COMPTE-RENDU SYSTÈMES SUR PUCES

« SRAM »

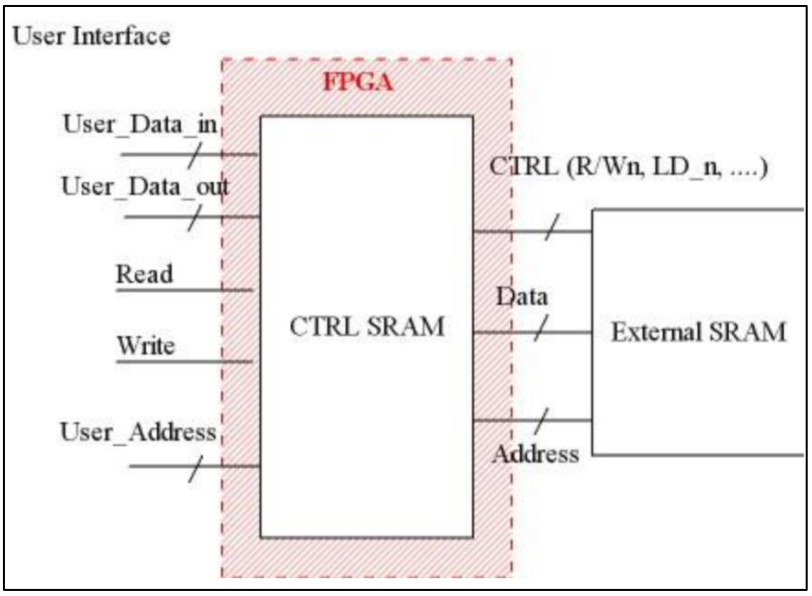
Réalisé par Sean Marotta & Pierre-Élie Morrot

Enseignant : Julien Dubois
Année : 2020 – 2021

➢ INTRODUCTION :

Dans ce TP, nous allons créer un contrôleur SRAM, un FPGA permettant de simplifier l'utilisation d'une SRAM (« Static Random Access Memory »). L'utilisateur aura juste besoin de donner une adresse et des données à envoyer s'il veut écrire ou bien juste lire à cette adresse.

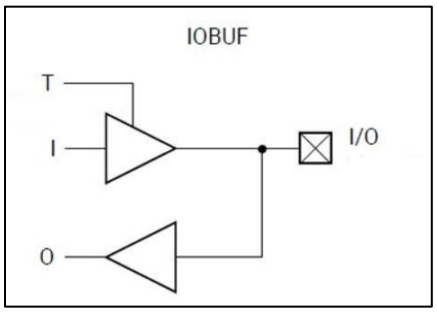
Voici un schéma représentant globalement la configuration des composants des bases :



Nous pouvons voir qu'il n'y a qu'un bus de données entre notre contrôleur et la SRAM. Cela signifie que ce bus est utilisé dans un sens lorsqu'on écrit dans la SRAM et dans un autre sens si l'on veut lire, il est donc bidirectionnel. On devra donc utiliser des composants appelés buffers à 3 états (IOBUF tri-state en anglais).

➢ IOBUF TRI-STATE :

Voici le schéma du composant :



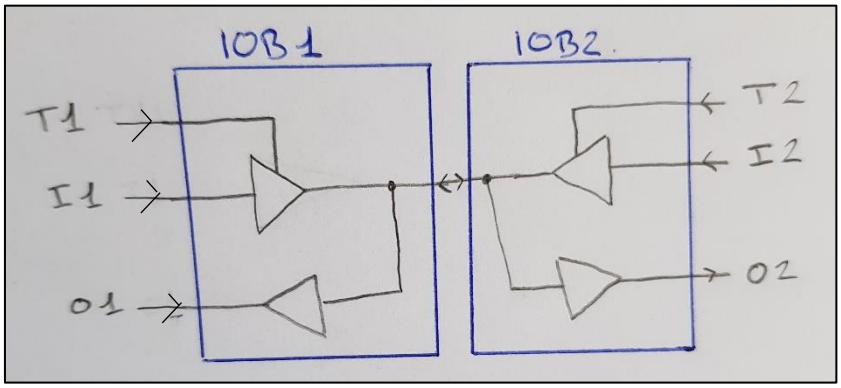
Ce composant IOBUF est composé de ports logiques (0 ou 1) qui sont :

- **Une entrée I :** on envoie nos données dans ce canal.
- **Une sortie O :** on récupère les données reçues par l'IOBUF.

- **Un canal entrée-sortie IO :** canal bidirectionnel permettant d'envoyer les données venant de I, ou de récupérer les données reçues par un composant externe relié à ce canal.
- **Et d'une entrée trigger T :** détermine l'état de l'IOBUF c'est-à-dire que lorsqu'il est à 0, l'IOBUF se met en mode WRITE, et lorsqu'il est à 1, l'IOBUF est en mode READ.
    - Si T = 0 alors IO <= I
    - Si T = 1 alors O <= IO

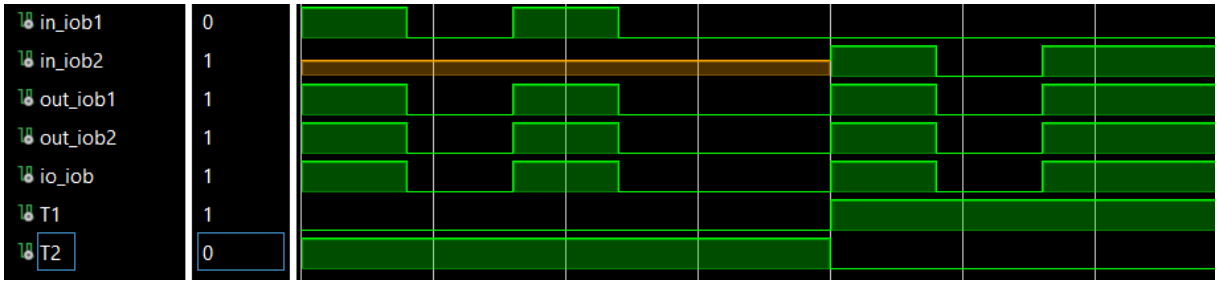
Comme on l'a dit, l'IOBUF travaille que sur 1 seul bit, or ici nos données sont sur 36 bits. Il faudra donc générer 36 IOBUFs pour bien transmettre toutes les données.

Pour bien comprendre le fonctionnement de ce composant, nous avons créer un petit Test Bench mettant en scène deux IOBUF branchés de cette manière :



Voici ce que l'on entre dans les différents signaux, et en dessous le résultat de la simulation :

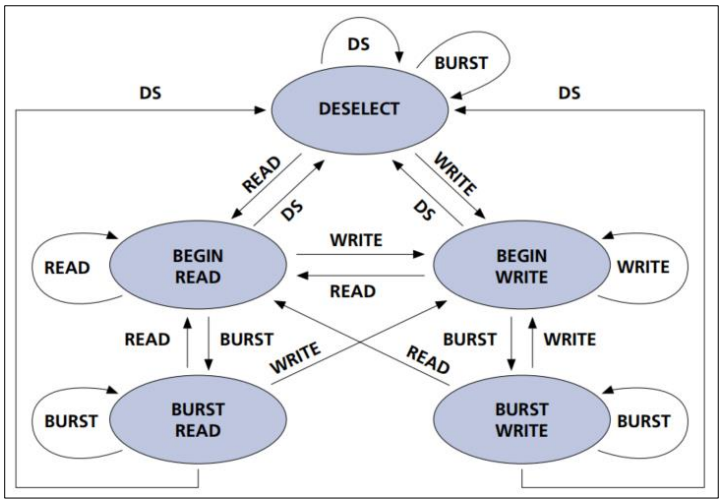
```
-- IOB1 en mode ecriture et IOB2 en mode lecture
T1 <= '0';
T2 <= '1';
in_iob1 <= '1';
wait for 40 ns;
in_iob1 <= '0';
wait for 40 ns;
in_iob1 <= '1';
wait for 40 ns;
in_iob1 <= '0';
wait for 80 ns;

 -- IOB1 en mode lecture et IOB2 en mode ecriture
T1 <= '1';
T2 <= '0';
in_iob2 <= '1';
wait for 40 ns;
in_iob2 <= '0';
wait for 40 ns;
in_iob2 <= '1';
wait for 80 ns;
```



Nous utiliserons donc ce composant à l'intérieur de notre contrôleur SRAM qui permettra entre autres d'échanger les données avec la SRAM.

➢ ÉTUDE DE LA SRAM :

Dans cette partie, nous allons nous intéresser sur la documentation de la SRAM fournie par le professeur. Ainsi, j'ai pu commencer à concevoir une machine à état en m'inspirant du schéma présent dans la documentation **p.11** et à l'aide du tableau de vérité **p.12**.



OPERATION	ADDRESS USED	CE#	CE2#	CE2	ZZ	ADV/ LD#	R/W#	BWx	OE#	CKE#	CLK	DQ	NOTES
DESELECT CYCLE	None	H	X	X	L	L	X	X	X	L	L→H	High-Z	
DESELECT Cycle	None	X	H	X	L	L	X	X	X	L	L→H	High-Z	
DESELECT Cycle	None	X	X	L	L	L	X	X	X	L	L→H	High-Z	
CONTINUE DESELECT Cycle	None	X	X	X	L	H	X	X	X	L	L→H	High-Z	1
READ Cycle (Begin Burst)	External	L	L	H	L	L	H	X	L	L	L→H	Q	
READ Cycle (Continue Burst)	Next	X	X	X	L	H	X	X	L	L	L→H	Q	1, 11
NOP/DUMMY READ (Begin Burst)	External	L	L	H	L	L	H	X	H	L	L→H	High-Z	2
DUMMY READ (Continue Burst)	Next	X	X	X	L	H	X	X	H	L	L→H	High-Z	1, 2, 11
WRITE Cycle (Begin Burst)	External	L	L	H	L	L	L	L	X	L	L→H	D	3
WRITE Cycle (Continue Burst)	Next	X	X	X	L	H	X	L	X	L	L→H	D	1, 3, 11

Les lignes jaunes représentent les états possibles et le surlignés rouges représente les bits de contrôles (permettant de passer d'un état à un autre).

Remarque : Pour la gestion du mode BURST, nous utiliserons le bit ADV/LD# (en vert). Cela introduit deux états en plus dans notre machine à états surligné en gris :

- READ Cycle (Continue Burst)
- WRITE Cycle (continue Burst).

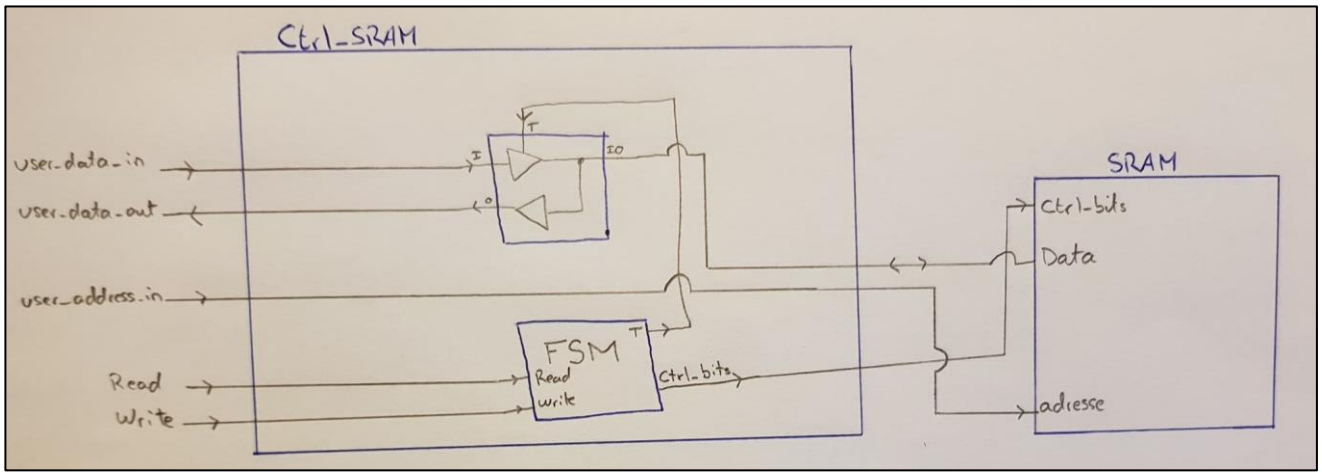
Ainsi, nous avons nos états nécessaires mais il manque encore des informations. Pour cela, nous commençons à travailler sur notre FPGA contrôleur SRAM.

➢ CONTRÔLEUR SRAM :

Notre contrôleur sera intermédiaire entre l'utilisateur et la SRAM. Comme nous l'avons dit, l'utilisateur devra fournir des informations qui permettra au FPGA de contrôler la SRAM. Voici l'ensemble des canaux nécessaires de notre contrôleur :

- 4 entrées :
  - Write : sur un seul bit, à 1 pour activer le mode écriture.
  - Read : sur un seul bit, à 1 pour activer le mode lecture.
  - User_data_in : sur 36 bits, ce sont tout simplement les données que l'utilisateur veut entrer dans la SRAM (pour le mode écriture).
  - User_address_in : sur 19 bits, l'adresse où l'utilisateur veut stocker ces données.
- 2 sorties :
  - User_data_out : sur 36 bits, les données récoltées de la SRAM (en mode lecture).
  - Ctrl_bit : sur 2 bits pour le moment, ce sont les bits de contrôle que le FPGA envoie à la SRAM (CE2, R/W#)
- 1 inout :
  - Data_inout : sur 36 bits, le canal directionnel géré par les IOBUFs.

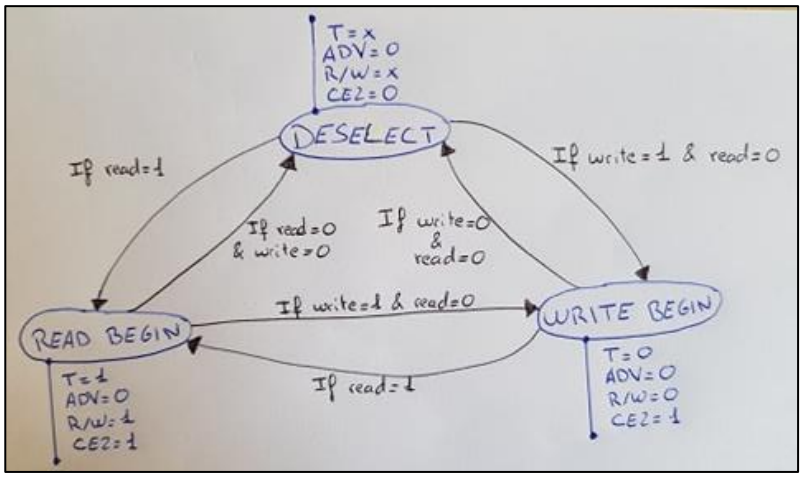
Voici le schéma de câblages :



J'ai représenté un seul IOBUF pour des raisons de lisibilité, mais en réalité il y en a 36 en parallèle déclarés de cette manière :

```
generate_buffer : for i in 0 to 35 generate
    io_buffer : IOBUF PORT MAP (address_user_in(i),data_inout(i),Trigger,address_user_out(i));
END GENERATE;
```

De plus, le bloc FSM (Finite State Machine) représente la machine à état qui ressemble à ceci :



Nous parlerons du ADV plus tard, il permettra la gestion du BURST.

Nous avons qu'une seule condition pour accéder à la lecture car il est prioritaire sur l'écriture.

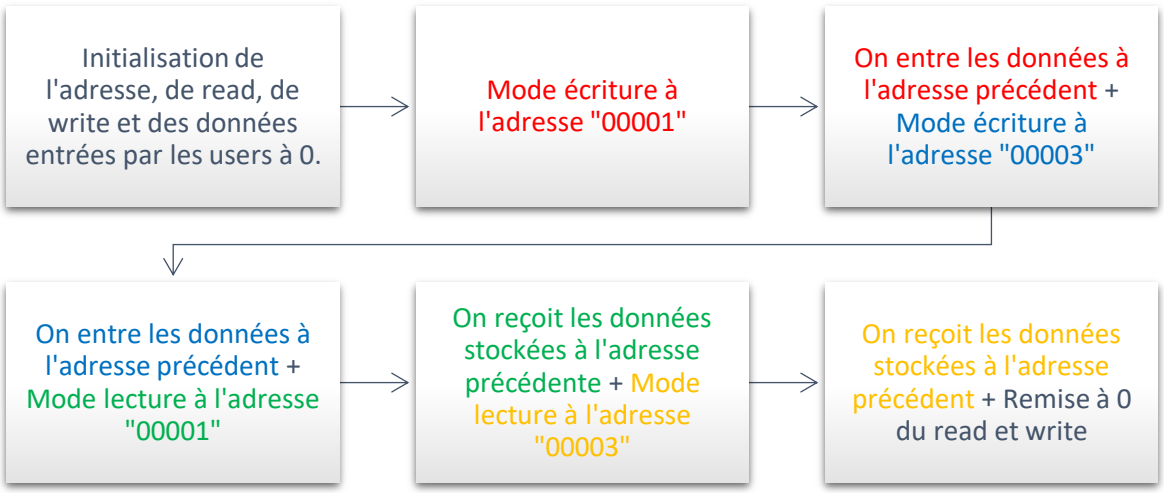
Dans le code, nous décrirons les états de cette manière :

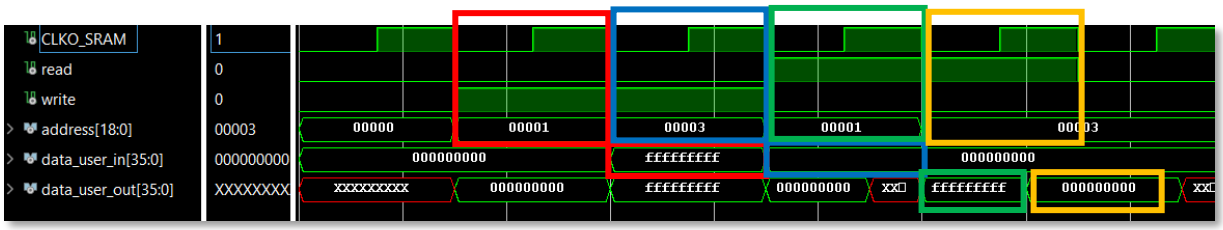
```
------------------------------------ ORDRE : "ADV ; R/W ; CE2 ; Trigger"
constant S_DESELECT : std_logic_vector(3 downto 0) := "0000";
constant S_READ     : std_logic_vector(3 downto 0) := "0111";
constant S_WRITE    : std_logic_vector(3 downto 0) := "0010";
```

Maintenant que le Block Design en VHDL du contrôleur SRAM est établi, nous pouvons le tester grâce à un Test Bench associé.

➢ TEST BENCH CONTRÔLEUR SRAM :

Dans le Test Bench, nous allons envoyer des données au contrôleur avec une adresse associée, et voir ensuite s'il est bien stocké dans la SRAM grâce au mode Lecture. Pour plus de détails, voici les différentes étapes que nous réaliserons pour bien le tester :





On peut voir que notre contrôleur fonctionne très bien. Désormais, nous pouvons passer au BURST afin d'approfondir nos connaissances dans le fonctionnement d'une SRAM.

- **CONTRÔLEUR SRAM AVEC LE MODE BURST :**