Laboratoire 3: Conception d'une interface simple

Département: **TIC**

Unité d'enseignement: **ARE**

Auteur(s):

- **BOUGNON-PEIGNE Kévin**
- **CECCHET Costantino**

Professeur:

- **MESSERLI Etienne**

Assistant:

- **CONVERS Anthony**

Date:

- **Novembre 2023**

Sommaire

Introduction

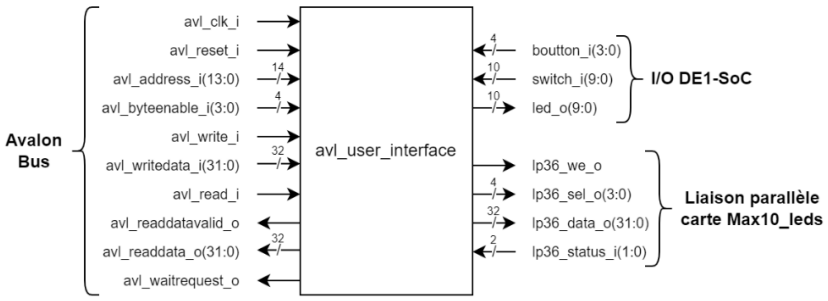
Pour ce laboratoire, il est demandé de réaliser une interface simple, connectée sur le bus Avalon interconnectant l'HPS (microcontroller) et l'FPGA de la carte DE1-SoC.

L'arborescence de fichiers de base du projet a été fourni par les responsables de cours.

Conception de l'interface

Plan d'adressage

Comme le montre le bloc de l'interface a concevoir:



Elle ne reçoit que 14bits pour l'adresse.

Ceci s'explique par le fait que le bus de données est sur **32bits**. Dès lors, une valeur 32bits doit pouvoir s'adresser de telles façons à accèder à chacun de ces bytes, soit:

```
|   REGISTRE     32     BITS    |
| ------------------------------ |
| BYTE3 | BYTE2 | BYTE1 | BYTE0 |
|      0x3     0x2     0x1     0x0 Offset à partir de l'adr. de base du reg.
```

Ce faisant, les adresses dans le plan d'adressage doivent être alignées sur 2bits, autrement dit, alignées sur 4. **Ce qui enlève les 2 premiers bits de poids faibles.**

Le bus AXI lightweight. Le bus Avalon est un sous-bloc avec 64KBytes

Ensuite, le manuel de référence technique du processeur nous indique que le bus Avalon débute à l'adresse 0xFF20'0000, avec une mémoire allouée de 2MB. **Il en est déduisible que les 11 derniers bits de poids forts sont décodés par ce dernier.** ✓

De plus, la zone attribuée aux étudiants est présenté avec le tableau ci-dessous:

Offset on bus AXI lightweight HPS2FGPA	Fonctionnalités
0x00_0000 - 0x00_0003	Constante ID 32Bits (Read only)
0x00_0004 - 0x00_FFFF	reserved
0x01_0000 - 0x01_FFFF	**Zone à disposition des étudiants**
0x02_0000 - 0x1F_FFFF	not used

Une constante ID de 32bits est retrouvée, mais cette dernière est implémentée par le *design* des responsables de cours. De fait, l'interface à concevoir est derrière le *design* sus-mentionné **et donc, les bits utilisés par l'interface à concevoir sont alors les 14bits d'adresse 15 à 2.** ✓

Le plan d'adressage a ensuite été définit comme suit: Indiquer: base adresse 0xFF20_0000

Offset

Address (CPU Side) [16..0]	Address (Itf side) [15..2]	Definition	R/W
0x1_0000	0x0000	Constant ID (**0xDEADBEEF**)	R
0x1_0004	0x0001	Constant 2 (Debug)	R/W
0x1_0008	0x0002	IN: Switches	R
0x1_0010	0x0004	IN: Keys	R
0x1_0020	0x0008	OUT: LEDs	R/W
0x1_0040	0x0010	OUT: MAX10-LEDs	R/W
0x1_0080	0x0020	OUT: MAX10-cfg (status[5..4] + sel[3..0])	R/W[1]
0x1_0100	0x0040	IN: MAX10-busy (write_enable)	R
0x1_0200	0x0080	*reserved*	-
0x1_0400	0x0100	*reserved*	-
0x1_....	0x0...	*reserved*	-
0x1_....	0x0...	*reserved*	-
0x1_FFF8	0x0800	*reserved*	-
0x1_FFFC	0x0000	*reserved*	-

Si R/W c'est un registre

Manque details pour les bits utilisés ?

Write_enable pour un read?

Ne correspond pas à l'adresse CPU

[1] *status* est *read only*, car ces bits indiquent si la MAX10 est prête à l'emploi ou non.

Justification du plan d'adressage:

- La première constante permet d'identifier l'interface à concevoir, conformément à la donnée du laboratoire.
- La seconde n'est présente qu'à des fins de vérifications intermédiaires, portant sur la lecture et l'écriture sur le bus.
- Les entrées utilisateurs (switches & bouttons) ont été séparées, afin d'éviter d'effectuer du masquage et des shift pour obtenir le périphérique désiré.
- Comme il a été définit un registre 32bits pour les LEDs de la MAX10, les LEDs de la DE1SOC se trouvent également isolée dans une adresse.
- Pour les bits de configuration de la MAX10, ils ont été groupés. L'ordre était de mettre les bits de status au plus haut (selon les éventuels autres bits de config.), car ces derniers ne seront utilisés qu'à l'enclenchement du programme.

  Les bits de sélection de la zone active de la MAX10 est au plus bas, pour n'avoir qu'à faire un masquage pour obtenir la valeur.
- Pour finir, le bit *"busy"* représentant le *write_enable* a été isolé, afin que le programme puisse attendre sur la fin d'écriture avec une simple boucle:

```
while( BUSY_REG ) ;
```

Déclenchement pour l'ecriture max10?

Quant aux adresses, avec le nombre de registres définis et la taille à disposition, il a été décidé de faire en sorte de n'avoir qu'un bit actif par registre.

Votre choix d'adressages n'est pas très optimum, vous avez beaucoup de zones d'adresses de perdu

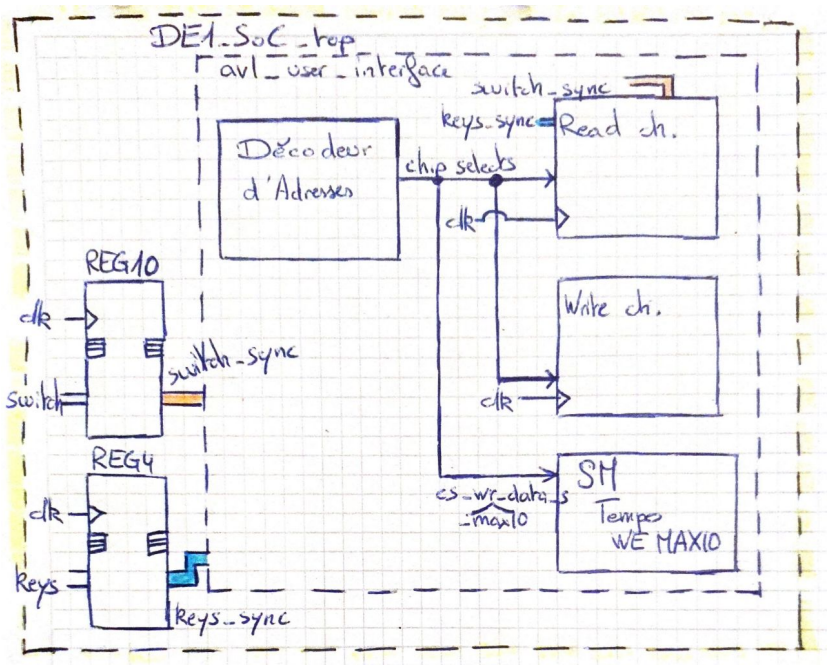
Cela implique que vous "consommez" de nombreuses adresses.
Cela doit être indiqué dans votre plan d'adressage.

Exemple:
OUT: MAX10-LEDs utilise les adresses 0x0010 à 0x001F !!!!

Finalement dans le VHDL, vous comparez une valeur fixe et pas un no de bit !!

Réalisation du circuit

La circuiterie a été coupée en 3 parties majeures; **lecture**, **écriture** et une troisième pour la gestion de la **validité d'écriture sur la MAX10** (signal *lp36_we_o*).



cs_wr_max10 provient seulement du decodeur d'adresses?

Busy?

Manque un peu d'information, le schéma n'est pas très clair.

Note: Les interrupteurs et les boutons arrivant sur l'interface ont été syncronisés au niveau de la DE1. Ceci dans le but de ne pas perturber la simulation, lorsque l'on change la valeur de ces derniers entre 2 lectures consécutives.

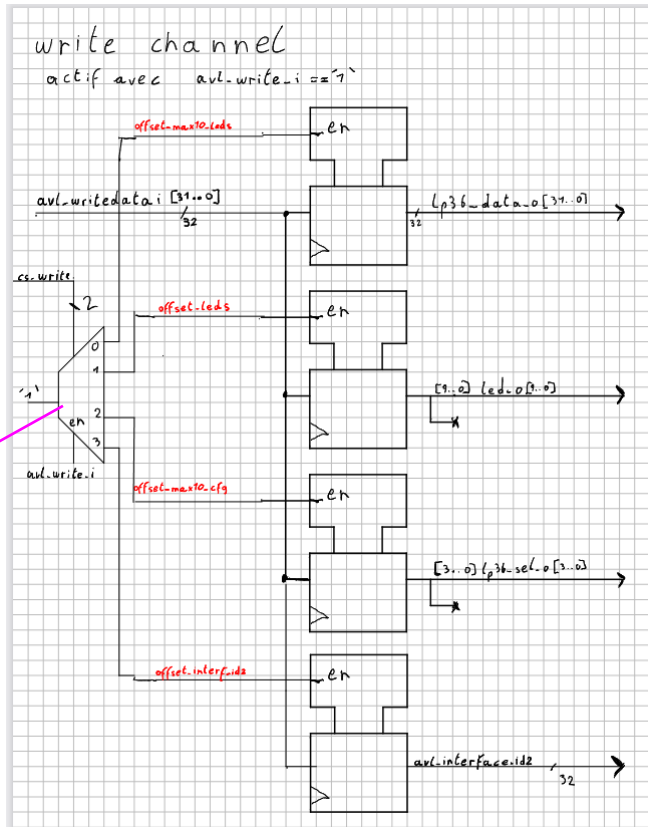
De plus, comme les portes trois-états ne sont pas disponibles dans une puce FPGA, il faut passer par du dé/multiplexage.

Canal d'écriture

Ce canal possède un DEMUX qui permet de choisir lequel des 4 registres écrivables (voir tableau en titre **Plan d'adressage**) sera écrit.

Le schéma résultant se présente ainsi:

Liens entre CS et adresses
par rapport à votre plan d'adressage?



Avec votre solution:
j'ai compris que vous utilisez directement 1 bit ?

Pas besoin de décodeur

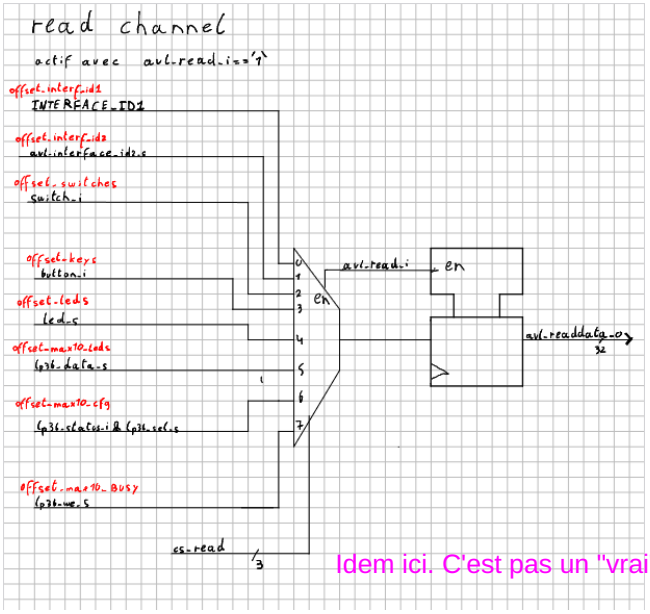
Dans la description du *design*, c'est ce canal qui traite le reset des différents registres.

Remarque: Le registre de la constante ID de **debug** se reset à sa valeur initiale (`0xCAFE0369`) et non pas à 0.

Canal de lecture

Pour celui de lecture, c'est un MUX qui permet de choisir parmis les 9 registres lisibles (voir tableau en titre **Plan d'adressage**) celui qui sera lu.

En voici son schéma:



Lors de la lecture d'une donnée, un décalage d'une période d'horloge est effectuée pour obtenir le signal *read_datavalid*.

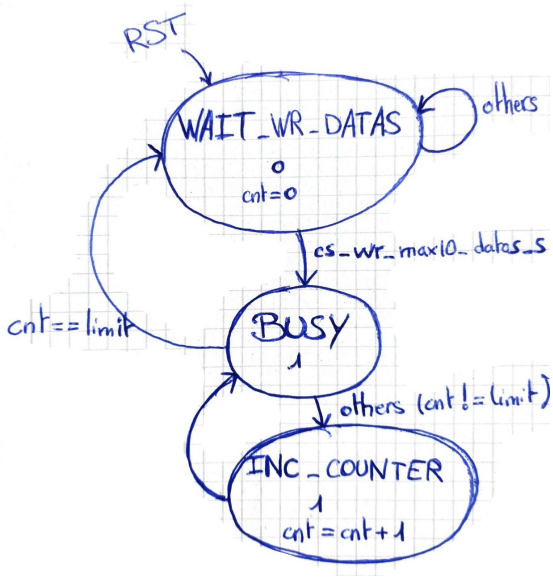
Décalage d'une periode par rapport a quel signal?
Alignement avec signal read_data?

Machine d'état pour l'écriture de la MAX10

Pour l'écriture de la MAX10, la création d'une machine d'état est nécessaire.

Selon les spécifications du bus, le signal *lp36_we* doit être actif pendant une période **d'au moins 1[us]**, afin de valider une écriture fiable des bits de sélections et des données des LEDs de la MAX10.

Voici le diagramme de la machine d'état, où les '0' et '1' isolés sont les valeurs de *lp36_enable* à chaque état:



Pendant cette période de 1[us], le bit de *busyness* de notre interface est maintenu actif.

Comme il sera possible de le voir en **annexes**, grâce à quelques mesures, le signal peut avoir quelques perturbations et a alors une légère gigue.

C'est pourquoi, une marge de 10% (purement subjectif) a été prise en considération, lors du calcul de la limite à compter.

Pour la gestion de la *SM*, un compteur est implémenté, afin de compter le nombre de cycle nécessaire au maintien du signal.

Initialement, la limite a compté respecte la règle: $\frac{T_{maintien}}{T_{clk}}$. Toutefois, comme le montre le diagramme d'état, le maintien du signal est fait dans 2 états, il faut donc diviser par le nombre d'états dans lesquels le maintien est fait.

Le calcul est alors:

$$Limit = \frac{T_{maintien}}{T_{clk} * N_{states}}$$

avec: $T_{maintien} = 1.1[us]$, $T_{clk} = \frac{1}{50'000'000} = 20[ns]$ et $N_{states} = 2$.

Et donc, avec transformation des valeurs en [ns], on obtient:

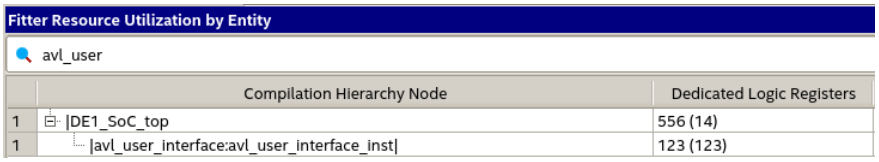
$$Limit = \frac{1'100}{20*2} = 27.5 \approx 28$$

$$RealT_{maintien} = 28 * 20 * 2 = 1'120[ns] => 1.12[us]$$

Synthèse

Après discussion avec l'enseignant, la vue RTL et la quantité de logique ne sont plus très parlantes, de part l'explosion de logiques. Cependant, une autre information, permettant un contrôle plus approchable, est la quantité de registres dédiées au *design*.

Sous le report de synthétisation: `Fitter > Resource Section > Resource Utilization by Entity`, à l'aide du filtre et en cherchant *avl_user_interface*, on trouve cette information:


Fitter Resource Utilization by Entity

avl_user

	Compilation Hierarchy Node	Dedicated Logic Registers
1	|DE1_SoC_top	556 (14)
1	|avl_user_interface:avl_user_interface_inst|	123 (123)

Chaque registre peut être compté, afin de retrouver la valeur présente ci-dessus:

Column 1	Size (bits)	Column 2	Size (bits)
Constante ID de debug	**32**	Sel + Status	**6**
Données pour LP36	**32**	Machine d'états	**3**
Registre de lecture	**32**	CS d'écriture sur MAX10	**1**
Données des LEDs sur DE1SoC	**10**	*write enable*	**1**
Compteur de tempo.	**6**	/	/

Ce qui amène alors à:

$$DedicatedLogicRegisters = 32 + 32 + 32 + 10 + 6 + 6 + 3 + 1 + 1 = \mathbf{1}23$$

✓

Les constantes, tel que la constante ID du périphérique, sont connectées en dures et ne comptent alors pas dans le compte de registres.

Simulation

Différentes phases de simulations ont été exécutées. Ces dernières peuvent être représentées avec les séquences sauvegardées, à l'aide de la console.

Voici leur test respectif:

0. Test des lectures des IDs, ainsi que l'écriture de la constante de debug

- Test de deux lectures consécutives, en forçant les signaux nécessaires

1. Test des lectures des entrées utilisateurs: Bouttons et interrupteurs
2. Test de l'écriture et relecture des LEDs de la DE1
3. Test de l'écriture, relecture des LEDs sur la MAX10, ainsi que le maintien de *lp36_we* soit active pendant 1[us]

Pour ne pas surcharger le rapport, l'analyse des chronogrammes, avec effet sur la console est annexé à la **fin**.

✓

Remarques code vhdl avl_user_interface:
- signal avl_readdatavalid_s généré dans le même process que avl_readdata_s, pas d'interet de le séparer.
- Attention!!! Process fut_state_dec est mélangé entre process synchrone et asynchone, c'est une pratique fausse et à ne pas faire -> séparer en 2 process.

Programme C

Pour le programme, la base a été reprise du dernier laboratoire.

En prémice du programme principale, un programme de test a été écrit (fichier: test_program.c.bak). Ceci dans le but de tester les accès aux différents registres, ainsi que leur utilisation propre (gestion luminosité des LEDs, lecture des interrupteurs, . . .).

Un module **interface** implemente les fonctions de lecture et d'écriture de l'interface, afin de satisfaire les contraintes décrites par la donnéedu labo. .

Le module se contente du minimum pour cette interface, mais elle est facilement modulable.

Ajouter des fonctions dédiées pour les accès aux leds, switchs, keys. Cela serra plus lisible dans la partie application et plus simple à utiliser.

Selon la compréhension du cahier des charges, voici quelques clarifications quant au comportement du programme:

- Pression sur KEY3 -> extinction des LEDs

  Pour satisfaire ce point, il semblait plus naturel de garder les LEDs éteintes, tant que le bouton était maintenu pressé. À noter que durant la présentation, un pseudo PWM entre allumage et extinction de LEDs se créait sur la zone active de la MAX10.

- Pression sur KEY2 -> Shift d'une taille de byte sur la gauche

  Le mode décale la valeur des switchs de 8bits, tout en gardant les états des LEDs précédents. De plus, les autres LEDs que le groupe de 8bits sur lesquels les switchs sont reportés sont éteintes si KEY3 est pressé.

Explications sur les tests des fonctions+programme réalisés sur la carte? avec les résultats obtenus?

Conclusion

Pour conclure, le laboratoire a été réalisé avec succès! Le cahier des charges est rempli et tant le *design*, que le programme, sont tout 2 facilement modifiables si besoin.

Après rédaction du rapport, 1 mitigation possibles serait de:

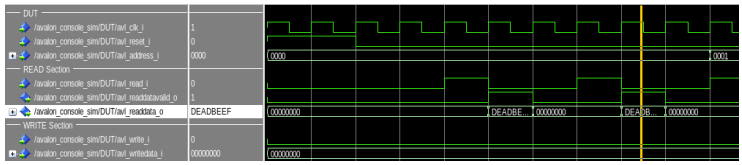
- Grouper les états responsables du maintien de *lp36_we* en un seul, de sorte à compter avec une résolution de 1 le nombre de cycle pour atteindre la limite.

Annexe(s)

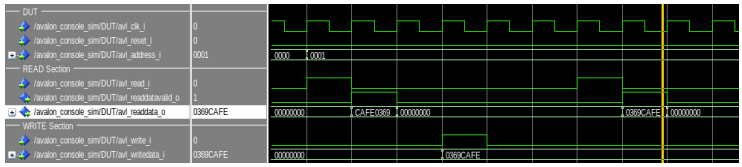
- Simulation: Chronogrammes et Consoles
  - IDs contrôle
  - Lecture des entrées utilisateurs
  - Écriture/Relecture des LEDs sur DE1SoC
  - Maintien du write enable d'écriture sur MAX10
- Mesures write enable
  - Write enable de 1.1us - 1
  - Write enable de 1.1us - 2
  - Write enable de 1us
  - Indication sur la mesure

Simulation: Chronogrammes et Consoles

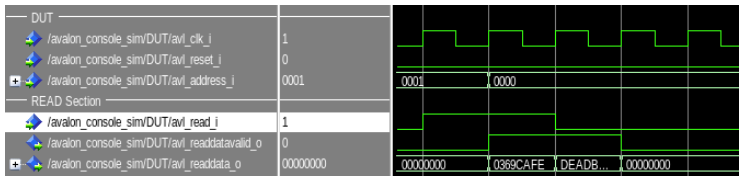
IDs contrôle



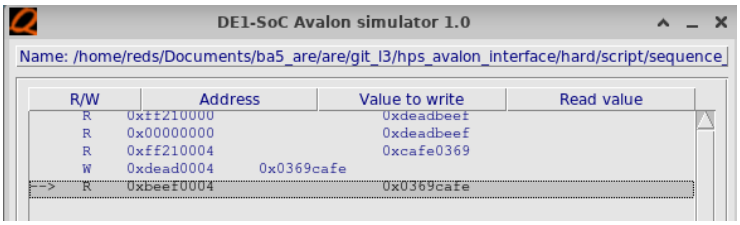
Ici, il est constaté que la valeur lue de l'ID (côté étudiant) est faite correctement et que le bit de *read_datavalid* est active, un coup d'horloge après *read_i*.



Ce chronogramme permet de voir que la lecture, la modification et la relecture de la constante de *debug* fonctionne.

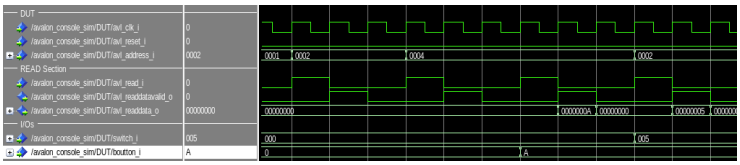


En forçant les valeurs de *read_i* et de *avl_address_i*, le dernier chronogramme montre que la lecture consécutive de deux registres différents fonctionnent.



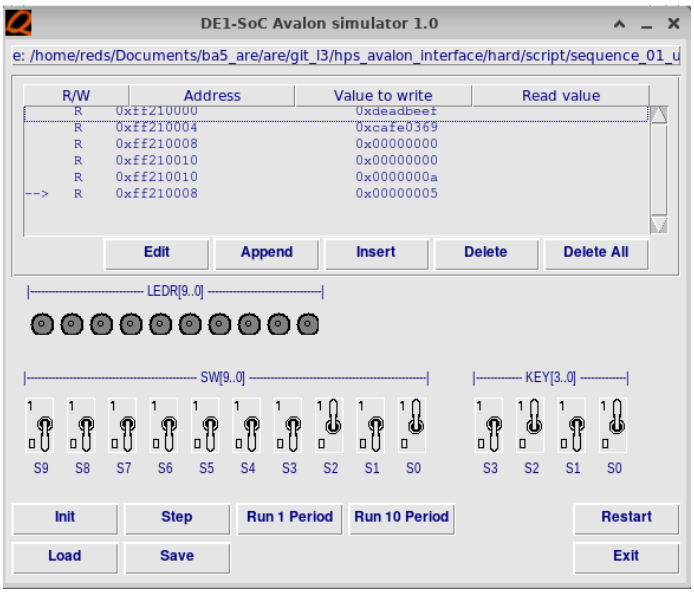
Sur la console, ces instructions permettent de confirmer que seul les bits 15 à 2 sont utilisés pour adresser l'interface.

Lecture des entrées utilisateurs

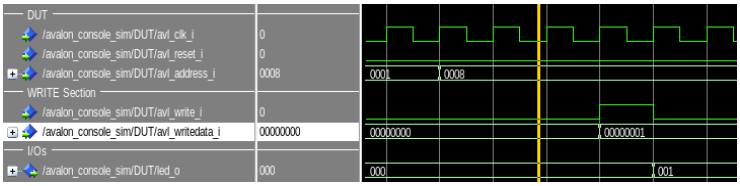


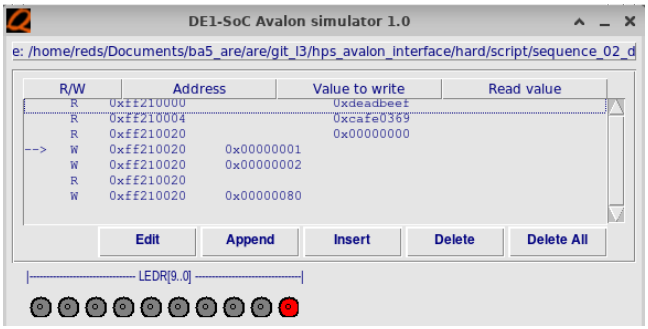
Ce chronogramme permet de valider que les valeurs des interrupteurs et des boutons sont bien reportées, lors de leur lecture.

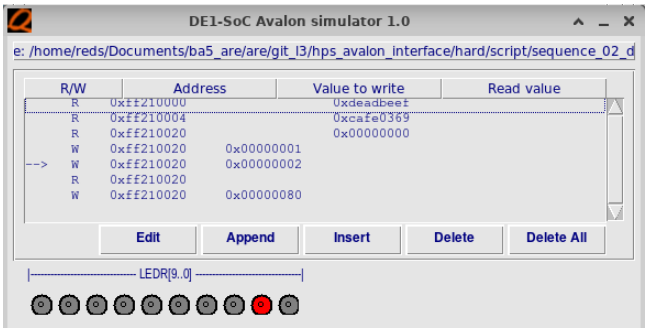
Ce qui est validé, également avec la console ci-dessous:

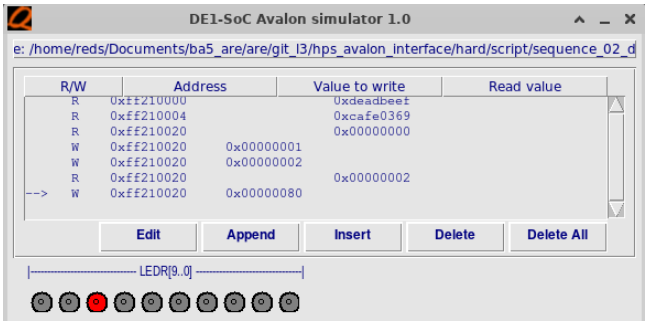


Écriture/Relecture des LEDs sur DE1SoC





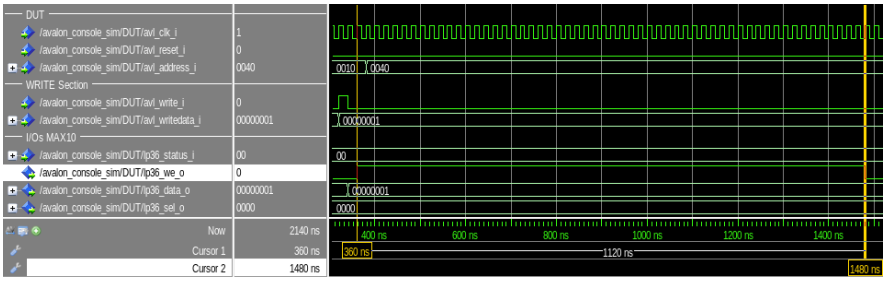




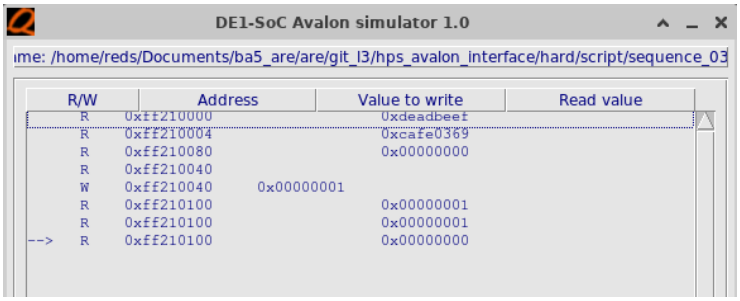
✓

Le chronogramme et les consoles vérifient le fonctionnement souhaité, transmis par les chronogrammes du bus Avalon (dans le dossier /doc, mis à disposition).

Maintien du *write enable* d'écriture sur MAX10



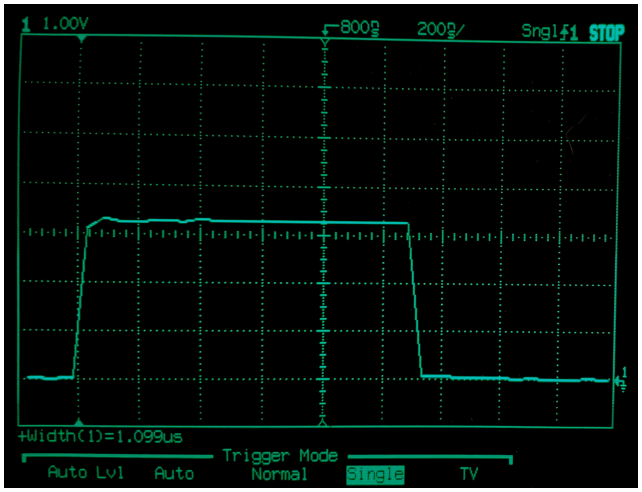
Le bit *write enable* est simulé correctement à 1'120[ns], comme calculé au titre **Machine d'état pour l'écriture de la MAX10**.



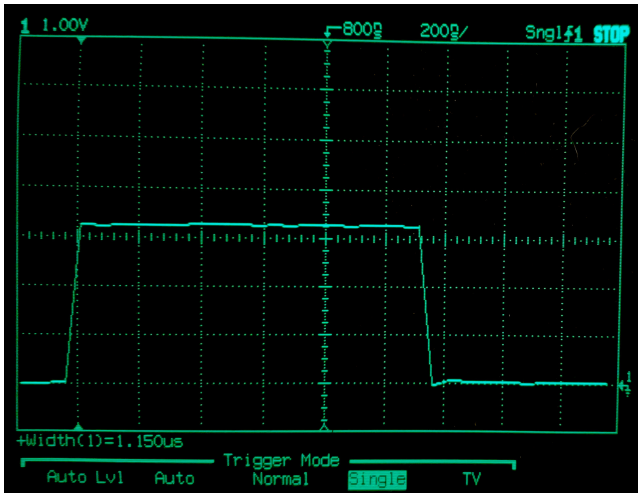
La console permet de voir que passer le délai de 1.12[us], le registre du *write enable* (ou appelé *busy* dans la solution adoptée) redescend bien à 0.

Mesures *write enable*

***Write enable* de 1.1us - 1**



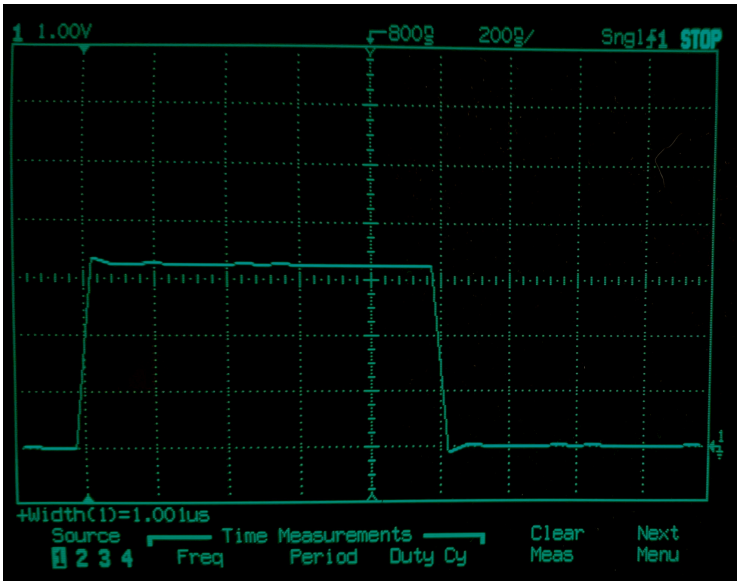
***Write enable* de 1.1us - 2**



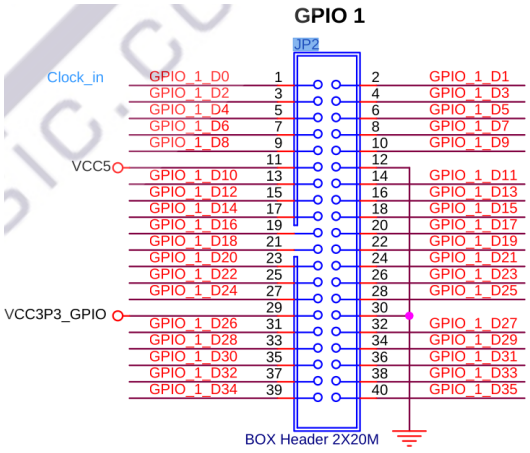
Comme mentionné en titre **Machine d'état pour l'écriture de la MAX10**, on voit que sur 1.12[us], on retrouve une légère fluctuation.

En prenant la mesure initiale (voir page suivante), il serait alors possible de descendre en dessous de la [us].

***Write enable* de 1us**



Indication sur la mesure



Pour effectuer les mesures, la sonde d'un oscilloscope a été planté en pin n°2, avec un point de référence en pin n°12, à la masse.