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

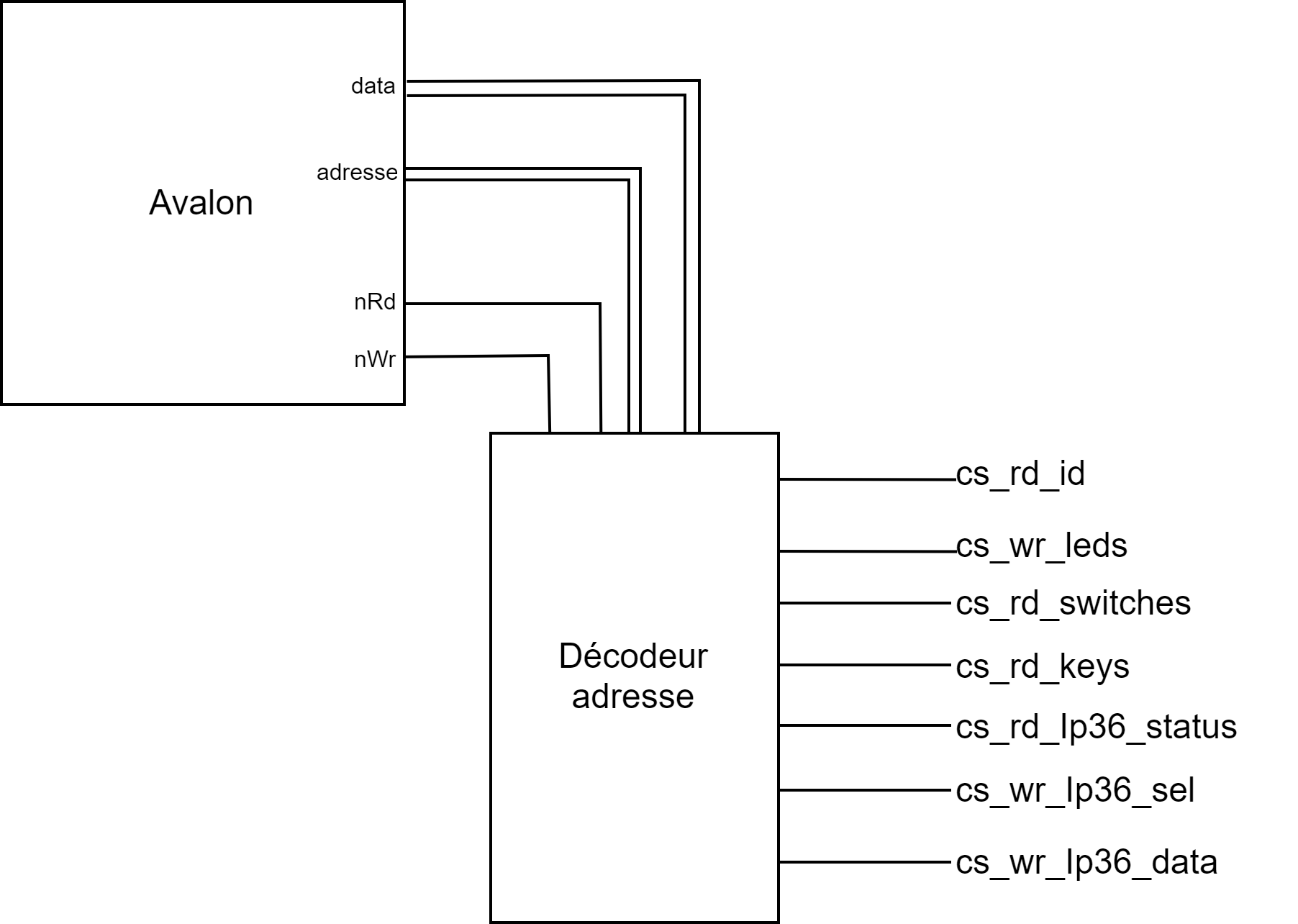
Dans le plan d’adressage, la taille de la zone disponible pour votre interface correspond telle aux 14 bits d’adresse défini dans le bus Avalon ? Pourquoi ?

Parce que chaque adresse Avalon correspond à 4 octets (32 bits), ainsi les 14 bits d'adresse du côté FPGA couvrent une zone équivalente à celle des 16 bits d'adresse du côté CPU qui est disponible 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 |

## 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

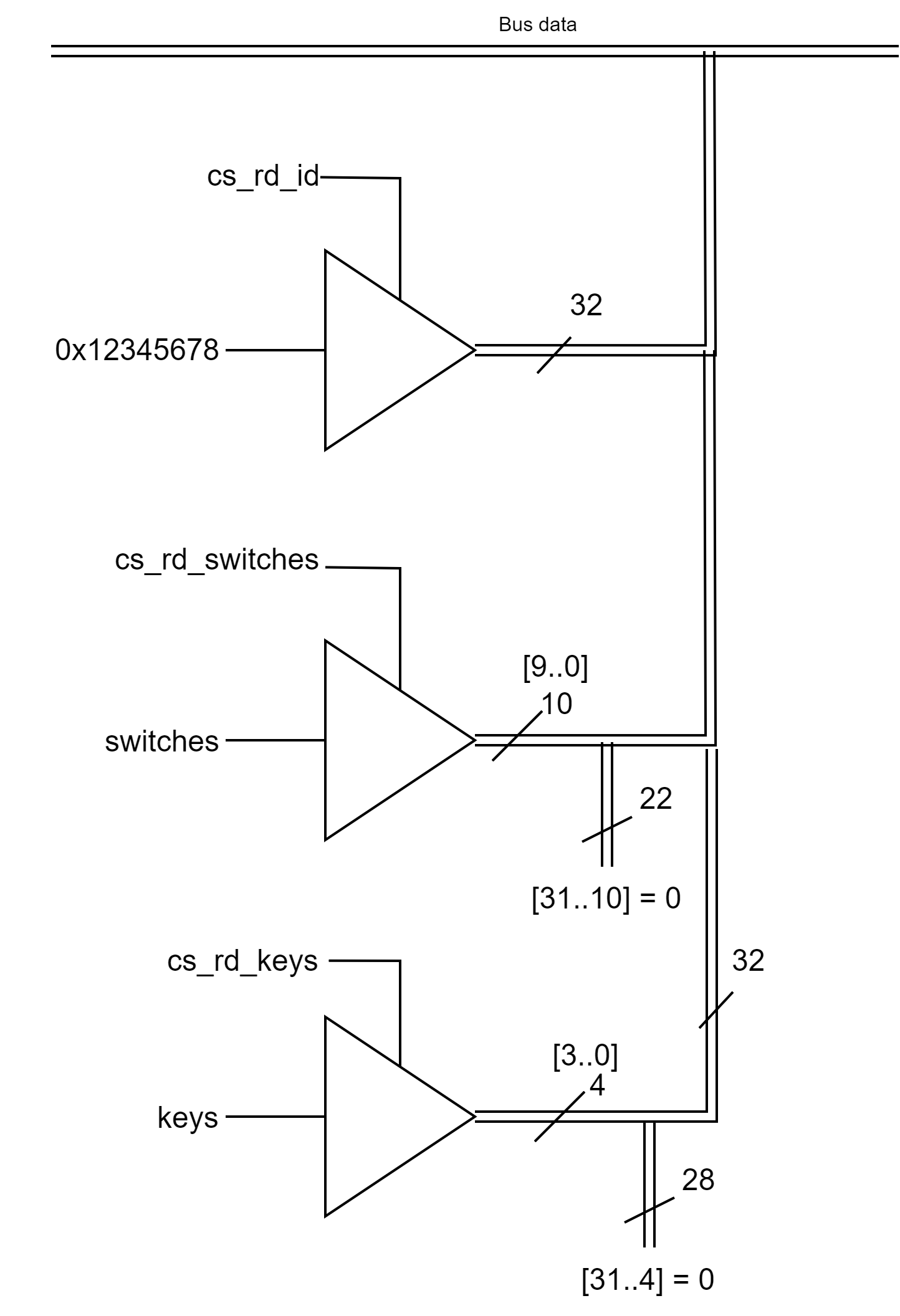
cs\_rd\_lp36\_data => addr = 0x01\_0018 \* rd

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

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

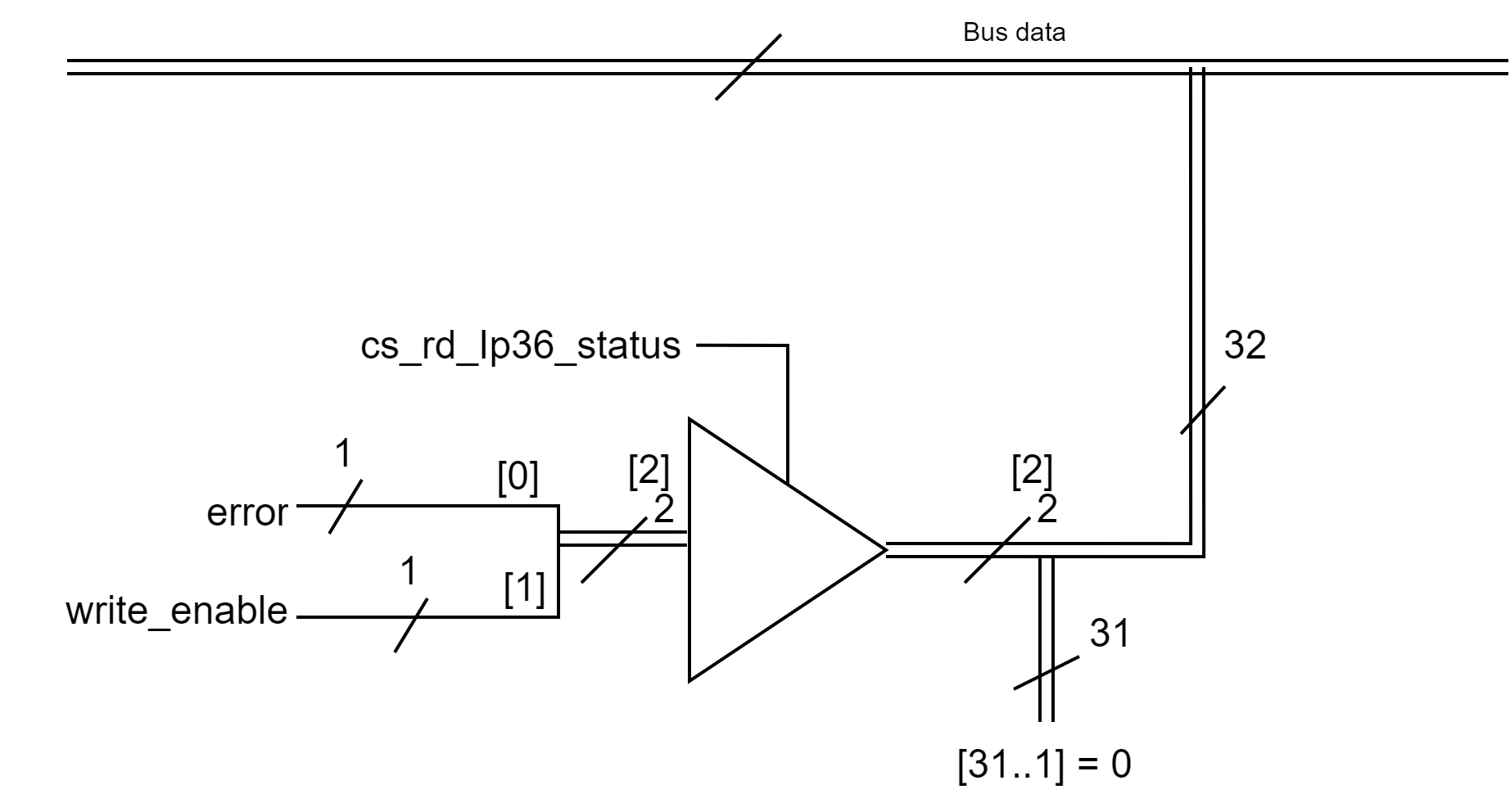
### Read ID, switches et keys

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

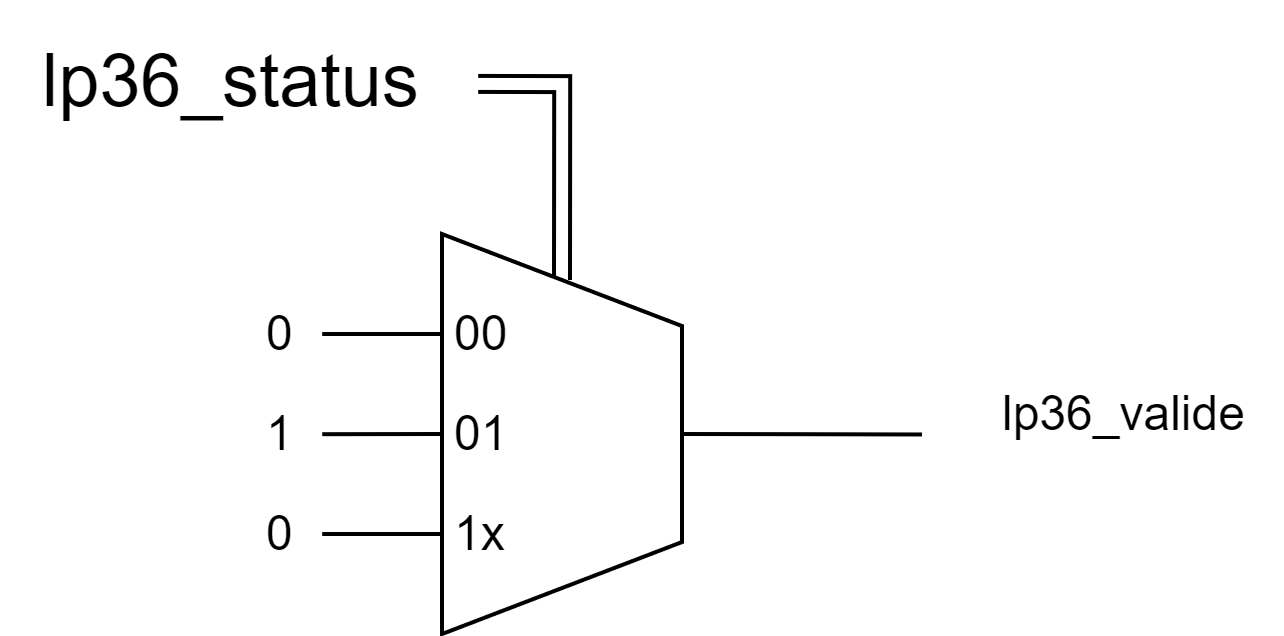


### Read status lp36

Pour la lecture de l'erreur du lp36, on fait la même chose que pour les lectures précédentes, mais l'erreur est interprétée en fonction de la valeur du signal lp36\_status et on plus de l'erreur, on retourne aussi le signal write\_enable qui indique si on est en train d'écrire sur le lp36. Le signal write\_enable est activé par la MSS.

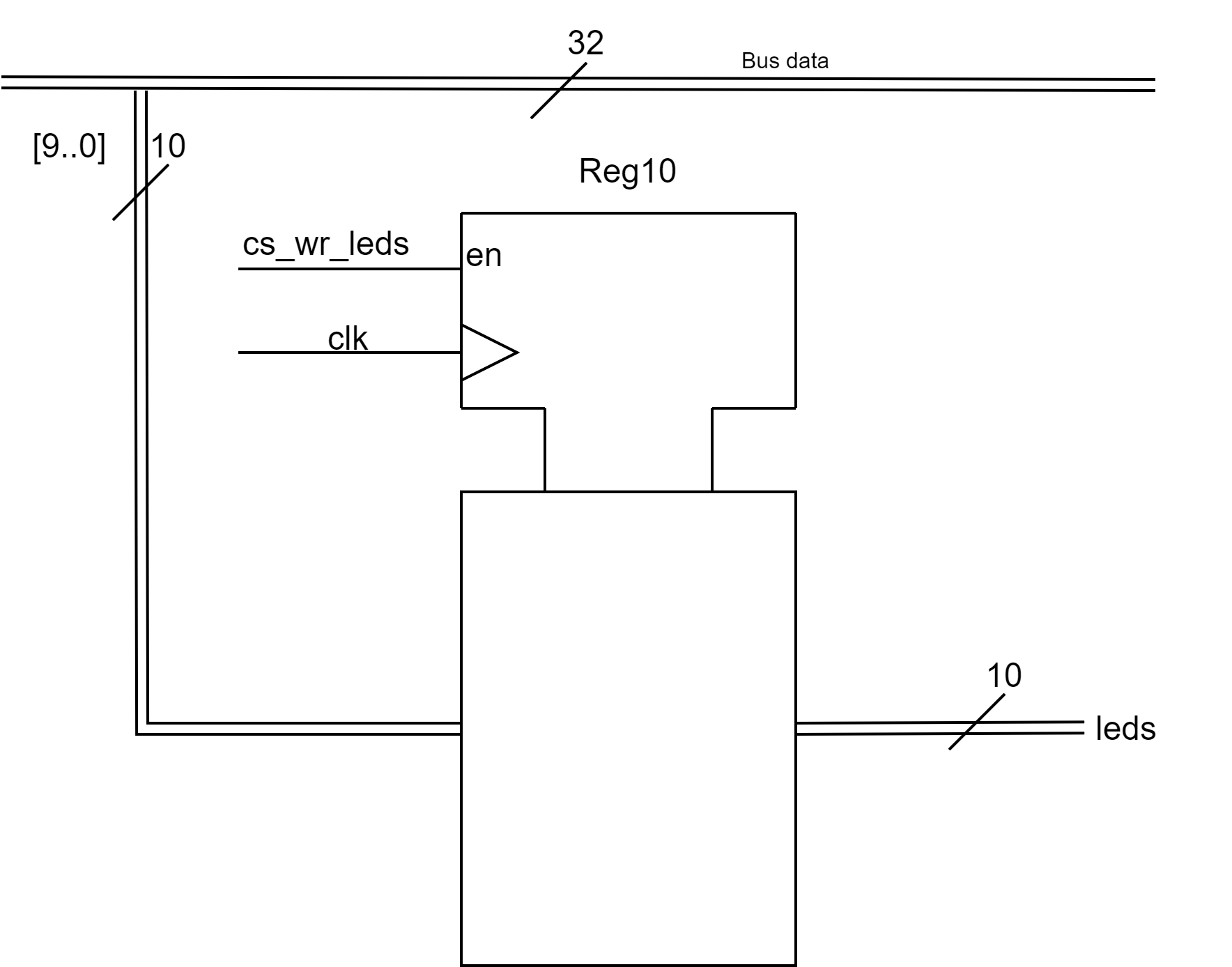


Le status du lp36 est sur 2 bits, donc on peut avoir 4 valeurs différentes, mais seulement les deux premières sont utilisées pour indiquer s’il y a une erreur ou non. Vu que les deux autres sont "réservées", on ne retourne pas d'erreur si le signal est à 2 ou 3.



### Write leds

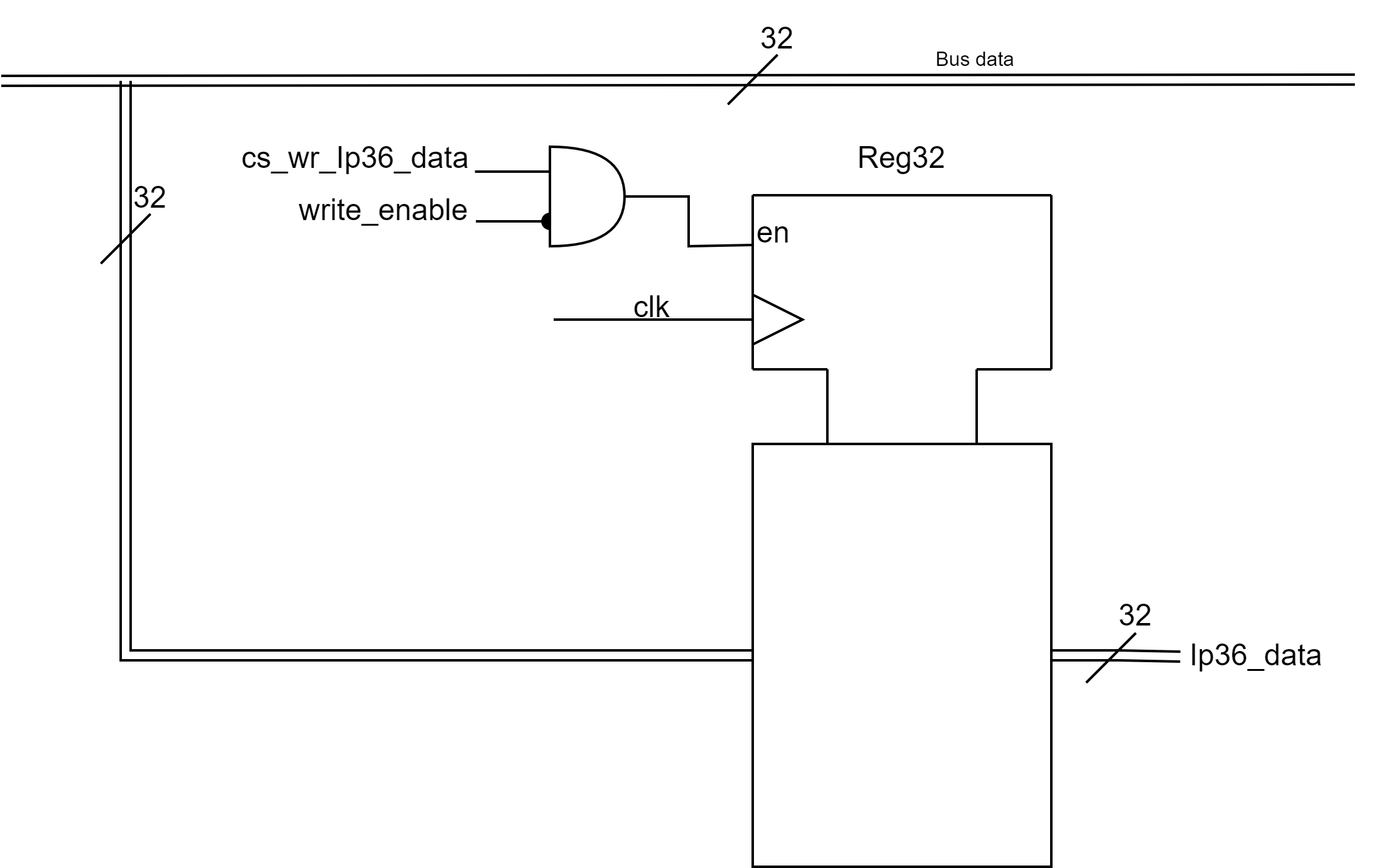
Pour l'écriture des leds, on active le signal de contrôle wr\_leds et on enregistre les données dans un registre qui est ensuite utilisé pour allumer les leds.



### Write lp36 data

Pour l'écriture des données du lp36, on active le signal de contrôle wr\_lp36\_data et on enregistre les données dans un registre qui est ensuite utilisé pour envoyer les données au lp36.

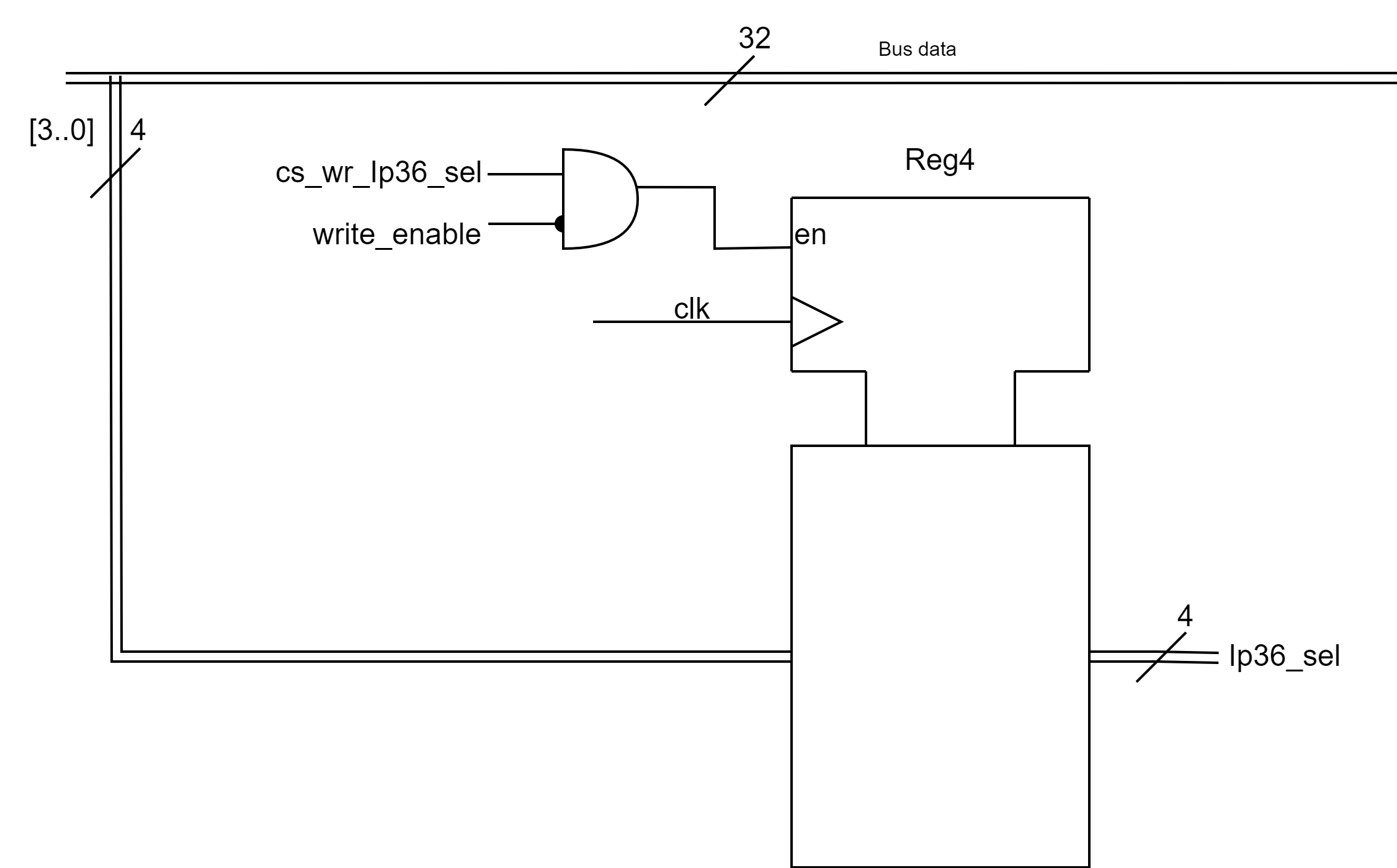
Durant l'écriture sur la Max10, on bloque l'écriture depuis le CPU des données pour ne pas créer des erreurs.



### Write lp36 sel

Pour l'écriture du sélecteur du lp36, on active le signal de contrôle wr\_lp36\_sel et on enregistre les données dans un registre qui est ensuite utilisé pour envoyer les données au lp36.

Durant l'écriture sur la Max10, on bloque l'écriture depuis le CPU du sélecteur pour ne pas créer des erreurs.



### Liaison avec la Max10

Pour la liaison avec la Max10, on doit cadencer les données pour avoir une écriture valide. Pour cela, on utilise un MSS qui va permettre d'envoyer les données et d'informer le CPU de l'état de l'écriture.

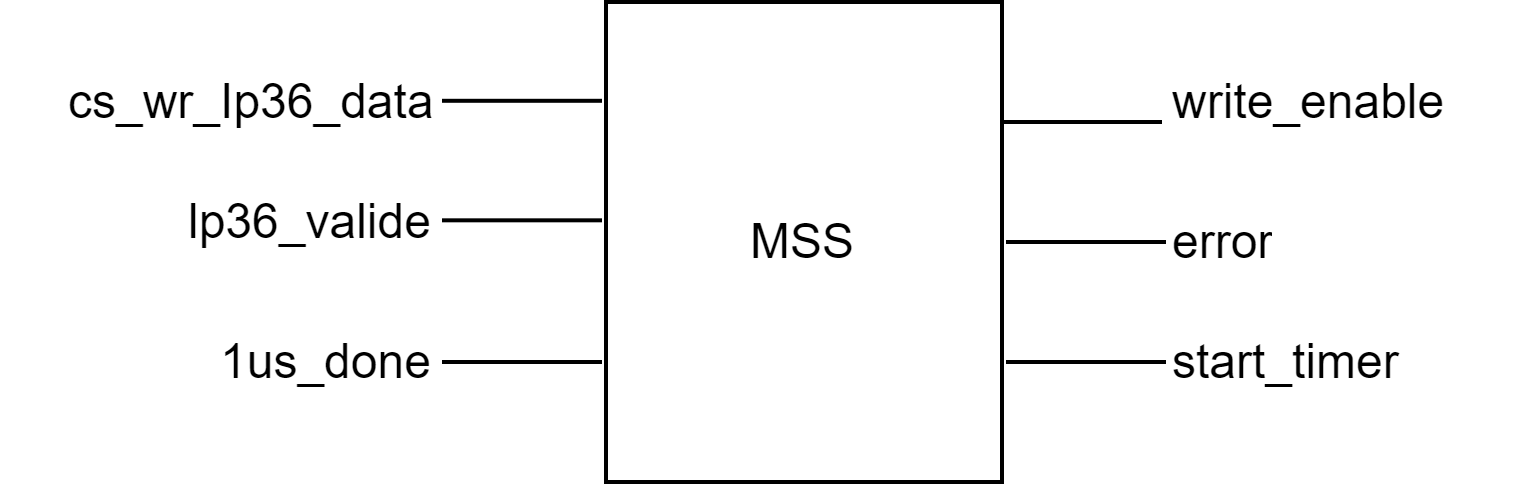
Le CPU a deux informations à lire: l'état de l'écriture(write\_enable) et l'état du lp36(error).

Si le CPU veut écrire, il doit vérifier que le signal write\_enable est à 0. Si c'est le cas, il peut écrire la sélection des leds pilotés et les données dans le registre de données. Une fois déposées, le CPU doit relire le même registre pour vérifier qu'il n'y a pas eu d'erreur. Si il y a une erreur, la sélection des leds pilotés n'est pas valide.

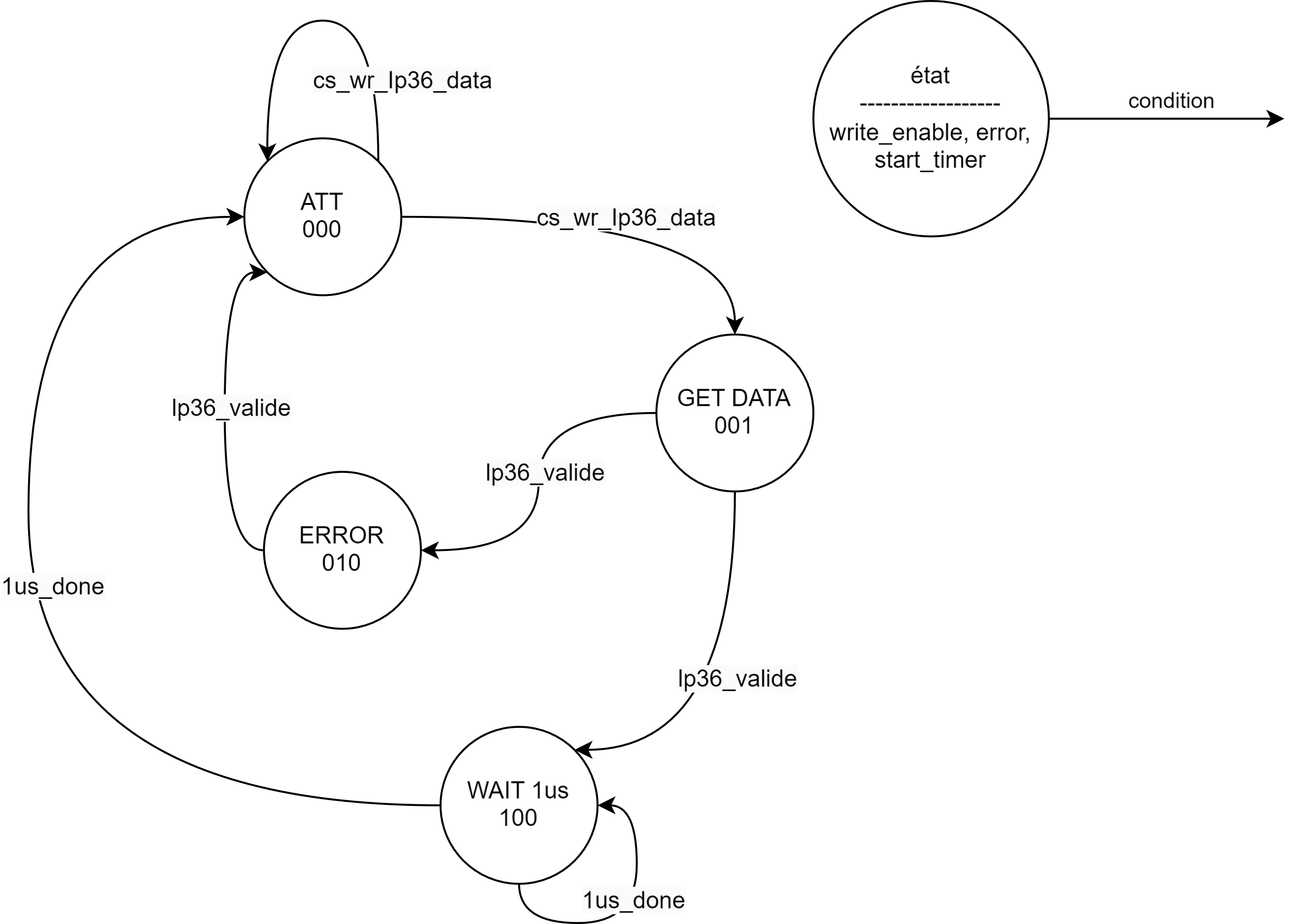
On a décidé d'utilisé un feedback du write\_enable pour indiquer au CPU que l'écriture a été effectuée, même si le signal sera allumé que pendant un cycle d’écriture (1us) pour éviter des problèmes de timing. Cela ne devrait pas arrivé, car on dépendant d'input du user sur les boutons et switches pour changer les valeurs pour le Max10.

### MSS pour la liaison Max10

On a décidé d'utilisé une machine séquentielle synchrone pour la liaison avec la Max10. La raison principale étant que l'écriture doit être actif pendant un cycle d'écriture (1us).



Pour cela, on a décidé d'utiliser un MSS avec 4 états: ATT, GET\_DATA, WAIT 1US et ERROR.



### 1us

Pour avoir un cycle d'écriture de 1us, on doit déterminer combien de cycles de l'horloge du FPGA correspondent à 1us. Pour cela, on utilise la fréquence de l'horloge du bus Avalon qui est de 50MHz. Cela nous donne que un cycle de l'horloge correspond à 20ns. Pour avoir 1us, on doit donc attendre 50 cycles de l'horloge.

### Code C

Dans le code C, on a choisi d'écrire toujours 0 pour les bits qui nous concerne pas (exemple : écriture dans les 10 LEDs: valeur écrite: 0x000003FF) bien que par la suite la partie FPGA s'assure n'écrire que les bits qui le doivent (les 10 derniers dans notre exemple). L'usage de macro C nous garantit une meilleur lisibilité du code et simplifie grandement les adaptation futures de celui-ci (par exemple si notre plan d'adressage change). Comme mentionné plus tôt dans ce rapport, pour pouvoir écrire dans lp36\_sel il faut que lp36\_wr soit à zéro et que le statut de la max10 soit valide (c'est à dire lp36\_status == 0b01). Ceci est donc fais dans le code avec une boucle d'écriture dans lp36\_sel jusqu'à ce que ces deux conditions soit valide. L'usage d'une boucle peut sembler risqué mais wr étant maintenu que 1us la boucle ne devrait jamais durer trop longtemps et donc ne devrais pas mettre à mal le système.

# Réalisation et implantation

# Simulation

# Conclusion

Dans ce laboratoire, on a créé une interface entre le CPU et la Max10. On a utilisé le bus Avalon pour communiquer entre les deux. On a créé un décodeur d'adresse pour gérer les différentes lectures et écritures. On a aussi créé un MSS pour la liaison avec la Max10. On a utilisé la console TCL-TK pour simuler les entrées du CPU et tester l'interface. Finalement, on a montré à Anthony Convers que notre interface fonctionne correctement le 12.11.2024.

Date : 13.11.2024

Noms des étudiants : Urs Behrmann Guillaume Gonin

# Annexes ….

Documents annexés avec numérotation et commentaires.