

Laboratoire 5: Conception d'une interface fiable

Département: **TIC**

Unité d'enseignement: **ARE**

Auteur(s):

- **CECCHET Costantino**
- **JALUBE Miguel**

Professeur:

- **MESSERLI Etienne**

Assistant:

- **CONVERS Anthony**

Date:

- **Décembre 2023**

Sommaire

- Introduction
- Conception de l'interface
 - Plan d'adressage
 - Réalisation du circuit
 - * Canal d'écriture
 - * Canal de lecture
 - * Machine d'état pour la lecture fiable
 - Synthèse
- Simulation
- Programme C
- Conclusion
- Annexe(s)

Introduction

Pour ce laboratoire, il est demandé de réaliser une interface fiable, 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:

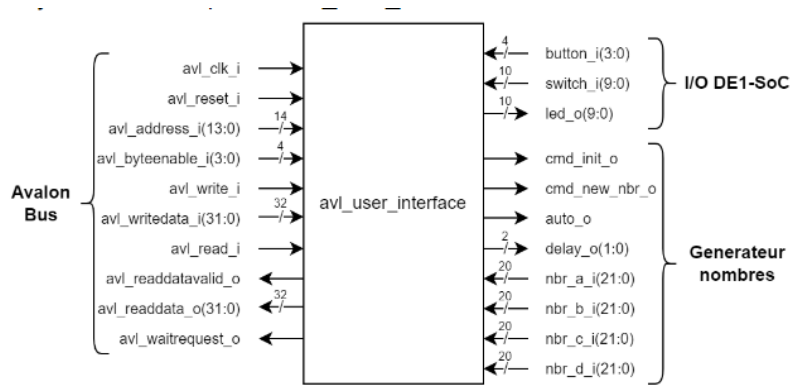


Figure 1: bloc_schem.png

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 accéder 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.**

Ensuite, le manuel de référence technique du processeur nous indique que le bus AXI lightweight 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 aiaint comme adresse de base 0xFF200000 a ensuite été définit comme suit:

Address (CPU Side) [16..0]	Address (Itf side) [15..2]	Definition	R/W
0x1_0000	0x0000	Constant ID (0xDEADBEEF)	R
0x1_0004	0x0001	IN: Keys	R
0x1_0008	0x0003	IN: Switches	R
0x1_000C	0x0003	OUT/IN: LEDs	R/W
0x1_0010	0x0004	OUT/IN: Status/Config. ¹	R/W
0x1_0014	0x0005	OUT/IN: Mode/Speed	R/W
0x1_0018	0x0006	<i>unused</i>	-
0x1_001C	0x0007	OUT: Photo	W
0x1_0020	0x0008	IN: nbr_a	R
0x1_0024	0x0009	IN: nbr_b	R
0x1_0028	0x0010	IN: nbr_c	R
0x1_002C	0x0011	IN: nbr_d	R
0x1_0030	0x0012	<i>reserved</i>	-
0x1_003C	0x1000	<i>not used</i>	-

¹*Status/Config.*: en suivant le plan d'adressage fourni par les responsables de cours, les bits de status sont *read only* et les bits de configuration sont *write only*.

Justification du plan d'adressage:

- La première constante permet d'identifier l'interface à concevoir, conformément à la donnée du laboratoire
- Pour les offset 0x18 et 0x1C, il a été décidé de n'utiliser que l'offset 0x1C et son bit 0 afin de gérer la prise de photo des registres 0x20 à 0x2C. Nous avons décidé de ne pas utiliser l'offset 0x18, car la prise de photo et son acquittement peuvent être gérés par le bit 0 de l'offset 0x1C.

Réalisation du circuit

La circuiterie a été coupée en 3 parties majeures; **lecture**, **écriture** et une machine d'état pour la lecture fiable.

TODO : mettre le schéma de l'interface

interface__split

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.

TODO : mettre le schéma du DEMUX READ

Le schéma résultant se présente ainsi:

demux_write

Canal de lecture

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

En voici son schéma:

mux_read

Lors de la lecture d'une donnée, un décalage d'une période d'horloge est effectuée pour obtenir le signal *avl_read_datavalid* à partir du signal *avl_read_i*.

Machine d'état pour la lecture fiable

Pour la lecture fiable, une machine d'état a été implémentée, afin de gérer un signal de prise de photo, ainsi que son acquittement.

Voici le diagramme de la machine d'état.

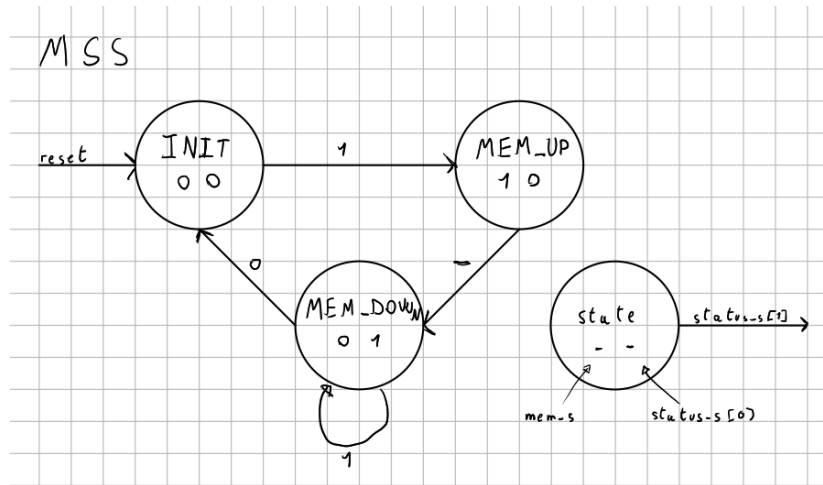


Figure 2: MSS

Cette machine a été conçue sur la base de ces diagrammes de flux:

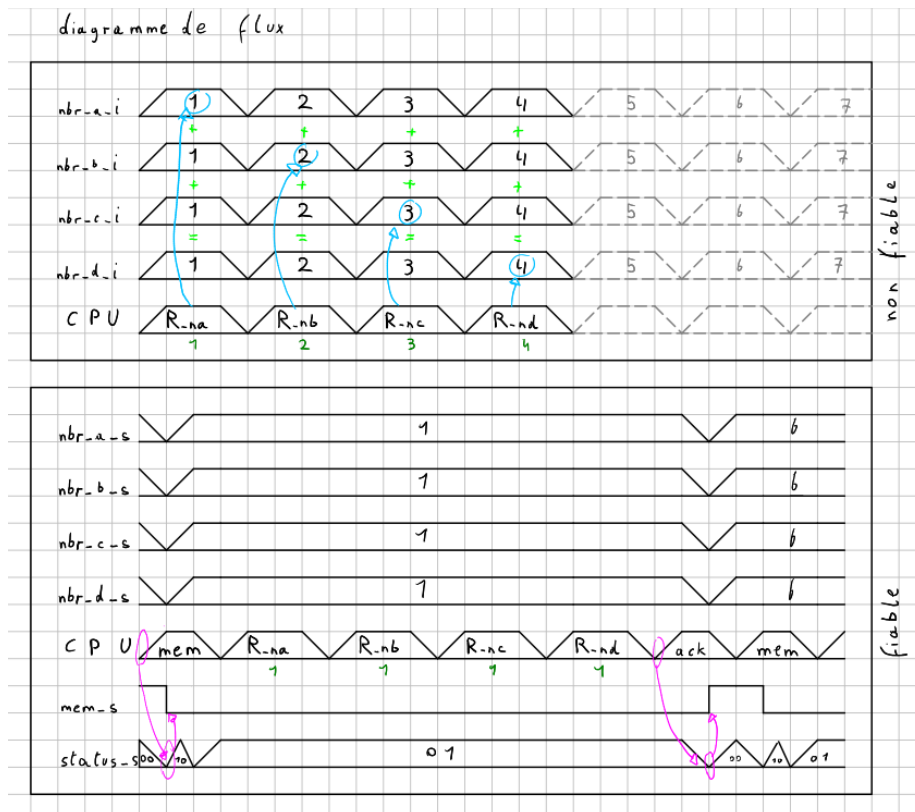


Figure 3: Diagramme de flux

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:

TODO : verifier les Nobmre de registres

Compilation Hierarchy Node	Dedicated Logic Registers
DE1_SoC_top	643 (0)
[avl_user_interface:avl_user_interface_inst]	121 (121)

Figure 4: dedicated_registers

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

	Size (bits)		Size (bits)
Données des LEDs sur DE1SoC	10	Status	2
Mode + Speed	3	New + Init	2
Nombres	22*4	readdatavalid	1
readdata	1	constante nombres	2*4
Nbr_save	1	mem	1
Machine d'état	3		

Ce qui amène alors à:

$$DedicatedLogicRegisters = 10 + 1 + 2 + 3 + 2 + 22 * 4 + 1 + 1 + 2 * 4 + 1 + 3 = 121$$

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
1. Test des lectures des entrées utilisateurs: Bouttons et interrupteurs
2. Test de l'écriture et relecture des LEDs de la DE1

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

Programme C

Pour le programme, la base a été reprise du dernier laboratoire. Cependant, le programme a été modifié pour satisfaire les contraintes du cahier des charges.

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

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

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

- Lors de l'écriture sur les registre de initialisation et de reset pour le generateur de nombres, le programme cree une pulse sur ce dernier. Ceci à été fait pour ne pas avoir à gérer cette pulse grâce a une machine d'état dans le vhdl.
- Lors de la lecture des nombres en mode fiable, deux solutions ont été discutée avec le professeur:
 - La première consiste à demander une prise de photo, et laisser a l'utilisateur la mission de lire les nombres avant de relancer une prise de photo. Cette solution est simple à implémenter.
 - La deuxième consiste à demander une prise de photo et d'attendre un acquittement de la part de l'utilisateur avant de pouvoir relancer une prise de photo. Cette solution est plus complexe à implémenter.

Nous avons implémenté cette solution, car elle est plus proche du cahier des charges.

Test

TODO : mettre les photos des tests

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.

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).