HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Reconfigurable & embedded
Digital Systems

Laboratoire 03 – Conception d'une interface simple

Départements : TIC

Unité d'enseignement ARE

Auteurs : **Urs Behrmann**
Guillaume Gonin

Professeur : **Etienne Messerli**
Assistant : **Anthony Convers**

Classe : **ARE**
Salle de labo : **A07**

Date : **13.11.2024**

1 Introduction

L'objectif de ce laboratoire est de concevoir une interface simple sur le bus Avalon, connectée au système à processeur HPS (hard processor system). On devra établir un plan d'adressage pour accéder et contrôler différents périphériques, notamment les entrées/sorties de la carte DE1-SoC (LEDs, boutons, interrupteurs) et une liaison parallèle de 36 lignes directes avec la carte Max10_leds. Enfin, on écrira un programme pour piloter cette interface en respectant les spécifications demandées.

2 Analyse et conception

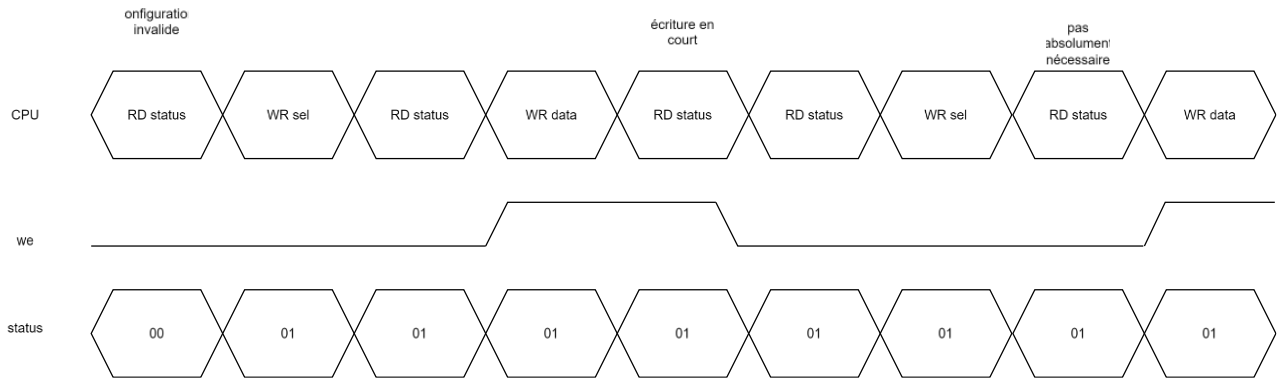
2.1 Plan d'adressage

Dans le plan d'adressage, la taille de la zone disponible pour notre interface correspond-elle aux 14 bits d'adresse définis dans le bus Avalon ? Pourquoi ?

Oui, la zone d'adressage de 14 bits du bus Avalon est suffisante pour notre interface, car chaque adresse du bus Avalon représente 4 octets (32 bits). Cela signifie que 14 bits d'adresse permettent de couvrir $2^{14}\times4=2^{16}$ octets, soit l'équivalent de 16 bits d'adressage côté CPU. Ainsi, les 14 bits d'adresse du côté FPGA couvrent bien la zone de 16 bits d'adresse disponible côté CPU pour notre interface.

Offset on bus AXI lightweight HPS-to-FPGA (relative to BA_LW_AXI)	**Lecture (Rd='1')**	**Écriture (Wr='1')**
0x00_0000 – 0x00_0003	Constante design ID 32 bits	Réservés
0x00_0004 – 0x00_FFFF	Réservés	Réservés
0x01_0000 – 0x01_0003	Constante interface ID 32 bits	Réservés
0x01_0004 – 0x01_0007	Réservés	leds (9..0), réservés (31..10)
0x01_0008 – 0x01_000B	Switches (9..0), réservés (31..10)	Réservés
0x01_000C – 0x01_000F	Keys (3..0), réservés (31..4)	Réservés
0x01_0010 – 0x01_0013	lp36_status (0), write_enable (1), réservés (31..2)	Réservés
0x01_0014 – 0x01_0017	Réservés	lp36_sel (3..0), réservés (31..4)
0x01_0018 – 0x01_001B	lp36_data (31..0)	lp36_data (31..0)
0x01_001C – 0x01_FFFF	Réservés	Réservés

2.2 Chronogramme



Il faut lire le registre de statut dans lequel les informations pour le write enable (WE) et le statut (status) sont accessibles.

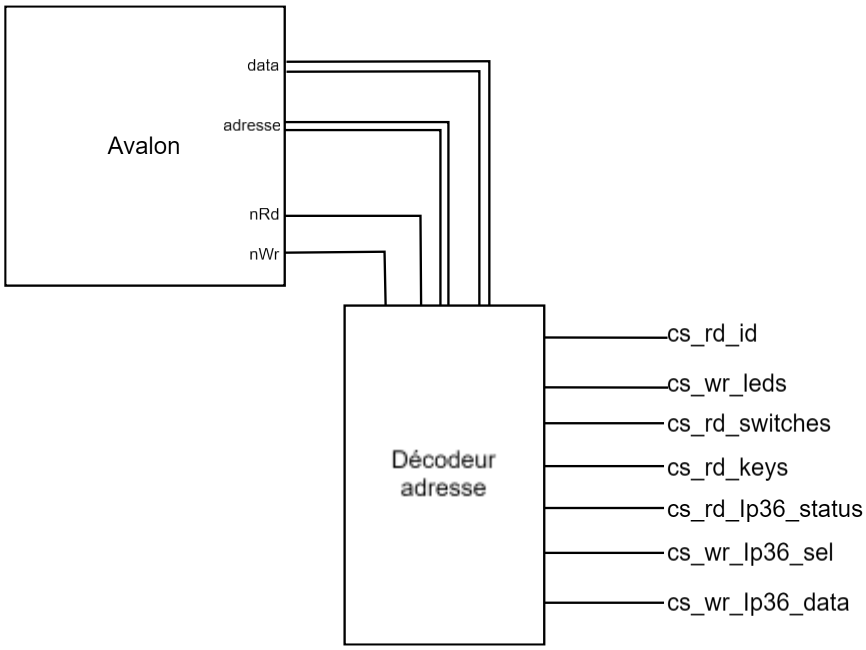
Si le statut n'est pas « 01 », il faut modifier la sélection pour corriger la configuration.

Lors de l'envoi des données, la séquence d'écriture commence et dure 1 µs. Pendant ce temps, le write enable est activé, ce qui empêche l'envoi de nouvelles données et la modification de la sélection. Cela permet de garantir que l'écriture se déroule correctement.

Une fois l'écriture terminée, il est possible d'envoyer une nouvelle sélection et un nouveau jeu de données. Il n'est pas nécessaire de vérifier le statut après l'écriture de la sélection, car celle-ci sera vérifiée uniquement avant l'envoi des données.

2.3 Schéma bloc de l'interface Avalon

2.3.1 Décodeur d'adresse

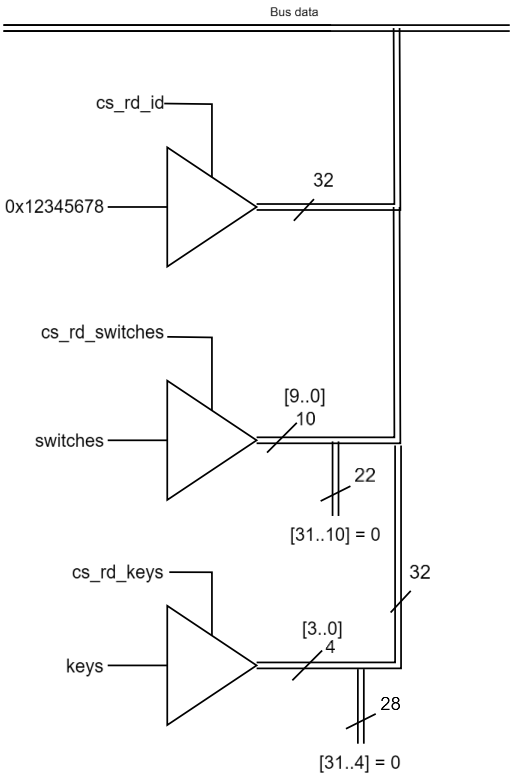


Equations décodeur d'adresse

cs_rd_id => addr = 0x01_0000 * rd
cs_wr_leds => addr = 0x01_0004 * wr
cs_rd_switches => addr = 0x01_0008 * rd
cs_rd_keys => addr = 0x01_000C * rd
cs_rd_lp36_status => addr = 0x01_0010 * rd
cs_rd_lp36_data => addr = 0x01_0018 * rd
cs_wr_lp36_sel => addr = 0x01_0014 * wr
cs_wr_lp36_data => addr = 0x01_0018 * wr

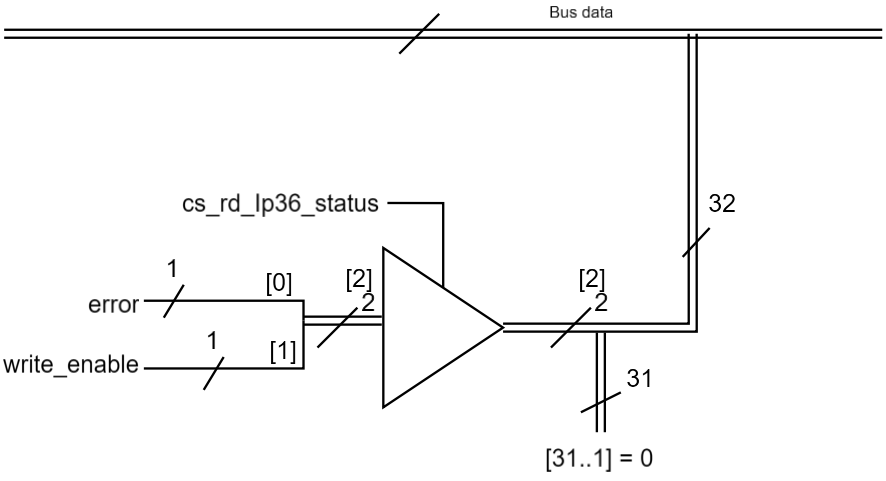
2.3.2 Read ID, switches et keys

Pour les lectures, les données sont directement envoyées sur le bus Avalon. Les signaux de contrôle sont activés pour chaque opération de lecture. Les signaux de données sont spécifiquement activés pour les lectures de l'ID du design, des *switches* et des *keys*.

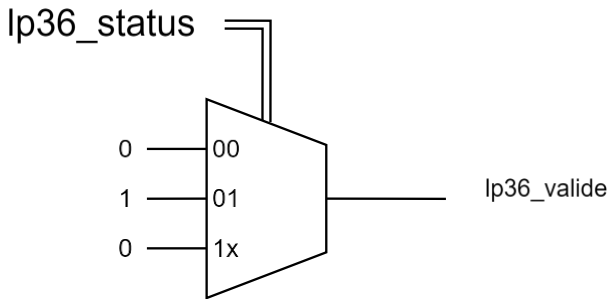


2.3.3 Read status lp36

Pour la lecture de l'erreur du lp36, le même procédé est utilisé que pour les lectures précédentes. Cependant, l'erreur est interprétée en fonction de la valeur du signal *lp36_status*. En plus de l'erreur, le signal *write_enable* est également retourné pour indiquer si une écriture est en cours sur le lp36. Ce signal *write_enable* est activé par la MSS.

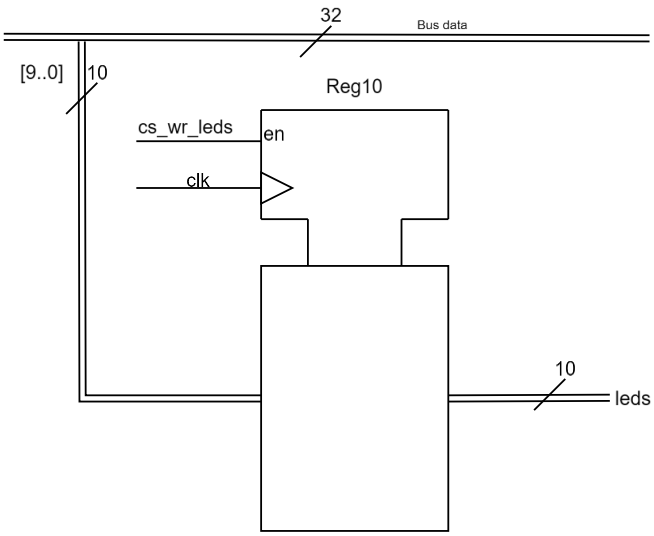


Le statut du *lp36* est codé sur 2 bits, ce qui permet d'obtenir 4 valeurs différentes. Cependant, seules les deux premières valeurs sont utilisées pour indiquer s'il y a une erreur ou non. Comme les deux autres valeurs sont "réservées", aucune erreur n'est retournée si le signal est à 2 ou 3.



2.3.4 **Write leds**

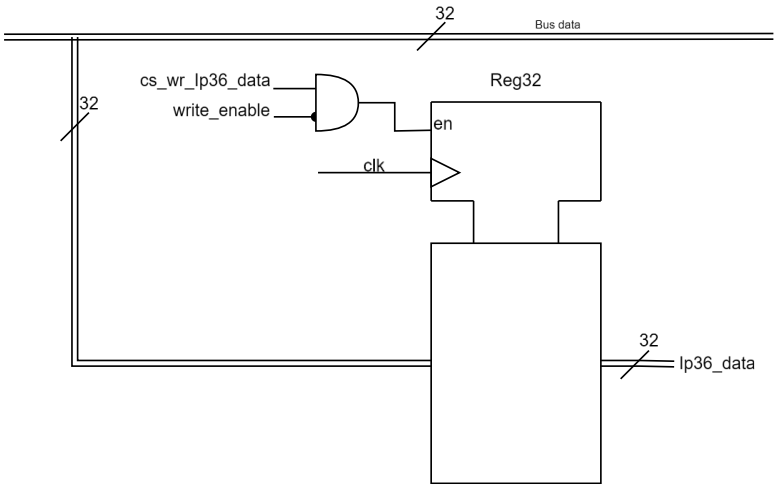
Pour l'écriture des LEDs, le signal de contrôle *wr_leds* est activé, et les données sont enregistrées dans un registre qui est ensuite utilisé pour allumer les LEDs.



2.3.5 Write lp36 data

Pour l'écriture des données sur le *lp36*, le signal de contrôle *wr_lp36_data* est activé, et les données sont enregistrées dans un registre qui est ensuite utilisé pour envoyer les informations au *lp36*.

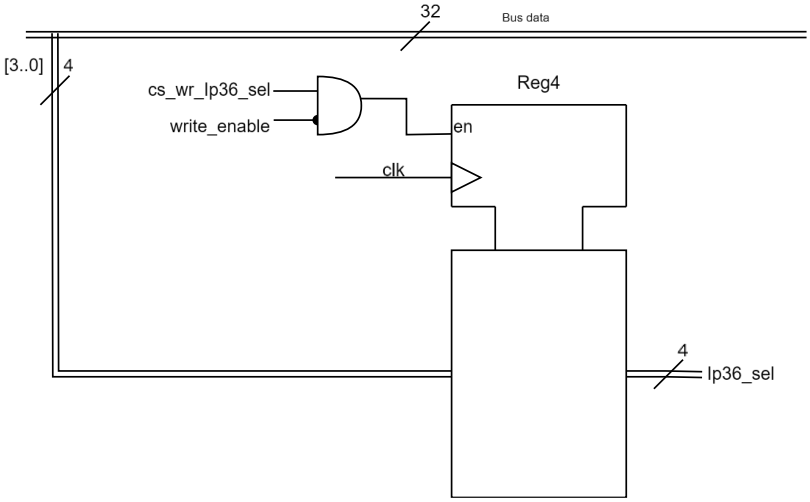
Pendant l'écriture sur le *Max10*, l'écriture de nouvelles données depuis le CPU est bloquée afin d'éviter toute erreur.



2.3.6 Write lp36 sel

Pour l'écriture du sélecteur du *lp36*, le signal de contrôle *wr_lp36_sel* est activé, et les données sont enregistrées dans un registre qui est ensuite utilisé pour transmettre le sélecteur au *lp36*.

Pendant l'écriture sur le *Max10*, l'écriture du sélecteur depuis le CPU est bloquée afin d'éviter toute erreur.



2.3.7 Liaison avec la Max10

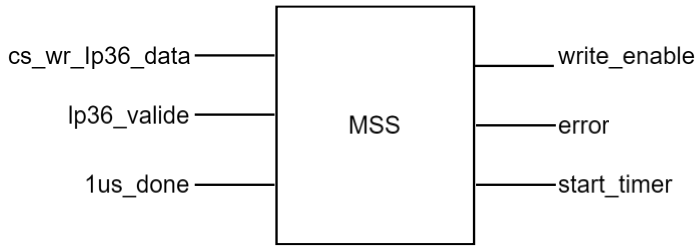
Pour la liaison avec le *Max10*, il est nécessaire de synchroniser les données afin de garantir une écriture valide. Pour cela, un MSS est utilisé pour envoyer les données et informer le CPU de l'état de l'écriture.

Le CPU doit surveiller deux informations : l'état de l'écriture (*write_enable*) et l'état du *lp36* (*error*). Si le CPU souhaite écrire, il doit d'abord vérifier que le signal *write_enable* est à 0. Si c'est le cas, il peut alors écrire la sélection des LEDs pilotées et les données dans le registre de données. Une fois les données déposées, le CPU doit relire ce même registre pour vérifier qu'aucune erreur ne s'est produite. En cas d'erreur, cela signifie que la sélection des LEDs pilotées n'est pas valide.

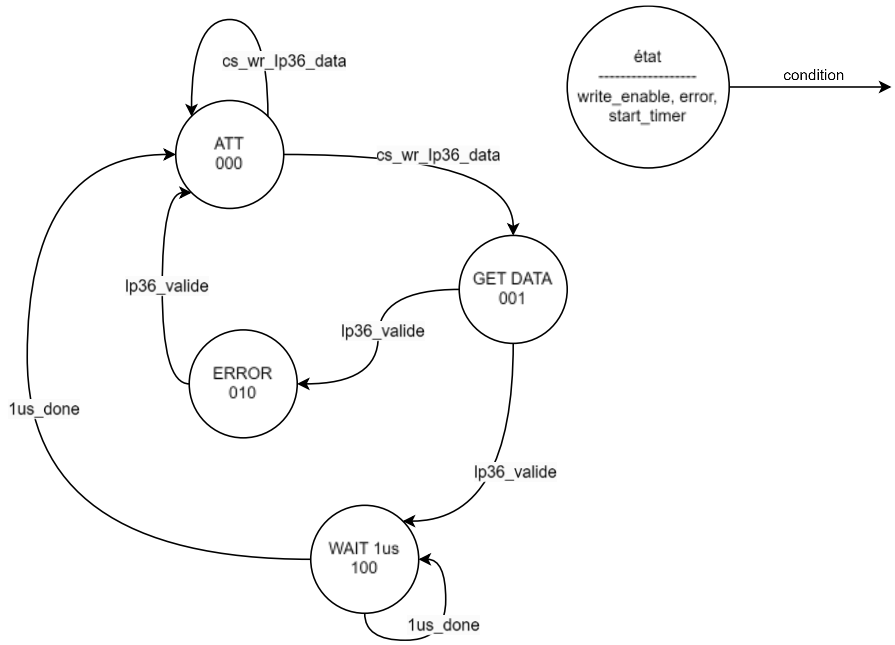
Nous avons décidé d'utiliser un retour d'information (*feedback*) du signal *write_enable* pour indiquer au CPU que l'écriture a été effectuée, même si ce signal ne sera actif que pendant un cycle d'écriture (1 µs) pour éviter des problèmes de synchronisation temporelle. Cela ne devrait pas poser de problème, car la mise à jour des valeurs pour le *Max10* dépend des entrées de l'utilisateur via les boutons et les *switches*.

2.3.8 MSS pour la liaison Max10

Nous avons décidé d'utiliser une machine séquentielle synchrone pour la liaison avec le *Max10*, principalement parce que le signal d'écriture doit être actif pendant un seul cycle d'écriture (1 µs).



Pour cela, nous avons décidé d'utiliser une machine à états synchronisée (*MSS*) avec 4 états : **ATT**, **GET_DATA**, **WAIT_1US** et **ERROR**.



2.3.9 **1us**

Pour obtenir un cycle d'écriture de 1 µs, il est nécessaire de déterminer combien de cycles de l'horloge du FPGA correspondent à cette durée. La fréquence de l'horloge du bus Avalon étant de 50 MHz, chaque cycle d'horloge correspond donc à 20 ns. Ainsi, pour atteindre 1 µs, il faut attendre 50 cycles d'horloge.

On va utiliser un compteur qu'on a créé en CSN le semestre passé pour compter le bon nombre de cycle.

2.3.10 **Code C**

Dans le code C, nous avons choisi d'écrire toujours 0 pour les bits qui ne nous concernent pas (par exemple, lors de l'écriture pour les 10 LEDs : valeur écrite = 0x000003FF). Toutefois, la partie FPGA s'assure ensuite de ne modifier que les bits nécessaires (les 10 derniers dans notre exemple). L'utilisation de macros en C garantit une meilleure lisibilité du code et simplifie grandement les futures adaptations (par exemple, en cas de modification du plan d'adressage).

Comme mentionné précédemment dans ce rapport, pour pouvoir écrire dans lp36_sel, il faut que lp36_wr soit à zéro et que le statut du *Max10* soit valide (c'est-à-dire lp36_status == 0b01). Cela est implémenté dans le code à l'aide d'une boucle qui tente l'écriture dans lp36_sel jusqu'à ce que ces deux conditions soient remplies. L'utilisation d'une boucle peut sembler risquée, mais comme wr n'est maintenu actif que pendant 1 µs, la boucle ne devrait jamais durer trop longtemps et ne devrait donc pas compromettre le fonctionnement du système.

3 Simulation / Tests

3.1 Test de la partie C

Voici le banc de test que nous avons utilisé :

No	Fonctionnalité/s testée/s	Scénario	Résultat attendu
1	Réplication des switcht 0-8 sur les leds 0-8	Éteindre tous les switchs puis allumer l'un après l'autre ou tous à la fois.	Les leds s'allument ou s'éteignent en fonctions des switchs correspondant.
2	Sélection des leds avec les switchs 9-10	Tester toutes les combinaisons de SW9-10 avec un autre switch d'allumé	Les leds s'allument au bon endroit (carré, ligne, demi-cercle1, demi-cercle2).
3	Choix du comportement des leds avec les keys 0-1	Appuyer sur key0, puis key1, enfin key0 et 1 simultanément	Les leds s'allument avec 0101.. puis avec 1010.. puis 1111.. et au relâchement réplique les switchs.
4	Reset des leds avec key 3	Appuyer sur key 3	Les leds s'éteignent (sauf celles sélectionnées car réplique les switchs 8-0).
5	Décalage avec key 2	Appuyer sur key 2 et modifier les switchs, le refaire encore 4 fois.	Les leds bouge d'une ligne a la prochaine vers le bas. Dès qu'il atteigne le bas, il remonte sur la première ligne.

Résultat des test C :

No	Statut
1	OK
2	OK
3	OK
4	OK
5	OK

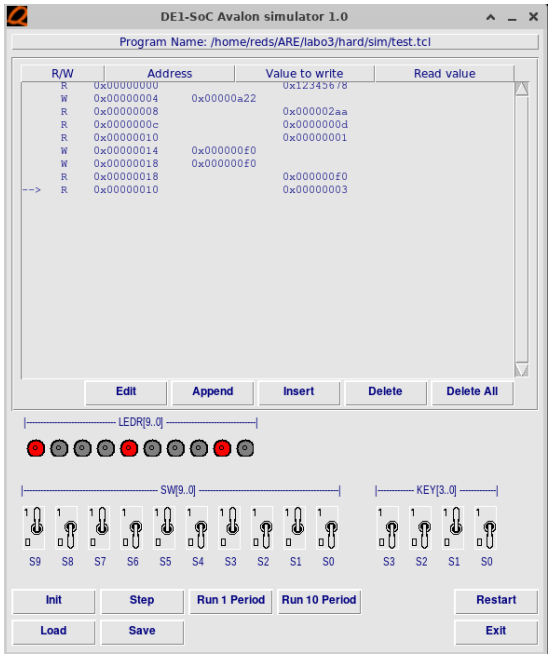
3.2 Test du VHDL avec la console TCL-TK

Pour tester l'interface, on a utilisé la console TCL-TK pour simuler les entrées du CPU. On a créé une série de commandes dans la console pour simuler les lectures et écritures sur l'interface.

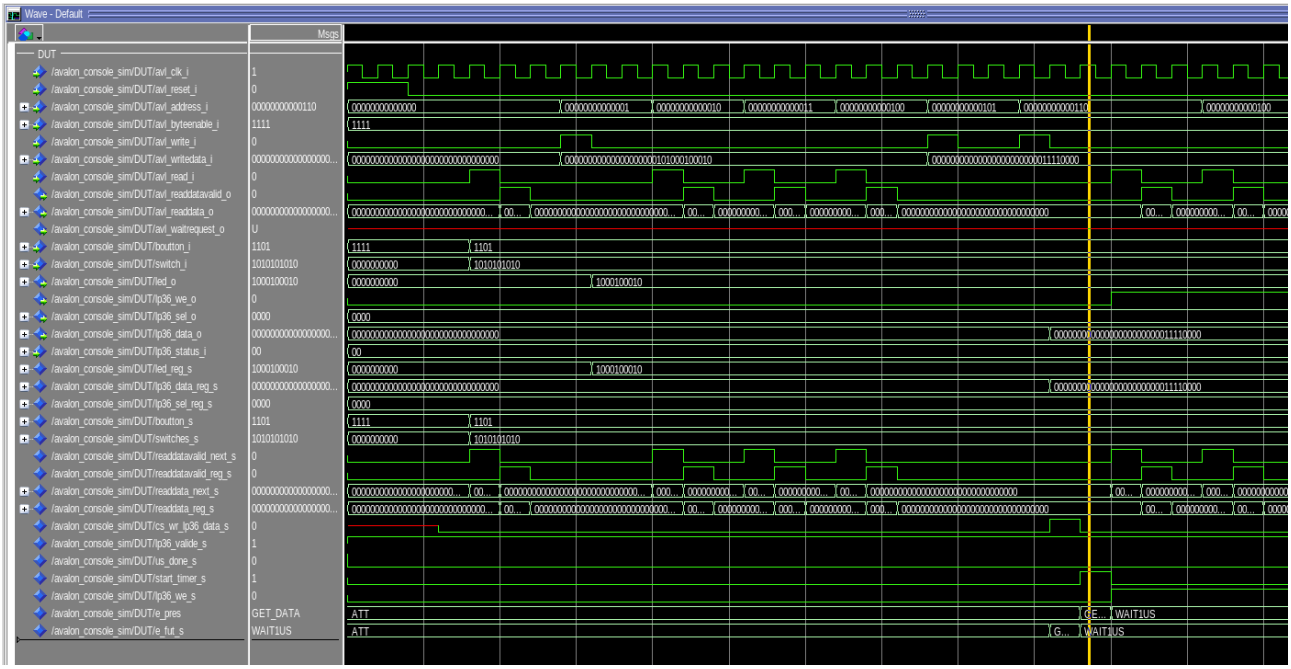
Commande	Description	Résultat attendu
R 0x0000	Lecture de l'ID de l'interface	0x12345678
W 0x0004	Ecriture des leds	0x2AA
R 0x0008	Lecture des switches	0x2AA
R 0x000C	Lecture des keys	0x2
R 0x0010	Lecture du status du lp36	0x1
W 0x0014	Ecriture du lp36_sel	0x1
W 0x0018	Ecriture du lp36_data	0xf0
R 0x0010	Lecture du status du lp36	0x10

3.2.1 Screenshots de la simulation

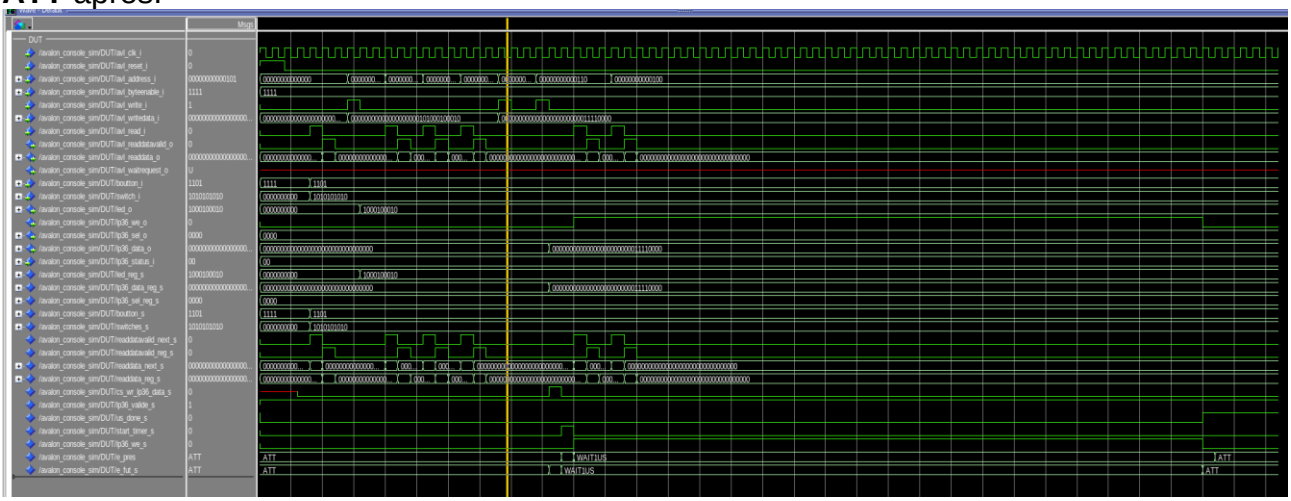
Ici les commandes qui ont été listé dans le tableau plus haut. Le seul problème qu'il y avait était que les leds ne s'allumait pas correctement. On s'attendait que 0xa22 fasse qu'une led sur deux s'allument, mais on peut voir sur le screenshot que seulement 3 leds se sont allumée. Sinon la simulation a fonctionné correctement.



Ici un screenshot du chronogramme, on peut bien voir le write enable qui s'active lors qu'on a envoyé les datas.



Le write enable reste activé pendant les 50 coups de clock et la mss repasse en mode **ATT** après.



4 Conclusion

Dans ce laboratoire, nous avons créé une interface entre le CPU et le *Max10*. Le bus Avalon a été utilisé pour communiquer entre les deux. Nous avons développé un décodeur d'adresse pour gérer les différentes opérations de lecture et d'écriture, ainsi qu'une machine à états synchrone (*MSS*) pour la liaison avec le *Max10*. La console TCL-TK a été utilisée pour simuler les entrées du CPU et tester l'interface. Enfin, nous avons démontré à Anthony Convers que notre interface fonctionnait correctement le 12.11.2024.

Nous pensons que la description VHDL et le code C auraient pu être encore simplifiés si nous avions disposé de plus de temps.

Date : 13.11.2024

Noms des étudiants : Urs Behrmann Guillaume Gonin

5 Annexes ….

5.1 Hps_applications.c

```
/*****************************************************************************************
 * HEIG-VD
 * Haute Ecole d'Ingenerie et de Gestion du Canton de Vaud
 * School of Business and Engineering in Canton de Vaud
 *****************************************************************************************
 * REDS Institute
 * Reconfigurable Embedded Digital Systems
 *****************************************************************************************
 *
 * File               : hps_application.c
  * Author             : Guillaume Gonin, Urs Behrmann
 * Date               : 5.11.2024
 *
 * Context            : ARE lab
 *
 *****************************************************************************************
 * Brief: Conception d'une interface simple sur le bus Avalon avec la carte DE1-SoC
 *
 *****************************************************************************************
 * Modifications :
 * Ver    Date       Student      Comments
 * 1      5.11.2024   GoninG       Starter routine done (not tested)
 * 2      5.11.2024   GoninG       Loop routine done (not tested)
 * 3      8.11.2024   GoninG       Updated the way we use lp36_wr and read lp36_status
(not tested)
 * 4      8.11.2024   GoninG       Removed some bugs after test
 * 5      9.11.2024   GoninG       Code refactored with macro
 * 6      11.11.2024  GoninG       Debuging and testing, Bugs fixed
 * 7      12.11.2024  GoninG       Shifting working the right way (hopefully)
 * 8      12.11.2024  BehrmannU    Refactoring after validation
*****************************************************************************************/
#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include "axi_lw.h"

int __auto_semihosting;

// Base address
#define INTERFACE_BASE_ADD          ((AXI_LW_HPS_FPGA_BASE_ADD) +
0x010000)
```

```
// ACCESS MACROS
#define INTERFACE_REG(_x_)          (*(volatile uint32_t *)(INTERFACE_BASE_ADD + _x_)) // _x_ is an offset with respect to the base address

// Address Plan
#define CONST_AXI_LW_OFF            0x0
#define CONST_AXI_LW_MASK           0xFFFFFFFF //32 bits

#define CONST_INT_OFF               0x0
#define CONST_INT_MASK              0xFFFFFFFF //32 bits
#define LEDS_OFF                    0x4
#define LEDS_MASK                   0x3FF //10 bits
#define SWITCHS_OFF                 0x8
#define SWITCHS_MASK                0x3FF //10 bits
#define KEYS_OFF                    0xC
#define KEYS_MASK                   0xF //4 bits
#define LP36_STATUS_OFF             0x10
#define LP36_STATUS_MASK            0x1 //2 bits
#define LP36_WR_OFF                 0x10
#define LP36_WR_MASK                0x2 //1 bits
#define LP36_SEL_OFF                0x14
#define LP36_SEL_MASK               0x3 //2 bits, technically 4 bits but only 2 used
#define LP36_DATA_OFF               0x18
#define LP36_DATA_SEC1_MASK         0x3FFFFFFF //30 bits
#define LP36_DATA_SEC2_MASK         0x3FFFFFFF //30 bits
#define LP36_DATA_LINE_MASK         0xFFFFFFFF //32 bits
#define LP36_DATA_SQUA_MASK         0x1FFFFFF //25 bits

#define LEDS_PATTERN_A              0b10101010101010101010101010101010
#define LEDS_PATTERN_B              0b01010101010101010101010101010101
#define LEDS_PATTERN_C              0b11111111111111111111111111111111

// READ / WRITE Macros
#define READ_CONST_AXI_LW()         (AXI_LW_REG(CONST_AXI_LW_OFF) & CONST_AXI_LW_MASK) //using mask isn't useful because our interface do it aswell but it stay a good habit
#define READ_CONST_INT()            (INTERFACE_REG(CONST_INT_OFF) & CONST_INT_MASK)
#define READ_KEYS()                 (~(INTERFACE_REG(KEYS_OFF) & KEYS_MASK)) //inverse keys value because active low
#define READ_SWITCHS()              (INTERFACE_REG(SWITCHS_OFF) & SWITCHS_MASK)
#define READ_LP36_STATUS()          (INTERFACE_REG(LP36_STATUS_OFF) & LP36_STATUS_MASK)
#define READ_LP36_WR()              ((INTERFACE_REG(LP36_WR_OFF) & LP36_WR_MASK) >> ((int)(LP36_WR_MASK/2))) // SHR to get the bit on bit 0
#define READ_LP36_DATA()            (INTERFACE_REG(LP36_DATA_OFF) & 0xFFFFFFFF)
```

```
#define WRITE_LEDS(_x_)             (INTERFACE_REG(LEDS_OFF) = ((_x_) &
LEDS_MASK)) // _x_is an 32 bits value
#define WRITE_LP36_SEL(_x_)         (INTERFACE_REG(LP36_SEL_OFF) = ((_x_) &
LP36_SEL_MASK))
#define WRITE_LP36_DATA(_x_, _MASK_)   (INTERFACE_REG(LP36_DATA_OFF) =
((_x_) & _MASK_)) // _MASK_ is the mask to apply, depends on lp36_sel

// Local use
#define VALID_CONFIG_STATUS          0x1
#define SW7_0_MASK                  0xFF
#define KEY1_0_MASK                 0x3
#define KEY2_MASK                   0x4
#define KEY3_MASK                   0x8
#define SQUARE_LINE_SIZE            5
#define SQUARE_FIRST_LINE_MASK                                  0x1F
#define SQUARE_SECOND_LINE_MASK                                 0xE0

#define CANT_WRITE_SEL              ((READ_LP36_WR() == 1) ||
(READ_LP36_STATUS() == 0)) //if lp36_wr == 1 or FPGA thrown an error we can't write

enum LP36Select {
    SECONDARY_1 = 0,
    SECONDARY_2 = 1,
    TWO_LINE = 2,
    SQUARE = 3
};

void all_max10_leds_off(void) {
    int select_vals[] = {
        SECONDARY_1,
        SECONDARY_2,
        TWO_LINE,
        SQUARE
        };

    int mask_vals[] = {
        LP36_DATA_SEC1_MASK,
        LP36_DATA_SEC2_MASK,
        LP36_DATA_LINE_MASK,
        LP36_DATA_SQUA_MASK
        };

    for (int i = 0; i < 4; ++i) {

                        while (CANT_WRITE_SEL);

                    WRITE_LP36_SEL(select_vals[i]);

        WRITE_LP36_DATA(CONST_INT_OFF, mask_vals[i]);
```

```
    }
}

int main(void){

    printf("Laboratoire: Conception d'une interface simple \n");

    /* Au démarrage, le programme doit remplir les conditions suivantes :
       1. Vérifier que le statut de la carte Max10_leds est une configuration valide. Sinon
          afficher un message d'erreur dans la console ARM-DS et quitter le programme.
       2. Les 10 leds DE1-SoC sont éteintes.
       3. Toutes les leds de la carte Max10_leds sont éteintes (leds secondes, 2 lignes
          de leds, carré de leds).
       4. Afficher la constante ID du bus AXI lightweight HPS-to-FPGA au format
          hexadécimal dans la console de ARM-DS.
       5. Afficher la constante ID de votre interface sur le bus Avalon au format
          hexadécimal dans la console de ARM-DS. */

    int config_status = READ_LP36_STATUS();
    if(config_status != VALID_CONFIG_STATUS) {
       printf("MAX10 Config status invalid: %x\n", config_status);
       return -1;
    }

    // Reset all leds of Cyclone V and Max10
    WRITE_LEDS(0);
    all_max10_leds_off();

    // Output the constant values of the Avalon bus and our interface
    printf("AXI LW Const32: %x\n", (unsigned)READ_CONST_AXI_LW());
    printf("Our interface Const32: %x\n", (unsigned)READ_CONST_INT());

                                int offset = 0;
    int last_key2_val = 0;

    while (1) {
       /*  Ensuite pendant l'exécution du programme, à tout instant les actions suivantes
doivent
       être respectées :
       1. Copie de la valeur des 10 interrupteurs (SW) sur les 10 leds de la DE1-SoC.
       2. L'état de SW9-8 permet de sélectionner les leds à mettre à jour sur la carte
          Max10_leds :
          - SW9-8 = 00 : Leds secondes DS30.. .1.
          - SW9-8 = 01 : Leds secondes DS60...31.
          - SW9-8 = 10 : Les 2 lignes de leds DL.
          - SW9-8 = 11 : Le carré des leds DM.
       3. L'état de KEY1-0 permet de définir la valeur affichée sur les leds sélectionnés
          de la carte Max10_leds :
          - KEY1-0 = 00 : Copie de la valeur des 8 interrupteurs (SW0 to SW7) sur
```

```
        les poids faibles. Les leds de poids forts sont éteintes.
      - KEY1-0 = 01 : Afficher la valeur 1010…1010.
      - KEY1-0 = 10 : Afficher la valeur 0101…0101.
      - KEY1-0 = 11 : Afficher la valeur 1111…1111.
   4. Pression sur KEY2 :
      Lors de la sélection du carré des leds DM ainsi que la copie de la
      valeur des 8 interrupteurs : Faire décaler d'une ligne vers le bas la
      valeur des 8 interrupteurs affichés sur le carré des leds DM.
   5. Pression sur KEY3 :
      Eteindre toutes les leds de la carte Max10_leds. */

                                        // check if the MAX10 is always connected and the good
status
            config_status = READ_LP36_STATUS();
                                    if(READ_LP36_STATUS() != VALID_CONFIG_STATUS) {
                                                                printf("MAX10 Config
status invalid: %x\n", config_status);
                                                                return -1;
                                                            }

    // Read the switches and keys
    int switches = READ_SWITCHS();
    int keys = READ_KEYS();

    // Copy the switches to the leds
    WRITE_LEDS(switches);

    // Check if we need to turn off all the leds
    if (keys & KEY3_MASK) {
                        all_max10_leds_off();
                                continue;
    }

    // Select the leds to update
    int select_mode = (switches >> 8) & 0x3;
    int mask_to_use, sel_value;

    switch (select_mode) {
       case 0:
          mask_to_use = LP36_DATA_SEC1_MASK;
          sel_value = SECONDARY_1;
          break;
       case 1:
          mask_to_use = LP36_DATA_SEC2_MASK;
          sel_value = SECONDARY_2;
          break;
       case 2:
          mask_to_use = LP36_DATA_LINE_MASK;
          sel_value = TWO_LINE;
```

```
            break;
        case 3:
            mask_to_use = LP36_DATA_SQUA_MASK;
            sel_value = SQUARE;
            break;
    }

    // Write the selected leds
    do {
        WRITE_LP36_SEL(sel_value);
    } while (CANT_WRITE_SEL);

    // Write the data to the leds
    int display_pattern = keys & KEY1_0_MASK;
    int switches_low = switches & SW7_0_MASK;
    int value_to_write = 0;

    if (display_pattern == 0) {
        if (mask_to_use == LP36_DATA_SQUA_MASK) {
            // Isolate the first and second line values from switches_low
            int first_line_value = switches_low & SQUARE_FIRST_LINE_MASK;    //
Extracts bits for the first line (SW0 to SW4)
            int second_line_value = (switches_low & SQUARE_SECOND_LINE_MASK) >>
5; // Extracts bits for the second line (SW5 to SW9)

            // Shift the isolated values to their correct positions based on offset
            int shifted_first_line = first_line_value << (SQUARE_LINE_SIZE * offset); // Shift
first line to correct position
            int shifted_second_line = second_line_value << (SQUARE_LINE_SIZE * ((offset
+ 1) % SQUARE_LINE_SIZE)); // Shift second line to next position

            // Combine both shifted values into square_shifted_value
            value_to_write = shifted_first_line | shifted_second_line;
        } else {
            value_to_write = switches_low;
        }
    } else if (display_pattern == 1) {
        value_to_write = LEDS_PATTERN_A;
    } else if (display_pattern == 2) {
        value_to_write = LEDS_PATTERN_B;
    } else if (display_pattern == 3) {
        value_to_write = LEDS_PATTERN_C;
    }

    // Write the value to the leds of the Max10
    WRITE_LP36_DATA(value_to_write, mask_to_use);

    // Check if we need to shift the square
```

```
        if ((keys & KEY2_MASK) && display_pattern == 0 && mask_to_use ==
LP36_DATA_SQUA_MASK && !last_key2_val) {
            offset = (offset + 1) % SQUARE_LINE_SIZE;
        }
        last_key2_val = keys & KEY2_MASK ? 1 : 0;
    }

    return 0;
}
```

5.2 Avl_user_interface.vhd

```
--------------------------------------------------------------------------------
-- HEIG-VD ////////////////////////////////////////////////////////////////////
-- Haute Ecole d'Ingenerie et de Gestion du Canton de Vaud
-- School of Business and Engineering in Canton de Vaud
--------------------------------------------------------------------------------
-- REDS Institute ////////////////////////////////////////////////////////////
-- Reconfigurable Embedded Digital Systems
--------------------------------------------------------------------------------
--
-- File               : avl_user_interface.vhd
-- Author             : Urs Behrmann
-- Date               : 06.11.2024
--
-- Context            : Avalon user interface
--
--------------------------------------------------------------------------------
-- Description :
--
--------------------------------------------------------------------------------
-- Dependencies : None
--
--------------------------------------------------------------------------------
-- Modifications :
-- Ver    Date        Engineer    Comments
-- 0.0    See header  UB          Initial version
-- 1.0    12.11.2024  UB          Final version

--------------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY avl_user_interface IS
  PORT (
    -- Avalon bus
    avl_clk_i : IN STD_LOGIC;
    avl_reset_i : IN STD_LOGIC;
    avl_address_i : IN STD_LOGIC_VECTOR(13 DOWNTO 0);
    avl_byteenable_i : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
    avl_write_i : IN STD_LOGIC;
    avl_writedata_i : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
    avl_read_i : IN STD_LOGIC;
    avl_readdatavalid_o : OUT STD_LOGIC;
    avl_readdata_o : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
    avl_waitrequest_o : OUT STD_LOGIC;
```

```
    -- User interface
    boutton_i : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
    switch_i : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
    led_o : OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
    lp36_we_o : OUT STD_LOGIC;
    lp36_sel_o : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
    lp36_data_o : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
    lp36_status_i : IN STD_LOGIC_VECTOR(1 DOWNTO 0)
  );
END avl_user_interface;

ARCHITECTURE rtl OF avl_user_interface IS

  --| Components declaration |-----------------------------------------------------------

  COMPONENT timer IS
    GENERIC (
      T1_g : NATURAL RANGE 1 TO 1023 := 50);
    PORT (
      clock_i : IN STD_LOGIC;
      reset_i : IN STD_LOGIC;
      start_i : IN STD_LOGIC;
      trigger_o : OUT STD_LOGIC
    );
  END COMPONENT;
  FOR ALL : timer USE ENTITY work.timer;

  --| Constants declarations |-----------------------------------------------------------
  CONSTANT INTERFACE_ID_C : STD_LOGIC_VECTOR(31 DOWNTO 0) :=
x"12345678";
  CONSTANT OTHERS_VAL_C : STD_LOGIC_VECTOR(31 DOWNTO 0) :=
x"00000000";

  CONSTANT ID_ADDRESS : INTEGER := 0;
  CONSTANT LED_ADDRESS : INTEGER := 1;
  CONSTANT SWITCH_ADDRESS : INTEGER := 2;
  CONSTANT BOUTTON_ADDRESS : INTEGER := 3;
  CONSTANT LP36_VALID_AND_WE_ADDRESS : INTEGER := 4;
  CONSTANT LP36_SEL_ADDRESS : INTEGER := 5;
  CONSTANT LP36_DATA_ADDRESS : INTEGER := 6;

  --| Signals declarations   |-----------------------------------------------------------
  SIGNAL led_reg_s : STD_LOGIC_VECTOR(9 DOWNTO 0);
  SIGNAL lp36_data_reg_s : STD_LOGIC_VECTOR(31 DOWNTO 0);
  SIGNAL lp36_sel_reg_s : STD_LOGIC_VECTOR(3 DOWNTO 0);

  SIGNAL boutton_s : STD_LOGIC_VECTOR(3 DOWNTO 0);
  SIGNAL switches_s : STD_LOGIC_VECTOR(9 DOWNTO 0);
```

```
  SIGNAL readdatavalid_next_s : STD_LOGIC;
  SIGNAL readdatavalid_reg_s : STD_LOGIC;
  SIGNAL readdata_next_s : STD_LOGIC_VECTOR(31 DOWNTO 0);
  SIGNAL readdata_reg_s : STD_LOGIC_VECTOR(31 DOWNTO 0);

  SIGNAL cs_wr_lp36_data_s : STD_LOGIC;
  SIGNAL lp36_valide_s : STD_LOGIC;
  SIGNAL us_done_s : STD_LOGIC;
  SIGNAL start_timer_s : STD_LOGIC;
  SIGNAL lp36_we_s : STD_LOGIC;

  --| Types |-------------------------------------------------------------
  TYPE state_t IS (
    --General state
    ATT,
    GET_DATA,
    WAIT1US,
    -- Error
    ERR
  );
  SIGNAL e_pres, e_fut_s : state_t;

BEGIN

  -- Input signals

  boutton_s <= boutton_i;
  switches_s <= switch_i;

  lp36_valide_s <= '1' WHEN lp36_status_i = "01" ELSE
    '0';

  -- Output signals

  avl_readdatavalid_o <= readdatavalid_reg_s;
  avl_readdata_o <= readdata_reg_s;

  led_o <= led_reg_s;

  lp36_sel_o <= lp36_sel_reg_s;
  lp36_data_o <= lp36_data_reg_s;
  lp36_we_o <= lp36_we_s;

  -- Read access part
  -- Read register process

  read_decoder_p : PROCESS (ALL)
  BEGIN
    readdatavalid_next_s <= '0'; --valeur par defaut
```

```
  readdata_next_s <= (OTHERS => '0'); --valeur par defaut

  IF avl_read_i = '1' THEN
    readdatavalid_next_s <= '1';

    CASE (to_integer(unsigned(avl_address_i))) IS

      WHEN ID_ADDRESS =>
        readdata_next_s <= INTERFACE_ID_C;

      WHEN SWITCH_ADDRESS =>
        readdata_next_s(9 DOWNTO 0) <= switch_i;

      WHEN BOUTTON_ADDRESS =>
        readdata_next_s(3 DOWNTO 0) <= boutton_s;

      WHEN LP36_VALID_AND_WE_ADDRESS =>
        readdata_next_s(0) <= lp36_valide_s;
        readdata_next_s(1) <= lp36_we_s;

      WHEN LP36_DATA_ADDRESS =>
        readdata_next_s <= lp36_data_reg_s;

      WHEN OTHERS =>
        readdata_next_s <= OTHERS_VAL_C;

    END CASE;
  END IF;
END PROCESS;

-- Read register process
read_register_p : PROCESS (avl_reset_i, avl_clk_i)
BEGIN
  IF avl_reset_i = '1' THEN

    readdatavalid_reg_s <= '0';
    readdata_reg_s <= (OTHERS => '0');

  ELSIF rising_edge(avl_clk_i) THEN

    readdatavalid_reg_s <= readdatavalid_next_s;
    readdata_reg_s <= readdata_next_s;

  END IF;
END PROCESS;

-- Write access part

write_register_p : PROCESS (
```

```
  avl_reset_i,
  avl_clk_i,
  avl_write_i,
  avl_writedata_i,
  led_reg_s,
  lp36_data_reg_s,
  lp36_sel_reg_s,
  cs_wr_lp36_data_s
  )
BEGIN

  IF avl_reset_i = '1' THEN

    led_reg_s <= (OTHERS => '0');
    lp36_data_reg_s <= (OTHERS => '0');
    lp36_sel_reg_s <= (OTHERS => '0');

  ELSIF rising_edge(avl_clk_i) THEN

    cs_wr_lp36_data_s <= '0';

    IF avl_write_i = '1' THEN

      CASE (to_integer(unsigned(avl_address_i))) IS

        WHEN LED_ADDRESS =>
          led_reg_s <= avl_writedata_i(9 DOWNTO 0);

        WHEN LP36_SEL_ADDRESS =>
          -- Write only if not in transfering mode
          IF lp36_we_s = '0' THEN
            lp36_sel_reg_s <= avl_writedata_i(3 DOWNTO 0);
          END IF;

        WHEN LP36_DATA_ADDRESS =>
          -- Write only if not in transfering mode
          IF lp36_we_s = '0' THEN
            lp36_data_reg_s <= avl_writedata_i;
            cs_wr_lp36_data_s <= '1';
          END IF;

        WHEN OTHERS =>
          NULL;

      END CASE;
    END IF;
  END IF;
END PROCESS;
```

```
-- Interface management

-- Timer management

timer_boutton : timer
GENERIC MAP(T1_g => 50)
PORT MAP(
  clock_i => avl_clk_i,
  reset_i => avl_reset_i,
  start_i => start_timer_s,
  trigger_o => us_done_s
);

-- State machine
-- This process update the state of the state machine
fsm_reg : PROCESS (avl_reset_i, avl_clk_i) IS
BEGIN
  IF (avl_reset_i = '1') THEN
    e_pres <= ATT;
  ELSIF (rising_edge(avl_clk_i)) THEN
    e_pres <= e_fut_s;
  END IF;
END PROCESS fsm_reg;

dec_fut_sort : PROCESS (
  e_pres,
  cs_wr_lp36_data_s,
  lp36_valide_s,
  us_done_s,
  start_timer_s,
  lp36_we_s
  ) IS
BEGIN
  -- Default values for generated signal
  start_timer_s <= '0';
  lp36_we_s <= '0';

  CASE e_pres IS
    WHEN ATT =>
      IF cs_wr_lp36_data_s = '1' THEN
        e_fut_s <= GET_DATA;
      ELSE
        e_fut_s <= ATT;
      END IF;
    WHEN GET_DATA =>
      IF lp36_valide_s = '0' THEN
        e_fut_s <= ERR;
      ELSE
        e_fut_s <= WAIT1US;
```

```
        start_timer_s <= '1';
      END IF;
    WHEN WAIT1US =>
      IF us_done_s = '0' THEN
        e_fut_s <= WAIT1US;
        lp36_we_s <= '1';
      ELSE
        e_fut_s <= ATT;
      END IF;
    WHEN ERR =>
      IF lp36_valide_s = '1' THEN
        e_fut_s <= ATT;
      ELSE
        e_fut_s <= ERR;
      END IF;
    WHEN OTHERS =>
      e_fut_s <= ATT;
  END CASE;
 END PROCESS dec_fut_sort;
END rtl;
```

5.3 Timer.vhd

```
--------------------------------------------------------------------------------
-- HEIG-VD, Haute Ecole d'Ingenierie et de Gestion du canton de Vaud
-- Institut REDS, Reconfigurable & Embedded Digital Systems
--
-- Fichier      : timer.vhd
-- Auteur       : Etienne Messerli, le 05.05.2016
--
-- Description  : Detection d'un clic et double clic
--                Projet repris du labo Det_Clic_DblClic 2012
--
-- Utilise      : Labo SysLog2 2016
--| Modifications |------------------------------------------------------------
-- Ver   Date      Qui        Description
-- 1.0   05.05.16  EMI          version initiale
-- 1.1   19.11.20  SMS          remplacement des constantes par des g□n□riques
--------------------------------------------------------------------------------

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

entity timer is
    generic (
        T1_g : natural range 1 to 1023 := 2);
    port (
        clock_i    : in  std_logic;
        reset_i    : in  std_logic;
        start_i    : in  std_logic;
        trigger_o : out std_logic
        );
end timer;

architecture comport of timer is

            signal timer_pres : std_logic_vector(8 downto 0);
             signal timer_fut : std_logic_vector(8 downto 0);

                  signal t1 : std_logic_vector(8 downto 0);

begin

               t1 <= std_logic_vector(to_unsigned(T1_g, 9));

           timer_fut <=  (others => '0') when (start_i = '1') else
                std_logic_vector(unsigned(timer_pres) + 1);
```

```
  process(reset_i, clock_i)
  begin
                                                    if reset_i = '1' then
                                                            timer_pres <= (others
=> '0');
                                                elsif rising_edge(clock_i) then
                                                            timer_pres <=
timer_fut;
                                                    end if;
  end process;

  trigger_o <= '0' when (timer_pres < t1) else '1';

end comport;
```