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**

# 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.

# Analyse et conception

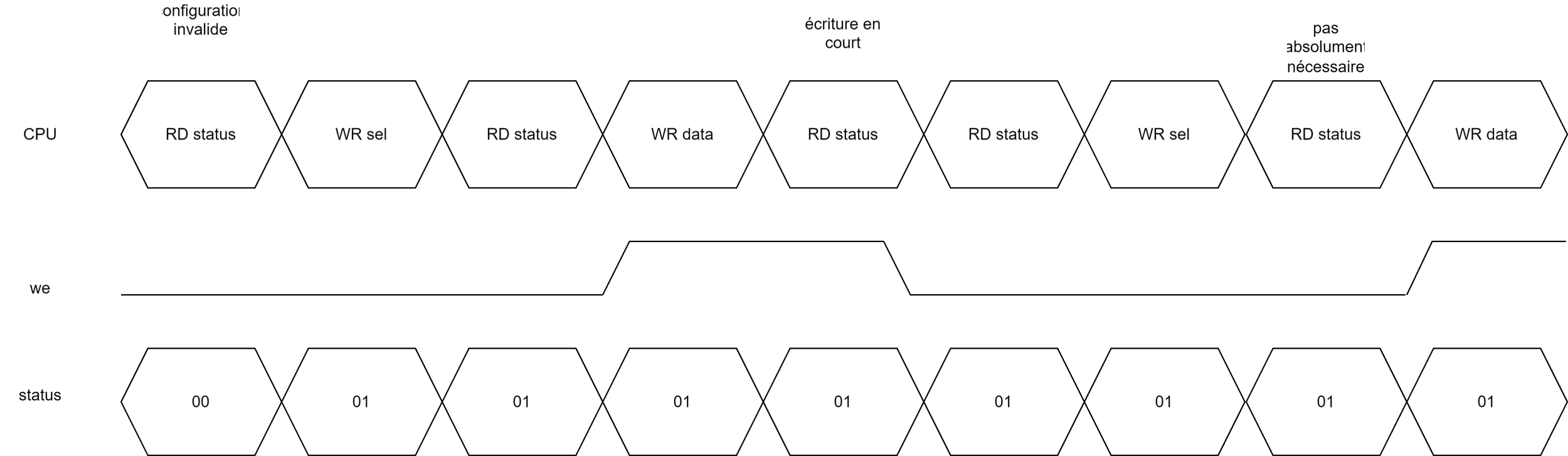
## Plan d’adressage

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 214×4=216 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 |

## Chronogramme



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

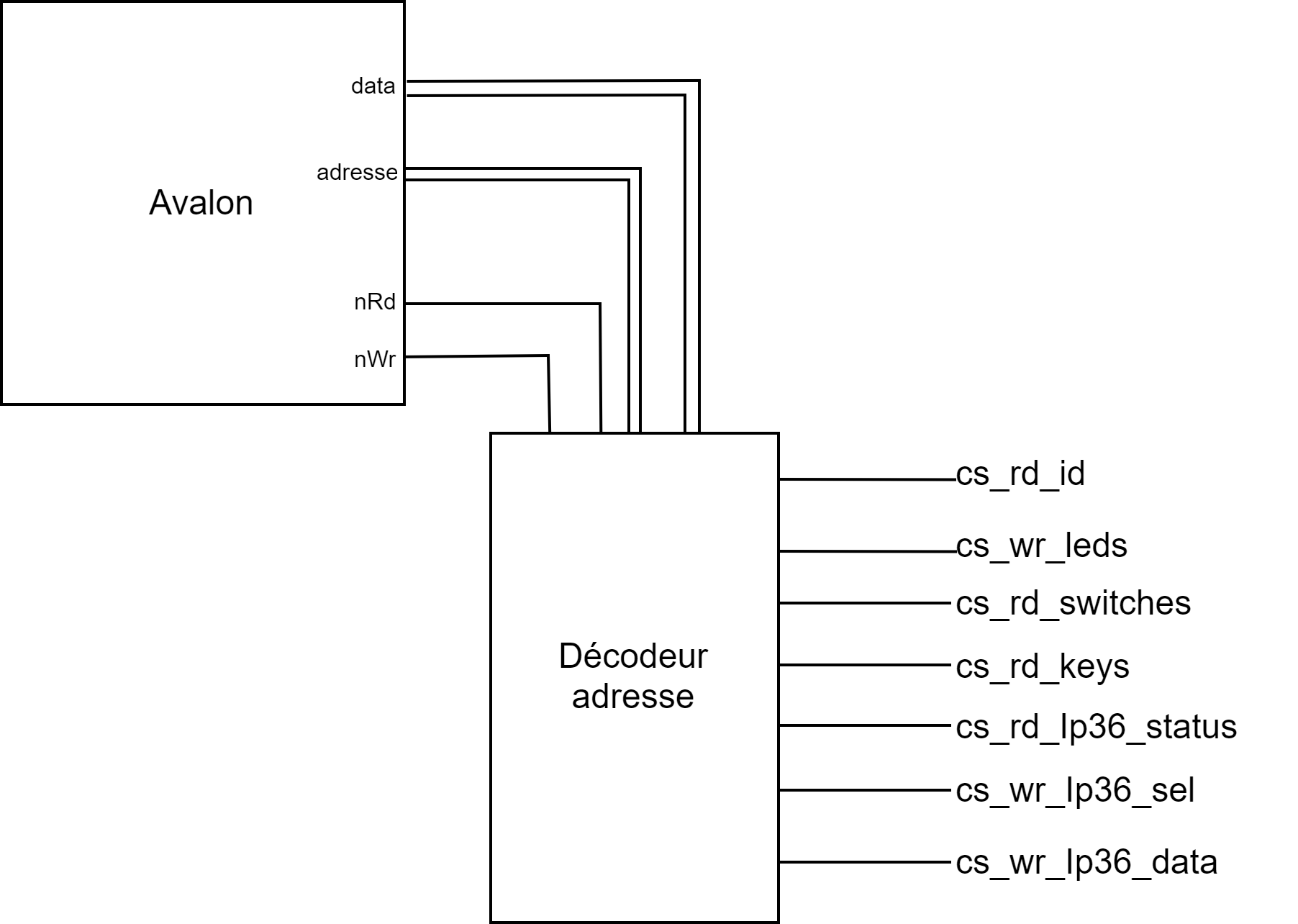
Si le statut n’est pas « 01 », il faut modifier la sélection pour corriger la configuration.

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.

## Schéma bloc de l’interface Avalon

### 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

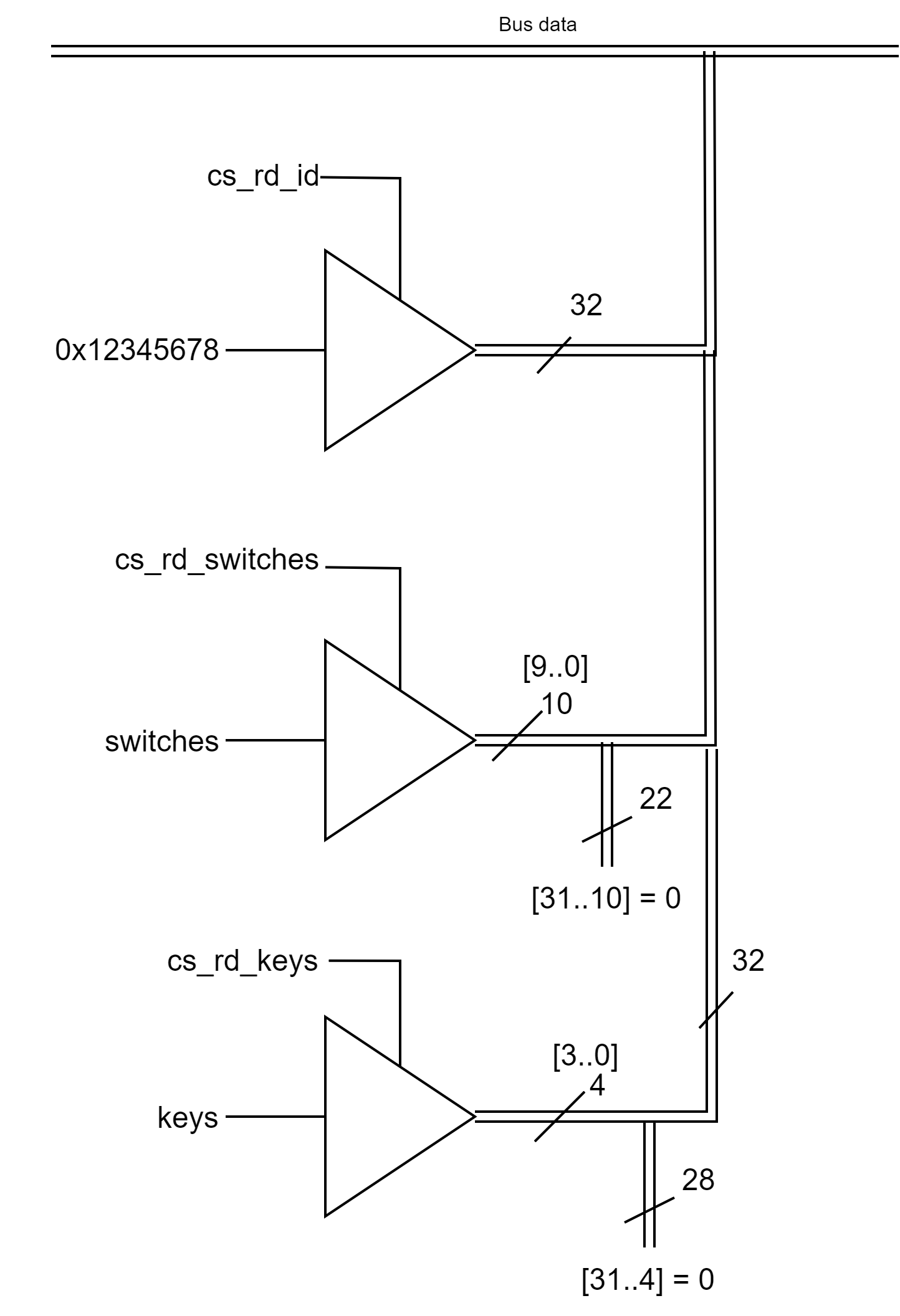
cs\_rd\_lp36\_data => addr = 0x01\_0018 \* rd

cs\_wr\_lp36\_sel => addr = 0x01\_0014 \* wr

cs\_wr\_lp36\_data => addr = 0x01\_0018 \* wr

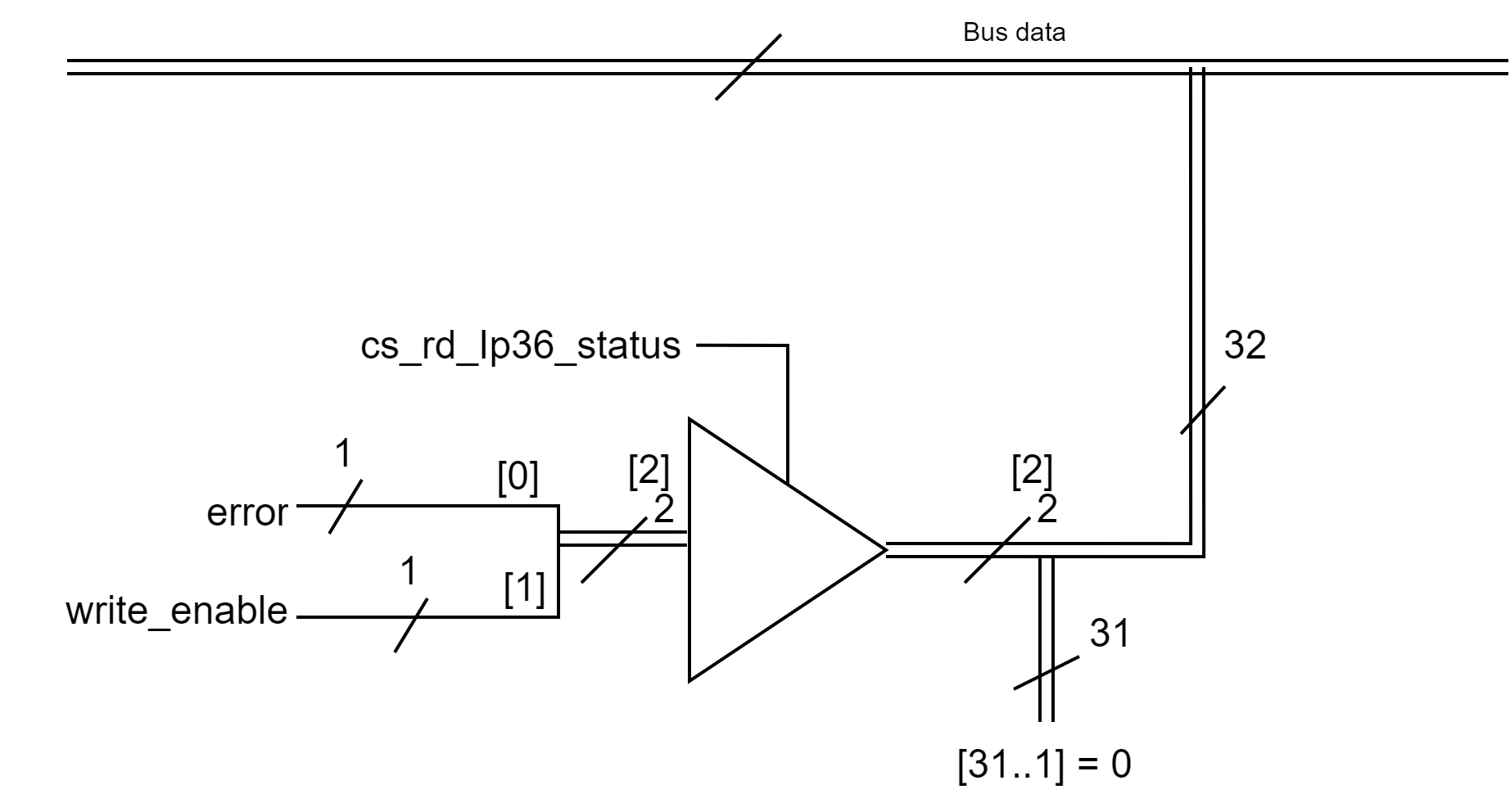
### Read ID, switches et keys

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*.

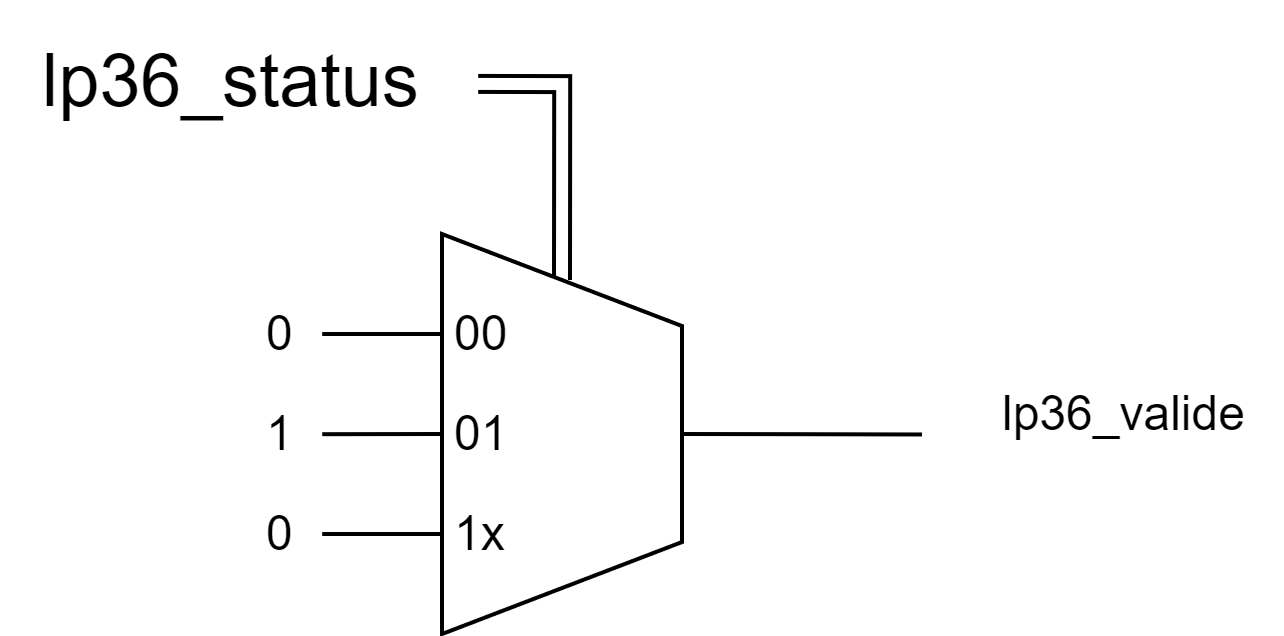


### Read status lp36

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.

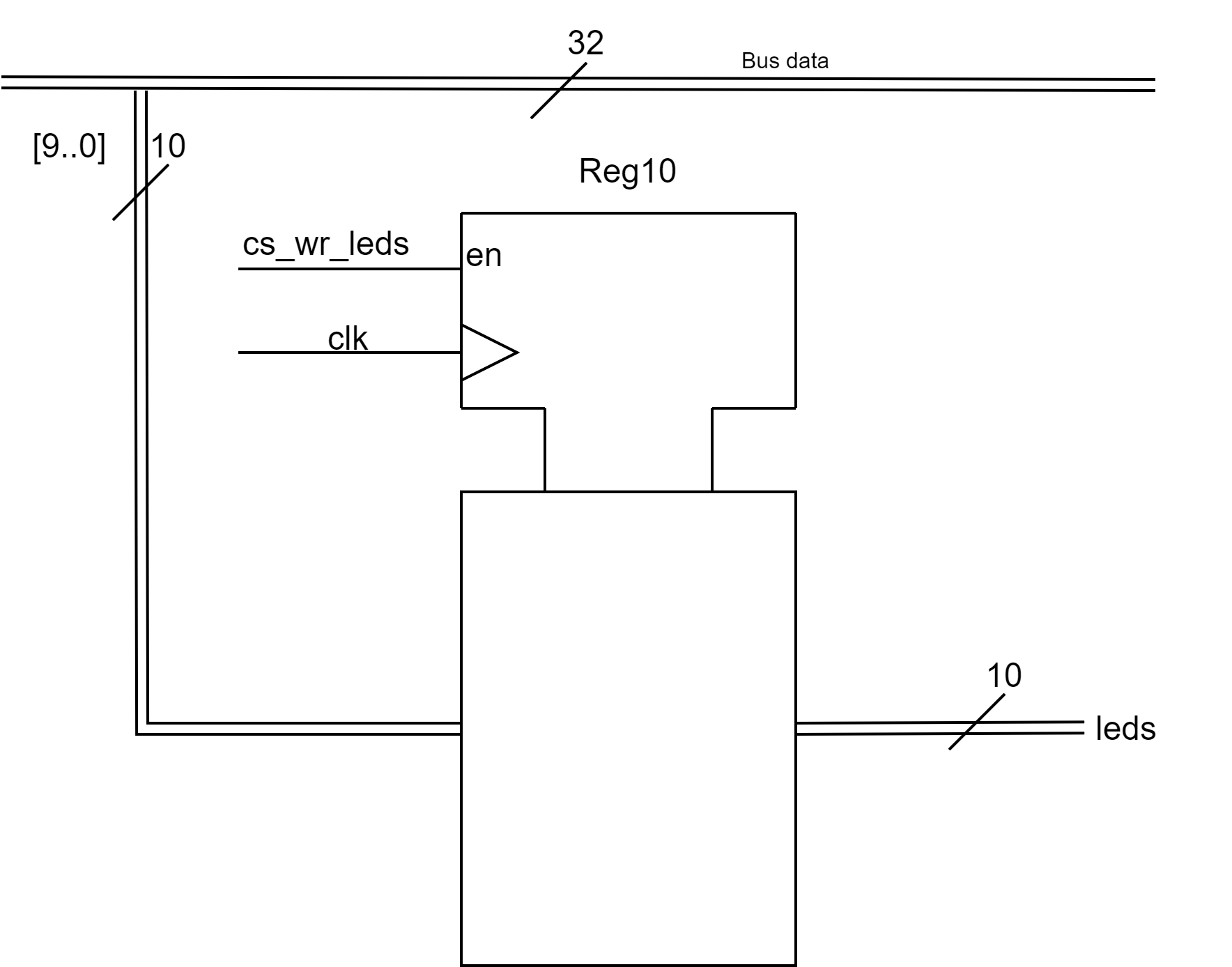


Le statut du *lp36* est codé sur 2 bits, ce qui permet d'obtenir 4 valeurs différentes. Cependant, seules les deux premières valeurs sont utilisées pour indiquer s'il y a une erreur ou non. Comme les deux autres valeurs sont "réservées", aucune erreur n'est retournée si le signal est à 2 ou 3.



### Write leds

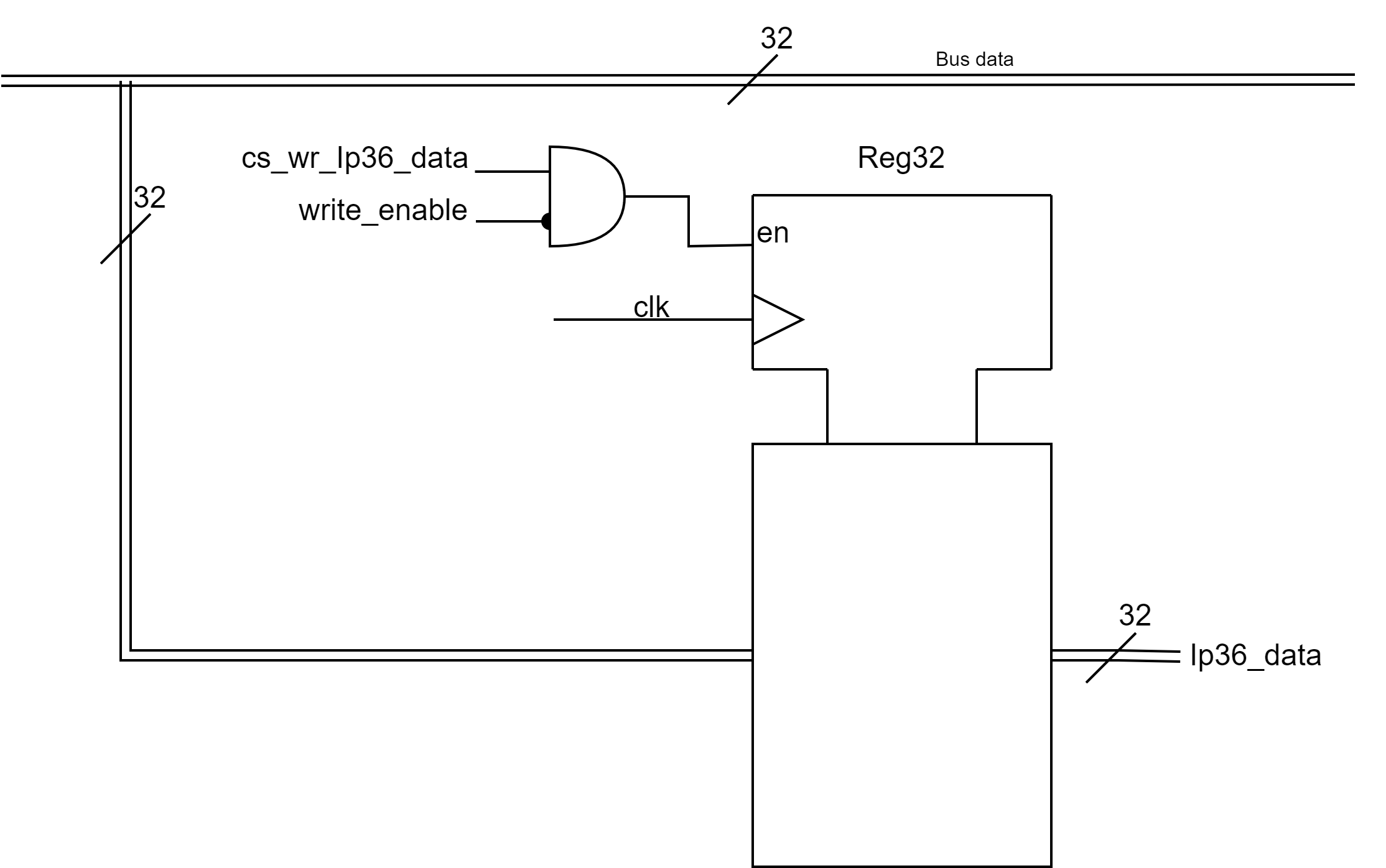
Pour l'écriture des LEDs, le signal de contrôle *wr\_leds* est activé, et les données sont enregistrées dans un registre qui est ensuite utilisé pour allumer les LEDs.



### 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.

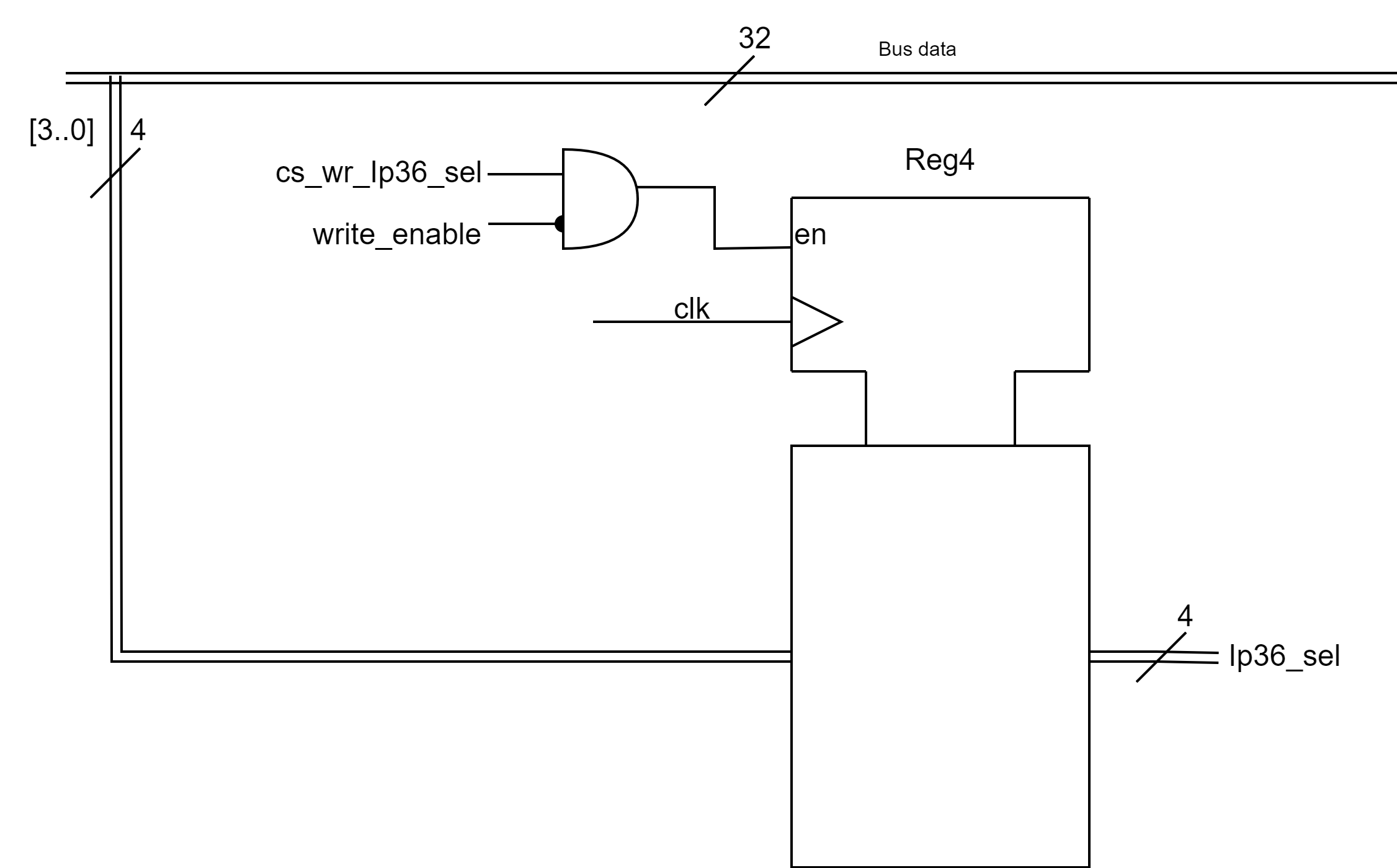
Pendant l'écriture sur le Max10, l'écriture de nouvelles données depuis le CPU est bloquée afin d'éviter toute erreur.



### 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.



### Liaison avec la Max10

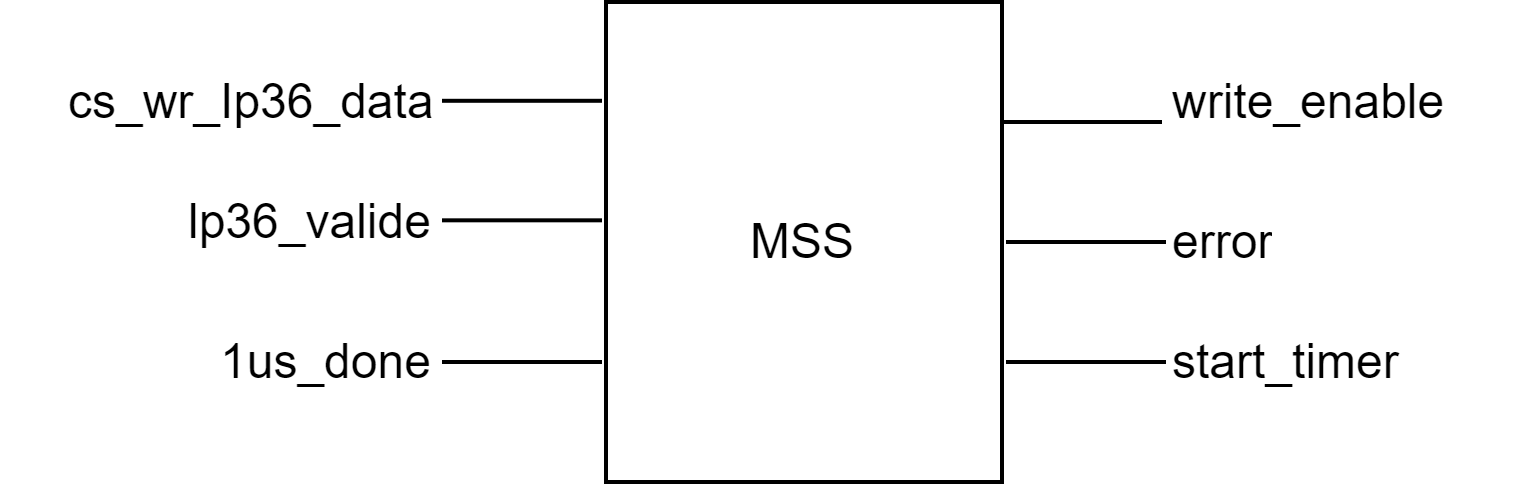
Pour la liaison avec le Max10, il est nécessaire de synchroniser les données afin de garantir une écriture valide. Pour cela, un MSS est utilisé pour envoyer les données et informer le CPU de l'état de l'écriture.

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.

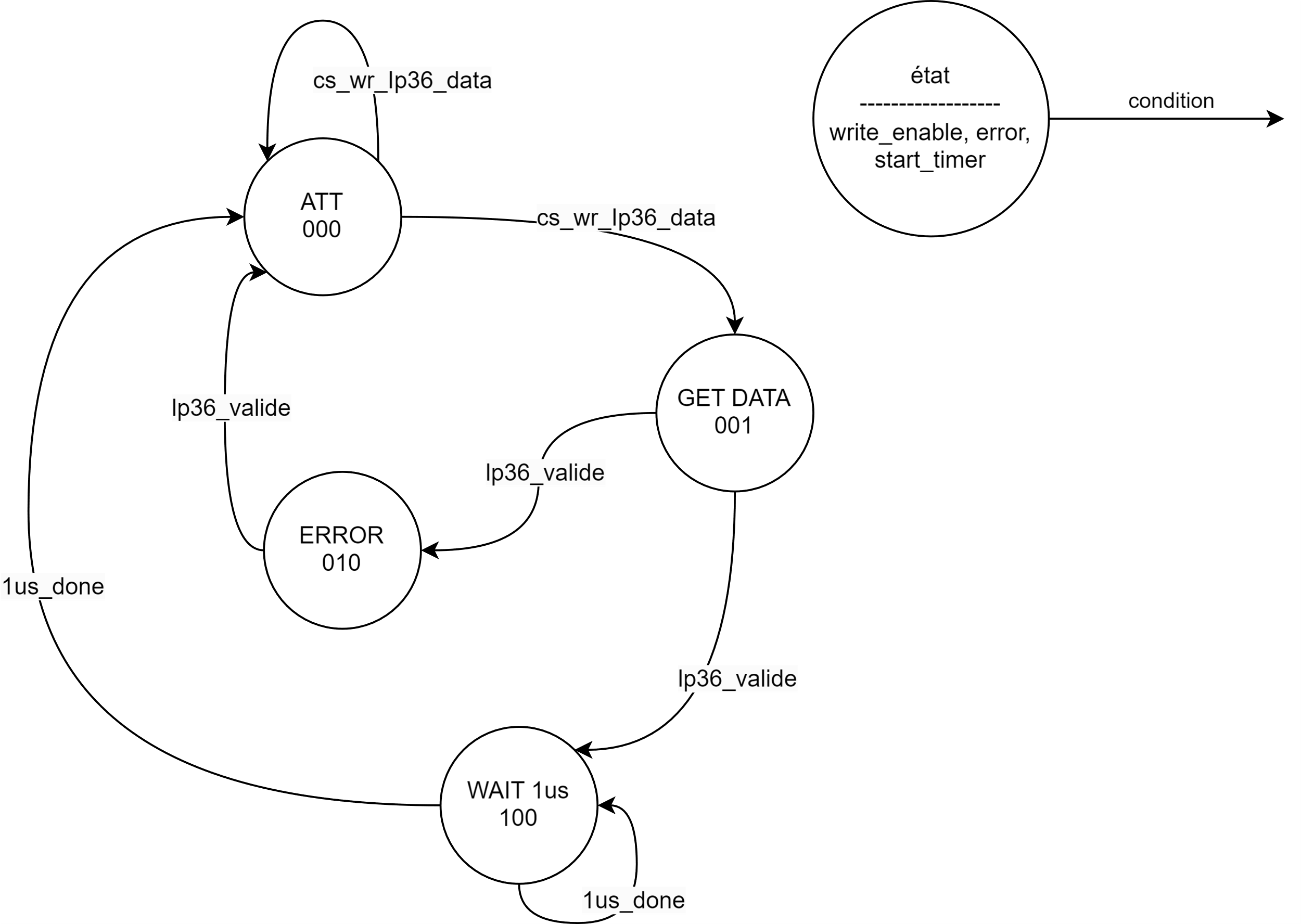
Nous avons décidé d'utiliser un retour d'information (feedback) du signal write\_enable pour indiquer au CPU que l'écriture a été effectuée, même si ce signal ne sera actif que pendant un cycle d'écriture (1 µs) pour éviter des problèmes de synchronisation temporelle. Cela ne devrait pas poser de problème,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

### 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**.



### 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.

### 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.

# Simulation / Tests

## 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 |

## 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 |

### 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.

Ein Bild, das Text, Screenshot, Display, Software enthält.

Automatisch generierte Beschreibung

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

Ein Bild, das Text, Screenshot, Display, Software enthält.

Automatisch generierte Beschreibung

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

Ein Bild, das Screenshot, Text, Multimedia-Software, Software enthält.

Automatisch generierte Beschreibung

# 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

# Annexes ….

## 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;

}

## 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;

## 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;