

Bordeaux INP – ENSEIRB-MATMECA

Filière Systèmes Électroniques Embarqués

Conception de systèmes numériques

PR209 Projet expérimental de conception de circuit
numérique

Jeu de dames

CHOLLET Benjamin- CLAIN Adrien

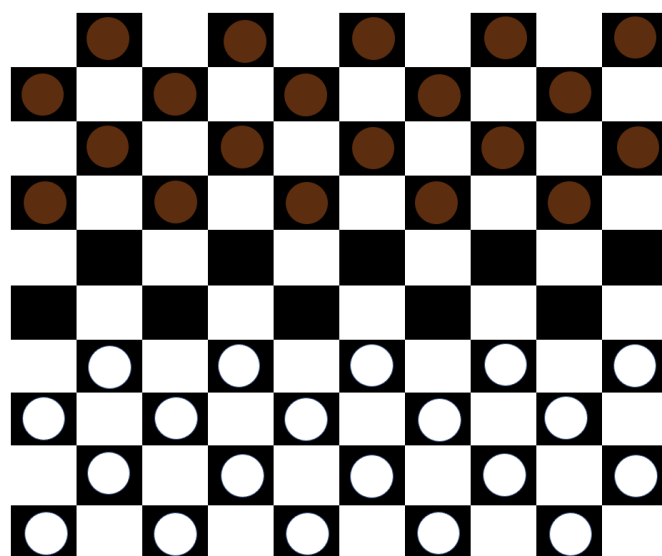


Table des matières

I.	Introduction.....	3
II.	Le jeu de dame	3
1.	Présentation du jeu.....	3
2.	Règles du jeu.....	4
III.	Principe de base du projet.....	4
1.	Affichage.....	4
2.	Mise en place de mémoire.....	5
IV.	Architecture du projet.....	6
1.	Acquisition.....	7
2.	Traitement.....	11
3.	Affichage.....	14
a.	Affichage du jeu	14
b.	Affichage du menu	17
c.	Remise à zéro de l’affichage.....	18
4.	Mémoire	19
5.	Contrôle.....	20
V.	Utilisation des ressources.....	23
VI.	Amélioration du projet	24
VII.	Conclusion.....	24
VIII.	Annexes	25
1.	Table de correspondance.....	25
2.	Menu du jeu	26
3.	Affichage du plateau sans remise à zéro de l’écran	26
4.	Affichage du plateau de jeu.....	26

I. Introduction

Dans le cadre de la matière « Projet expérimental de conception de circuit numérique », nous cherchons à mettre en place une description matérielle d'un célèbre jeu de réflexion, le jeu de dames, dont les règles de fonctionnement sont données dans le paragraphe suivant.

Nous avons à notre disposition une carte Nexys A7, qui embarque une cible FPGA de Xilinx, dont l'aperçu est disponible à la Figure 1.

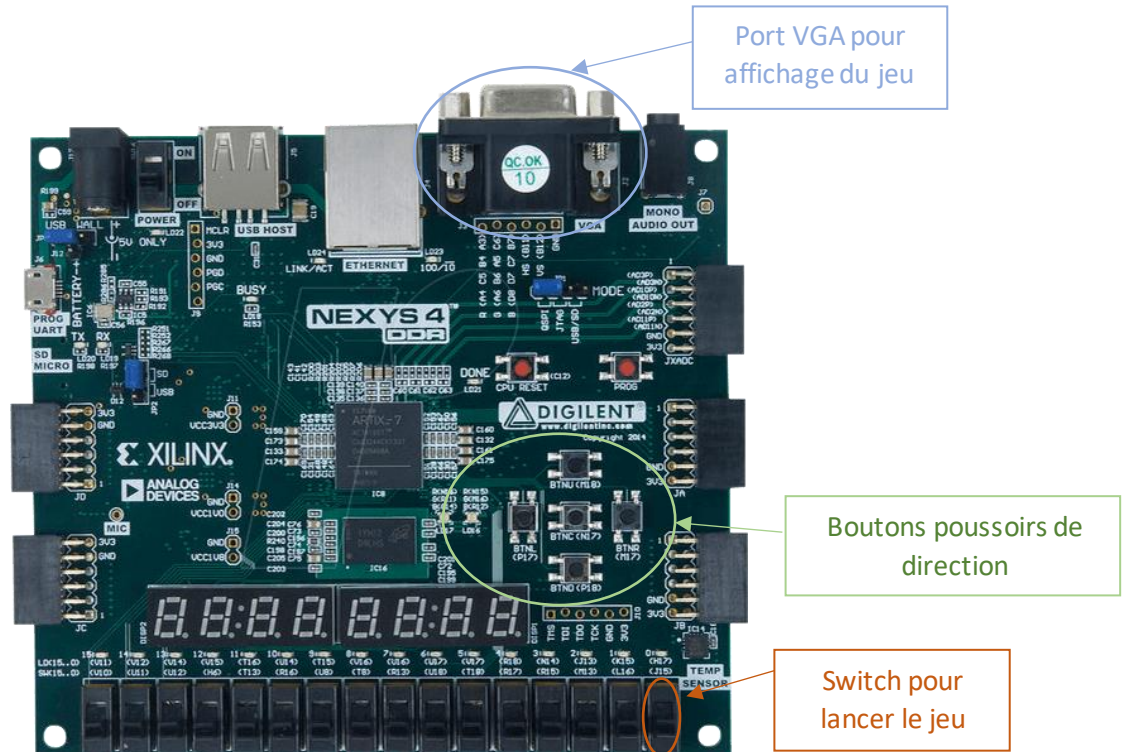


Figure 1 : Carte Nexys A7

La carte dispose pour cela d'un large panel d'accessoires (port USB et micro USB, port jack, interrupteurs, ...).

II. Le jeu de dame

1. Présentation du jeu

Le jeu de dames est un jeu de société combinatoire abstrait qui permet l'affrontement de deux joueurs, en capturant ou immobilisant toutes les pièces de l'adversaire.

Le jeu nécessite un damier, un plateau "classique" de 10 par 10 cases, style échec, et 40 pions, dont 20 de couleur blanche, et les 20 autres de couleur noire.

2. Règles du jeu

Les pions se placent uniquement sur les cases foncées du damier, et de la même manière que les échecs, les deux couleurs se font face, et c'est le joueur ayant les pions blancs qui lance la partie.

Le but de ce jeu est de capturer ou d'immobiliser tous les pions de l'adversaire. Les pions sont donc capables de se déplacer en diagonale, d'une case à la fois, et uniquement vers l'avant.

Lorsqu'une case voisine (en diagonale) est prise par un pion adverse, et que la case derrière est libre, la capture de ce pion est obligatoire, et peut s'effectuer vers l'avant mais aussi vers l'arrière. Il est également possible d'effectuer une « raffle », ou une capture de plusieurs pions en un seul coup.

Lorsqu'un pion atteint la dernière rangée, il se transforme en dame, ce qui lui octroie des compétences supplémentaires. Contrairement au pion, la dame peut se déplacer d'autant de cases qu'elle le souhaite, toujours en diagonale, du moment qu'il n'y a aucun pion sur sa route.

III. Principe de base du projet

Avant de se lancer dans l'explication de l'architecture du projet, nous allons rappeler le principe de fonctionnement du projet.

1. Affichage

Des ressources pour la gestion de l'affichage via le port VGA sont mise à disposition par Monsieur Bornat sur son site web. Nous avons ainsi décidé d'utiliser une description matérielle permettant d'afficher une image avec une résolution de 320 par 240 pixels avec un codage binaire par pixel de 12 bits afin d'avoir des couleurs précises (Figure 4).

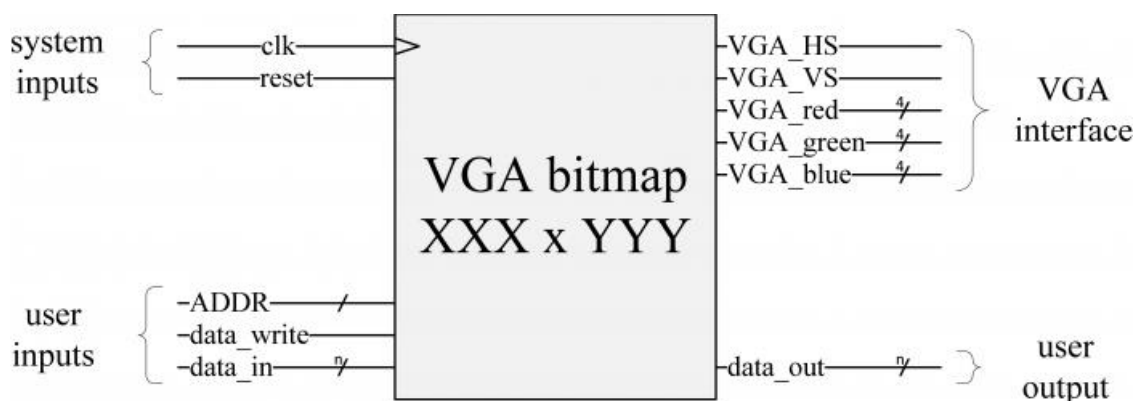


Figure 2 : Module VGA pour la gestion de l'affichage

Pour faciliter la gestion de l’affichage, nous avons décidé de diviser notre image en 100 blocs (24x24) comme montré à la figure suivante.

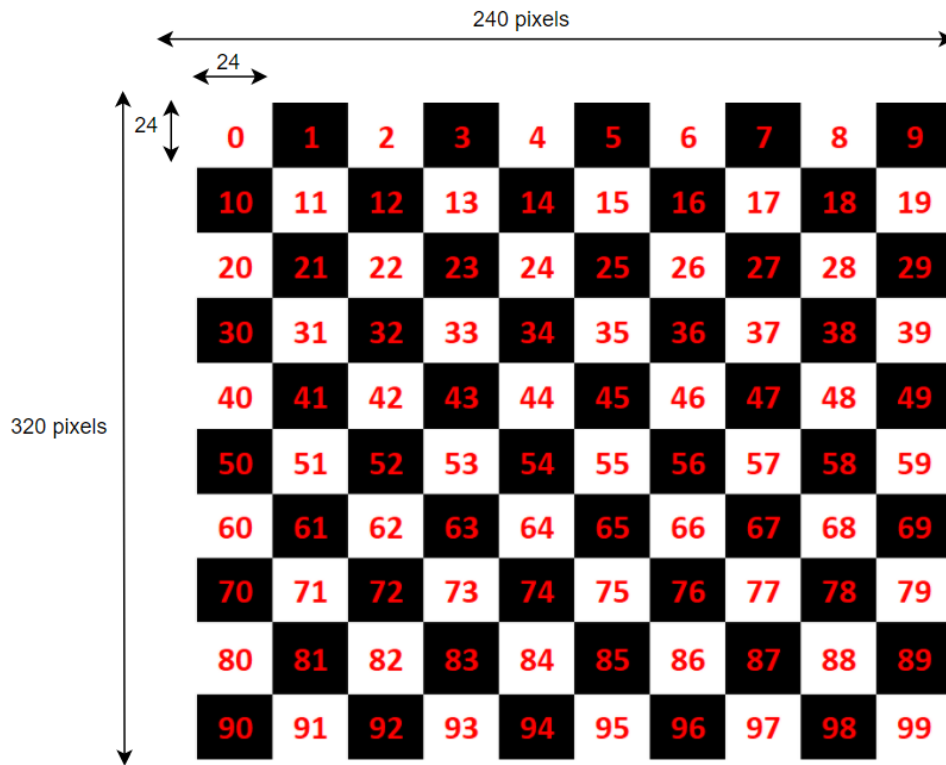


Figure 3 : Plateau de jeu divisé en 100 blocs

Chaque case contiendra ainsi une valeur en hexadécimale permettant d’identifier le type de bloc à afficher (Annexe : Table de correspondance). Nous avons identifié 12 objets devant être affichés par le jeu, et nous les avons ainsi créés grâce au logiciel Paint.net. La génération d’un code binaire de 12 bits sur le canal RGB est réalisé grâce à un code Matlab par objets.

Il est important de noter que l’orientation du plateau reste fixe durant toute la partie, et les pions blancs se placent sur la partie basse du plateau.

Par soucis d’affichage, les pions et dames noirs seront représentés par la couleur marron.

2. Mise en place de mémoire

Dans ce projet, nous utilisons un bloc mémoire de 100 cases pour la création du plateau de jeu mais aussi 12 blocs mémoires de 576 cases afin de stocker le code binaire de chaque bloc généré par Matlab dans notre bloc affichage.

IV. Architecture du projet

En suivant les règles précédentes, nous cherchons à implémenter un jeu de dames sur FPGA.

La carte utilisée nous permet d'étendre les fonctionnalités, et de rajouter un écran, par l'intermédiaire d'un port VGA. C'est sur ce dernier que nous pouvons visualiser la partie en cours.

Pour l'acquisition des données, on se servira de boutons poussoirs et d'un interrupteur, installés sur la carte, qui permettront de lancer le jeu et de sélectionner des pions.

La première étape du projet a donc été de mettre en place de manière précise l'architecture du projet afin de pouvoir se répartir les tâches en binôme tout en travaillant avec les mêmes types et nom de signaux.

L'architecture globale simplifiée est donnée selon la Figure 4 :

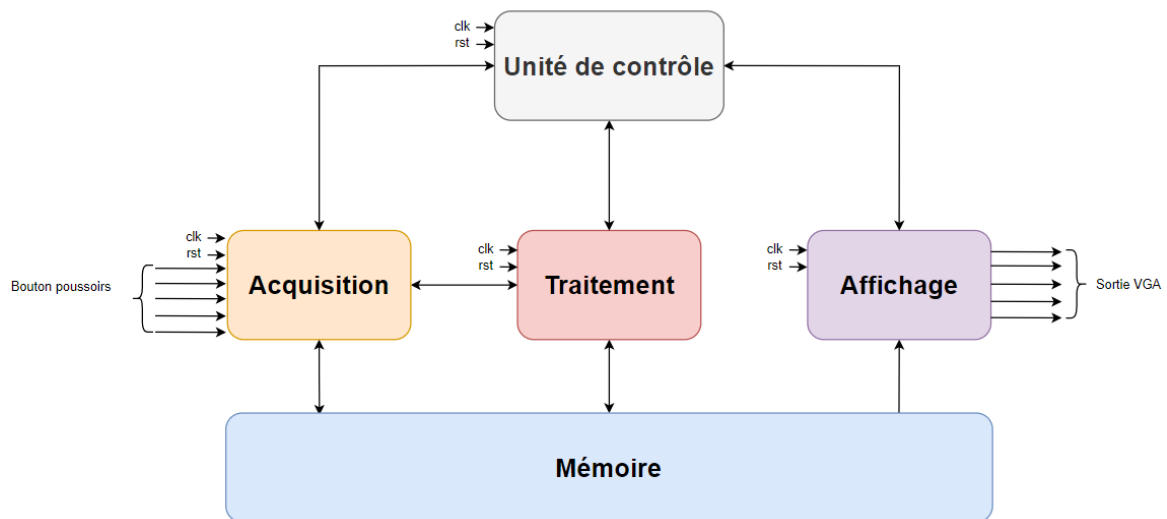


Figure 4 : Architecture globale simplifiée

Nous retrouvons ainsi trois grands blocs :

- L'acquisition : choix des actions (sélections, déplacement et capture)
- Le traitement : traiter les actions de l'utilisateur
- La mémoire : stockages de différentes données
- La partie opérative : contrôle des différents modules

Nous allons maintenant détailler les différentes descriptions matérielles réalisées en VHDL dans les parties suivantes.

1. Acquisition

Fonctionnement :

Ce bloc a pour principale mission de récupérer les informations du joueur, comme la sélection de la pièce, à l'aide des boutons directionnels. Pour cela, il agit directement sur la mémoire qui contient l'emplacement exacte de tous les pions, et modifie au passage, dès l'appuie sur un bouton, les cases affectées par le déplacement de la sélection. Un appuie sur le bouton central permet de sélectionner le pion sur lequel le joueur va interagir (déplacement ou capture d'un pion adverse).

Architecture du module :

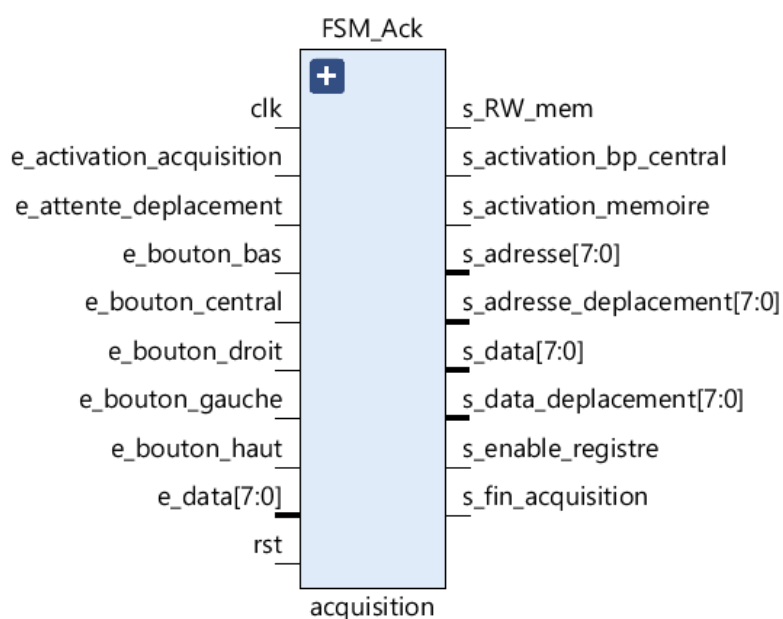


Figure 5 : Architecture du module d'acquisition

Module acquisition			
Entrées	Nombre de bits	Sorties	Nombre de bits
Clk	1	s_RW_mem	1
Rst	1	s_activation_bp_central	1
e_activation_acquisition	1	s_activation_memoire	1
e_attente_deplacement	1	s_adresse	8
e_bouton_bas	1	s_adresse_deplacement	8
e_bouton_haut	1	s_data	8
e_bouton_gauche	1	s_data_deplacement	8
e_bouton_droit	1	s_enable_registre	1
e_bouton_central	1	s_fin_acquisition	1
e_data	8		

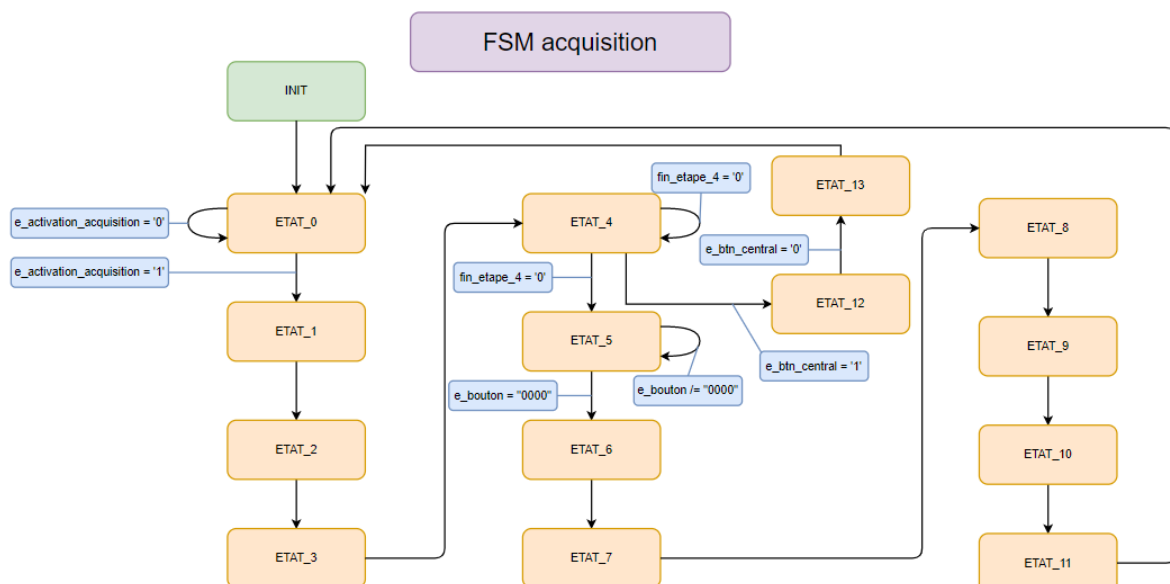
Entrées :

- **Clk** : horloge de 100 MHz
- **Rst** : remise à zéro
- **e_activation_acquisition** : signal d'activation envoyé par la FSM générale pour activer l'acquisition
- **e_attente_deplacement** : signal envoyé par le bloc traitement qui est en attente de réception de données
- **e_bouton_bas** : bouton bas
- **e_bouton_haut** : bouton haut
- **e_bouton_gauche** : bouton gauche
- **e_bouton_droit** : bouton droit
- **e_bouton_central** : bouton central
- **e_data** : données envoyées par la mémoire

Sorties :

- **s_RW_mem** : choix du mode de lecture ou d'écriture dans la mémoire
- **s_activation_bp_central** : sortie au bloc traitement et permet de prévenir si l'utilisateur appuie sur le bouton central
- **s_activation_memoire** : signal d'activation de la mémoire
- **s_adresse** : envoi de l'adresse mémoire que l'on souhaite utiliser
- **s_adresse_deplacement** : envoi de l'adresse de la case choisie par l'utilisateur si un appui sur le bouton central a été détecté
- **s_data** : envoi de la valeur que l'on souhaite écrire dans la mémoire
- **s_data_deplacement** : envoi de la valeur de la case choisie par l'utilisateur si un appui sur le bouton central a été détecté
- **s_enable_registre** : activation du registre placé avant le bloc traitement
- **s_fin_acquisition** : sortie à 1 si la FSM arrive à la dernière étape

FSM du module :



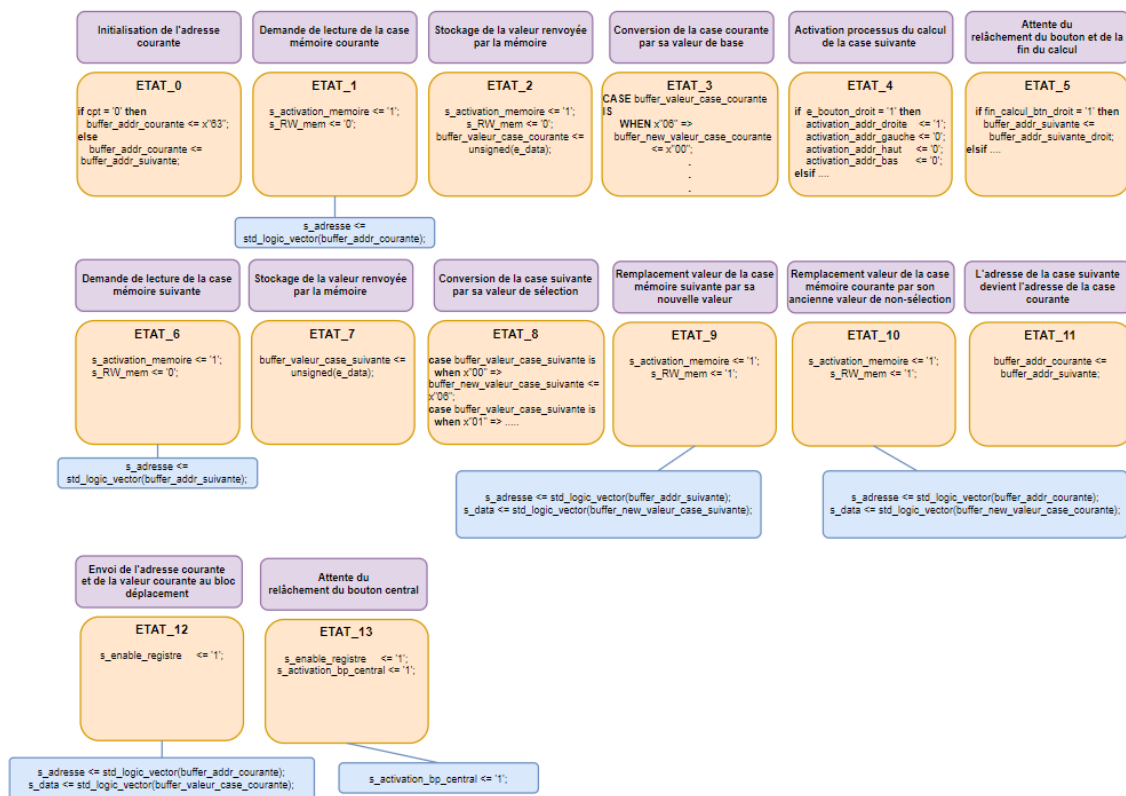


Figure 6 : FSM du module acquisition

Cette FSM va permettre d'activer des processus de mémorisation grâce à des signaux d'activation afin de venir stocker différentes adresses et valeurs du plateau de jeu.

La FSM va également, grâce à ses sorties reliées à la mémoire, remplacer les valeurs du plateau de jeux grâce à ces différents processus de calcul d'adresse et de valeurs. Ceci afin de pouvoir déplacer la sélection de la case.

Simulation :

Nous réalisons une simulation d'un appui du bouton gauche par l'utilisateur afin de déplacer le curseur de sélection de case et nous retrouvons les chronogrammes de la

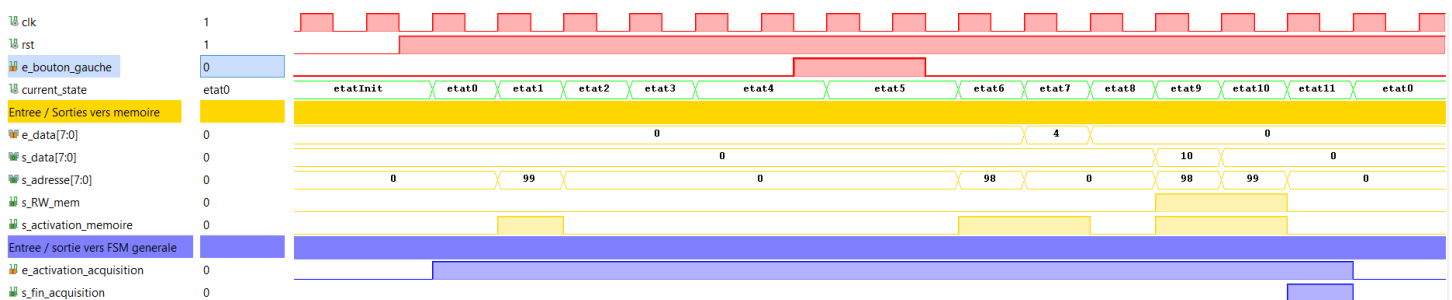


Figure 7.

Figure 7 : Simulation du bloc d'acquisition sur un appui de bouton

Si nous nous basons sur la Figure 6, nous passons bien de l'état d'initialisation à l'étape 0 de la FSM lorsque le module reçoit l'activation de la FSM générale. L'état 1 demande l'accès de la case mémoire n°99 et reçoit la donnée un front d'horloge plus tard, ici x00 car c'est une case blanche (cf. Table de correspondance).

Ensuite, nous passons bien d'étape en étape à chaque front montant de l'horloge jusqu'à l'étape 4 qui attend que l'utilisateur appuie sur un bouton poussoir. L'étape 5 attend bien le relâchement du bouton poussoir pour passer à l'étape suivante.

A l'étape 6, le bloc acquisition demande la lecture de la case mémoire suivante calculée à l'étape 4 lors de l'appui du bouton gauche. Nous avons une demande d'accès mémoire à la case n°98, ce qui est correct au vu du processus calculant un appui à gauche ($ADDR = ADDR - 1 = 99 - 1 = 98$).

A l'étape 7, le module reçoit bien une donnée provenant de la mémoire (x04) et la stocke dans une mémoire tampon (buffer).

L'étape 9, va bien écrire dans la mémoire la nouvelle valeur de la case suivante. En effet, nous remplaçons une case blanche par une case blanche sélectionnée (Figure 8)

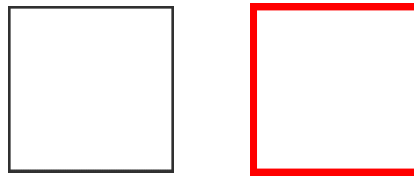


Figure 8 : Case blanche (x00) et sélection de case blanche (x0A)

L'étape 10, remet quand elle correctement la case n°99 qui était la case courante à son état initiale, soit une case blanche.

L'étape 11 va mettre à jour la case courante, ici c'est la case n°98.

Nous revenons bien à l'étape 0 quand tout est terminé.

2. Traitement

Fonctionnement :

Chaque pion est capable de se déplacer uniquement en avant et d'une case, hormis les dames. Un bloc est dédié au calcul de ce procédé, en vérifiant la possibilité de déplacement, suivant plusieurs conditions :

- Le pion ou la dame à déplacer, doit être de la couleur du joueur (ne pas déplacer un pion adverse)
- La case souhaitée doit être vide (pas de pions ou dames amis/adverses sur la case) et de couleur noire
- La case souhaitée doit être dans le voisinage de la pièce, dans sa diagonale

Si ces conditions sont réunies, le pion peut être déplacé en mémoire. Le contenu des cases en mémoire est alors inversé.

Le pion blanc peut uniquement se déplacer sur l'une des deux cases marquées par une flèche verte. Le principe est le même pour les dames, cependant, elles ne sont pas réglementées par le déplacement d'une seule case, ainsi que par le sens (avant ou arrière). Toutefois, elles sont heurtées à la présence ou non de pion/dame sur leur chemin.

Ce module est composé de trois blocs distincts ayant chacun un rôle prédéfini :

- Un bloc "décodeur", qui a pour rôle de décomposer l'entrée **e_data** en deux signaux permettant le calcul de la position
- Un bloc "déplacement", qui organise la partie opérative liée au déplacement
- Un bloc 'codeur', qui fait l'opération inverse du décodeur

Architecture du module :

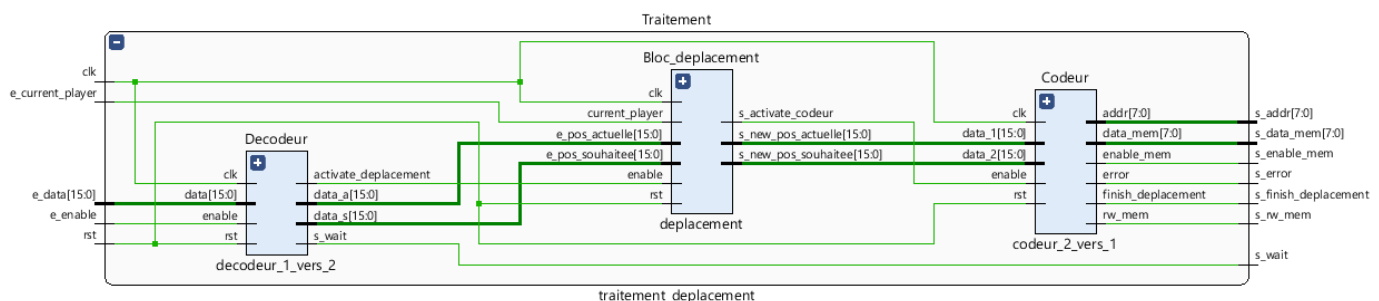


Figure 9 : Architecture du module traitement

Module traitement			
Entrées	Nombre de bits	Sorties	Nombre de bits
<i>clk</i>	1	<i>s_addr</i>	8
<i>rst</i>	1	<i>s_data_mem</i>	8
<i>e_current_player</i>	1	<i>s_rw_mem</i>	1
<i>e_enable</i>	1	<i>s_enable_mem</i>	1
<i>e_data</i>	16	<i>s_finish_deplacement</i>	1
		<i>s_error</i>	1

Entrées :

- **clk** : horloge de la carte à 100 MHz
- **rst** : remise à zéro du système
- **e_current_player** : joueur actuel en train de jouer, '0' pour le joueur blanc et '1' pour le joueur noir
- **e_enable** : signal provenant de la FSM principale autorisant la mise en fonctionnement du bloc
- **e_data** : entrée sur 16 bits qui comprend l'adresse mémoire et la donnée associée

Pour faciliter le traitement de nos blocs (un seul signal à manipuler au lieu de deux), nous avons regroupé deux signaux de 8 bits chacun en un.

Le signal **e_data** permet de connaître la donnée contenue à la case dont le numéro est représenté par l'adresse. La décomposition du signal est donnée par le tableau ci-dessous.

e_data (16 bits)	
<i>Adresse mémoire(8 bits)</i>	<i>Bloc en mémoire (8 bits)</i>

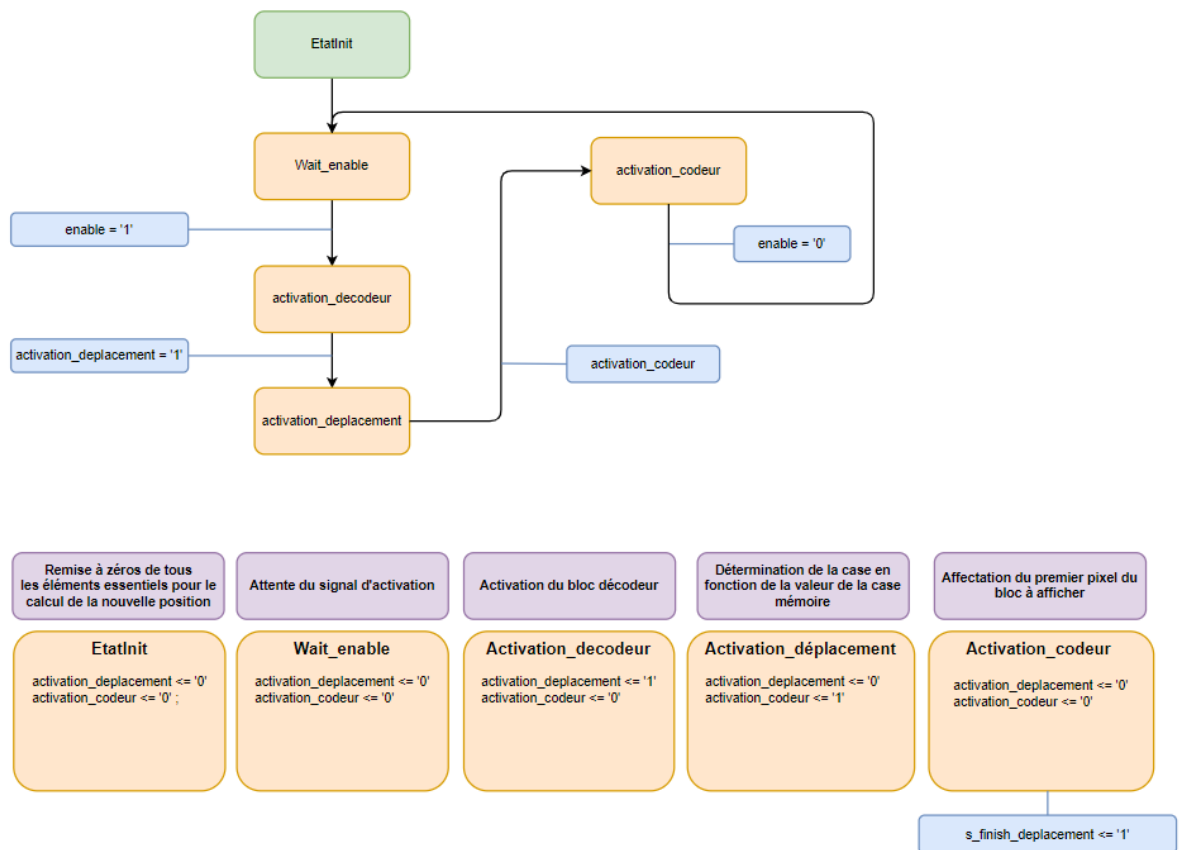
Sorties :

- **s_addr** : adresse à laquelle on va venir écrire la donnée
- **s_data_mem** : donnée à écrire dans la mémoire à l'adresse *s_addr*
- **s_rw_mem** : permet la lecture (0 logique) ou l'écriture (1) en mémoire
- **s_enable_mem** : autorisation de lecture ou écriture sur la mémoire
- **s_finish_deplacement** : indique à la FSM principale que le calcul est fini.
- **s_error** : indique à la FSM principale qu'une erreur a été détectée durant le calcul. Si *s_error* et *s_finish_deplacement* sont à '1', alors la FSM relance la procédure de déplacement

Les cas d'erreur sont les suivants :

- Le joueur a cherché à déplacer un pion/dame adverse
- La case souhaitée n'est pas vide
- Le joueur a cherché à déplacer son pion sur une case qui n'est pas sur sa diagonale

FSM du module :



Le fonctionnement de bloc est le suivant:

- Sélection de la pièce à déplacer
- Activation du bloc décodeur
- Attente de la seconde sélection
- Activation du bloc déplacement
- Vérification des conditions de déplacement
- Activation du codeur
- Si pas d'erreur, écriture en mémoire

3. Affichage

a. Affichage du jeu

Fonctionnement :

Lorsqu'il est activé par la FSM générale, ce module permet de lire en mémoire tous les blocs 24 par 24 pixels et de piloter le module VGA (Figure 2)

Architecture du module :

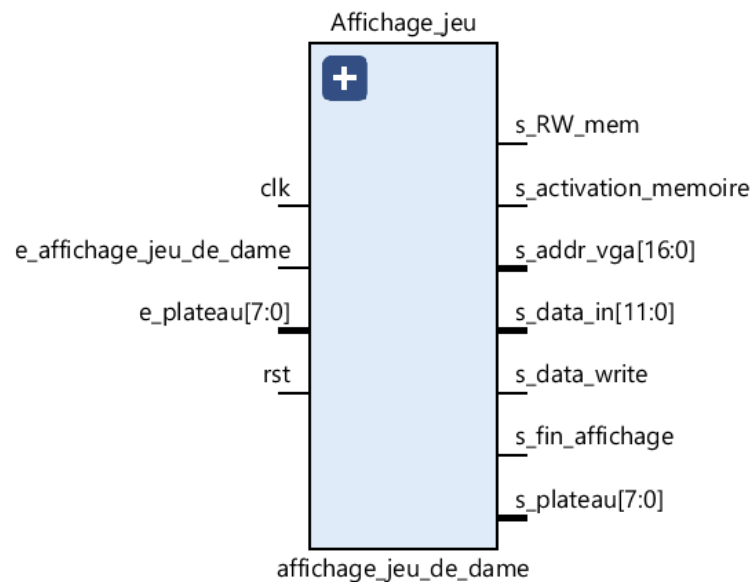


Figure 10 : Architecture du module d'affichage du jeu

Module affichage jeu de dame			
Entrées	Nombre de bits	Sorties	Nombre de bits
<i>clk</i>	1	<i>s_rw_mem</i>	1
<i>rst</i>	1	<i>s_activation_memoire</i>	1
<i>e_affichage_jeu_de_dame</i>	1	<i>s_addr_vga</i>	17
<i>e_plateau</i>	8	<i>s_data_in</i>	12
		<i>s_data_write</i>	1
		<i>s_fin_affichage</i>	1
		<i>s_plateau</i>	8

Entrées :

- **Clk** : horloge de 100 MHz
- **Rst** : remise à zéro du système
- **e_affichage_jeu_de_dame** : signal d'activation du module envoyé par la FSM générale
- **e_plateau** : données envoyées par la mémoire

Sorties :

- **s_rw_mem** : permet la lecture (0 logique) ou l'écriture (1) en mémoire
- **s_activation_memoire** : sortie d'activation de la mémoire
- **s_addr_vga** : adresse pixel envoyé au module VGA (Figure 2)
- **s_data_in** : données envoyées au VGA
- **s_data_write** : activation du VGA
- **s_fin_affichage** : signal envoyé à la FSM générale quand le module à fini l'affichage du plateau
- **s_plateau** : adresse envoyée à la mémoire pour lire la valeur de cette même case

FSM du module :

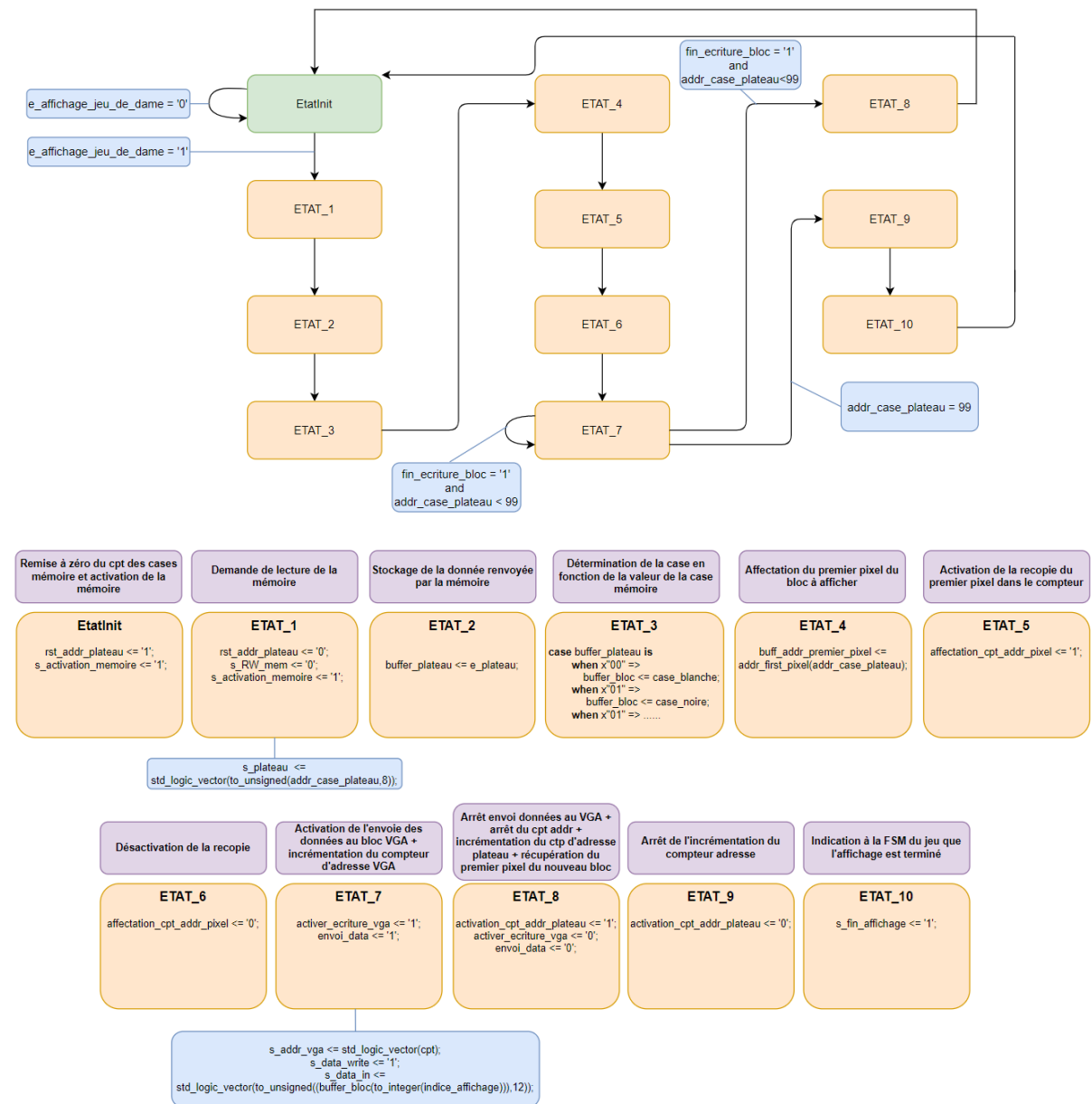


Figure 11 : FSM du module d'affichage jeu

Simulation :



Figure 12 : Simulation du module affichage jeu

Le module s'active bien lorsqu'il reçoit le signal d'activation `e_affichage_jeu_de_dame` est envoyé par l'unité de contrôle. La FSM est maintenant à l'étape1 et demande une lecture de la case mémoire n°0. L'étape 2 permet de stocker la valeur renvoyée par la mémoire, ici x00. L'étape 3 détermine avec sa table de conversion le bloc à afficher (cf. Table de correspondance). Les étapes 4 à 6 s'effectuent et à l'étape 7 un compteur d'adresse s'active permettant l'écriture sur le module VGA. L'adresse commence à 39 car nous voulons que notre plateau soit centré sur l'écran et les premières valeurs sont à 4095 ce qui représente du blanc. Sachant que la case n°0 de la case mémoire contient une case blanche, ceci semble cohérent.

Maintenant, il faut vérifier si le compteur s'arrête à la bonne adresse. Nous avons calculé que pour afficher la première case blanche en haut à gauche, le compteur d'adresse pixel doit s'arrêter à

l'adresse 7422. Nous remarquons que cela est bien le cas sur la deuxième simulation si nous regardons la valeur de **s_addr_VGA**.

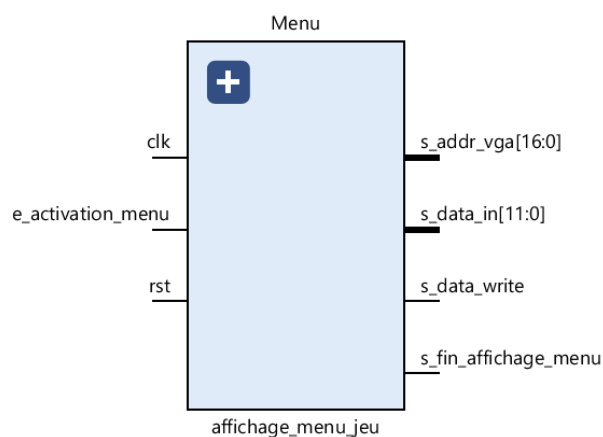
Enfin, le compteur de case allant de 0 à 99, il faut vérifier que le compteur d'adresse VGA s'arrête au dernier pixel que nous voulons afficher et qui correspond à la dernière case (en bas à droite). Le dernier pixel calculé en amont vaut 76 758. En regardant la troisième simulation nous pouvons remarquer que le compteur s'arrête bien à l'adresse pixel voulue, et que les dernières valeurs envoyées au VGA concordent avec les valeurs d'une case blanche. De plus, les état 9 et 10 sont bien effectués et la sortie **s_fin_affichage** est bien à l'état haut quand l'affichage est terminé.

b. Affichage du menu

Fonctionnement :

Nous avons décidé de mettre en place un menu du jeu avant de commencer la partie. Ce menu est une référence à la célèbre série d'échec *Le jeu de la Dame* (cf. Menu du jeu). Ce module contient une mémoire de 76 800 cases et contient des valeurs binaires sur 12 bits exportés grâce à notre code Matlab. Un compteur (0 à 76 799) va permettre d'affecter toutes les adresses du VGA à une valeurs binaires prédéfinis dans le bloc mémoire.

Architecture du module :



Module affichage menu			
Entrées	Nombre de bits	Sorties	Nombre de bits
<i>clk</i>	1	<i>s_addr_VGA</i>	17
<i>rst</i>	1	<i>s_data_in</i>	12
<i>e_activation_menu</i>	1	<i>s_data_write</i>	1
		<i>s_fin_affichage_menu</i>	1

Entrées :

- **Clk** : horloge de 100MHz
- **Rst** : remise à zéro du système
- **e_activation_menu** : signal d'activation envoyé par la FSM générale

Sorties :

- **s_addr_vga** : adresse envoyée au module VGA (Figure 2)
- **s_data_in** : données envoyées au VGA
- **s_data_write** : activation du VGA
- **s_fin_affichage_menu** : sortie mise à 1 quand le menu est affiché

c. Remise à zéro de l'affichage

Fonctionnement :

L'image du plateau de jeu étant plus petite que l'image du menu, si nous ne remettons pas les pixels à 0 (en noir) alors il y aura la présence des bords de l'image du menu (cf. Affichage du plateau sans remise à zéro de l'écran). Ce module permet donc la remise à zéro de l'écran, il est similaire au module affichage menu mais nous n'utilisons pas de bloc mémoire car nous n'avons besoin que d'une seule couleur qui est le noir.

Architecture du module :

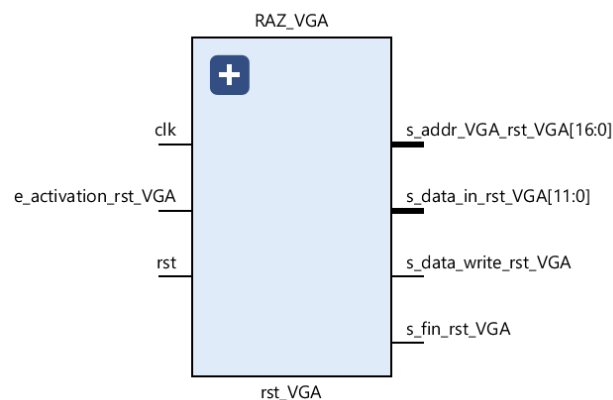


Figure 13 : Architecture de la remise à zéro de l'écran

Module remise à zéro de l'écran			
Entrées	Nombre de bits	Sorties	Nombre de bits
clk	1	s_addr_VGA	17
rst	1	s_data_in	12
e_activation_rst_VGA	1	s_data_write	1
		s_fin_rst_VGA	1

Entrées :

- **Clk** : horloge de 100MHz
- **Rst** : remise à zéro du système
- **e_activation_rst_VGA** : signal d'activation envoyé par la FSM générale

Sorties :

- **s_addr_vga** : adresse envoyée au module VGA (Figure 2)
- **s_data_in** : données envoyées au VGA
- **s_data_write** : activation du VGA
- **s_fin_rst_VGA** : sortie mise à 1 quand remise à zéro est terminée

4. Mémoire

Fonctionnement :

Ce module permet d'avoir un bloc mémoire de 100 cases regroupant les différentes cases du plateau de jeu.

Architecture du module :

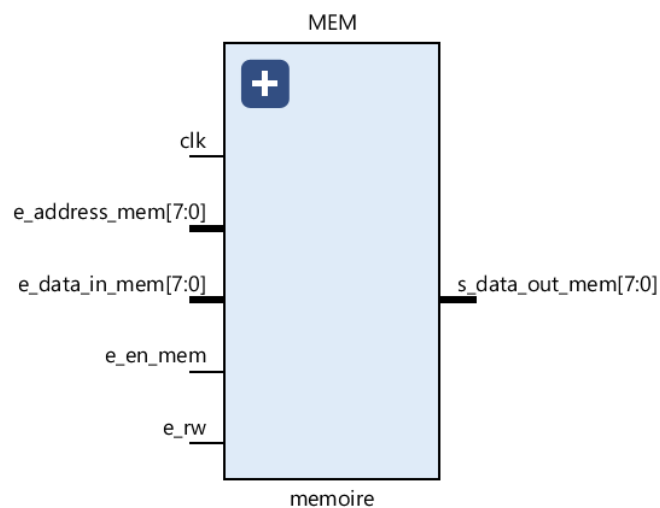


Figure 14 : Architecture de la mémoire

Module mémoire			
Entrées	Nombre de bits	Sorties	Nombre de bits
clk	1	S_data_out_mem	8
E_address_mem	8		
E_data_in_mem	8		
E_en_mem	1		
E_rw	1		

Entrées :

- **Clk** : horloge de 100 MHz
- **e_address_mem** : adresse permettant de sélectionner la case à lire ou à écrire
- **e_data_in_mem** : données d'entrée
- **e_en_mem** : activation de la mémoire
- **e_rw** : choix entre le mode écriture et lecture

Sorties :

- **s_data_out_mem** : données de sortie

5. Contrôle

Fonctionnement :

L'unité de contrôle permet de séquencer grâce à des signaux d'activations tous les modules de traitement, d'acquisition, d'affichage mais aussi les nombreux multiplexeurs présents dans ce projet.

Architecture du module :

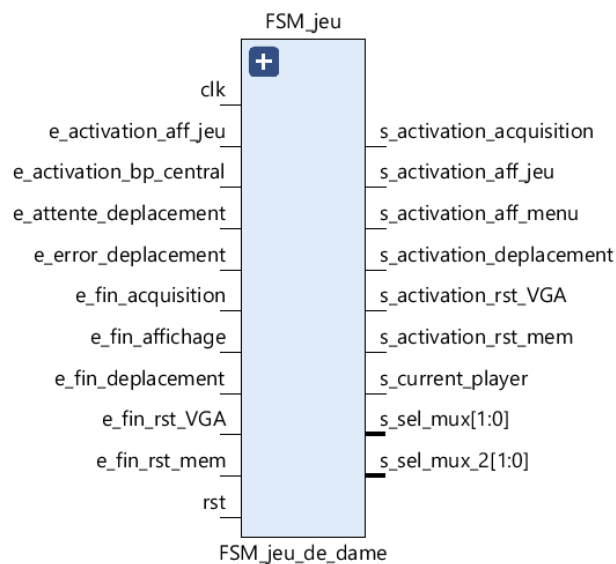


Figure 15 : Architecture de l'unité de contrôle

Module unité de contrôle			
Entrées	Nombre de bits	Sorties	Nombre de bits
<i>clk</i>	1	<i>s_activation_acquisition</i>	1
<i>rst</i>	1	<i>s_activation_aff_jeu</i>	1
<i>e_activation_aff_jeu</i>	1	<i>s_activation_aff_menu</i>	1
<i>e_attente_deplacement</i>	1	<i>s_activation_deplacement</i>	1
<i>e_error_deplacement</i>	1	<i>s_activation_rst_VGA</i>	1
<i>e_fin_acquisition</i>	1	<i>s_activation_rst_mem</i>	1
<i>e_fin_deplacement</i>	1	<i>s_current_player</i>	1
<i>e_fin_rst_VGA</i>	1	<i>s_sel_mux</i>	2
<i>e_fin_rst_mem</i>	1	<i>s_sel_mux2</i>	2
<i>e_activation_bp_central</i>	1		
<i>e_fin_affichage</i>	1		

Entrées :

- **Clk** : horloge de 100 MHz
- **rst** : remise à zéro du système
- **e_activation_aff_jeu** : entrée reliée à un interrupteur permettant de passer de l’affichage du menu à celui du plateau de jeu
- **e_attente_deplacement** : signal provenant du module traitement qui attend une donnée pour déplacer un pion
- **e_error_deplacement** : signal d’erreur envoyé par le module traitement
- **e_fin_acquisition** : signal de fin d’acquisition
- **e_fin_deplacement** : signal de fin de déplacement
- **e_fin_rst_VGA** : signal de fin de remise à zéro de l’écran
- **e_fin_rst_mem** : signal de fin de remise à zéro du bloc mémoire
- **e_activation_bp_central** : signal envoyé par le module d’acquisition si le bouton central est appuyé
- **e_fin_affichage** : signal de fin d’affichage

Sorties :

- **s_activation_acquisition** : signal d’activation du module d’acquisition
- **s_activation_aff_jeu** : signal d’activation du module affichage jeu
- **s_activation_aff_menu** : signal d’activation du module affichage menu
- **s_activation_deplacement** : signal d’activation du module traitement
- **s_activation_rst_VGA** : signal d’activation du module de remise à zéro du VGA
- **s_activation_rst_mem** : signal d’activation du module de remise à zéro de la mémoire
- **s_current_player** : signal sélection de joueur
- **s_sel_mux** : signal de sélection multiplexeur post mémoire
- **s_sel_mux2** : signal de sélection multiplexeur post VGA

FSM du module :

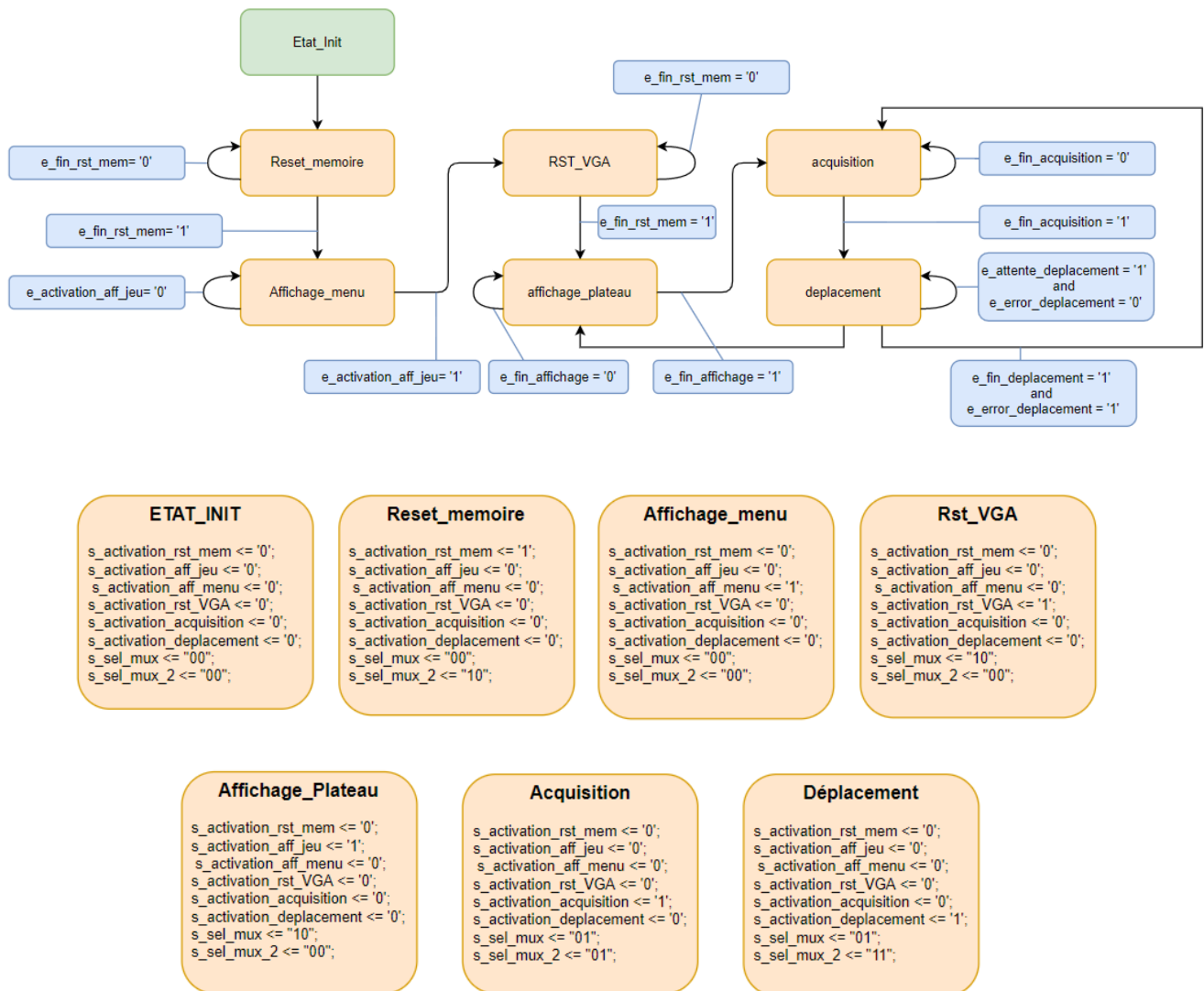


Figure 16 : FSM unité de contrôle

La FSM va dans un premier temps mettre la mémoire plateau à son état initial comme montré en annexe (cf. Affichage du plateau de jeu). Une fois que cela est fait, la FSM active le module d'affichage du menu du *Jeu de la dame*. Si le joueur active l'interrupteur 1 alors une remise à zéro de l'écran est nécessaire pour enlever les bordures du menu sur le plateau de jeu. Quand l'affichage du plateau est fait, la FSM active l'acquisition de données et active le traitement si un bouton poussoir est activé. Le système va donc boucler entre l'acquisition, le traitement et l'affichage du jeu.

V. Utilisation des ressources

Etant donné que nous avons un projet utilisant un affichage, un très grand nombre de BRAM et de LUT sont utilisés afin de stocker les blocs mémoires implémentés. Nous utilisons pratiquement la moitié des LUT et BRAM pour ce projet, mais cela n'a rien de surprenant.

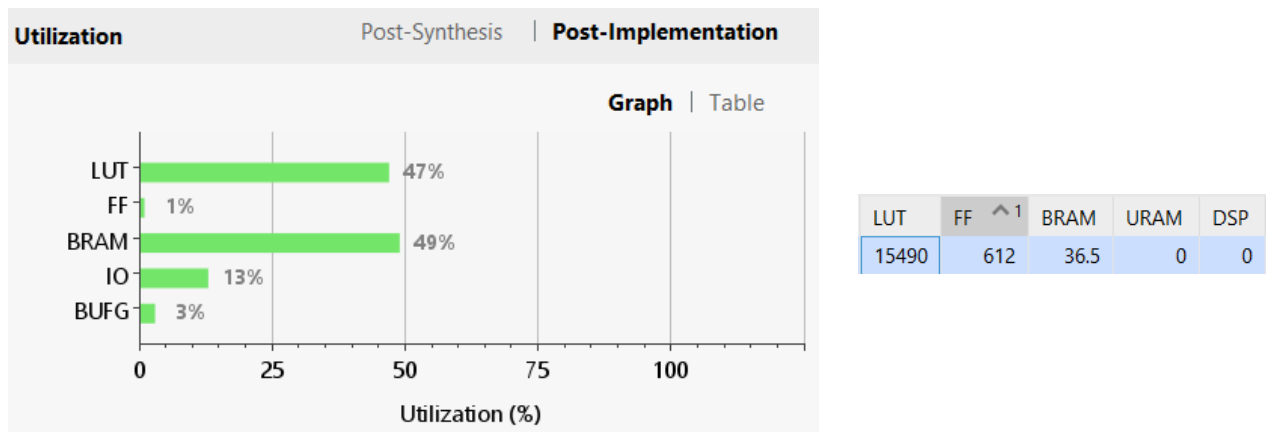


Figure 17 : Utilisation des ressources matérielles

Au début du projet nous avons eu des problèmes de chemin critique au niveau du module VGA, pour contrer cela, nous avons décidé de rajouter un registre avant ce module afin d'améliorer le chemin critique. Nous avons ainsi le rapport de timing suivant :

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 0,484 ns	Worst Hold Slack (WHS): 0,070 ns	Worst Pulse Width Slack (WPWS): 4,500 ns
Total Negative Slack (TNS): 0,000 ns	Total Hold Slack (THS): 0,000 ns	Total Pulse Width Negative Slack (TPWS): 0,000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 2212	Total Number of Endpoints: 2212	Total Number of Endpoints: 722

Figure 18 : Rapport de timing

VI. Amélioration du projet

A ce jour, nous ne pouvons seulement déplacer le curseur de sélection de case et la partie déplacement d'un pion n'a pas été validée. Ainsi, le cahier des charges initial n'étant pas respecté nous pouvons facilement décrire les axes d'amélioration par ordre de complexité:

- Finaliser la partie déplacement
- Ajouter un bloc pour la capture
- Instaurer les règles
- Ajouts des dames

VII. Conclusion

Ce projet nous a permis d'accroître nos compétences dans la compréhension et la mise en œuvre du langage de description matériel VHDL. Beaucoup d'erreurs ont été commises durant ce projet par manque d'expérience dans ce domaine (ex : calcul dans une FSM). Ce qui a entraîné un grand retard sur la finalité du projet.

Les premières séances ont été consacrées à la mise en place de l'architecture globale du projet, une cette étape validée nous avons attaqué la partie description matérielle avec nos propres moyens et avec l'aide expert de Monsieur Jégo. Chaque module décrit a été testé en simulation avant d'être utilisée sur la carte, mais nous avons compris au fil du temps de ne pas nous fier aux simulations. Ainsi nous avons donc dû améliorer notre utilisation du langage VHDL.













Nous avons utilisé nos compétences en traitement d'image pour exportés des images en binaire. Nous avons aussi consacré quelques heures sur le design des images des blocs sur Paint.net.

Même si le projet n'a pas abouti comme nous l'aurions souhaité, nous sommes fiers d'avoir choisi ce sujet qui semblait ambitieux de notre point de vue. Le projet était épanouissant dans le sens où ne faisons un sujet choisi par nous-même.

Notre montée en compétence sur langage VHDL et notre expérience dans la conception de circuit numérique est considérable.

VIII. Annexes

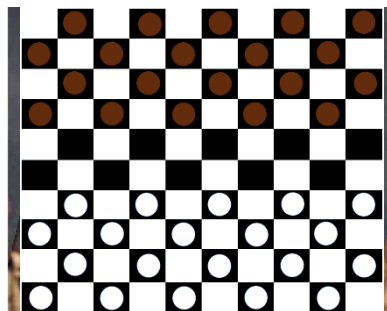
1. Table de correspondance

Image	Signification case mémoire	Valeur en hexadécimal
	Case blanche	X'00'
	Case noire	X'01'
	Pion marron	X'02'
	Dame marron	X'03'
	Pion blanc	X'04'
	Dame blanche	X'05'
	Sélection case blanche	X'06'
	Sélection case noire	X'07'
	Sélection pion marron	X'08'
	Sélection dame marron	X'09'
	Sélection pion blanc	X'0A'
	Sélection dame blanche	X'0B'

2. Menu du jeu



3. Affichage du plateau sans remise à zéro de l'écran



4. Affichage du plateau de jeu

