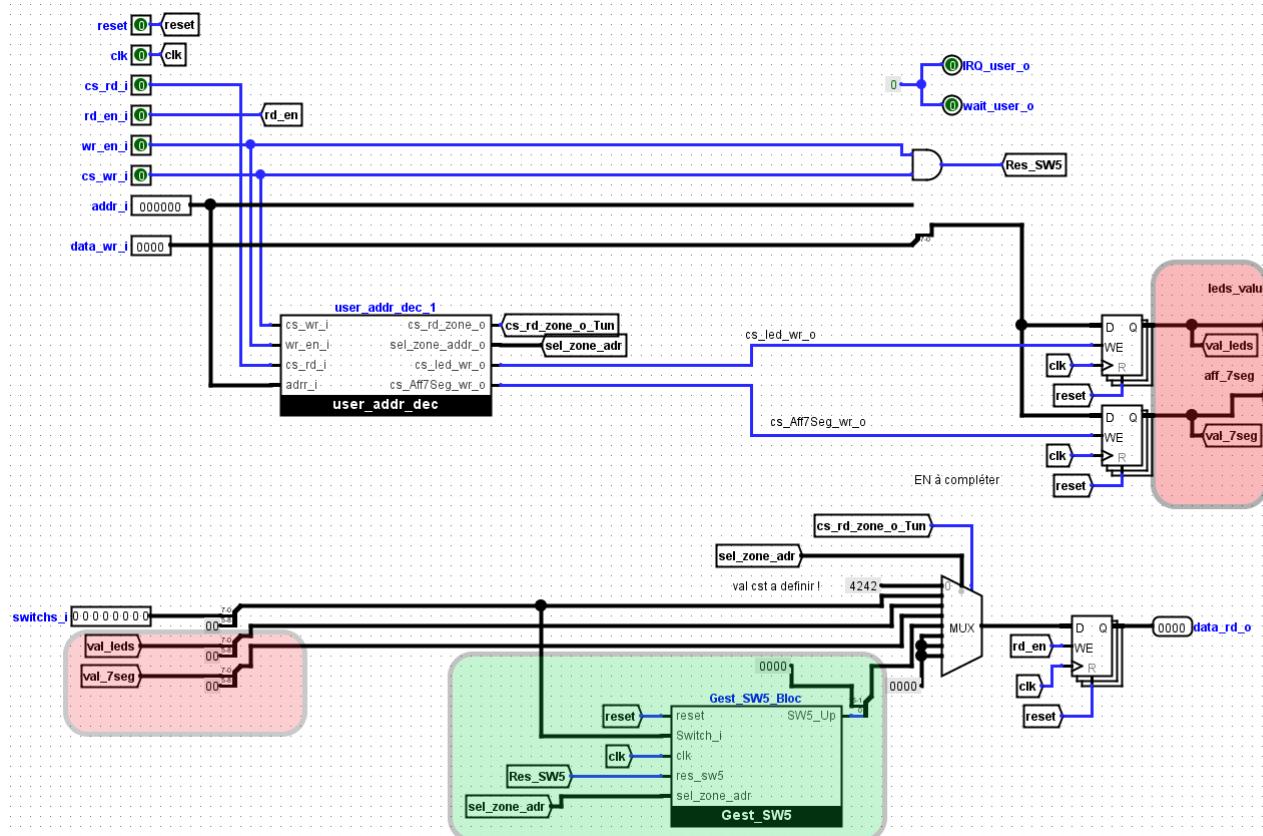


FIGURE 9 – Implémentation complète partie 2

Les deux fonctionnalités ont été ajouté, en rouge ci-dessus la relecture des leds, et en vert la gestion du bit d'état interne concernant la pression du bouton SW5. Pour la relecture des leds, nous avons juste relié l'état de la sortie LED aux entrées, conformément au plan d'adressage.

Le bloc vert (Gest_SW5_Bloc) contient le système suivant :

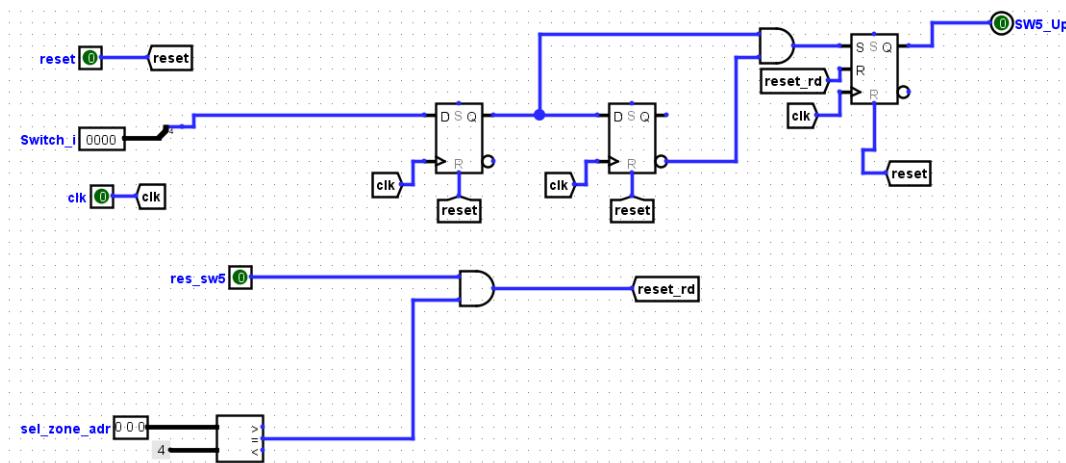


FIGURE 10 – Bloc gestion SW5

Les différentes entrées sont les suivantes :

- reset : Le signal reset de l'interface
- Switch_i : L'état des switch, d'où on va extraire le bit 4 qui correspond au SW5
- clk : L'horloge du système
- res_sw5 : Un signal correspondant à un AND entre cs_wr_i et wr_en_i qui sont des signaux de l'interface
- sel_zone_adr : Zone d'adresse dans laquelle on veut lire / écrire. On regarde si elle correspond à la valeur 4 qui correspond à l'adresse de notre flag dans le plan d'adressage (4 -> 8).

On utilise une détection de flanc montant afin d'activer le bit d'état si le bouton SW5 est pressé. On utilise pour stocker l'état de ce bit d'état un S-R Flip-Flop qui permet d'avoir un reset synchrone lorsque l'on souhaite désactiver notre bit d'état. Comme dis dans la partie d'analyse, l'écriture dans la zone acquitSW5 va permettre de reset notre bit d'état.

Voila pour l'implémentation des fonctionnalités supplémentaires de l'interface. Maintenant, pour la partie logicielle, voila ce que j'ai conçu :

```

1  /*****
2  * HEIG-VD, Institut REDS

```

```
3  *
4  * File      : labo2_io_fpga.c (Adaptation du filchier ...
5  *             : rehtar_fpga_std.c)
6  * Author    : Alexandre MALKI
7  * Created on : 09.03.2017
8  *
9  * Description : Programme c pour le labo2 io fpga sp6.
10 *                 Realisation d'une application avec les io de ...
11 *                 la fpga
12 *
13 * Ver Date      Author      Description
14 * 0.4 09.03.2017 AMX          Adaptation pour le labo2 io fpga
15 *
16 *
17 * Modifi  le : 01.04.2019
18 * Par        : Pierrick Muller
19 *
20 *
21 */
22
23 #include <common.h>
24
25 typedef volatile unsigned short vushort;
26
27 /*Les acces a la FPGA sont sur 16 bits. Il faut utiliser un type ...
28   short */
29 /*Le type volatile indique que cette donnee ne peut pas etre ...
30   mise dans le cash*/
31 /*ADDRESSES DU PLAN D'ADDRESSAGE SP6*/
32 #define FPGA_SP6_BASE_ADDR      0x19000000
33 #define FPGA_SP6_CST           *(vushort *) (FPGA_SP6_BASE_ADDR)
34 #define FPGA_SP6_BUTTONS        *(vushort *) (FPGA_SP6_BASE_ADDR ...
35   + 0x2)
36 #define FPGA_SP6_LEDS           *(vushort *) (FPGA_SP6_BASE_ADDR ...
37   + 0x4)
38 #define FPGA_SP6_AFF7SEG        *(vushort *) (FPGA_SP6_BASE_ADDR ...
39   + 0x6)
40 #define FPGA_SP6_FLAG_SW5        *(vushort *) (FPGA_SP6_BASE_ADDR ...
41   + 0x8)
42
43 /*ADDRESSES DU PLAN D'ADDRESSAGE SP3*/
44 #define FPGA_SP3_BASE_ADDR      0x1A000000
45 #define FPGA_SP3_BUTSWITCH     *(vushort *) (FPGA_SP3_BASE_ADDR + 0x0A)
46
47 /*MASQUE SP6*/
48 #define MASK_8BITS              0xFF
49 #define MASK_SW_3                0x4
50 #define MASK_ROUND               0x20
51 #define MASK_AFF_G               0x40
52
53 /*MASQUE SP3*/
54 #define MASK_SW_SP3_7            0x40
55 /* Creer les define necessaires pour votre programme ...
```

```
      */  
51  
52  
53  
54 /* Adresse pour les acces de SW4 du CPU */  
55 #define GPIO_DATAIN_REGISTER_BANK4 *(volatile unsigned int *) ...  
     (0x49054038)  
56 #define MASK_SW_4 ...  
     0x40000000  
57  
58  
59 /* Uniquement declaration des fonctions ici ! */  
60 void delay(unsigned int time);  
61  
62 int lab2_io_fpga(void)  
63 {  
64     //Variables utilitaires  
65     vushort constProg = FPGA_SP6_CST;  
66     vushort tempAff = 0;  
67     vushort tempAffMid = 0;  
68     uchar stateSw3 = 0;  
69  
70     //Réinitialisation des sorties  
71     FPGA_SP6_AFF7SEG &= ~MASK_8BITS;  
72     FPGA_SP6_LEDS &= ~MASK_8BITS;  
73  
74     //Affichage de la constante  
75     printf("Uboot: Reptar io fpga sp6\n");  
76     printf("Const offset 0x0000 : %d (0x%04x)\n", constProg, ...  
            constProg);  
77  
78 /*Quitte l'application quand le SW4 du CPU est presse*/  
79     while (!(GPIO_DATAIN_REGISTER_BANK4 & MASK_SW_4)) {  
80  
81         //Gestion de l'appui sur un bouton pour les leds  
82         if((FPGA_SP6_BUTTONS & MASK_8BITS) != 0x0)  
83         {  
84             //Allumage ou eteignage en fonction de l'état de ...  
                 SW7 de la SP3  
85             if((FPGA_SP3_BUTSWITCH & MASK_SW_SP3_7) == ...  
                 MASK_SW_SP3_7)  
86             {  
87                 FPGA_SP6_LEDS |= (FPGA_SP6_BUTTONS & MASK_8BITS);  
88             }  
89             else  
90             {  
91                 FPGA_SP6_LEDS &= ~(FPGA_SP6_BUTTONS & MASK_8BITS);  
92             }  
93         }  
94  
95         //Gestion de l'afficheur 7 segment  
96         if((FPGA_SP6_BUTTONS & MASK_SW_3) || (FPGA_SP6_FLAG_SW5 ...  
             == 1))  
97         {  
98             //On s'assure que l'appui du bouton ne compte pas ...
```

```
        plusieurs fois et on controle que l'appui ne soit ...
        pas bloquant
100    if(stateSw3 == 0 || (FPGA_SP6_FLAG_SW5 == 1))
101    {
102        stateSw3 = 1;
103
104        //Gestion du segment du milieu
105        if((tempAff & MASK_ROUND) == MASK_ROUND)
106        {
107            tempAffMid ^= MASK_AFF_G;
108            tempAff = 1;
109            FPGA_SP6_AFF7SEG = tempAffMid | tempAff;
110        }
111        else
112        {
113            //Gestion de l'avancement du tour
114            if(tempAff == 0)
115            {
116                tempAff = 1;
117            }
118            else
119            {
120                tempAff = (tempAff << 1) ;//+ 1;
121            }
122            FPGA_SP6_AFF7SEG = tempAff | tempAffMid;
123        }
124        //Permet d'éviter les sauts
125        delay(15000000);
126        //Acquittement du flag (0xFF peut être une autre ...
127        valeur, ce qui compte c'est d'écrire à cette adresse
128        FPGA_SP6_FLAG_SW5 = 0xFF;
129    }
130    //Permet d'éviter les sauts
131    delay(15000);
132
133    }
134    else
135    {
136        stateSw3 = 0;
137    }
138}
139
140    return (0);
141}
142
143 //Fonction permettant d'effectuer une attente.
144 void delay(unsigned int time)
145{
146    unsigned int i = 0;
147    while(i < time)
148    {
149        i++;
150    }
151}
```

Le programme ressemble beaucoup au programme de base, du moins dans sa deuxième partie (Gestion affichage 7 segments). les difficultés concernant ce programme venait de l'équilibrage du temps de délai nécessaire pour éviter que les sauts existent et de la gestion du fait que l'appui sur SW3 était bloquant. En effet, étant donné la gestion des appuis multiples avec la variable stateSw3, le programme bloquait l'appui sur SW5 et l'avancement de l'afficheur 7 segment. En s'assurant que l'état stateSw3 est à 0 ou que le flag SW5 est actif, on peut s'arranger pour que l'appui sur SW3 ne soit pas bloquant. C'est donc la solution qui a été choisie.

Voilà ce qui clôt l'implémentation logicielle et physique de cette partie 2. Passons maintenant aux différentes simulations qui ont été effectuées.

4 Simulation

Partie 1 :

Toutes les simulations de cette partie ont été effectuées sur le circuit suivant :

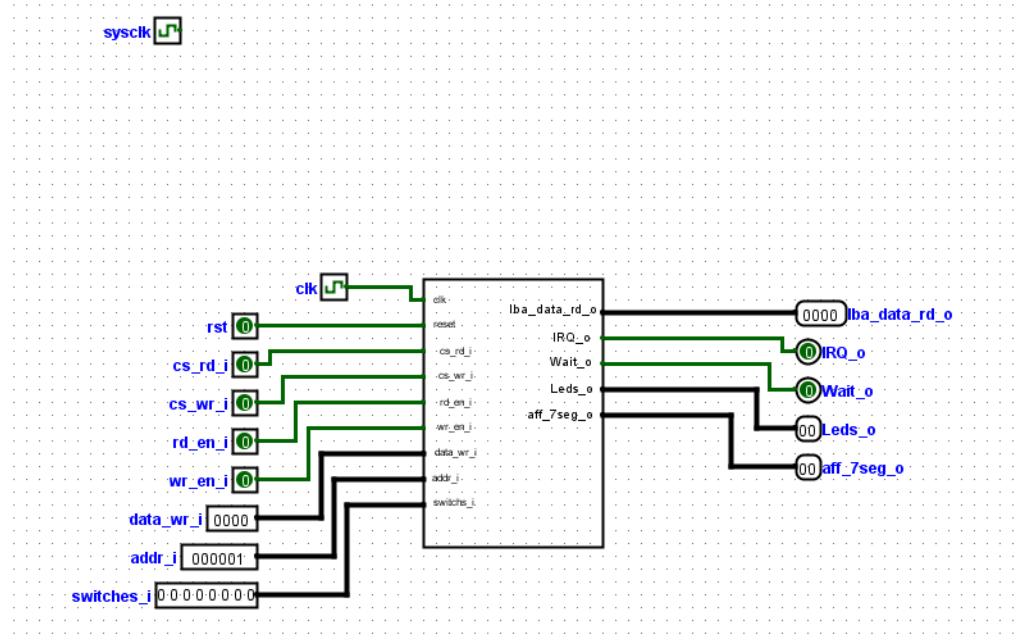


FIGURE 11 – Circuit de simulation

Ci dessous, la simulation de la lecture des switchs sur le circuit :

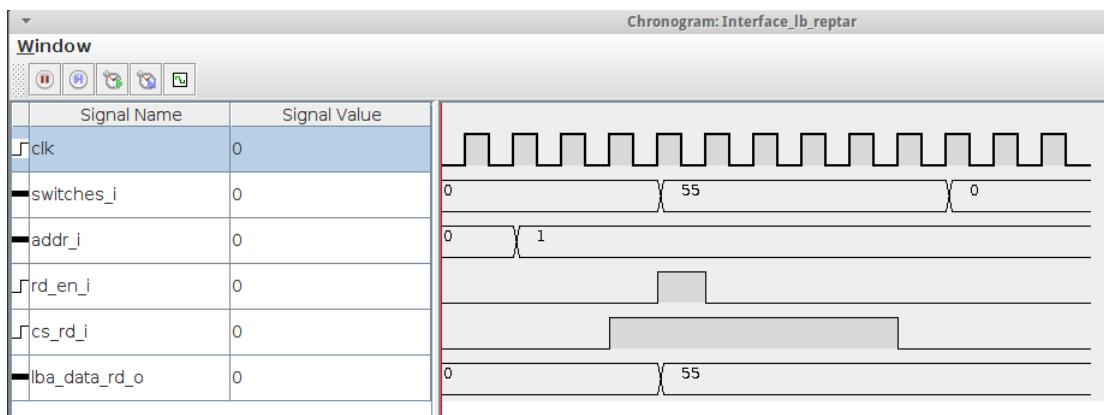


FIGURE 12 – Simulation de la lecture des switchs

En respectant les timings du bus local reptar, j'ai effectuer la simulation ci-dessus sous logisim. On peut voir que la donnée n'est chargé sur la sortie que lorsque le rd_en_i est actif, comme l'on pouvait s'y attendre en ayant étudié les bus reptar et l'interface logisim.

les deux simulations ci dessous montrent le fonctionnement des écritures sur les sorties :

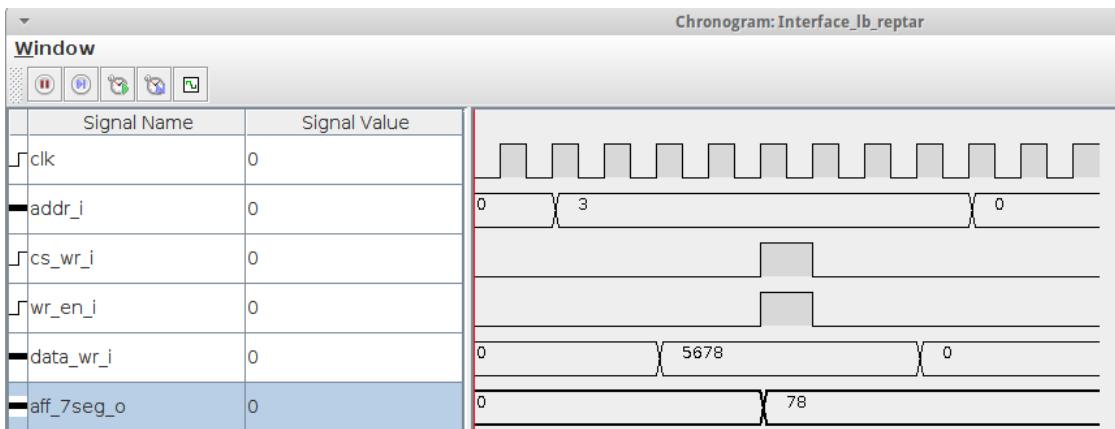


FIGURE 13 – Simulation de l'écriture sur l'afficheur 7 segments

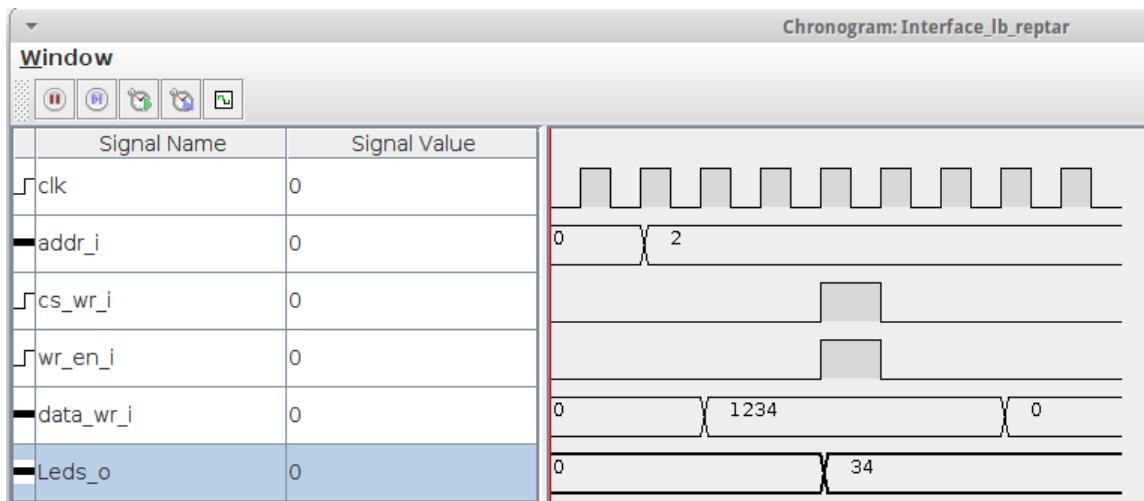


FIGURE 14 – Simulation de l'écriture sur les leds

On voit que le fonctionnement est en adéquation avec le fonctionnement du timing bus local de la carte REPTAR. Le fonctionnement est le même pour les deux , la valeur des sorties étant mise à jour des qu'une écriture se fait.

Partie 2 :

Deux simulations différentes ont été fait sur cette partie, la première concernant la relecture des leds, que voici :

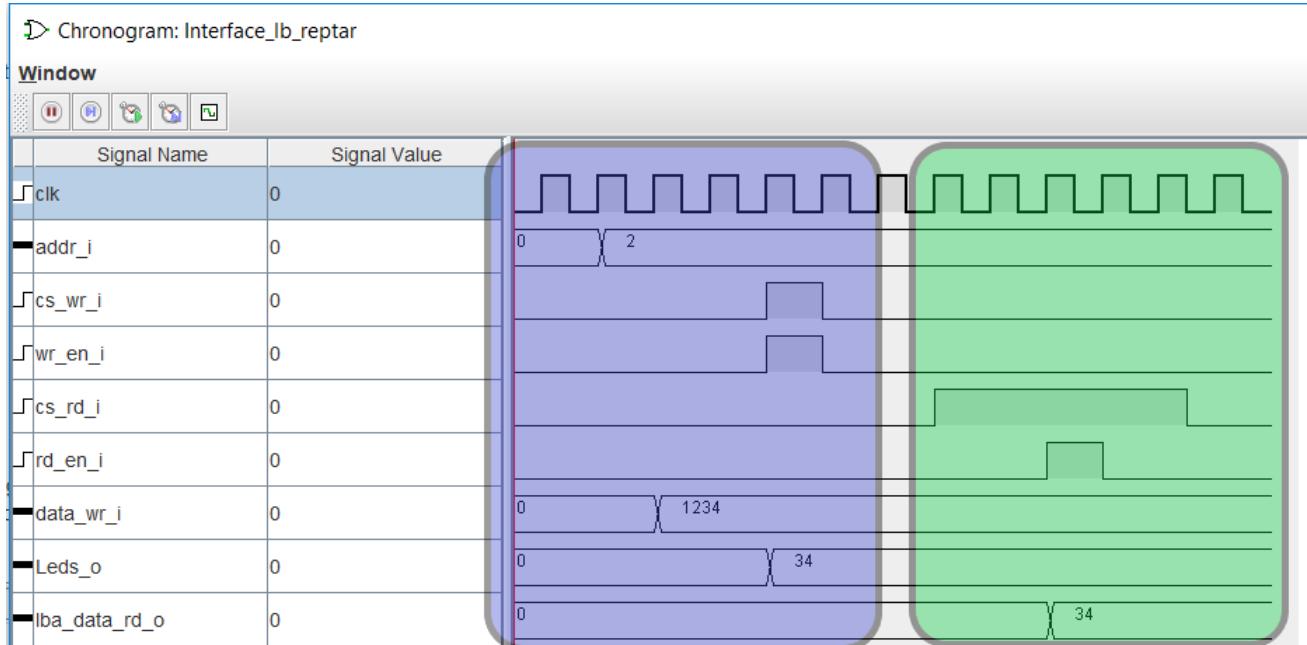


FIGURE 15 – Simulation de l'écriture et de la relecture sur les leds

Cette simulation est séparé en deux partie. La première en bleu correspond à l'écriture des leds (Comme vu plus haut) et la deuxième partie représente la lecture

de ces mêmes leds. On voit que le tout est fonctionnel en respectant les timings du bus local de la carte Reptar.

La deuxième solution représente une activation de la valeur du flag d'état actif si SW5 à été pesé et un acquittement de ce flag :

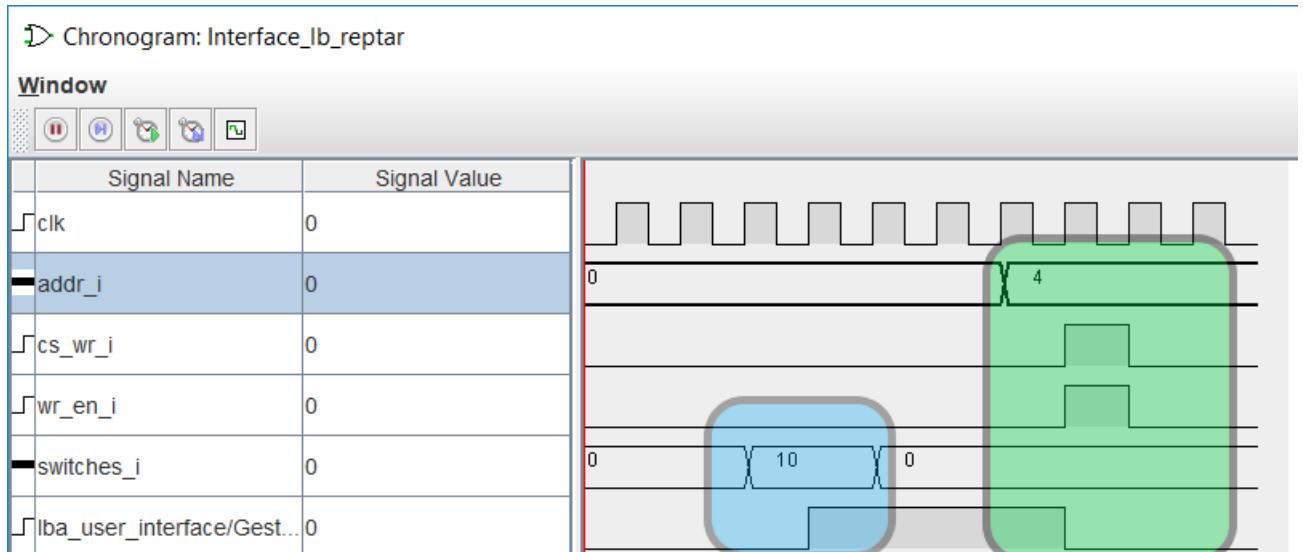


FIGURE 16 – Simulation de la gestion du flag SW5

La simulation est de nouveau séparée en deux bloc, le premier en bleu représentant l'activation du flag SW5 sur le flanc montant suivant l'activation du switch SW5 (0x10 correspondant au cinquième bit). Le second en vert montre l'acquittement de la part du CPU avec l'écriture à l'adresse du flag d'une valeur. La simulation est donc un succès.

5 Conclusion

Ce laboratoire m'a permis de mettre en pratique la création d'une interface depuis sa création sur le papier (analyse) jusqu'à son implémentation logicielle (Programme C) en passant par son implémentation matérielle (Logisim).

Il m'a permis de me familiariser avec le timing bus local de la carte Reptar, et notamment avec les signaux Rd_en et Wr_en. Le découpage en deux parties m'as permit de mieux structurer mon apprentissage et a facilité la deuxième partie.

Les notions vues en cours se retrouvent dans ce laboratoire, et les compétences théoriques acquises permettent de mener au bout le laboratoire. Les problèmes principaux viennent de la compréhension du timing bus local et de la création des programmes C.

Toutes les manœuvres devant être contrôlées l'ont été par l'ingénieur ou par le professeur.

6 Signatures

Yverdon-les-Bains le 13 octobre 2019

Pierrick Muller