PROJET EXPÉRIMENTAL DE CONCEPTION DE CIRCUIT NUMÉRIQUE

JEU MORPION

MODULE PR209

Sylvain MARIEL
Thomas MOREAU

Mai 2013
Formation SEE
ENSEIRB-Matmeca

Sommaire

1.Présentation

1.1. Principe du jeu

L'application que nous avons choisi de réaliser est un jeu de type "Morpion". Le Morpion est un jeu de réflexion se pratiquant à deux joueurs au tour par tour et dont le but est de créer le premier un alignement sur une grille. Celle-ci se compose de 9 cases (3lignes et 3 colonnes) où chaque joueur vient y inscrire un symbole.

Grille de Morpion

1.2. Rappel du cahier des charges

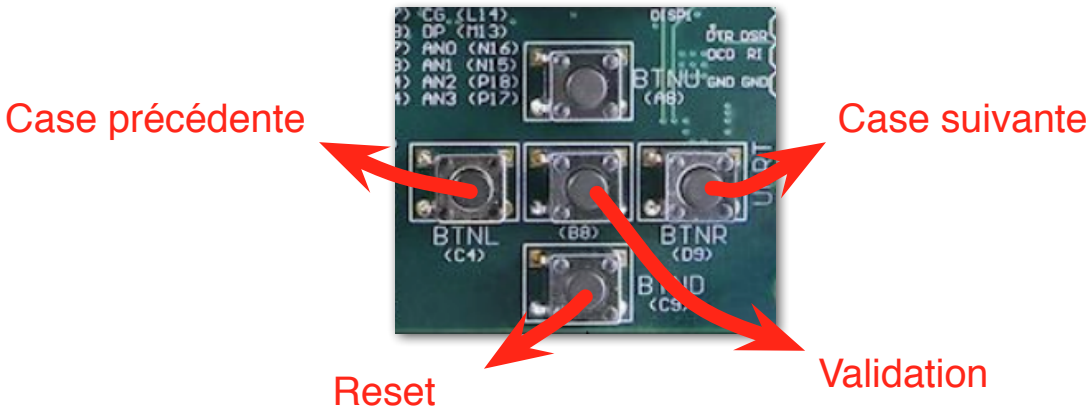
Le cahier des charges définit avant le commencement du projet, définit les parties de la carte Digilent Nexys 3 que nous utilisons ainsi que leur utilité.

Le jeu du Morpion utilise deux périphériques :

- Le port VGA, sur lequel sera relié un écran d'ordinateur, et servira de contrôle visuel.
- Les boutons poussoirs permettront aux joueurs d'interagir avec le système.

Les boutons poussoirs sont utilisés pour :

- Se déplacer dans la grille en sélectionnant une case.
- Valider la sélection de la case afin d'y déposer un symbole.

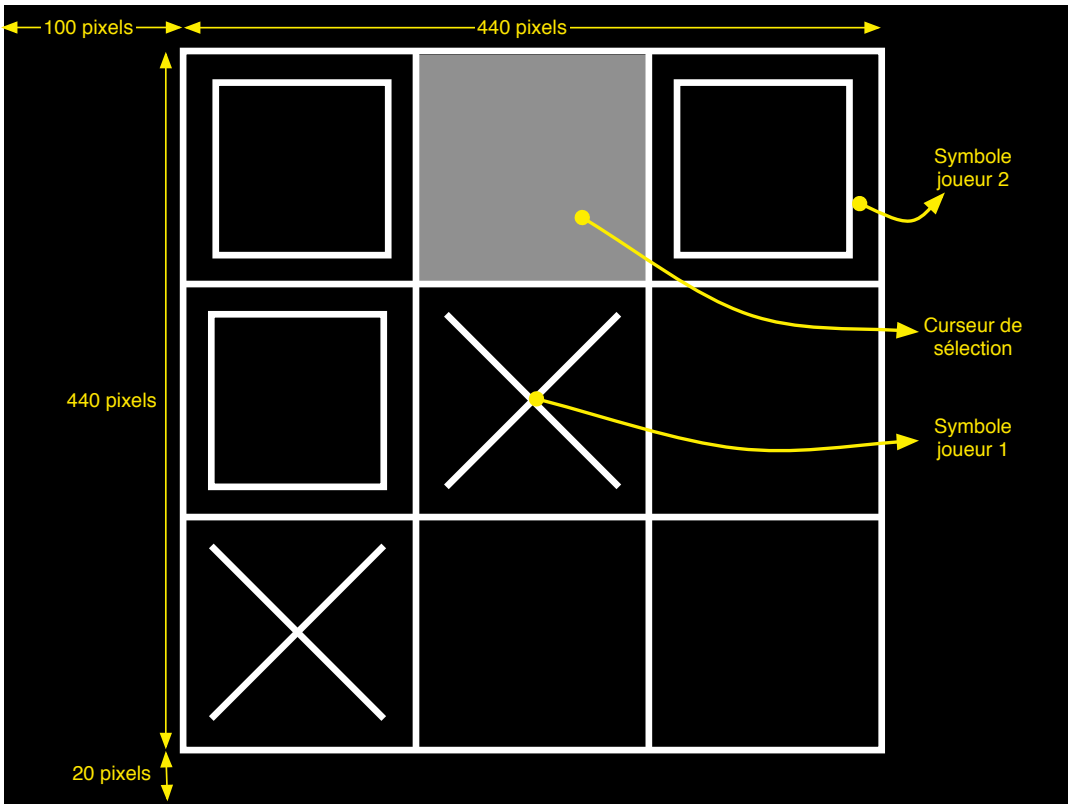


Action des différents boutons poussoirs

L‘écran a pour but :

- Afficher la grille de jeu et visualiser le positionnement de curseur. L’utilisateur doit pouvoir observer la case qu’il sélectionne.
- Visualiser les cases validées en affichant les symboles placés par les joueurs.

D'après la documentation de la carte Digilent Nexys 3, le port VGA utilisé nous permet d'afficher sur une zone de 640 par 480 pixels. Ainsi, nous proposons la gestion de l'affichage minimal suivant :



Affichage minimal attendu

1.3. Description fonctionnelle

D'un point de vue fonctionnel, nous avons donc décidé d'architecturer notre projet de la manière suivante :

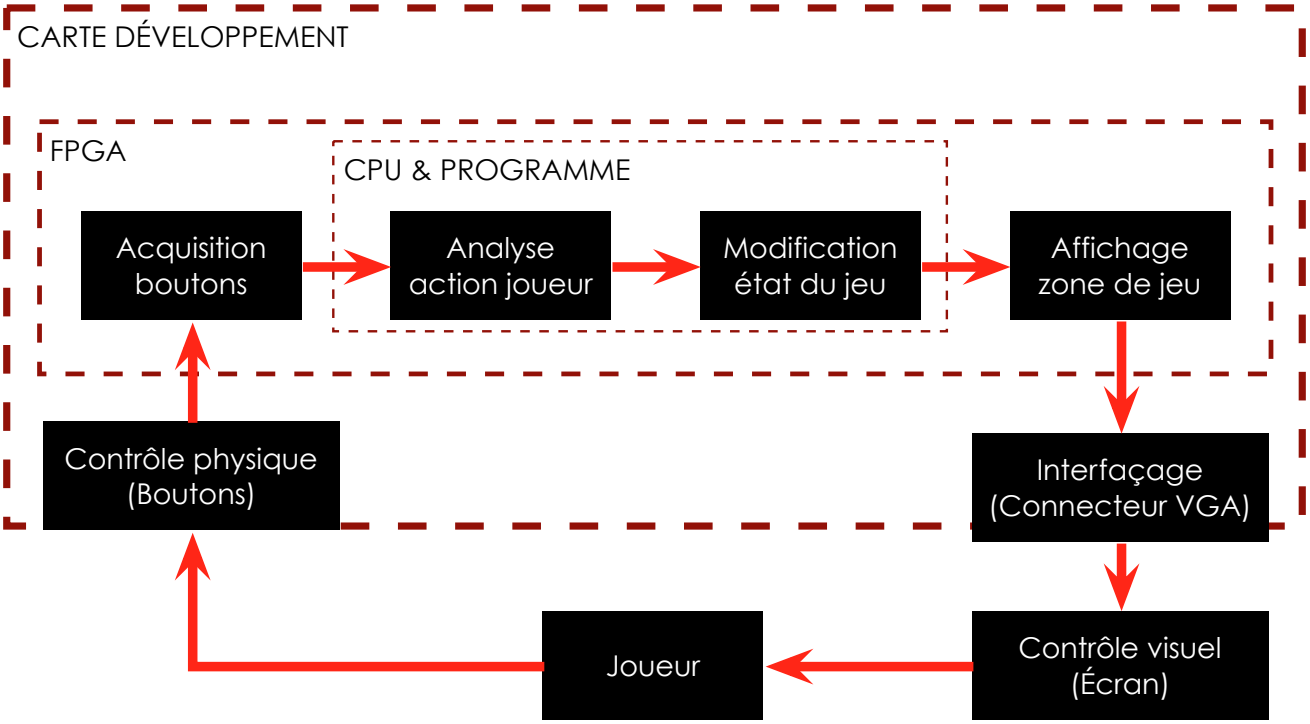


Schéma fonctionnel du système

Comme nous l'indiquions, le joueur aura deux contrôles pour interagir avec le jeu : les boutons et l'écran. Ils sont respectivement l'entrée et la sortie du système. Nous avons alors construit ce dernier entre ces deux points. Il se découpe en quatre grandes parties, chacune dépendant de la précédente.

1. L'acquisition des actions du joueur
2. L'analyse ou détermination du type d'action
3. La modification de l'état de la partie
4. La génération ou mise à jour de la zone de jeu

2. Le projet “Morpion”

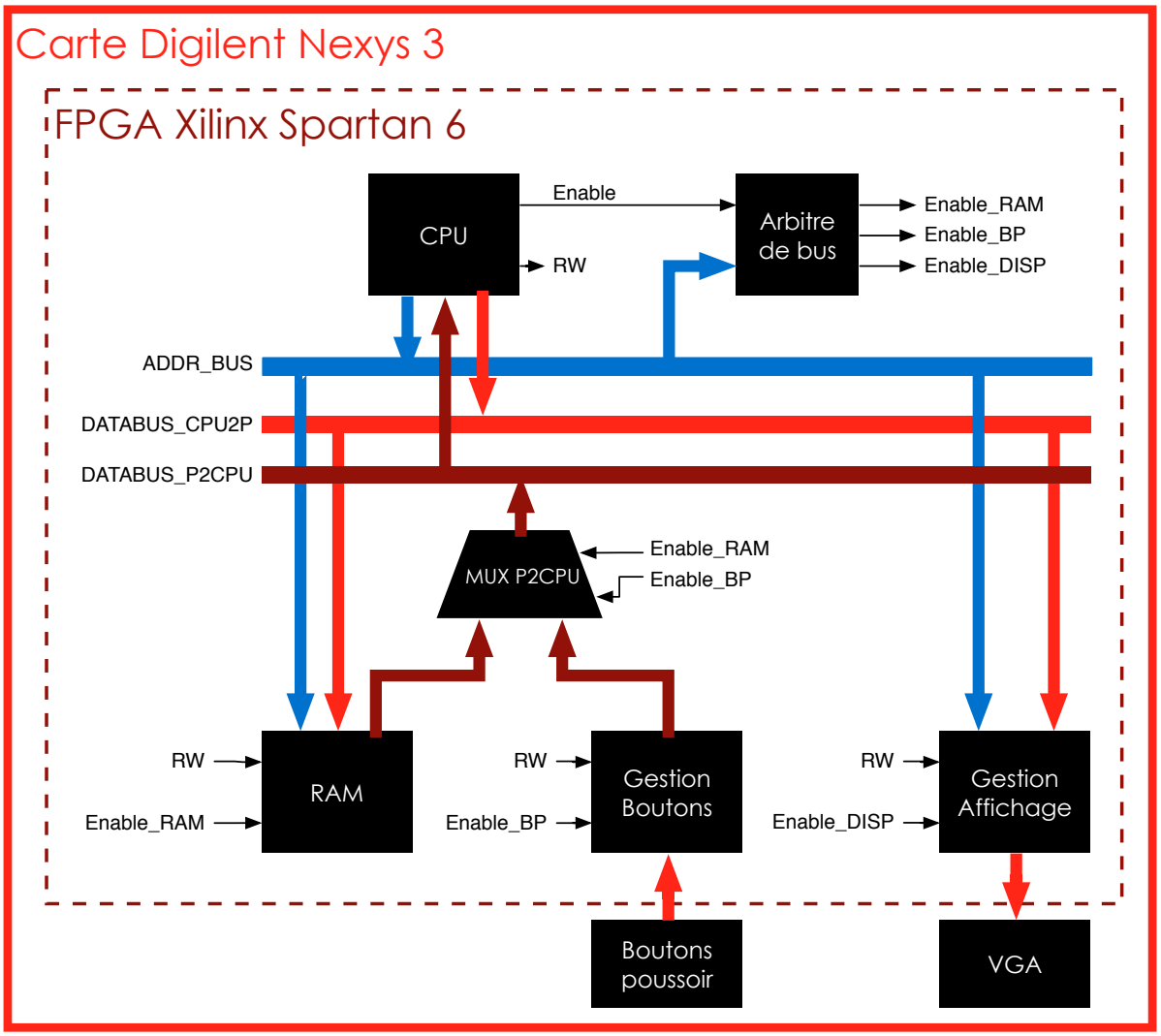
2.1. Organisation du système

a. L'architecture

Tout d’abord, le projet qui nous devons mettre en place doit utiliser le processeur 8 bits que nous avions conçu précédemment, dans le module EN 217. Nous disposons également d’une RAM 64 octets.

Nous avons alors décidé d'intégrer ces deux composants directement au coeur de notre architecture. Nous avons en effet opté sur une architecture "en bus", ou le CPU se retrouve comme maître du système, et chaque composant souhaitant dialoguer devient un de ses périphériques. La RAM est alors le premier périphérique du processeur.

Pour coller avec notre architecture fonctionnelle, nous avons réalisé deux périphériques de plus : un composant de gestion des boutons poussoirs (l'entrée), et un gestionnaire de l'affichage (la sortie). En les ajoutant, l’architecture matérielle peut être représentée de la manière suivante :



Architecture globale

Les différents joueurs pourront interagir de deux manières avec le jeu : en observant l'écran où se situe la grille du jeu, pour choisir une case qu'ils souhaitent et en appuyant sur les boutons poussoirs afin de se positionner dessus et valider par la suite.

L'appui des boutons engendre différentes opérations :

- une acquisition de l'opération produite par le joueur (déplacement ou sélection d'une case) par un module "Gestion des boutons"
- une analyse de son action et modification de l'état du jeu, effectué par le programme (processeur)
- un affichage de l'opération demandée par le module "Gestion affichage".

b. Communication entre entités

De cette architecture, on observe que la communication entre le CPU et les différents périphériques s'effectuent par le biais de bus. Nous avons en effet essayé d'instaurer une communication semblable à celle que l'on trouve au sein des microcontrôleurs (habituellement bus de contrôle, donnée et adresse). Dans notre cas, on retrouve :

- Deux bus de données, périphérique vers CPU et CPU vers périphérique. Comme leurs noms l'indiquent, ils permettent la circulation des données dans deux sens, mais jamais entre deux périphériques.
- Un bus d'adresse. Il permet au CPU de désigner une donnée précise sur un des périphériques.
- Des signaux de contrôles. RW pour indiquer l'action du processeur (lecture ou écriture de donnée), et Enable pour activer un périphérique.

Le signal Enable sortant du processeur est unique, et ne permet pas de cibler un périphérique donné. C'est ici qu'intervient l'arbitre de bus. Celui-ci a pour rôle de transformer l'unique signal Enable, en un signal pour chaque périphérique. Pour cela, il a besoin de l'adresse émise par le CPU, afin de savoir avec quel périphérique il veut communiquer. En effet, chaque adresse n'appartient qu'à un seul périphérique. Le bus d'adresse étant sur 6 bits, nous disposons de 2^6 adresses, soit 64. Ainsi il nous a été nécessaire d'établir un plan d'adressage, pour savoir à qui appartient chaque adresse.

Périphérique	Nb @	@ Début	@ Fin	Équation du signal «Enable» correspondant
RAM	56	0	55	/A5 + (A5 • /A4) + (A5 • A4 • /A3)
Gestion Boutons	1	56	56	(A5 • A4 • A3 • /A2 • /A1 • /A0)
Gestion Affichage	3	57	59	(A5 • A4 • A3 • /A2 • /A1 • A0) + (A5 • A4 • A3 • /A2 • A1)
Extensions futures	4	60	63	(A5 • A4 • A3 • A2)

Plan d'adressage du système où An correspond au énième bit d'adresse

Grâce à celui-ci, nous avons déterminé les équations booléennes nécessaires à l'arbitre de bus pour générer tous les signaux "Enable". Les adresses prévues pour d'éventuelles extensions ne sont pas utilisées et donc ignorées.

Par ailleurs, comme l'indique le tableau, nous avons diminué la taille de la RAM pour la passer sur 56 octets.
Enfin, notons que deux de nos périphériques partagent le même bus. Pour éviter tous conflit, nous avons inséré un multiplexeur commandé par les signaux "Enable". Ainsi, si nous venions à rajouter des périphériques, il nous suffirait de faire grossir ce multiplexeur (ou en rajouter dans le cas de l'autre bus) pour tout faire fonctionner.

2.2.CPU et programme : maîtres du système

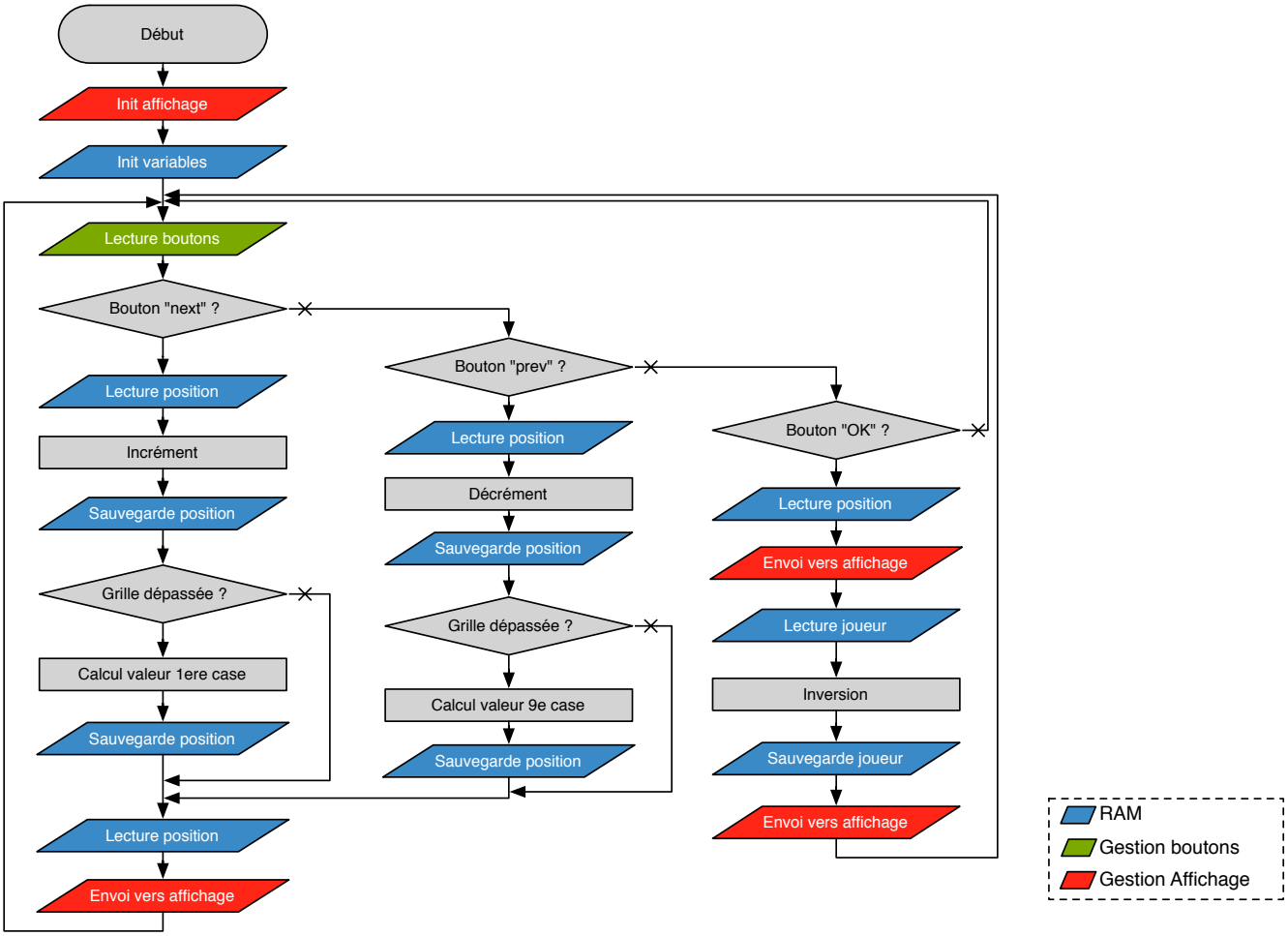
a. Le CPU

Pour notre projet, nous avons souhaité utiliser le processeur comme cerveau de notre architecture. Il est en charge de cadencer l'ensemble du système puisqu'il est le seul lien entre les entrées et les sorties. Nous ne rentrerons pas dans le détail de sa conception puisque nous l'avons déjà fait dans un rapport remis précédemment. Nous pouvons simplement préciser que avons extrait l'ensemble des périphériques réalisés lors du module précédent pour n'y garder que l'unité de traitement et l'unité de contrôle, sans y apporter aucun changement. Ainsi, pour rappel, ses caractéristiques sont les suivantes :

- Jeux d'instructions sur 8 bits
- 4 instructions différentes, dont deux opérations par l'UAL (NOR, ADD, STA, JCC)
- Adresses sur 6 bits
- Un seul registre de donnée (8 bits)

b. Le programme

Le CPU n'étant qu'un outil, il est nécessaire de lui faire exécuter un programme. Celui-ci que nous avons écrit permet de réaliser les deux tâches d'écrites sur le schéma fonctionnel précédent : analyser l'action du joueur et modifier l'état du jeu. Pour plus de clarté, nous l'étudierons sous la forme d'un logigramme.



Logigramme du logiciel assembleur embarqué

Le logiciel, disponible en annexe, se découpe en 7 grandes parties :

1. Initialisation des valeurs en RAM et sur affichage
2. Scrutation des boutons
3. Analyse de la valeur des boutons
4. Bouton "next" : incrément
5. Bouton "prev" : décrément
6. Bouton "OK" : validation position et changement de joueur
7. Envoi de la nouvelle position

Le programme est écrit de telle façon qu'il réalise une boucle infinie sur la scrutation des boutons. Ainsi, comme une grande partie des logiciels embarqués, il ne s'arrête jamais.

Les conditions sur le logigramme sont réalisées grâce aux instructions JCC. En effet, une condition booléenne revient en réalité à scinder un chemin en deux suivant un certain état. L'instruction JCC nous permet donc de faire un JUMP (chemin n°1) ou d'effacer la retenue et de continuer (chemin n°2). C'est-à-dire qu'il faut jouer avec la retenue de l'UAL pour créer des conditions.

Enfin, la communication avec les boutons ou l'affichage se fait comme avec la RAM. L'instruction STA nous permet d'envoyer le contenu du registre accumulateur vers l'adresse indiquée. Les instructions ADD et NOR nous permettent de charger la valeur à l'adresse indiquée et d'effectuer l'opération. Dans les deux cas, il suffit de faire correspondre l'adresse avec le périphérique désiré.

Notons que le logigramme parait redondant en certains points, et fait penser que le logiciel pourrait être simplifié. Malheureusement, la réalisation des conditions se fait par modification du registre accumulateur. Dans le cas où une valeur doit être réutilisée, il devient nécessaire de sauvegarder, réaliser la condition, et recharger la valeur.

Pour terminer, trois envois vers l'affichage sont réalisés dans ce programme. Les situations et types d'informations étant différents pour chacun, les adresses sont différentes.

Au final, après corrections du bug et ajouts de fonctions, le programme et les données occupent 54 octets sur les 56 de la RAM. Le CPU ayant un rôle central, nous avons essayé de porter le plus possible d'intelligence dans le programme, tout en l'optimisant au maximum de notre aisance avec le code assembleur. Nous n'avons en effet que très peu l'habitude de programmer à si bas niveau.

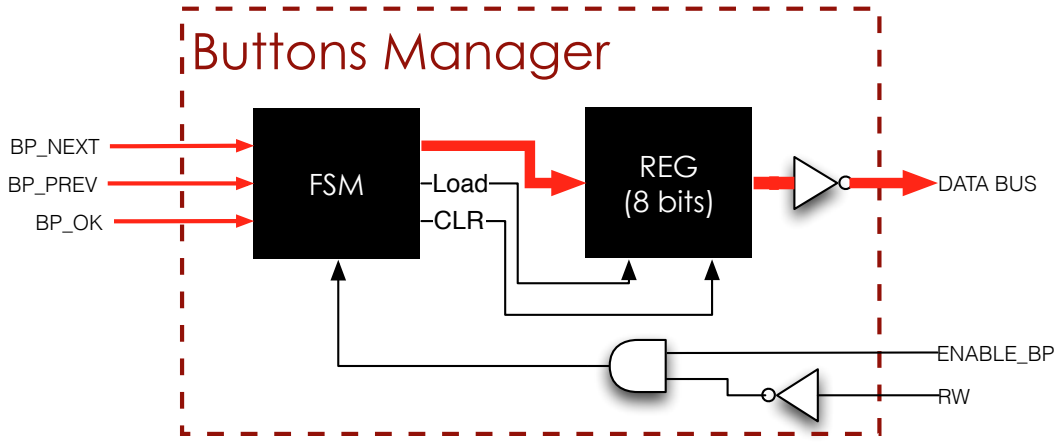
En adoptant une telle architecture, nous savions que le programme allait occuper un grand rôle et que les 54 octets de la mémoire induits par l'adressage sur 6 bits pourraient nous causer des problèmes. Lors de la rédaction du cahier des charges, nous avions effectivement réfléchi à modifier le processeur pour augmenter le bus d'adresse, et ainsi nous permettre de disposer d'une gigantesque RAM. Mais nous avons choisi de ne pas le faire pour deux raisons. La première étant que nous disposions d'un processeur entièrement fonctionnel et testé. Nous n'avions (et n'avons toujours) aucune idée du temps à passer sur ces modifications. La seconde étant que l'objectif pédagogique principal que nous avons retenu de ce projet est de nous familiariser avec la description de circuits en VHDL. L'augmentation significative de la mémoire nous aurait alors permis de placer la quasi-totalité de la réalisation de ce jeu au sein du code assembleur. Cela aurait en effet été un système élégant. Mais nous avons préféré adopter l'architecture que nous présentons dans ce document. Ce choix est en grande partie subjectif.

2.3 Gestion des boutons

Le périphérique de gestion des boutons à deux objectifs :

- Réceptionner les diverses informations données par le joueur
- Travailler ces informations pour les envoyer, par la suite, au CPU

Pour atteindre ces deux buts, nous avons organisé la gestion des boutons de la manière suivante :

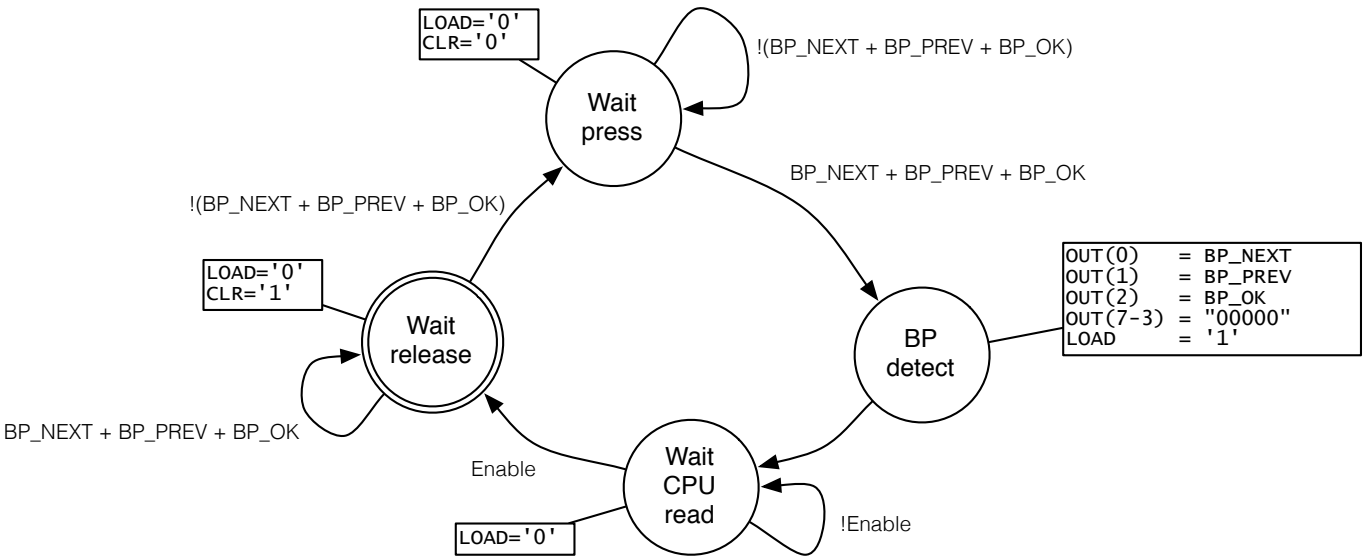


Architecture du périphérique de gestion des boutons

Comme tous les périphériques, la gestion des boutons travaille seulement si c'est à son tour. C'est à dire que le CPU est en mode scrutation d'un appui de bouton poussoir. Pour cela, il est nécessaire que le "ENABLE_BP" géré par l'arbitre de bus soit à l'état haut. De plus, pour anticiper certains bugs du programme, nous effectuons une vérification du "RW". Pour que le périphérique modifie son information de sortie, ce signal doit se trouver à l'état "READ". Une fois ces deux conditions validées, la machine d'état peut fonctionner.

Le registre permet au périphérique d'envoyer au CPU les informations dans un format que nous décrirons ultérieurement. On observe que ce registre possède deux entrées : "Load" et "CLR".
Le signal "Load" permet d'indiquer au registre de transmettre l'information liée aux boutons lorsqu'il y a un appui. Le signal "CLR", quant à lui, permet la remise à zéro de l'information, c'est dire que le registre va transmettre une information au CPU, qui indique qu'aucun appui sur bouton n'a été fait.

Une fois qu'un joueur appuie sur un bouton, une information est envoyée au périphérique de gestion des boutons sous forme de signal. Cette information est recueillie par la machine d'état, puis traitée. Le fonctionnement de celles-ci est décrit ci-dessous :



Principe de fonctionnement de la machine d'état du périphérique de la gestion des boutons

Ce module est composé de 4 états. Le premier, "Wait release", permet d'initialiser ou de réinitialiser le registre servant d'interface avec le CPU en positionnant le signal "CLR" à '1'. Rappelons que cette action indique au processeur qu'aucun appui n'est effectué. Cet état permet donc l'attente du relâchement du bouton et donc de prendre en compte le cas ou les joueurs débuteraient une partie en pressant un bouton par mégarde.

Le second état est "WAIT PRESS". Celui-ci permet l'attente d'un appui. Une fois que cela arrive, nous nous dirigeons vers le troisième état "BP detect". Cet état va envoyer l'information liée au bouton vers le registre, ainsi que le signal "Load" (positionné à l'état haut), permettant au registre de mettre à disposition cette information au programme.

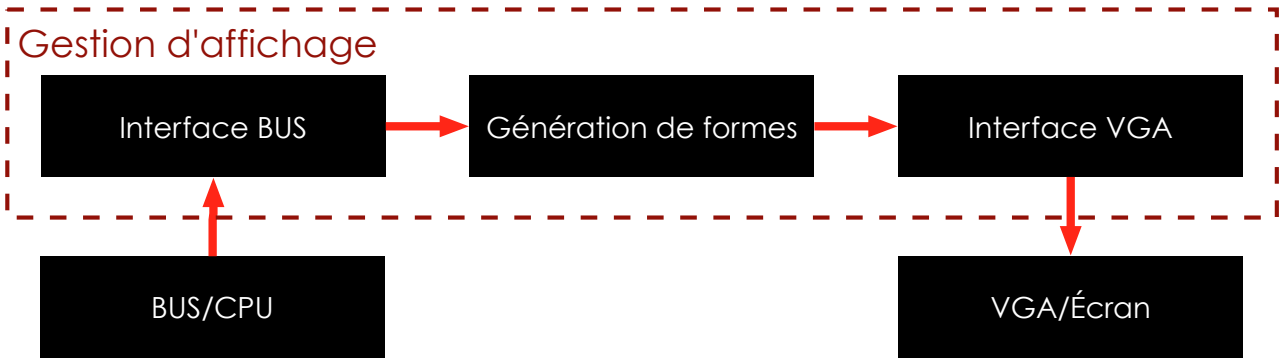
Un coup d'horloge plus tard, nous nous retrouvons au dernier état. Nous avons mis en place cet état, dans un souci de synchronisation. Cet état permet au registre d'envoyer l'information jusqu'à ce que le CPU ait bien reçu l'information et qu'il se remette en mode scrutation d'un appui. Puis un cycle recommence.

La syntaxe de l'information envoyée au CPU est régie selon la loi suivante : bit 7 à 3 étant à '1' et bit 2 à 0 dépendent de l'appui qu'effectue l'opérateur. Un '1' est équivalent à un non-appui et un '0' à l'inverse. Cette manière de procédé permet de prendre en compte les différents appuis de manière très simple. Par exemple, l'appui sur aucun des boutons envoi au CPU l'information suivante : "1111111". Cette syntaxe nous est très utile lors du traitement du programme. Comme il n'existe aucune condition en assembleur, une simple addition de 1 effectuée par le processeur, va permettre la création d'une retenue (dans le cas d'un non-appui). Celle-ci fera office de condition lors des commandes d'un saut ("JCC"). Par exemple, lors d'un appui sur le bouton "NEXT" , le CPU reçoit "11111110" et l'absence de retenue après l'addition va provoquer un saut sur une parcelle de programme destinée à découvrir quel bouton a été pressé.

Pour conclure, nous avons dû faire face à un problème de "rebond" généré par l'appui sur les boutons. Ces rebonds mécaniques induisent des successions rapides de 1 et de 0, pouvant faire croire à de multiples appuis. En effet, notre système fonctionnait à ce moment-là à plusieurs dizaines de MHz. Pour y remédier, nous avons décidé de grandement descendre la vitesse de fonctionnement de certains éléments. Ainsi, la gestion des boutons est passée à 3 kHz, éliminant de fait ces problèmes de rebonds. Cependant, le processeur ne devant pas fonctionner plus vite que ses entrées, nous l'avons également descendu à cette vitesse. Pour cela, nous avons utilisé les signaux CE (clock enable) simulant une division d'horloge.

2.4 Gestion de l'affichage

Le principe du périphérique de gestion d'affichage est simple : interpréter les informations fournies par le CPU en éléments graphiques pour le joueur. Pour cela, nous avons découpé ce composant en trois entités :



Architecture du périphérique de gestion d'affichage

Les informations envoyées par le CPU sur le bus de données sont sauvegardées par le premier bloc d'interface. Les données mises à dispositions sont transformées et interprétées par la génération de formes. Comme son nom l'indique, son rôle est de générer des éléments graphiques. Ces éléments graphiques sont alors stockés et traduis au format "VGA" pour être lus par un écran.

a. Interface CPU

Le périphérique ne fait que recevoir des données du CPU, comme en témoigne le programme que nous avons décrit précédemment. Comme nous l'indiquions, ce périphérique dispose de trois adresses pour trois données distinctes :

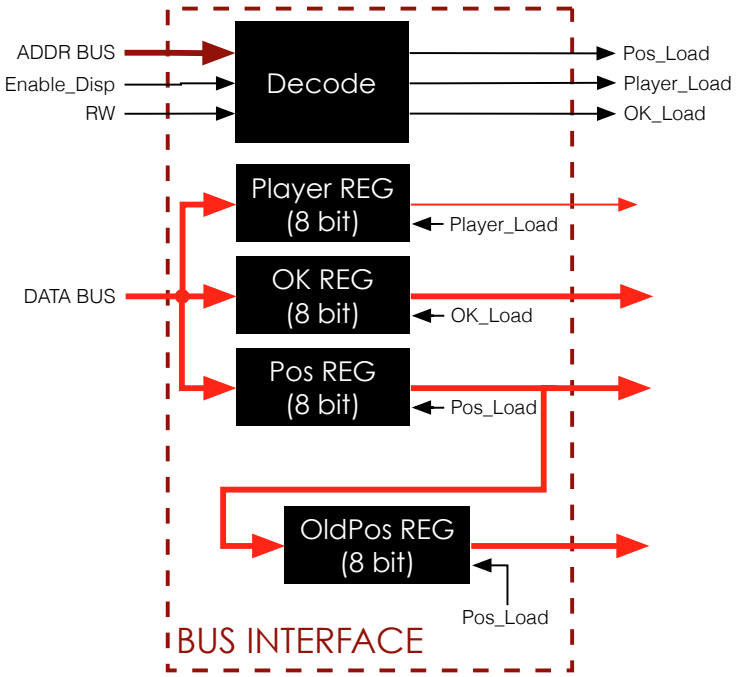
Type de donnée	Nom	Adresse
Joueur dont c'est le tour de jouer	Joueur (J)	0x39
Position sur la grille de la dernière sélection	Position (POS)	0x3A
Position sur la grille de la dernière validation	Validation (OK)	0x3B

Chaque information est alors stockée dans un registre différent. Au cours de l'avancement du projet, pour des raisons de commodité, nous avons décidé de rajouter un quatrième registre contenant l'ancienne position sélectionnée (OLDPOS).

Lorsque la donnée arrive du bus, le choix du registre se fait par un dernier composant, "décodeur". À la manière de l'arbitre de bus, celui-ci va transformer le signal Enable en trois signaux distincts en fonction de l'adresse reçue. Ces trois signaux sont alors l'image d'une action réalisée par le joueur :

- **Pos_load** : changement de la position du curseur
- **Player_load** : changement de joueur
- **OK_load** : validation d'une case

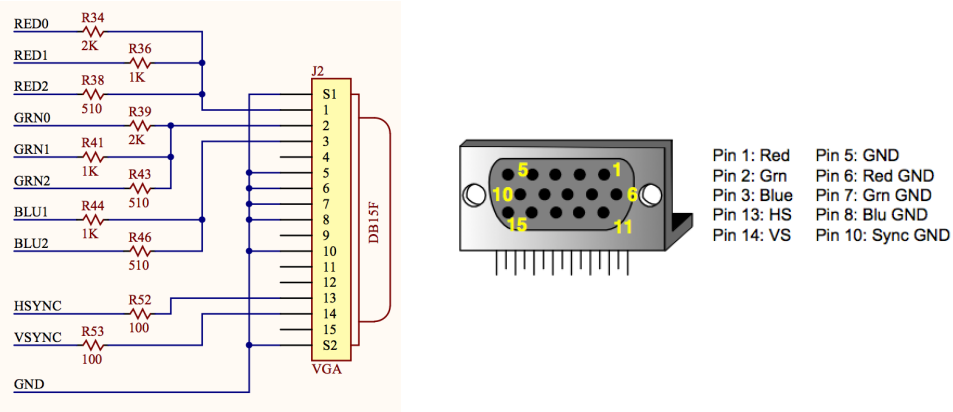
Ainsi, l'architecture que nous avons décrite en VHDL est la suivante :



Bloc d'interface de gestion d'affichage avec bus

b. Interface VGA

Le VGA, pour Video Graphics Array, est un standard d'affichage pour ordinateur créé par IBM en 1987. Il désigne à la fois un mode d'affichage (640*480 pixels) et une connectique. Présente sur notre carte de développement Digilent Nexys 3, cette connectique est la suivante :

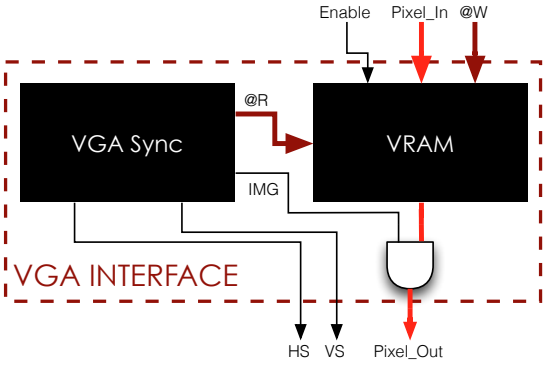


Connectique VGA sur carte Nexys 3

En plus de la masse, on distingue donc une pin pour chaque couleur et deux signaux de synchronisation horizontale et verticale sur le connecteur DB15. Le problème avec l'utilisation du VGA et d'un FPGA est que les signaux de couleur sont analogiques. Ainsi, Digilent a fait le choix pour sa carte de dédier un total de 8 pins du FPGA pour les couleurs. En utilisant des résistances de différentes valeurs derrière ces pins, il est alors possible de créer différents niveaux de tensions suivant les combinaisons d'états logiques choisis. On obtient alors 8 nuances de rouges et de verts, et 4 de bleu. Ils expliquent par ailleurs leur choix de n'utiliser que deux bits pour le bleu, car l'oeil humain est moins sensible à cette couleur.

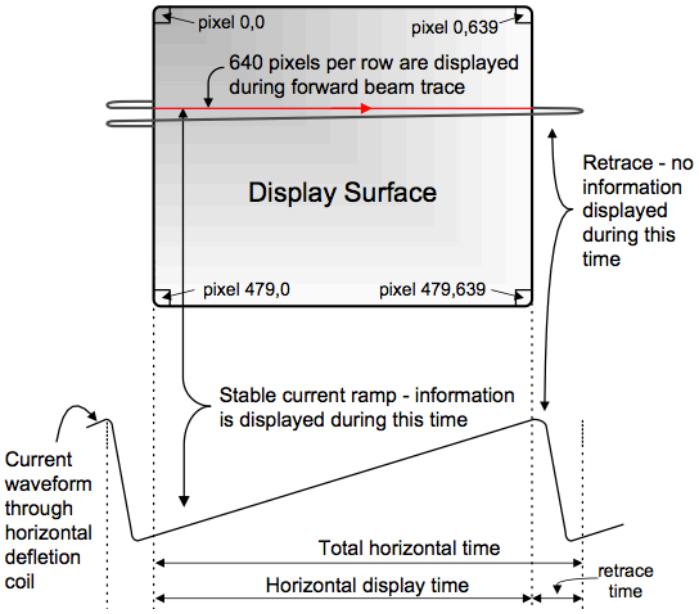
Au sein du FPGA, la couleur d'un pixel (unité minimale adressable par un contrôleur vidéo) est donc définie par ces 8 bits. La deuxième information nécessaire est la position du pixel. Le VGA impose pour cela de passer par des signaux de synchronisation horizontale et verticale. C'est donc une information temporelle qui définit la position d'un pixel. C'est en effet un choix historique du fait de la technologie employée à l'époque sur les écrans cathodiques. Le canon à électron étant dévié par des électroaimants commandés par les signaux de synchronisation, les pixels sont envoyés par "balayages", les uns à la suite des autres, colonne par colonne et ligne par ligne. L'enjeu est donc de positionner la valeur de chaque pixel au bon nomment.

Cette entité est donc composée de deux éléments. Le premier (Synchronisation) s'occupe de la gestion temporelle, et donc des signaux de synchronisation. Il sait à chaque moment quel pixel est en train d'être affiché. Le deuxième est une mémoire vidéo (VRAM). Elle comprend la totalité des pixels de l'image, et est lue par le bloc de synchronisation. On obtient alors une architecture telles que celle-ci :



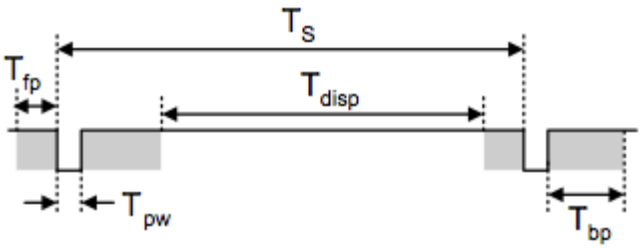
Bloc d'interface VGA

Comme décrit précédemment, la partie synchronisation VGA doit générer les signaux de synchronisations HS et VS et de coordonner l'adresse des données. L'image ci-dessous décrit le balayage effectué :



Balayage effectué par le canon à électron

Les synchronisations se décomposent en 4 temps :



Signal de synchronisation horizontale

- Le temps d'affichage est le moment ou les données peuvent être envoyées à l'écran
- L'impulsion (pulse) est le temps de saut de la droite de l'écran à la gauche
- Le "front porch" est le temps de transition entre la fin des pixels et le moment du saut
- Le "back porch" est le temps de transition entre le saut et le début de la nouvelle ligne de l'écran

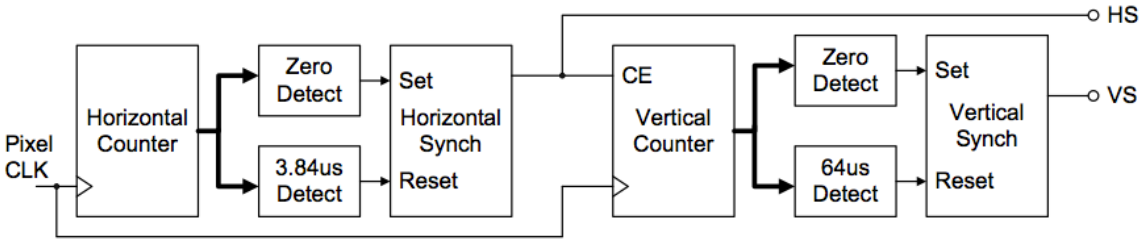
La synchronisation horizontale et verticale sont soumises aux même type de signal, cependant les durées sont différentes. Nous avons décidé d'utiliser une fréquence de rafraîchissement de 60 ±1Hz pour une résolution de 640 par 480 pixels. Les timings des signaux HS et VS sont indiquées dans le tableau ci-contre peuvent être dérivées, dont les temps de synchronisation sont donnés dans la documentation :

Symbol	Parameter	Vertical Sync			Horiz. Sync	
		Time	Clocks	Lines	Time	Clks
T_S	Sync pulse	16.7ms	416,800	521	32 us	800
T_{disp}	Display time	15.36ms	384,000	480	25.6 us	640
T_{pw}	Pulse width	64 us	1,600	2	3.84 us	96
T_{fp}	Front porch	320 us	8,000	10	640 ns	16
T_{bp}	Back porch	928 us	23,200	29	1.92 us	48

Durée de synchronisation horizontale et verticale pour affichage 640x480 @ 60 Hz

La génération des signaux de synchronisation HS et VS se fait à l'aide de compteur allant de 0 à 639 pour HS et de 0 à 479 pour VS. Cependant, il est nécessaire de retrouver le temps d'affichage, nécessaire pour savoir à quel moment la VRAM doit envoyer ses informations.

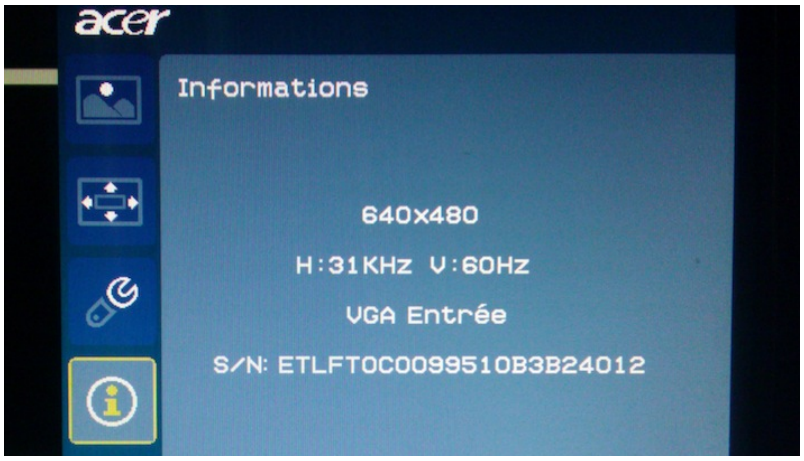
Digilent propose une architecture de ce type :



Génération de HS et VS préconisée par Digilent

Pour simplifier cette tâche, nous avons décidé de débuter les signaux de synchronisation par ce temps. Durant ce temps d'affichage, le module de synchronisation positionne à '1', un signal "IMG". Celui-ci permet d'envoyer au port VGA les informations de la VRAM durant le temps d'affichage.

Au final, notre moniteur numérique détecte parfaitement nos réglages, et nous en informe :



Détection du VGA émit par la carte

Mémoire vidéo (VRAM)

Afin de stocker l'image à afficher, nous avons choisi d'utiliser une mémoire vidéo. Elle contient la totalité des pixels visibles à l'écran. C'est en réalité une RAM double port. C'est-à-dire qu'elle est lue et écrite sur deux ports distincts. Ainsi, elle peut être lue par le composant de synchronisation (l'adresse étant généré à partir de ses compteurs) et écrite par un élément différent. Nous avons choisi de l'adresser sur 19 bits. En effet, nous disposons d'un affichage en 640*480, ou de 307200 pixels. Il nous faut donc 19 bits pour adresser ces valeurs (512k). Nous avons fait le choix de découper l'adresse en fonction des coordonnées. C'est-à-dire que les 10 bits de poids faible correspondent aux X, et les 9 de poids fort aux Y.
Comme nous l'indiquions, la valeur d'un pixel est définie sur 8 bits. Dans l'idéal nous aurions alors utilisé une VRAM de 512 ko. C'est d'ailleurs ce que nous avions prévu à tort au démarrage du projet. En effet, nous avons fait une erreur considérable, puisque notre FPGA Spartan 6 (en version XC6SLX16) dispose d'un maximum de 572 kilos... bits de RAM. En effet, il est équipé de 32 blocs de 18kb chacun, et non octets. Nous avons donc dû réfléchir à l'utilisation de ces blocs. Deux choix s'offraient à nous : diminuer la taille d'affichage ou diminuer la taille des pixels. C'est la deuxième solution que nous avons choisi, et sommes donc passé à un unique bit de donnée, répliqué en sortie de la RAM sur les 8 pins dédiées. Ainsi, nous sommes passés d'un affichage 256 couleurs à un affichage en noir et blanc sans nuances. Ce faisant, nous obtenons une RAM de 512kb, flirtant avec les limites du FPGA.

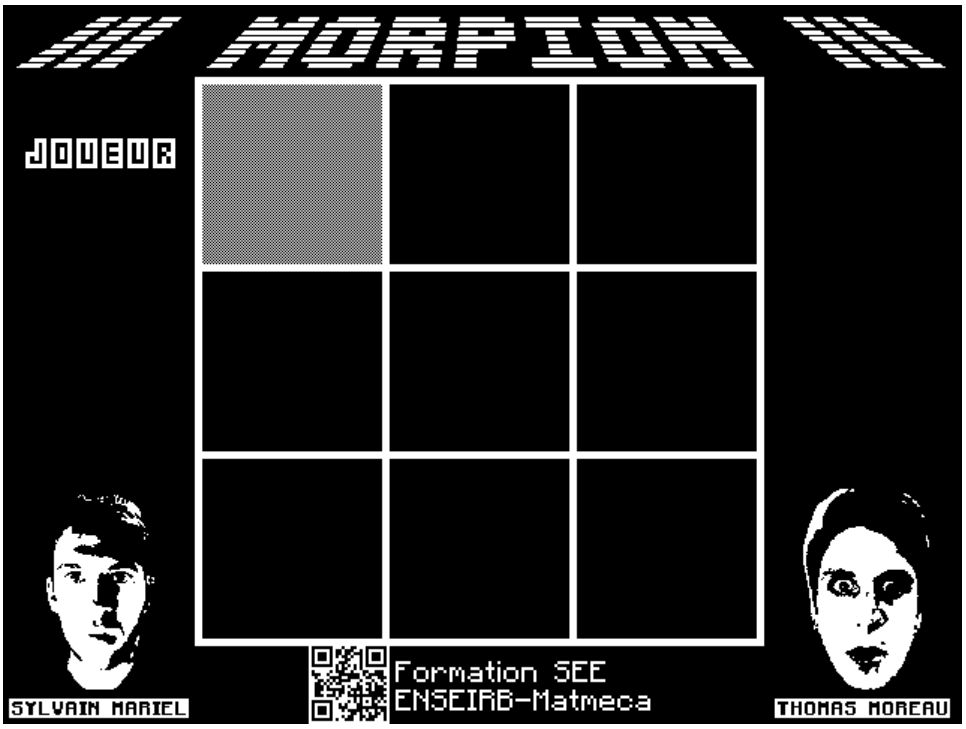
Étant donné la nature du jeu, un grand nombre d'informations à afficher est statique. Ces informations ne seront jamais modifiées. C'est le cas de la grille et de tous les éléments "décoratif" autour. Il est donc plus aisé d'initialiser notre VRAM avec ces valeurs. Le problème est que nous disposons d'une mémoire de 512kb. Si l'on veut initialiser chaque pixel, il est nécessaire de les écrire un à un. C'est un travail impossible à faire à la main, représentant plusieurs centaines de milliers de lignes de code. Nous avons donc choisi d'appliquer un certain nombre d'enseignements acquis lors du module de traitement de l'image. Ainsi, nous avons conçu des scripts matlab permettant de générer le fichier VHDL de la VRAM à partir d'une image.

L'image est tout d'abord acquise avec la fonction imread. Grâce à celle-ci, nous obtenons les matrices correspondant à chaque composante chromatique, où chaque pixel a une valeur de 0 à 255 (au format uint8). Nous allons tout d'abord passer cette valeur de 0 à 1 par une simple division par 255. Puis, pour chaque pixel, nous calculons son adresse à partir de ses coordonnées x et y. Il suffit d'effectuer un décalage du nombre de bits désiré :

$$addr = y*(2^{nb\_bits\_x}) + x;$$

La valeur du pixel est obtenue par un OU logique entre les trois composantes. Pour terminer, on crée la chaîne de caractère correspondant au code vhdl. Par exemple : 18 => '0', . L'ensemble des chaînes est assemblé et intégré dans un fichier VHDL vide de donnée.

Au moment où nous écrivons ces lignes, c'est l'image suivante qui est passée au script Matlab, dont sont générées les 400.000 lignes de code VHDL :

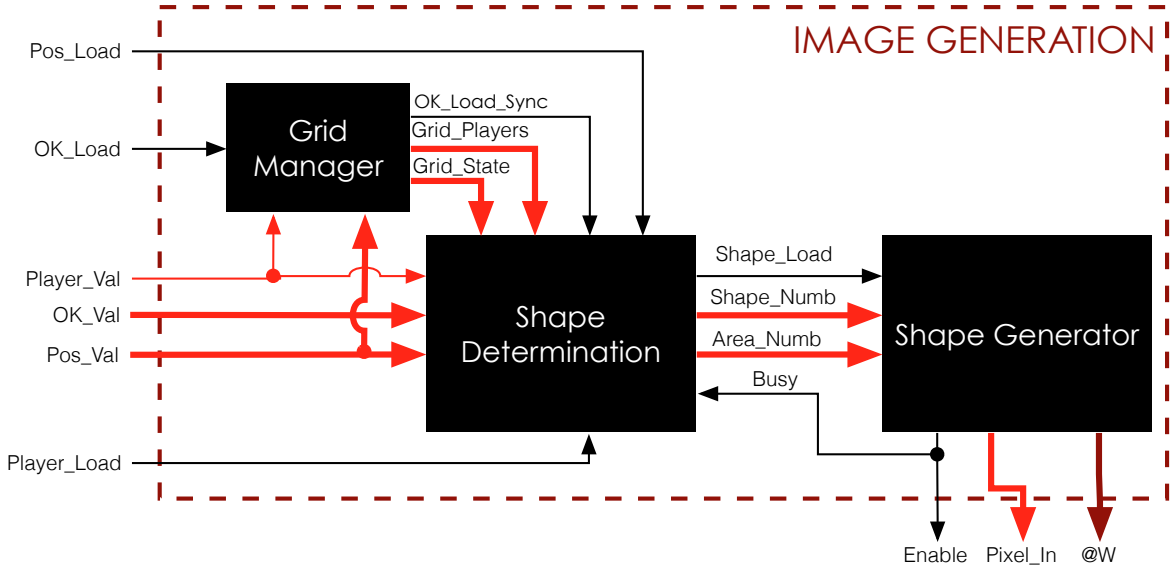


Contenu de la VRAM à l'initialisation

Pour plus de détail, le code Matlab est disponible en annexes.

c. Génération d'image

La génération de forme est la partie la plus complexe du projet. Ce module ce décompose en trois éléments :



Module Génération d'image de la gestion de l'affichage

- Une gestion de la grille permettant d'effectuer un historique des différentes validations effectuées par les deux joueurs.
- Un générateur de forme proposant de fournir à la VRAM Le contenu de la case pointée/ sélectionnée par le joueur
- Une détermination de forme gérant la liaison entre la gestion de la grille et le générateur de forme

La gestion de la grille

La gestion de la grille intervient seulement lorsqu'une case est validée. Comme dit précédemment, elle crée un historique des cases occupées et par quel joueur.

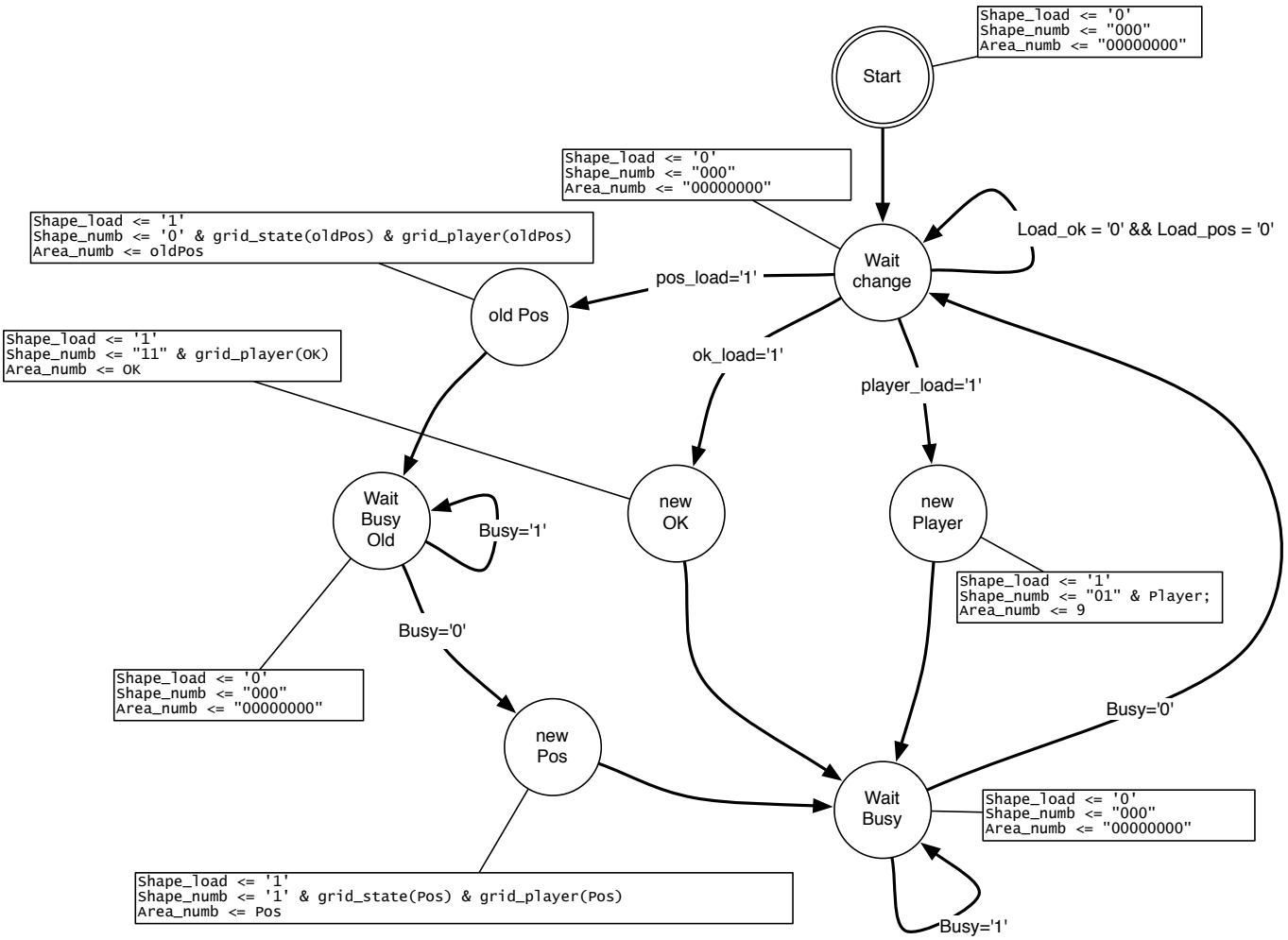
Sous la seule condition d'une validation (signal « OK_Load » généré par l'interface du bus), le module génère deux vecteurs, « Grid_State » et « Grid_Player ». Ces deux signaux sont de 9 bits, un pour chaque case de la grille. Le premier, « Grid_State », positionne un '1' sur le bit correspondant à la case où a été effectuée la validation. Le second génère un '1' ou un '0' sur le même bit que 'Grid_State' en fonction du joueur.
Cette façon de procéder permet d'avoir à disposition deux signaux qui indiquent quelles sont les cases sélectionnées et par quel joueur.

À noter que ce module permet aux joueurs de ne pas sélectionner une case qui l'est déjà. Cependant, ce contrôle n'empêche pas le changement du joueur effectué dans le programme. Nous aurions dû modifier le programme en conséquence, mais n'avons plus assez de RAM pour cela.

Lors de la simulation de ce gestionnaire, nous avons rencontré un problème de synchronisation. Il recevait, avant nos modifications, 'OK_val'. Nous avons pu observer via les différents testbenchs que le module sélectionnait l'ancienne case qui avait été validée. Nous avons visualisé que 'OK_val' proposait la valeur qui nous intéressée un coup d'horloge trop tard. C'est pourquoi, pour remédier à ce problème, nous avons décidé de nous fier à la valeur du signal 'Pos_val'. Cette valeur doit être la même que 'OK_val', cependant, elle est générée plus tôt, car le joueur a besoin de se positionner sur la case, avant même de la sélectionnée.

De plus, nous avons rajouté un signal 'Ok_Load_Sync' afin de synchroniser les autres éléments après la modification des états de la grille. Le problème de synchronisation a pu être annihilé.

La détermination de forme est effectué par une machine d'état, dont le fonctionnement est représenté ci-dessous :



Détermination de la forme de la case

Dans cette FSM, trois cas sont traités :

- Le changement de position
- La validation d'une case
- Le changement de joueur

Tout d'abord, le premier état "START" permet d'effectuer l'initialisation de l'affichage du jeu.

L'état "WAIT_CHANGE", comme son nom l'indique, attend un changement d'état de la partie. Soit Un changement de position, soit la validation d'une case, soit le changement de joueur.

Le traitement du changement de position s'effectue en 3 étapes.

1. "OLDPOS", consiste à modifier le contenu de l'ancienne case.
2. "WAIT_BUSY_OLD" va permettre l'attente que le changement s'effectue avant de passer à la suite. Pour cela, le signal "Busy", dois indiquer la bonne réception de la VRAM de ce changement, pour pouvoir passer à l'étape suivante
3. "NEW_POS" permet de modifier le contenu de la nouvelle case.

Le principe de ces trois étapes consiste à "désélectionner" l'ancienne case pour sélectionner la nouvelle, c'est-à-dire effectuer le déplacement du « sprite », tout en conservant les formes, si elles y sont. Pour cela, nous aurons besoin de l'état de la grille avec "Grid_State" et "Grid_Player", ainsi que la valeur de la nouvelle et de l'ancienne position pour pouvoir pointer les bonnes cases et savoir si les cases sont occupées ou non.

Lors d'une validation de cas, nous nous retrouvons à l'état "NEW_OK". Comme précédemment, grâce aux mêmes signaux, nous allons écrire le symbole correspondant au joueur qui à validé la case, dans celle pointée.

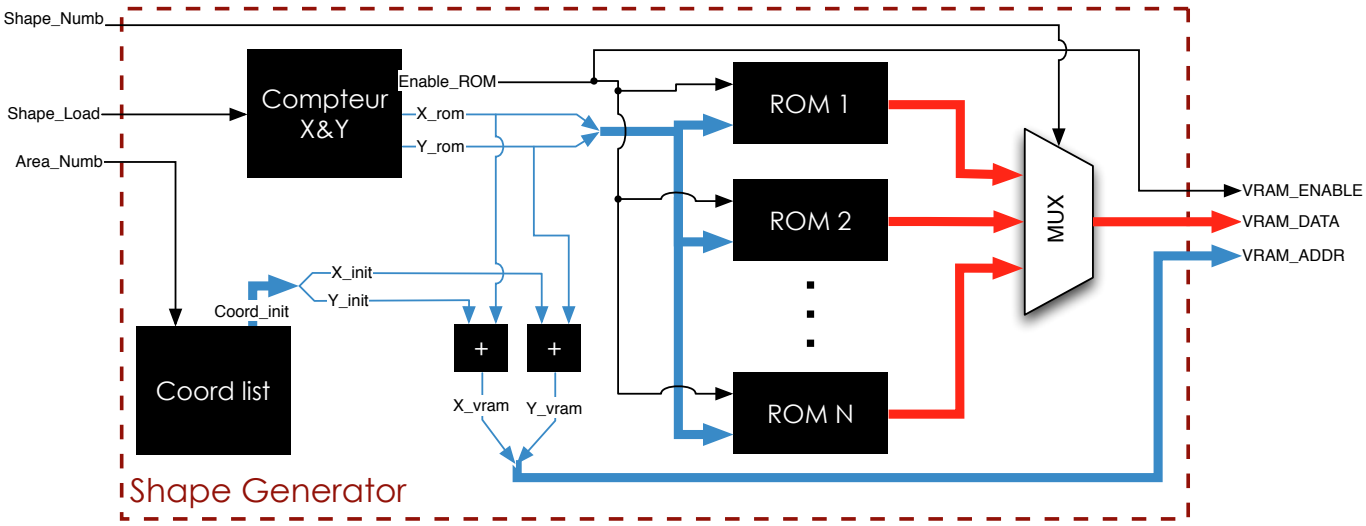
Le dernier cas correspond aux changements de joueur. Cet état est une amélioration du cahier des charges. Elle consiste à afficher à côté de la grille à qui est le tour de jouer.

Pour terminer ces 3 cas, nous allons attendre que la VRAM ait bien pris en compte les changements. Une fois que cela est fait, nous retournons à l'état "WAIT_CHANGE" .

Cette FSM produit trois signaux : "Shape_Load", "Shape_Numb" et Area_Numb".
Le signal "Shape_Load" sert d'autorisation de génération de forme. Il est à l'état '1' à chaque qui doit modifier une le contenue d'une case.
"Shape_Numb" correspond au numéro de la forme qui doit être affiché. Six formes différentes, le symbole du joueur non sélectionné et sélectionné ainsi que le symbole du joueur et la case vide.
"Area_Numb" fournit le numéro qui correspond à l'endroit où se trouve la modification de contenu. L'affichage possède 10 positions, les 9 cases de la grille ainsi que l'endroit qui indique à qui est le tour.

Le générateur de forme

Le générateur de forme est le module qui communique avec la VRAM. Son architecture est illustrée sur la figure ci-dessous :



Architecture du générateur de forme

Ce module réceptionne les trois signaux produits par la FSM : “Shape_Load”, “Area_Num” et Shape_Numb”.

Lors d’une demande de création de formes, le générateur d’image réceptionne l'ordre de créer une image. Un compteur de coordonnées en abscisse et en ordonnée va produire l’ensemble des coordonnées nécessaire à l’affichage de la forme. Les coordonnées générées débutent à 0.

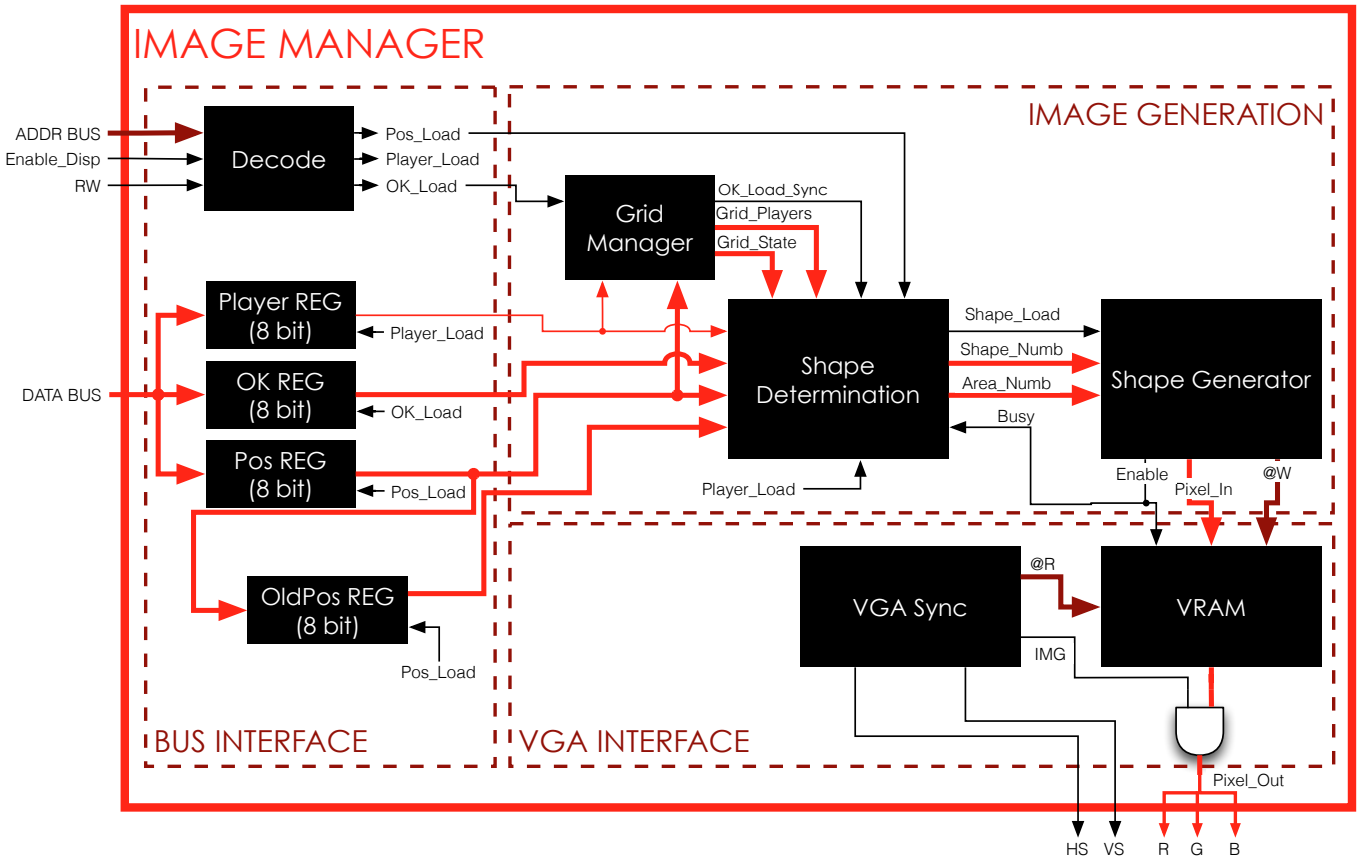
En aval se trouvent les 6 ROMs, contenant l’ensemble des pixels décrivant une forme : le symbole du joueur 1, celui du joueur 2 ainsi que de la case vide, sélectionné et non sélectionné (pour chacune). Ces ROMs fournissent la valeur du pixel adressé par le compteur.

Le choix de la forme s’effectue par un multiplexeur, dont le pilotage est géré par “Shape_Number”.

Il ne faut pas oublier l’adresse correspondant à la valeur du pixel envoyé à la VRAM. Les adresses générées par le compteur débutent à zéro. C’est-à-dire que toutes les formes créées vont être affichées dans le coin supérieur gauche. Pour corriger cela, un sous-module “Coord_list”, possède les coordonnées des pixels se trouvant dans le coin supérieur gauche de toutes les cases de la grille. Le signal “Area_Numb” fournit à ce sous-module, le numéro de la case pour que les coordonnées générées soient ajoutées aux coordonnées produites par le compteur.

Pour conclure, le module de génération d’image au complet a pour rôle d'écrire un sprite dans la VRAM à un emplacement précis, en fonction des ordres reçus du CPU.

Le périphérique de gestion d'affichage est alors le suivant :

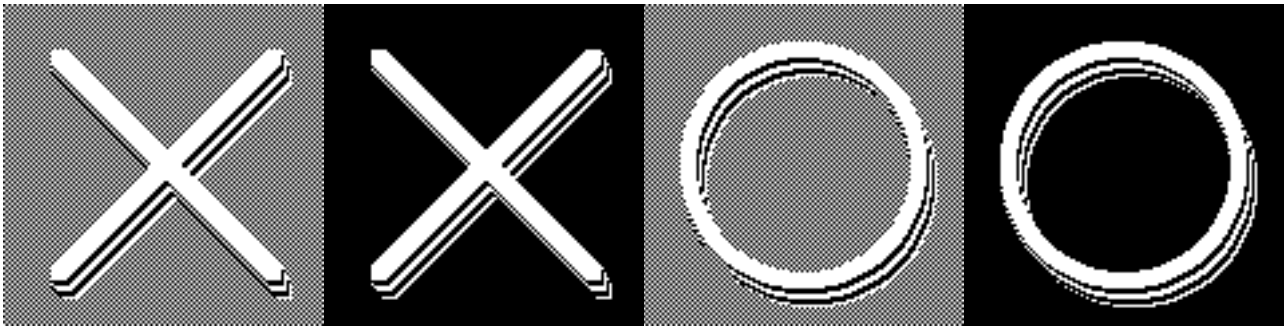


Périphérique de gestion d'affichage

2.5 Quelques améliorations

La première modification que nous avons effectuée concerne la zone d'affichage. Nous étions initialement parties sur l'idée d'une grille simple. Nous avons pu modifier cette zone grâce au code Matlab créé précédemment, en modifiant simplement l'image de fond. Nous avons pu effectuer différents rajouts comment le titre du jeu en haut de l'écran, le nom de l'école ainsi que le titre de la formation et les photos présentes sur les cotés inférieur de l'écran.

De plus, nous avons pu, grâce à Matlab ainsi que Photoshop, styliser les formes utilisées par les différents joueurs :



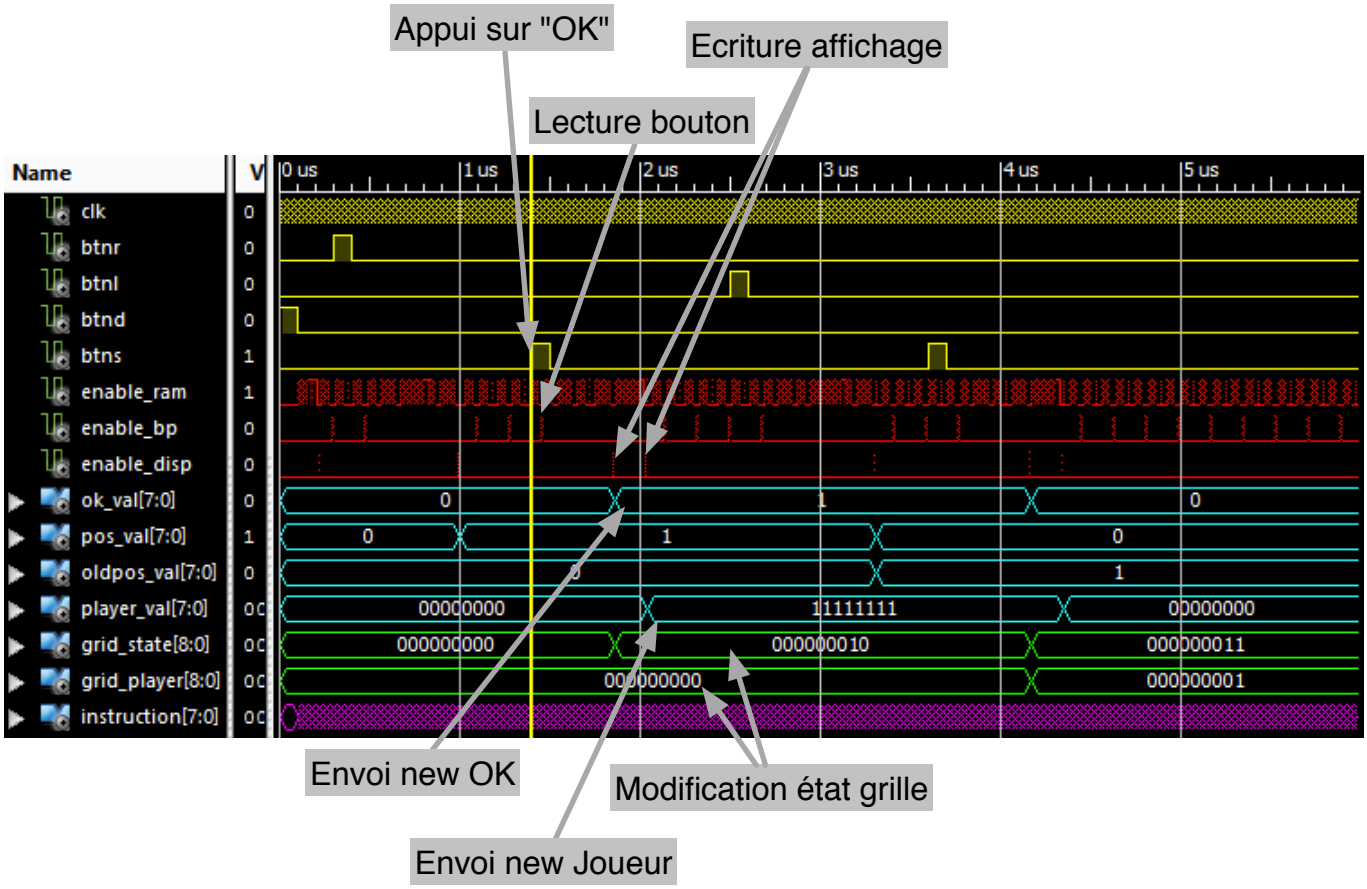
Formes identifiant les différents joueurs

Lors de la manipulation du jeu, nous avons remarqué, qu'il été nécessaire pour les joueurs d'avoir un repère visuel, définissant le joueur a qui été le tour. Pour mettre en place cette modification, nous avons créé un état supplémentaire dans la machine d'état "Shape Determination", qui lorsque le numéro du joueur est modifié, le symbole présent sur l'écran change.

3. Validation du fonctionnement

Outre les 34 fichiers VHDL que nous avons écrits pour décrire le système, nous avons rédigé un total de 17 testbench tout au long de la conception afin de certifier le bon fonctionnement de l'ensemble.

Étonnamment, ce chiffre peut paraître faible, mais il y a plusieurs explications. Tout d'abord, nous avons une dizaine de ROM/RAM presque identiques. Idem pour les différents registres ou multiplexeurs. De plus, notre architecture peut descendre jusqu'à 5 niveaux. Nous avons fait le choix de parfois utiliser le testbench du niveau le plus haut pour tester un niveau légèrement en dessous. Grâce aux outils de simulation puissants, nous pouvons aisément regarder l'évolution de l'ensemble du système comme d'un point très précis. Quoi qu'il en soit, nous pouvons aujourd'hui certifier le bon fonctionnement de l'ensemble des descriptions que nous avons réalisé. Que ce soit grâce aux testbenches fréquents ou grâce à des contrôles visuels. Mais étant donné le grand nombre de composants décrits, nous avons décidé de ne pas aborder leur validation en détail afin de ne pas alourdir ce rapport. Néanmoins, nous proposons une rapide analyse du testbench de l'élément de plus haut niveau. Pour cela, nous avons passé l'ensemble des CE (clock enable) à 100 MHz.



Testbench de haut niveau du système

On observe bien le déroulement du process : un bouton est appuyé, l'information est lue et interprétée par le processeur qui, à son tour, en informe la gestion d'affichage. Cette dernière modifie alors l'état de la grille.

Notons tout de même que nous ne pouvons pas simuler le fonctionnement de l'affichage d'un point de vue macroscopique puisque de nombreux (et longs) compteurs entrent en jeux. Mais ces éléments ont tous été validés indépendamment.

Sur ce même testbench, on peut lire un deuxième appui sur le bouton de validation, et observer les signaux d'état de la grille pour confirmer que l'action du deuxième joueur a été prise en compte (grid_player reçoit un '1' sur la case 0).

4. Conclusion

Un projet passionnant, c'est ce que nous ressentons en cette fin de module. Tout d'abord, nous avons fait le choix de passer de nombreuses heures à réfléchir à notre architecture. Nous pensons aujourd'hui que ce temps passé fut indispensable et indéniablement lié à la réussite de notre projet. Chacune de nos réflexions aboutissant systématiquement à un schéma, le travail d'écriture du VHDL a été grandement simplifié.

Cette architecture a justement été le sujet de nombreuses de nos discussions. Nous avons fait le choix de partir sur un système type "microcontrôleur" où le CPU est le centre de l'appareil, et les périphériques gravitent autour. Nous ne pouvons pas certifier que ce choix était forcément le meilleur, mais affirmons que ce n'était pas une mauvaise idée. En effet, un jeu vidéo est plus facile à réaliser en logiciel. Il fait intervenir des mécanismes et des calculs relativement spécifiques. Or le choix d'établir un processeur généraliste au centre de l'architecture nous permet de porter un maximum d'intelligence dans le logiciel. Ce choix était autant réfléchi que personnel.

Dans l'absolu, avec beaucoup plus de temps, nous aurions aimé concevoir une architecture entièrement générique. Cela nous aurait permis d'écrire la majorité des spécificités du jeu au sein du logiciel. En d'autres termes, nous aurions, avec bien plus de temps, préféré réaliser une console de jeu basée sur ce processeur 8 bits, plus qu'un simple jeu.

Nous avons également été confrontés à un problème que nous n'avions pas prévu. En effet, nous nous sommes retrouvés à court de RAM. Comme nous l'indiquions, nous avions prévu une mémoire vidéo de 512ko. Or le FPGA que nous utilisons ne dispose que de 572kb de RAM. Il nous a alors fallu adapter notre design, et avons choisi de sacrifier la couleur et de privilégier la résolution. Nous aurions pu faire l'inverse. Il est d'ailleurs intéressant de constater la gigantesque évolution des mémoires vidéos de nos cartes graphiques d'ordinateur en parallèle de l'évolution des tailles d'écran et la multiplication des couleurs.

Finalement, ce projet nous conforte dans notre désir de continuer l'apprentissage du VHDL. Nous trouvons en effet fabuleuse l'aisance de décrire des systèmes électroniques complexes en quelques lignes de codes.

Pour terminer, nous avons pris la décision d'utiliser un outil de développement participatif afin de simplifier notre gestion du projet. Malgrès quelques problèmes techniques, GitHub nous a grandement aidé à travailler en parallèle. Ainsi, pour plus de détails, nous vous proposons de vous rendre sur la page du projet pour disposer de l'ensemble des sources (projet open source) : https://github.com/Anozer/Morpion .

Annexes

Assembleur

```
;                       BT next     BT prev     BT val      Autres
INIT:
    NOR ALLONE      ;1  0000 0000
    STA POS         ;1  MEM[pos]
    STA JOUEUR      ;1  MEM[joueur]
    STA AFF_J       ;1  AFF[joueur]
START:
    LDA BP          ;2  1111 1110   1111 1101   1111 1011   1101 1111   11
    ADD ONE         ;1  1111 1111   1111 1110   1111 1100   1110 0000   00
    JCC BPDETECT    ;1  JUMP        JUMP        JUMP        JUMP        00
    JCC START       ;1                                                  JU

BPDETECT:
    NOT             ;1  0000 0000   0000 0001   0000 0011   0001 1111
    ADD ALLONE      ;1  1111 1111   0000 0000C  0000 0010C  0001 1110C
    JCC BT_NEXT     ;1  JUMP        0000 0000   0000 0010   0001 1110
    ADD ALLONE      ;1              1111 1111   0000 0001C  0001 1101C
    JCC BT_PREV     ;1              JUMP        0000 0001   0001 1101
    JCC BT_OK       ;1                          JUMP        JUMP

;                       Position 3 joueur 0
BT_OK:
    LDA POS         ;2  accu        = MEM[pos]
    STA AFF_OK      ;1  AFF[val]    = accu
    LDA JOUEUR      ;2  accu        = MEM[joueur]
    NOT             ;1  accu        = !accu
    STA JOUEUR      ;1  MEM[joueur] = accu
    STA AFF_J       ;1  AFF[joueur] = accu
    JCC START       ;1  retour

;                       Position 2      Position 8
BT_NEXT:
    LDA POS         ;2  0000 0010       0000 1000
    ADD ONE         ;1  0000 0011       0000 1001
    STA POS         ;1  MEM[pos]=accu   MEM[pos]=accu
    ADD VAL_247     ;1  1111 1010       0000 0000C
    JCC SENDPOS     ;1  JUMP            0000 0000
    STA POS         ;1                  MEM[pos]=accu
    JCC SENDPOS     ;1                  JUMP

;                       Position 3      Position 0
BT_PREV:            ;   1111 1111       1111 1111
    ADD ONE         ;1  0000 0000C      0000 0000C
    LDA POS         ;2  0000 0011C      0000 0000C
    ADD ALLONE      ;1  0000 0010C      1111 1111C
    JCC START       ;1  0000 0010       1111 1111
    STA POS         ;1  MEM[pos]=accu   MEM[pos]=accu
    ADD ONE         ;1  0000 0011       0000 0000C
    JCC SENDPOS     ;1  JUMP            0000 0000
    ADD VAL_8       ;1                  0000 1000
    STA POS         ;1                  MEM[pos]=accu
    JCC SENDPOS     ;1                  JUMP


;                       Position 3      Position 0
BT_PREV:            ;   1111 1111       1111 1111
    NOR ALLONE      ;1  0000 0000       0000 0000
```

```
58      ADD POS          ;1  0000 0011       0000 0000
59      ADD ALLONE       ;1  0000 0010C      1111 1111
60      ADD ZERO         ;1  0000 0010       1111 1111
61      STA POS          ;1  MEM[pos]=accu   MEM[pos]=accu
62      ADD ONE          ;1  0000 0011       0000 0000C
63      JCC SENDPOS      ;1  JUMP            0000 0000
64      ADD VAL_8        ;1                  0000 1000
65      STA POS          ;1                  MEM[pos]=accu
66      JCC SENDPOS      ;1                  JUMP
67
68  SENDPOS:
69      LDA POS          ;2  accu = MEM[pos]
70      STA AFF_POSMOD   ;1  AFF[pos]=accu
71      JCC START        ;1  retour
72
73  ; données mémoire
74
75  ; ALLONE     = 1111 1111
76  ; ZERO       = 0000 0000
77  ; ONE        = 0000 0001
78  ; VAL_247    = 1111 0111
79  ; VAL_8      = 0000 1000
80  ; POS        = 0000 0000
81  ; JOUEUR     = 0000 0000
82
83  ; 46 instructions + 7 données = 53 octets
84  ; BP         = 111000 = 0x38
85  ; AFF_J      = 111001 = 0x39
86  ; AFF_POS    = 111010 = 0x3A
87  ; AFF_OK     = 111011 = 0x3B
88
```

Matlab

Init_VRAM

```
clear all
close all
clc

% répertoires source et destination
path_img = 'IMG/VRAM.png';
path_vhdl = 'VHDL/VRAM.vhd';
default_VRAM = 'VHDL/default_VRAM.vhd';
data_type = 'std_logic';

% init
bitsX = 10;
bitsY = 9;

% config image
nb_X = 640;
nb_Y = 480;
cell_width = 120;
border_width = 5;
posJx = 4;
posJy = 120;
victJx = 516;
victJy = 120;

% init coord zones pour shape generator
x1 = ((nb_X - (cell_width*3+4*border_width))/2) +border_width-1;
x2 = x1+cell_width+border_width;
x3 = x2+cell_width+border_width;
x4 = x3+cell_width+border_width;
y1 = ((nb_Y - (cell_width*3+4*border_width))/2) +border_width-1;
y2 = y1+cell_width+border_width;
y3 = y2+cell_width+border_width;
y4 = y3+cell_width+border_width;


fprintf('\t0 => "%s",\t-- x%d y%d\n' ,coord2addr(x1,y1,'bin',bitsX,bitsY),(x1),y1);
fprintf('\t1 => "%s",\t-- x%d y%d\n' ,coord2addr(x2,y1,'bin',bitsX,bitsY),(x2),y1);
fprintf('\t2 => "%s",\t-- x%d y%d\n' ,coord2addr(x3,y1,'bin',bitsX,bitsY),(x3),y1);
fprintf('\t3 => "%s",\t-- x%d y%d\n' ,coord2addr(x1,y2,'bin',bitsX,bitsY),(x1),y2);
fprintf('\t4 => "%s",\t-- x%d y%d\n' ,coord2addr(x2,y2,'bin',bitsX,bitsY),(x2),y2);
fprintf('\t5 => "%s",\t-- x%d y%d\n' ,coord2addr(x3,y2,'bin',bitsX,bitsY),(x3),y2);
fprintf('\t6 => "%s",\t-- x%d y%d\n' ,coord2addr(x1,y3,'bin',bitsX,bitsY),(x1),y3);
fprintf('\t7 => "%s",\t-- x%d y%d\n' ,coord2addr(x2,y3,'bin',bitsX,bitsY),(x2),y3);
fprintf('\t8 => "%s",\t-- x%d y%d\n' ,coord2addr(x3,y3,'bin',bitsX,bitsY),(x3),y3);
fprintf('\t9 => "%s",\t-- x%d y%d : info joueur\n',coord2addr(posJx,  posJy,
'bin',bitsX,bitsY),posJx,posJy);
fprintf('\t10 => "%s"\t-- x%d y%d : victoire\n'   ,coord2addr(victJx,
victJy,'bin',bitsX,bitsY),victJx,victJy);

%%


%constantes
nb_bits = bitsX+bitsY;
addr_size  = sprintf('%d downto 0', nb_bits-1);
array_size = sprintf('(2**%d)-1 downto 0', nb_bits);
```

```
% Récup du VHDL de la VRAM à compléter
default_VRAM_file = fopen(default_VRAM, 'r+');
default_VRAM = fread(default_VRAM_file,'*char');
fclose(default_VRAM_file);

% calcul du VHDL des datas
disp('Generation VHDL...');
vhd = img2vhdl(path_img,bitsX,bitsY);

% completion du VHDL
disp('Ecriture du fichier...');
VRAM_vhdl = default_VRAM';
VRAM_vhdl = regexprep(VRAM_vhdl,'%ADDR_SIZE%', addr_size);
VRAM_vhdl = regexprep(VRAM_vhdl,'%ARRAY_SIZE%', array_size);
VRAM_vhdl = regexprep(VRAM_vhdl,'%DATA_TYPE%', data_type);
VRAM_vhdl = regexprep(VRAM_vhdl,'%DATAS%', vhd);

vhd_file = fopen(path_vhdl,'w+');
fwrite(vhd_file, VRAM_vhdl);
fclose(vhd_file);

fprintf('done\n');
```

Init_ROMS

```
clear all
close all
clc

% principe :
% 1) Analyse de chaque image du dossier source
% 2) Récupère un fichier VHDL décrivant une ROM
% 3) Pour chaque image, on complète crée une ROM VHDL au complet


% répertoires source et destination
path_img = 'IMG';
path_vhdl = 'VHDL';
default_ROM = 'VHDL/default_ROM.vhd';

% init
nb_bitsX = 7;
nb_bitsY = 7;

% types de fichier à convertir
types = {'jpg', 'jpeg', 'png', 'bmp', 'gif', 'tiff'};
exclude = {'VRAM.png'};

% tailles des elements VHDL
rom_addr_tab = ['(2**' num2str(nb_bitsX+nb_bitsY) ')-1 downto 0'];
rom_addr_size = [num2str(nb_bitsX+nb_bitsY-1) ' downto 0'];

% Récup du VHDL de la rom à compléter
default_ROM_file = fopen(default_ROM, 'r+');
default_ROM = fread(default_ROM_file,'*char');
fclose(default_ROM_file);



% récup de tous les fichiers du dossier
dir_img = dir(path_img);

% pour chaque fichier du dossier
for i=1:length(dir_img)

    % récup du nom et de l'extension
    img = dir_img(i).name;
    type  = regexprep(img,'.*\.(.+)$','$1');
    name = regexprep(img,'(.*)\..+$','$1');
    img_path = [path_img '/' img];
    rom_name = ['ROM_' name];
    rom_file = [rom_name '.vhd'];
    rom_path = [path_vhdl '/' rom_file];

    % check si l'extension est présente dans la liste
    if (~sum(strcmp(type,types)))
        continue
    end;

    if (sum(strcmp(img,exclude)))
        fprintf('/!\\ %s will be excluded.\n\n',img);
        continue
    end;
```

```
    % Récup du VHDL
    fprintf('Image:\t%s\nROM:\t%s\n...\n\n',img_path,rom_path);
    rom_data = img2vhdl(img_path, nb_bitsX, nb_bitsY);

    % Remplacement de la rom par défaut
    rom_vhdl = default_ROM';
    rom_vhdl = regexprep(rom_vhdl,'%ROM_NAME%', rom_name);
    rom_vhdl = regexprep(rom_vhdl,'%DATAS%', rom_data);
    rom_vhdl = regexprep(rom_vhdl,'%ROM_ADDR_TAB%', rom_addr_tab);
    rom_vhdl = regexprep(rom_vhdl,'%ROM_ADDR_SIZE%', rom_addr_size);
    rom_vhdl = regexprep(rom_vhdl,'%ROM_DATA_TYPE%', 'std_logic');
    %rom_vhdl = regexprep(rom_vhdl,'%ROM_DATA_TYPE%', 'std_logic_vector(7 downto
0)');

    % Création du fichier VHDL pour la nouvelle ROM
    rom_file_des = fopen(rom_path, 'w+');
    fwrite(rom_file_des, rom_vhdl);
    fclose(rom_file_des);
end;


fprintf('done\n');
```

img2vhdl

```
function [ result ] = img2vhdl(img_path, nb_bits_x, nb_bits_y)
    %img2vhdl
    %   -> transforme l'image donnée en VHDL pour ROM 8 bits
    %   img_path  : chemin de l'image
    %   nb_bits_x : nombre de bits pour l'adresse des X
    %   nb_bits_y : nombre de bits pour l'adresse des Y

    % recup de l'image
    img = imread(img_path);

    % passage de 8 bits à 3 ou 2
    img_R = (img(:,:,1)/255);
    img_G = (img(:,:,2)/255);
    img_B = (img(:,:,3)/255);

    % Pour chaque x/y
    data = [];
    for y = 1:size(img,1)
        for x = 1:size(img,2)
            % calcul de l'adresse du pixel
            addr = (y-1)*(2^nb_bits_x) + (x-1);

            % calcul de la valeur du pixel
            pixel = img_R(y,x) | img_G(y,x)| img_B(y,x);

            % création du VHDL du pixel
            line = sprintf('\t\t%d => ''%d'',\n',addr,pixel);
            data = [data line];
        end;
    end;

    result = data;
end
```

VHDL

```
1   ----------------------------------------------------------------------------------
2   -- Company: ENSEIRB-MATMECA
3   -- Engineer: Sylvain MARIEL (sylvain.mariel@otmax.fr)
4   -- Engineer: Thomas MOREAU  (thomas.moreau-33@hotmail.fr)
5
6   -- Create Date:    21/05/2013
7   -- Design Name:
8   -- Module Name:    Morpion - Behavioral
9   -- Project Name:
10  -- Target DeviCEs:
11  -- Tool versions:
12  -- Description:
13  --
14  -- Dependencies:
15  --
16  -- Revision:
17  -- Revision 0.01 - File Created
18  -- Additional Comments:
19  --
20  ----------------------------------------------------------------------------------
21  library IEEE;
22  use IEEE.STD_LOGIC_1164.ALL;
23  use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25  -- Uncomment the following library declaration if using
26  -- arithmetic functions with Signed or Unsigned values
27  --use IEEE.NUMERIC_STD.ALL;
28
29  -- Uncomment the following library declaration if instantiating
30  -- any Xilinx primitives in this code.
31  --library UNISIM;
32  --use UNISIM.VComponents.all;
33
34  entity Morpion is
35     port (Clk         : IN  STD_LOGIC;
36           BTNR        : IN  STD_LOGIC;
37           BTNL        : IN  STD_LOGIC;
38           BTND        : IN  STD_LOGIC;
39           BTNS        : IN  STD_LOGIC;
40           VGA_RED     : OUT STD_LOGIC_VECTOR(2 downto 0);
41           VGA_GREEN   : OUT STD_LOGIC_VECTOR(2 downto 0);
42           VGA_BLUE    : OUT STD_LOGIC_VECTOR(1 downto 0);
43           VGA_HS      : OUT STD_LOGIC;
44           VGA_VS      : OUT STD_LOGIC);
45  end Morpion;
46
47  architecture Behavioral of Morpion is
48
49     component CPU_8bits
50        Port (Reset          : in  STD_LOGIC;
51              Clk            : in  STD_LOGIC;
52              CE             : in  STD_LOGIC;
53              data_in        : in  STD_LOGIC_VECTOR(7 downto 0);
54              data_out       : out STD_LOGIC_VECTOR(7 downto 0);
55              addr           : out STD_LOGIC_VECTOR(5 downto 0);
56              Enable         : out STD_LOGIC;
57              RW             : out STD_LOGIC);
```

```
58      end component;
59
60      component BP
61         Port (Clk            : in  STD_LOGIC;
62               CE             : in  STD_LOGIC;
63               Reset          : in  STD_LOGIC;
64               BP_NEXT        : IN  STD_LOGIC;
65               BP_PREV        : IN  STD_LOGIC;
66               BP_OK          : IN  STD_LOGIC;
67               Enable         : IN  STD_LOGIC;
68               RW             : IN  STD_LOGIC;
69               DataBus_toCPU  : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
70      end component;
71
72      component busArbiter
73         Port (Enable      : in  STD_LOGIC;
74               AddrBus     : in  STD_LOGIC_VECTOR(5 downto 0);
75               Enable_RAM  : out STD_LOGIC;
76               Enable_BP   : out STD_LOGIC;
77               Enable_DISP : out STD_LOGIC);
78      end component;
79
80      component RAM_56
81         Port (AddrBus           : in  STD_LOGIC_VECTOR (5 downto 0);
82               DataBus_fromCPU   : in  STD_LOGIC_VECTOR (7 downto 0);
83               RW                : in  STD_LOGIC;
84               ENABLE            : in  STD_LOGIC;
85               clk               : in  STD_LOGIC;
86               Ce                : in  STD_LOGIC;
87               DataBus_toCPU     : out STD_LOGIC_VECTOR (7 downto 0));
88      end component;
89
90      component Display
91         Port (Clk               : in  STD_LOGIC;
92               CE                : in  STD_LOGIC;
93               Reset             : in  STD_LOGIC;
94               Enable            : IN  STD_LOGIC;
95               RW                : IN  STD_LOGIC;
96               AddrBus           : IN  STD_LOGIC_VECTOR (5 downto 0);
97               DataBus_fromCPU   : IN  STD_LOGIC_VECTOR (7 downto 0);
98               VGA_HS            : OUT STD_LOGIC;
99               VGA_VS            : OUT STD_LOGIC;
100              VGA_Red           : OUT STD_LOGIC_VECTOR (2 downto 0);
101              VGA_Green         : OUT STD_LOGIC_VECTOR (2 downto 0);
102              VGA_Blue          : OUT STD_LOGIC_VECTOR (1 downto 0));
103     end component;
104
105
106     signal Reset            : std_logic;
107     signal DataBus_cpu2p    : std_logic_vector(7 downto 0);
108     signal DataBus_p2cpu    : std_logic_vector(7 downto 0);
109     signal DataBus_bp2cpu   : std_logic_vector(7 downto 0);
110     signal DataBus_ram2cpu  : std_logic_vector(7 downto 0);
111     signal AddrBus          : std_logic_vector(5 downto 0);
112     signal RW               : std_logic;
113     signal Enable           : std_logic;
114     signal Enable_ram       : std_logic;
```

```
115      signal Enable_bp        : std_logic;
116      signal Enable_disp      : std_logic;
117
118      signal Cpt_ce           : std_logic_vector(14 downto 0);
119      signal CE_100M          : std_logic;
120      signal CE_25M           : std_logic;
121      signal CE_3K            : std_logic;
122
123   begin
124      Reset <= BTND;
125
126      CE_100M <= '1';
127
128      -- Compteur pour tous les CE
129      process (Clk,Reset) begin
130         if (Reset = '1') then
131            Cpt_ce <= (others => '0');
132         elsif (Clk'event and Clk = '1') then
133            Cpt_ce <= Cpt_ce + 1;
134         end if;
135      end process;
136
137      -- Division par 4 : 100 MHz -> 25 MHz
138      process (Cpt_ce) begin
139         if (Cpt_ce(1 downto 0) = "11") then
140            CE_25M <= '1';
141         else
142            CE_25M <= '0';
143         end if;
144      end process;
145
146      -- Division par 2^15 : 100 MHz -> 3.052 KHz
147      process (Cpt_ce) begin
148         if (Cpt_ce(14 downto 0) = (2**15)-1) then
149            CE_3K <= '1';
150         else
151            CE_3K <= '0';
152         end if;
153      end process;
154
155
156
157      -- MUX du bus de données (periph vers cpu)
158      DataBus_p2cpu <=  DataBus_ram2cpu   WHEN Enable_ram = '1' ELSE
159                        DataBus_bp2cpu    WHEN Enable_bp = '1' ELSE
160                        (others => '0');
161
162      The_CPU : CPU_8bits port map (
163         Reset,
164         Clk,
165         CE_3K,
166         DataBus_p2cpu,
167         DataBus_cpu2p,
168         AddrBus,
169         Enable,
170         RW
171      );
```

```
172
173      The_BP : BP port map (
174         Clk,
175         CE_3K,
176         Reset,
177         BTNR,
178         BTNL,
179         BTNS,
180         Enable_bp,
181         RW,
182         DataBus_bp2cpu
183      );
184
185      The_busArbiter : busArbiter port map (
186         Enable,
187         AddrBus,
188         Enable_ram,
189         Enable_bp,
190         Enable_disp
191      );
192
193      The_RAM : RAM_56 port map (
194         AddrBus,
195         DataBus_cpu2p,
196         RW,
197         Enable_ram,
198         Clk,
199         CE_3K,
200         DataBus_ram2cpu
201      );
202
203      The_display : Display port map (
204         Clk,
205         CE_25M,
206         Reset,
207         Enable_disp,
208         RW,
209         AddrBus,
210         DataBus_cpu2p,
211         VGA_HS,
212         VGA_VS,
213         VGA_Red,
214         VGA_Green,
215         VGA_Blue
216      );
217
218   end Behavioral;
219
220
```

```
1  ----------------------------------------------------------------------------------
2  -- Company: ENSEIRB-MATMECA
3  -- Engineer: Sylvain MARIEL (sylvain.mariel@otmax.fr)
4  -- Engineer: Thomas MOREAU  (thomas.moreau-33@hotmail.fr)
5
6  -- Create Date:    21/05/2013
7  -- Design Name:
8  -- Module Name:    VRAM - Behavioral
9  -- Project Name:
10 -- Target Devices:
11 -- Tool versions:
12 -- Description:
13 --
14 -- Dependencies:
15 --
16 -- Revision:
17 -- Revision 0.01 - File Created
18 -- Additional Comments:
19 --
20 ----------------------------------------------------------------------------------
21 library ieee;
22 use ieee.std_logic_1164.all;
23 use ieee.std_logic_unsigned.all;
24 use IEEE.NUMERIC_STD.ALL;
25
26 entity VRAM is
27    port (Clk      : in  std_logic;
28          CE       : in  std_logic;
29          Enable_w : in  std_logic;
30          Addr_w   : in  std_logic_vector  (18 downto 0);
31          Addr_r   : in  std_logic_vector  (18 downto 0);
32          Data_in  : in  std_logic;
33          Data_out : out std_logic);
34 end VRAM;
35
36 architecture Behavioral of VRAM is
37
38    type ram_type is array ((2**19)-1 downto 0) of std_logic;
39    signal VRAM: ram_type := (
40             --Initialisation
41             );
42 begin
43
44    process (Clk)
45    begin
46       if (Clk'event and Clk = '1') then
47          if (CE = '1') then
48             if (Enable_w = '1') then
49                VRAM (to_integer(unsigned(Addr_w))) <= Data_in;
50             end if;
51             Data_out <=  VRAM(to_integer(unsigned(Addr_r)));
52          else
53             NULL;
54          end if;
55       end if;
56    end process;
57
```

```
58   end Behavioral;
59
60
61
```

```
1   ----------------------------------------------------------------------------------
2   -- Company: ENSEIRB-MATMECA
3   -- Engineer: Sylvain MARIEL (sylvain.mariel@otmax.fr)
4   -- Engineer: Thomas MOREAU  (thomas.moreau-33@hotmail.fr)
5
6   -- Create Date:    21/05/2013
7   -- Design Name:
8   -- Module Name:    Disp_VGAsync - Behavioral
9   -- Project Name:
10  -- Target Devices:
11  -- Tool versions:
12  -- Description:
13  --
14  -- Dependencies:
15  --
16  -- Revision:
17  -- Revision 0.01 - File Created
18  -- Additional Comments:
19  --
20  ----------------------------------------------------------------------------------
21  library IEEE;
22  use IEEE.STD_LOGIC_1164.ALL;
23
24  -- Uncomment the following library declaration if using
25  -- arithmetic functions with Signed or Unsigned values
26  use IEEE.NUMERIC_STD.ALL;
27
28  -- Uncomment the following library declaration if instantiating
29  -- any Xilinx primitives in this code.
30  --library UNISIM;
31  --use UNISIM.VComponents.all;
32
33  entity Disp_VGAsync is
34     Port (Clk         : in  STD_LOGIC;
35           CE          : in  STD_LOGIC;
36           Reset       : in  STD_LOGIC;
37           IMG         : OUT STD_LOGIC;
38           HS          : OUT STD_LOGIC;
39           VS          : OUT STD_LOGIC;
40           VRAM_addr   : out STD_LOGIC_VECTOR(18 downto 0));
41  end Disp_VGAsync;
42
43  architecture Behavioral of Disp_VGAsync is
44
45  subtype coordX is integer range 0 to 800;
46  subtype coordY is integer range 0 to 521;
47  signal comptX : coordX := 0;
48  signal comptY : coordY := 0;
49  signal X : std_logic_vector(9 downto 0);
50  signal Y : std_logic_vector(8 downto 0);
51  signal img_x : std_logic;
52  signal img_y : std_logic;
53
54
55  begin
56
57     VRAM_addr <= Y & X;
```

```
58       img <= img_x AND img_y;
59
60       process (Clk, Reset)
61       begin
62          if (Reset = '1') then
63             comptX <= 0;
64             comptY <= 0;
65          elsif (Clk'event and Clk='1') then
66             if(CE = '1') then
67                if comptX < 799 then
68                   comptX <= comptX+1;
69                else
70                   comptX<= 0;
71                end if;
72
73                if comptX=0 then
74                   if comptY<520 then
75                      comptY <= comptY+1;
76                   else
77                      comptY <= 0;
78                   end if;
79                end if;
80             end if;
81          end if;
82       end process;
83
84
85       -- Génération de HSYNC + coord X
86       process (comptX)
87       begin
88          -- Display
89          if(comptX < 640) then
90             HS <= '1';
91             X <= std_logic_vector(to_unsigned(comptX, 10));
92             img_x <= '1';
93          else
94             img_x <= '0';
95
96             -- Front Porsh
97             if(comptX < (640 + 16)) then
98                HS <= '1';
99                X <= (others => '0');
100            else
101               -- Pulse
102               if(comptX < (640 + 16 + 96)) then
103                  HS <= '0';
104                  X <= (others => '0');
105               -- Back Porsh
106               else
107                  HS <= '1';
108                  X <= (others => '0');
109               end if;
110            end if;
111         end if;
112      end process;
113
114
```

```
115      -- Génération de VSYNC + coord Y
116      process (comptY)
117      begin
118         -- Display
119         if(comptY < 480) then
120            VS <= '1';
121            Y <= std_logic_vector(to_unsigned(comptY, 9));
122            img_y <= '1';
123         else
124            img_y <= '0';
125
126            -- Front Porsh
127            if(comptY < (480 + 10)) then
128               VS <= '1';
129            Y <= (others => '0');
130            else
131               -- Pulse
132               if(comptY < (480 + 10 + 2)) then
133                  VS <= '0';
134                  Y <= (others => '0');
135               -- Back Porsh
136               else
137                  VS <= '1';
138                  Y <= (others => '0');
139               end if;
140            end if;
141         end if;
142      end process;
143
144   end Behavioral;
145
146
```

```
1   ----------------------------------------------------------------------------------
2   -- Company: ENSEIRB-MATMECA
3   -- Engineer: Sylvain MARIEL (sylvain.mariel@otmax.fr)
4   -- Engineer: Thomas MOREAU  (thomas.moreau-33@hotmail.fr)
5
6   -- Create Date:    21/05/2013
7   -- Design Name:
8   -- Module Name:    Disp_VGAinterface - Behavioral
9   -- Project Name:
10  -- Target Devices:
11  -- Tool versions:
12  -- Description:
13  --
14  -- Dependencies:
15  --
16  -- Revision:
17  -- Revision 0.01 - File Created
18  -- Additional Comments:
19  --
20  ----------------------------------------------------------------------------------
21  library IEEE;
22  use IEEE.STD_LOGIC_1164.ALL;
23
24  -- Uncomment the following library declaration if using
25  -- arithmetic functions with Signed or Unsigned values
26  --use IEEE.NUMERIC_STD.ALL;
27
28  -- Uncomment the following library declaration if instantiating
29  -- any Xilinx primitives in this code.
30  --library UNISIM;
31  --use UNISIM.VComponents.all;
32
33  entity Disp_VGAinterface is
34     Port (Clk            : in  STD_LOGIC;
35           CE             : in  STD_LOGIC;
36           Reset          : in  STD_LOGIC;
37           VRAM_enableW   : in  STD_LOGIC;
38           VRAM_addrW     : in  STD_LOGIC_VECTOR(18 downto 0);
39           VRAM_dataIn    : in  STD_LOGIC_VECTOR(7 downto 0);
40           HS             : OUT STD_LOGIC;
41           VS             : OUT STD_LOGIC;
42           VRAM_dataOut   : OUT STD_LOGIC_VECTOR(7 downto 0));
43  end Disp_VGAinterface;
44
45  architecture Behavioral of Disp_VGAinterface is
46
47     component Disp_VGAsync
48        Port (Clk         : in  STD_LOGIC;
49              CE          : in  STD_LOGIC;
50              Reset       : in  STD_LOGIC;
51              IMG         : OUT STD_LOGIC;
52              HS          : OUT STD_LOGIC;
53              VS          : OUT STD_LOGIC;
54              VRAM_addr   : out STD_LOGIC_VECTOR(18 downto 0));
55     end component;
56
57     component VRAM
```

```
58          port (Clk      : in  std_logic;
59             CE          : in  std_logic;
60             Enable_w    : in  std_logic;
61             Addr_w      : in  std_logic_vector  (18 downto 0);
62             Addr_r      : in  std_logic_vector  (18 downto 0);
63             Data_in     : in  std_logic;
64             Data_out    : out std_logic);
65       end component;
66
67       signal VRAM_addrR : std_logic_vector(18 downto 0);
68       signal pixel_out : std_logic;
69       signal pixel_in : std_logic;
70       signal img : std_logic;
71    begin
72       VRAM_dataOut <= (others => pixel_out) WHEN img='1' ELSE "00000000";
73       pixel_in <= VRAM_dataIn(0);
74
75       VGA_sync : Disp_VGAsync
76       port map (
77          Clk         => Clk,
78          CE          => Ce,
79          Reset       => Reset,
80          IMG         => img,
81          HS          => HS,
82          VS          => VS,
83          VRAM_addr   => VRAM_addrR);
84
85
86       VGA_VRAM : VRAM
87       port map (
88          Clk         => Clk,
89          CE          => Ce,
90          Enable_w    => VRAM_enableW,
91          Addr_w      => VRAM_addrW,
92          Addr_r      => VRAM_addrR,
93          Data_in     => pixel_in,
94          Data_out    => pixel_out);
95
96    end Behavioral;
97
98
```

```
1  ----------------------------------------------------------------------------------
2  -- Company: ENSEIRB-MATMECA
3  -- Engineer: Sylvain MARIEL (sylvain.mariel@otmax.fr)
4  -- Engineer: Thomas MOREAU  (thomas.moreau-33@hotmail.fr)
5
6  -- Create Date:    21/05/2013
7  ----------------------------------------------------------------------------------
8
9  library IEEE;
10 use ieee.std_logic_1164.all;
11 use ieee.std_logic_unsigned.all;
12 use IEEE.NUMERIC_STD.ALL;
13
14 entity ROM_O is
15    port (CLK : in std_logic;
16          EN : in std_logic;
17          ADDR : in std_logic_vector(13 downto 0);
18          DATA : out std_logic);
19 end ROM_O;
20
21 architecture Behavioral of ROM_O is
22
23 type zone_memoire is array ((2**14)-1 downto 0) of std_logic;
24 constant ROM: zone_memoire := (
25       --Initialisation
26
27 begin
28
29    -- process ROM
30    process (CLK)
31    begin
32       if (CLK'event and CLK = '1') then
33          if (EN = '1') then
34             DATA <= ROM(to_integer(unsigned(ADDR)));
35          end if;
36       end if;
37    end process;
38
39 end Behavioral;
40
41
```

```
1   ----------------------------------------------------------------------------------
2   -- Company: ENSEIRB-MATMECA
3   -- Engineer: Sylvain MARIEL (sylvain.mariel@otmax.fr)
4   -- Engineer: Thomas MOREAU  (thomas.moreau-33@hotmail.fr)
5
6   -- Create Date:    21/05/2013
7   -- Design Name:
8   -- Module Name:    MUX_8to1 - Behavioral
9   -- Project Name:
10  -- Target Devices:
11  -- Tool versions:
12  -- Description:
13  --
14  -- Dependencies:
15  --
16  -- Revision:
17  -- Revision 0.01 - File Created
18  -- Additional Comments:
19  --
20  ----------------------------------------------------------------------------------
21  library IEEE;
22  use IEEE.STD_LOGIC_1164.ALL;
23
24  -- Uncomment the following library declaration if using
25  -- arithmetic functions with Signed or Unsigned values
26  --use IEEE.NUMERIC_STD.ALL;
27
28  -- Uncomment the following library declaration if instantiating
29  -- any Xilinx primitives in this code.
30  --library UNISIM;
31  --use UNISIM.VComponents.all;
32
33  entity MUX_8to1 is
34      Port ( Data_In_0 : in  STD_LOGIC_VECTOR (7 downto 0);
35             Data_In_1 : in  STD_LOGIC_VECTOR (7 downto 0);
36             Data_In_2 : in  STD_LOGIC_VECTOR (7 downto 0);
37             Data_In_3 : in  STD_LOGIC_VECTOR (7 downto 0);
38             Data_In_4 : in  STD_LOGIC_VECTOR (7 downto 0);
39             Data_In_5 : in  STD_LOGIC_VECTOR (7 downto 0);
40             Data_In_6 : in  STD_LOGIC_VECTOR (7 downto 0);
41             Data_In_7 : in  STD_LOGIC_VECTOR (7 downto 0);
42             Channel   : in  STD_LOGIC_VECTOR (2 downto 0);
43             Data_Out  : out STD_LOGIC_VECTOR (7 downto 0));
44  end MUX_8to1;
45
46  architecture Behavioral of MUX_8to1 is
47
48  begin
49
50
51     with Channel select
52        Data_Out <= Data_In_0 when "000",
53                    Data_In_1 when "001",
54                    Data_In_2 when "010",
55                    Data_In_3 when "011",
56                    Data_In_4 when "100",
57                    Data_In_5 when "101",
```

```
58                      Data_In_6 when "110",
59                      Data_In_7 when "111",
60                      "00000000" when others;
61
62    end Behavioral;
63
64
```

```
1  ----------------------------------------------------------------------------------
2  -- Company: ENSEIRB-MATMECA
3  -- Engineer: Sylvain MARIEL (sylvain.mariel@otmax.fr)
4  -- Engineer: Thomas MOREAU  (thomas.moreau-33@hotmail.fr)
5
6  -- Create Date:    21/05/2013
7  -- Design Name:
8  -- Module Name:    Disp_ImgGen_ShapeGenerator_Cpt - Behavioral
9  -- Project Name:
10 -- Target Devices:
11 -- Tool versions:
12 -- Description:
13 --
14 -- Dependencies:
15 --
16 -- Revision:
17 -- Revision 0.01 - File Created
18 -- Additional Comments:
19 --
20 ----------------------------------------------------------------------------------
21 library IEEE;
22 use IEEE.STD_LOGIC_1164.ALL;
23
24 -- Uncomment the following library declaration if using
25 -- arithmetic functions with Signed or Unsigned values
26 use IEEE.NUMERIC_STD.ALL;
27
28 -- Uncomment the following library declaration if instantiating
29 -- any Xilinx primitives in this code.
30 --library UNISIM;
31 --use UNISIM.VComponents.all;
32
33 entity Disp_ImgGen_ShapeGenerator_Cpt is
34    port (
35       Clk         : IN  STD_LOGIC;
36       Reset       : IN  STD_LOGIC;
37       Ce          : IN  STD_LOGIC;
38       Load        : IN  STD_LOGIC;
39       X_out       : OUT STD_LOGIC_VECTOR(6 downto 0);
40       Y_out       : OUT STD_LOGIC_VECTOR(6 downto 0);
41       Counting    : OUT STD_LOGIC);
42 end Disp_ImgGen_ShapeGenerator_Cpt;
43
44 architecture Behavioral of Disp_ImgGen_ShapeGenerator_Cpt is
45
46
47 constant X_max : integer := 119;
48 constant Y_max : integer := 119;
49
50
51 subtype coordX is integer range 0 to X_max;
52 subtype coordY is integer range 0 to Y_max;
53 signal comptX : coordX;
54 signal comptY : coordY;
55 signal enable_cpt : std_logic;
56
57
```

```
58   begin
59      Counting <= enable_cpt;
60      X_out <= std_logic_vector(to_unsigned(comptX, 7)) WHEN enable_cpt = '1' ELSE (
     others => '0');
61      Y_out <= std_logic_vector(to_unsigned(comptY, 7)) WHEN enable_cpt = '1' ELSE (
     others => '0');
62
63      -- Contrôle des compteurs sur enable_cpt
64      process (Clk, Reset)
65      begin
66         if (Reset = '1') then
67            enable_cpt <= '0';
68         elsif (Clk'event and Clk='1') then
69            if(CE = '1') then
70               if(Load = '1') then
71                  enable_cpt <= '1';
72               elsif (comptY = Y_max) then
73                  if(comptX = X_max) then
74                     enable_cpt <= '0';
75                  end if;
76               end if;
77            end if;
78         end if;
79      end process;
80
81      -- Compteurs X et Y
82      process (Clk, Reset)
83      begin
84         if (Reset = '1') then
85            comptX <= 0;
86            comptY <= 0;
87         elsif (Clk'event and Clk='1') then
88            if(CE = '1') then
89               if(enable_cpt = '1') then
90                  if comptX < X_max then
91                     comptX <= comptX+1;
92                  else
93                     comptX<= 0;
94                  end if;
95
96                  if comptX=X_max then
97                     if comptY< Y_max then
98                        comptY <= comptY+1;
99                     else
100                       comptY <= 0;
101                    end if;
102                 end if;
103              end if;
104           end if;
105        end if;
106     end process;
107
108
109  end Behavioral;
110
111
```

```
1  ----------------------------------------------------------------------------------
2  -- Company: ENSEIRB-MATMECA
3  -- Engineer: Sylvain MARIEL (sylvain.mariel@otmax.fr)
4  -- Engineer: Thomas MOREAU  (thomas.moreau-33@hotmail.fr)
5
6  -- Create Date:    21/05/2013
7  -- Design Name:
8  -- Module Name:    Disp_ImgGen_ShapeGenerator - Behavioral
9  -- Project Name:
10 -- Target Devices:
11 -- Tool versions:
12 -- Description:
13 --
14 -- Dependencies:
15 --
16 -- Revision:
17 -- Revision 0.01 - File Created
18 -- Additional Comments:
19 --
20 ----------------------------------------------------------------------------------
21 library IEEE;
22 use IEEE.STD_LOGIC_1164.ALL;
23 use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25 -- Uncomment the following library declaration if using
26 -- arithmetic functions with Signed or Unsigned values
27 use IEEE.NUMERIC_STD.ALL;
28
29 -- Uncomment the following library declaration if instantiating
30 -- any Xilinx primitives in this code.
31 --library UNISIM;
32 --use UNISIM.VComponents.all;
33
34 entity Disp_ImgGen_ShapeGenerator is
35    port (
36       Clk         : IN  STD_LOGIC;
37       Reset       : IN  STD_LOGIC;
38       Ce          : IN  STD_LOGIC;
39       Shape_Load  : IN  STD_LOGIC;
40       Shape_Numb  : IN  STD_LOGIC_VECTOR(2  downto 0);
41       Area_Numb   : IN  STD_LOGIC_VECTOR(7 downto 0);
42       VRAM_data   : OUT STD_LOGIC_VECTOR(7  downto 0);
43       VRAM_addr   : OUT STD_LOGIC_VECTOR(18 downto 0);
44       VRAM_enable : OUT STD_LOGIC);
45 end Disp_ImgGen_ShapeGenerator;
46
47 architecture Behavioral of Disp_ImgGen_ShapeGenerator is
48
49
50 component Disp_ImgGen_ShapeGenerator_Cpt
51    port (
52       Clk         : IN  STD_LOGIC;
53       Reset       : IN  STD_LOGIC;
54       Ce          : IN  STD_LOGIC;
55       Load        : IN  STD_LOGIC;
56       X_out       : OUT STD_LOGIC_VECTOR(6 downto 0);
57       Y_out       : OUT STD_LOGIC_VECTOR(6 downto 0);
```

```
58          Counting    : OUT STD_LOGIC);
59    end component;
60
61    component MUX_8to1
62        Port ( Data_In_0 : in  STD_LOGIC_VECTOR (7 downto 0);
63               Data_In_1 : in  STD_LOGIC_VECTOR (7 downto 0);
64               Data_In_2 : in  STD_LOGIC_VECTOR (7 downto 0);
65               Data_In_3 : in  STD_LOGIC_VECTOR (7 downto 0);
66               Data_In_4 : in  STD_LOGIC_VECTOR (7 downto 0);
67               Data_In_5 : in  STD_LOGIC_VECTOR (7 downto 0);
68               Data_In_6 : in  STD_LOGIC_VECTOR (7 downto 0);
69               Data_In_7 : in  STD_LOGIC_VECTOR (7 downto 0);
70               Channel   : in  STD_LOGIC_VECTOR (2 downto 0);
71               Data_Out  : out STD_LOGIC_VECTOR (7 downto 0));
72    end component;
73
74    component ROM_Vide
75       port (
76          Clk   : IN  STD_LOGIC;
77          En    : IN  STD_LOGIC;
78          ADDR  : IN  STD_LOGIC_VECTOR(13 downto 0);
79          DATA  : OUT STD_LOGIC);
80    end component;
81
82    component ROM_Vide_sel
83       port (
84          Clk   : IN  STD_LOGIC;
85          En    : IN  STD_LOGIC;
86          ADDR  : IN  STD_LOGIC_VECTOR(13 downto 0);
87          DATA  : OUT STD_LOGIC);
88    end component;
89
90    component ROM_O
91       port (
92          Clk   : IN  STD_LOGIC;
93          En    : IN  STD_LOGIC;
94          ADDR  : IN  STD_LOGIC_VECTOR(13 downto 0);
95          DATA  : OUT STD_LOGIC);
96    end component;
97
98
99    component ROM_O_sel
100   port (CLK : in std_logic;
101         EN : in std_logic;
102         ADDR : in std_logic_vector(13 downto 0);
103         DATA : out std_logic);
104   end component;
105
106
107   component ROM_X
108      port (
109         Clk   : IN  STD_LOGIC;
110         En    : IN  STD_LOGIC;
111         ADDR  : IN  STD_LOGIC_VECTOR(13 downto 0);
112         DATA  : OUT STD_LOGIC);
113   end component;
114
```

```
115   component ROM_X_sel
116      port (
117         Clk   : IN  STD_LOGIC;
118         En    : IN  STD_LOGIC;
119         ADDR  : IN STD_LOGIC_VECTOR(13 downto 0);
120         DATA  : OUT STD_LOGIC);
121   end component;
122
123   component ROM_victoire
124      port (
125         Clk   : IN  STD_LOGIC;
126         En    : IN  STD_LOGIC;
127         ADDR  : IN STD_LOGIC_VECTOR(13 downto 0);
128         DATA  : OUT STD_LOGIC);
129   end component;
130
131
132   signal sig_ShapeNumb : STD_LOGIC_VECTOR(2 downto 0);
133
134   signal Enable_rom : STD_LOGIC;
135   signal ROM_addr   : STD_LOGIC_VECTOR(13 downto 0);
136
137   signal coord_init : STD_LOGIC_VECTOR(18 downto 0);
138   signal X_init     : STD_LOGIC_VECTOR(9 downto 0);
139   signal Y_init     : STD_LOGIC_VECTOR(8 downto 0);
140   signal X_rom      : STD_LOGIC_VECTOR(6 downto 0);
141   signal Y_rom      : STD_LOGIC_VECTOR(6 downto 0);
142   signal X_vram     : STD_LOGIC_VECTOR(9 downto 0);
143   signal Y_vram     : STD_LOGIC_VECTOR(8 downto 0);
144
145   signal MUX_IN0    : STD_LOGIC;
146   signal MUX_IN1    : STD_LOGIC;
147   signal MUX_IN2    : STD_LOGIC;
148   signal MUX_IN3    : STD_LOGIC;
149   signal MUX_IN4    : STD_LOGIC;
150   signal MUX_IN5    : STD_LOGIC;
151   signal MUX_IN6    : STD_LOGIC;
152   signal MUX_IN7    : STD_LOGIC;
153
154   type coord_tab is array (10 downto 0) of std_logic_vector(18 downto 0);
155   constant Shape_Coord : coord_tab := (
156      0 => "0001101010010000101",   -- x134 y54
157      1 => "0001101010100000010",   -- x259 y54
158      2 => "0001101010101111111",   -- x384 y54
159      3 => "0101100100010000101",   -- x134 y179
160      4 => "0101100100100000010",   -- x259 y179
161      5 => "0101100100101111111",   -- x384 y179
162      6 => "1001011110010000101",   -- x134 y304
163      7 => "1001011110100000010",   -- x259 y304
164      8 => "1001011110101111111",   -- x384 y304
165      9 => "0011101110000000011",   -- x4 y120 : info joueur
166      10 => "0011101111000000011"   -- x516 y120 : victoire
167   );
168
169   begin
170
171      VRAM_enable <= Enable_rom;
```

```
172
173      process (Clk, Reset)
174      begin
175         if (Reset = '1') then
176            Coord_init <= Shape_Coord(0);
177            sig_ShapeNumb <= "000";
178         elsif (Clk'event and Clk='1') then
179            if(Shape_Load = '1') then
180               Coord_init <= Shape_Coord(to_integer(unsigned(Area_numb)));
181               sig_ShapeNumb <= Shape_Numb;
182            end if;
183         end if;
184      end process;
185
186
187      -- Coord de la case
188      X_init <= Coord_init( 9 downto  0);
189      Y_init <= Coord_init(18 downto 10);
190
191      -- Coord de la VRAM = case+rom
192      X_vram <= X_init + ("000" & X_rom);
193      Y_vram <= Y_init +  ("00" & Y_rom);
194
195      -- Adresses des mémoires
196      ROM_addr  <= Y_rom & X_rom;
197      VRAM_addr <= Y_vram & X_vram;
198
199
200      MUX_IN1 <= MUX_IN0;
201
202      Compteur : Disp_ImgGen_ShapeGenerator_Cpt
203      port map (
204         Clk      => Clk,
205         Reset    => Reset,
206         Ce       => Ce,
207         Load     => Shape_Load,
208         X_out    => X_rom,
209         Y_out    => Y_rom,
210         Counting => Enable_rom
211      );
212
213      MUX_ROM2OUT : MUX_8to1
214      Port map (
215         Data_In_0   => (others => MUX_IN0),
216         Data_In_1   => (others => MUX_IN1),
217         Data_In_2   => (others => MUX_IN2),
218         Data_In_3   => (others => MUX_IN3),
219         Data_In_4   => (others => MUX_IN4),
220         Data_In_5   => (others => MUX_IN5),
221         Data_In_6   => (others => MUX_IN6),
222         Data_In_7   => (others => MUX_IN7),
223         Channel     => sig_ShapeNumb,
224         Data_Out    => VRAM_data
225      );
226
227      ROM0 : ROM_O
228      port map (
```

```
229         Clk   => Clk,
230         En    => Enable_rom,
231         ADDR  => ROM_addr,
232         DATA  => MUX_IN2
233      );
234
235      ROM1 : ROM_X
236      port map (
237         Clk   => Clk,
238         En    => Enable_rom,
239         ADDR  => ROM_addr,
240         DATA  => MUX_IN3
241      );
242
243      ROM2 : ROM_Vide
244      port map (
245         Clk   => Clk,
246         En    => Enable_rom,
247         ADDR  => ROM_addr,
248         DATA  => MUX_IN0
249      );
250
251      ROM3 : ROM_Vide_Sel
252      port map (
253         Clk   => Clk,
254         En    => Enable_rom,
255         ADDR  => ROM_addr,
256         DATA  => MUX_IN4
257      );
258
259      ROM4 : ROM_O_sel
260      port map (
261         CLK => Clk,
262         EN => Enable_rom,
263         ADDR => Rom_addr,
264         DATA => MUX_IN6
265      );
266
267      ROM5 : ROM_X_Sel
268      port map (
269         Clk   => Clk,
270         En    => Enable_rom,
271         ADDR  => ROM_addr,
272         DATA  => MUX_IN7
273      );
274
275      ROM6 : ROM_victoire
276      port map (
277         Clk   => Clk,
278         En    => Enable_rom,
279         ADDR  => ROM_addr,
280         DATA  => MUX_IN5
281      );
282
283
284   end Behavioral;
285
```

```
1   ----------------------------------------------------------------------------------
2   -- Company: ENSEIRB-MATMECA
3   -- Engineer: Sylvain MARIEL (sylvain.mariel@otmax.fr)
4   -- Engineer: Thomas MOREAU  (thomas.moreau-33@hotmail.fr)
5
6   -- Create Date:    21/05/2013
7   -- Design Name:
8   -- Module Name:    Disp_ImgGen_ShapeDetermination - Behavioral
9   -- Project Name:
10  -- Target Devices:
11  -- Tool versions:
12  -- Description:
13  --
14  -- Dependencies:
15  --
16  -- Revision:
17  -- Revision 0.01 - File Created
18  -- Additional Comments:
19  --
20  ----------------------------------------------------------------------------------
21  library IEEE;
22  use IEEE.STD_LOGIC_1164.ALL;
23  use ieee.std_logic_unsigned.all;
24  use IEEE.NUMERIC_STD.ALL;
25
26  -- Uncomment the following library declaration if using
27  -- arithmetic functions with Signed or Unsigned values
28  --use IEEE.NUMERIC_STD.ALL;
29
30  -- Uncomment the following library declaration if instantiating
31  -- any Xilinx primitives in this code.
32  --library UNISIM;
33  --use UNISIM.VComponents.all;
34
35  entity Disp_ImgGen_ShapeDetermination is
36     Port( Clk         : IN  STD_LOGIC;
37           Rst         : IN  STD_LOGIC;
38           CE          : IN  STD_LOGIC;
39           Busy        : IN  STD_LOGIC;
40           Pos_Load    : IN  STD_LOGIC;
41           Ok_Load     : IN  STD_LOGIC;
42           Player_Load : IN  STD_LOGIC;
43           Ok          : IN  STD_LOGIC_VECTOR(7 downto 0);
44           Pos         : IN  STD_LOGIC_VECTOR(7 downto 0);
45           Player      : IN  STD_LOGIC;
46           OldPos      : IN  STD_LOGIC_VECTOR(7 downto 0);
47           Grid_State  : IN  STD_LOGIC_VECTOR(8 downto 0);
48           Grid_Player : IN  STD_LOGIC_VECTOR(8 downto 0);
49           Win_J1      : IN  STD_LOGIC;
50           Win_J2      : IN  STD_LOGIC;
51           Shape_Load  : OUT STD_LOGIC;
52           Shape_Numb  : OUT STD_LOGIC_VECTOR(2 downto 0);
53           Area_Numb   : OUT STD_LOGIC_VECTOR(7 downto 0));
54  end Disp_ImgGen_ShapeDetermination;
55
56  architecture Behavioral of Disp_ImgGen_ShapeDetermination is
57
```

```
 58   -- Types et signaux correspondant aux états de la FSM
 59   type etats is (START, WAIT_CHANGE, NEW_OK, NEW_POS, OLD_POS, WAIT_BUSY_OLD, WAIT_BUSY,
       NEW_PLAYER, WINNER, INIT, BUSY_INIT);
 60   signal etat_present     :etats := START;
 61   signal etat_futur       :etats := START;
 62
 63   begin
 64
 65   -- Actualisation des etats presents à chaque coup d'horloge et gestion du reset
 66      process (Clk, Rst)
 67      begin
 68         if (Rst = '1') then
 69               etat_present <= START;  -- En cas de reset, initialisation état START
 70
 71         elsif (Clk'event and Clk = '1') then
 72
 73            if(CE = '1') then
 74               etat_present <= etat_futur; -- Actualisation des états
 75            else
 76               etat_present <= etat_present;
 77            end if;
 78
 79         end if;
 80
 81      end process;
 82
 83      -- Definition des etats futurs en fonction de la FSM
 84      process (etat_present, Pos_Load, Ok_Load, Busy, player_load, Win_J1, Win_J2 )
 85      begin
 86         CASE etat_present IS
 87            WHEN START        =>
 88               etat_futur <= INIT;
 89
 90            WHEN INIT         =>
 91               etat_futur <= BUSY_INIT;
 92
 93            WHEN BUSY_INIT          =>
 94               if (busy = '0') then
 95                  etat_futur <=  WAIT_CHANGE;
 96               else
 97                  etat_futur <= BUSY_INIT;
 98               end if;
 99
100            WHEN WAIT_CHANGE  =>
101               if (Ok_Load = '1') then
102                  etat_futur <= NEW_OK;
103               elsif (Player_Load = '1') then
104                  etat_futur <= NEW_PLAYER;
105               elsif (Pos_Load = '1') then
106                  etat_futur <= OLD_POS;
107               elsif (Win_J1= '1' OR Win_J2 ='1') then
108                  etat_futur <= WINNER;
109               else
110                  etat_futur <= WAIT_CHANGE;
111               end if;
112
113            WHEN NEW_OK       =>
```

```
114               etat_futur <= WAIT_BUSY;
115
116            WHEN NEW_PLAYER         =>
117               etat_futur <= WAIT_BUSY;
118
119            WHEN OLD_POS      =>
120               etat_futur <= WAIT_BUSY_OLD;
121
122            WHEN NEW_POS      =>
123               etat_futur <= WAIT_BUSY;
124
125            WHEN WAIT_BUSY_OLD =>
126               if (busy = '0') then
127                  etat_futur <= NEW_POS;
128               else
129                  etat_futur <= WAIT_BUSY_OLD;
130               end if;
131
132            WHEN WAIT_BUSY    =>
133               if (busy = '0') then
134                  etat_futur <= WAIT_CHANGE;
135               else
136                  etat_futur <= WAIT_BUSY;
137               end if;
138
139            WHEN WINNER =>
140                  etat_futur <= WINNER;
141
142            WhEN OTHERS       =>
143               etat_futur <= START;
144         END CASE;
145      end process;
146
147      process (etat_present, grid_state, grid_player, oldPos, pos, OK, player, Win_J1,
       Win_J2)
148      begin
149         CASE etat_present IS
150            WHEN START           =>
151               Shape_load  <= '0';
152               Shape_numb  <= "000";
153               Area_numb   <= (others => '0');
154
155            WHEN INIT         =>
156               Shape_load  <= '1';
157               Shape_numb  <= "000";
158               Area_numb   <= "00001010";
159
160            WHEN BUSY_INIT       =>
161               Shape_load  <= '0';
162               Shape_numb  <= "000";
163               Area_numb   <= (others => '0');
164
165            WHEN WAIT_CHANGE     =>
166               Shape_load  <= '0';
167               Shape_numb  <= "000";
168               Area_numb   <= (others => '0');
169
```

```
170            WHEN NEW_OK          =>
171               Shape_load  <= '1';
172               Shape_numb  <= "11" & grid_player(to_integer(unsigned(Pos)));
173               Area_numb   <= Pos;
174
175            WHEN NEW_PLAYER            =>
176               Shape_load  <= '1';
177               Shape_numb  <= "01" & Player;
178               Area_numb   <= "00001001";
179
180            WHEN NEW_POS         =>
181               Shape_load  <= '1';
182               Shape_numb  <= '1' & grid_state(to_integer(unsigned(Pos))) & grid_player(
       to_integer(unsigned(Pos)));
183               Area_numb   <= Pos;
184
185            WHEN OLD_POS         =>
186               Shape_load  <= '1';
187               Shape_numb  <= '0' & grid_state(to_integer(unsigned(oldPos))) &
       grid_player(to_integer(unsigned(oldPos)));
188               Area_numb   <= oldPos;
189
190            WHEN WAIT_BUSY_OLD   =>
191               Shape_load  <= '0';
192               Shape_numb  <= "000";
193               Area_numb   <= (others => '0');
194
195            WHEN WAIT_BUSY       =>
196               Shape_load  <= '0';
197               Shape_numb  <= "000";
198               Area_numb   <= (others => '0');
199
200            WHEN WINNER       =>
201               Shape_load  <= '1';
202               Shape_numb  <= "101";
203               Area_numb   <= "00001010";
204
205            WHEN OTHERS          =>
206               Shape_load  <= '0';
207               Shape_numb  <= "000";
208               Area_numb   <= (others => '0');
209
210         END CASE;
211
212      end process;
213
214
215   end Behavioral;
216
217
```

```
1  ----------------------------------------------------------------------------------
2  -- Company: ENSEIRB-MATMECA
3  -- Engineer: Sylvain MARIEL (sylvain.mariel@otmax.fr)
4  -- Engineer: Thomas MOREAU  (thomas.moreau-33@hotmail.fr)
5
6  -- Create Date:    21/05/2013
7  -- Design Name:
8  -- Module Name:    Disp_ImgGen_GridManager - Behavioral
9  -- Project Name:
10 -- Target Devices:
11 -- Tool versions:
12 -- Description:
13 --
14 -- Dependencies:
15 --
16 -- Revision:
17 -- Revision 0.01 - File Created
18 -- Additional Comments:
19 --
20 ----------------------------------------------------------------------------------
21 library IEEE;
22 use IEEE.STD_LOGIC_1164.ALL;
23 use IEEE.STD_LOGIC_UNSIGNED.ALL;
24 use IEEE.NUMERIC_STD.ALL;
25
26
27 -- Uncomment the following library declaration if instantiating
28 -- any Xilinx primitives in this code.
29 --library UNISIM;
30 --use UNISIM.VComponents.all;
31
32 entity Disp_ImgGen_GridManager is
33    Port( Clk            : IN  STD_LOGIC;
34          Rst            : IN  STD_LOGIC;
35          CE             : IN  STD_LOGIC;
36          Ok_Load        : IN  STD_LOGIC;
37          Player         : IN  STD_LOGIC;
38          OK             : IN  STD_LOGIC_VECTOR(7 downto 0);
39          Ok_Load_sync   : OUT STD_LOGIC;
40          Grid_State     : OUT STD_LOGIC_VECTOR(8 downto 0);
41          Grid_Player    : OUT STD_LOGIC_VECTOR(8 downto 0));
42 end Disp_ImgGen_GridManager;
43
44 architecture Behavioral of Disp_ImgGen_GridManager is
45 signal Tmp_State  : std_logic_vector(8 downto 0) := "000000000";
46 signal Tmp_Player : std_logic_vector(8 downto 0) := "000000000";
47 begin
48
49    Grid_State  <= Tmp_State;
50    Grid_Player <= Tmp_Player;
51
52    process (Clk,Rst)
53       begin
54
55       if (Rst = '1') then
56       Tmp_State   <= "000000000";
57       Tmp_Player  <= "000000000";
```

```
58
59          elsif (Clk'event AND Clk = '1') then
60
61             if (CE = '1') then
62
63                if (Ok_Load = '1') then
64
65                   if (Tmp_State(to_integer(unsigned(OK))) = '0') then
66                      Tmp_State(to_integer(unsigned(OK))) <= '1';
67                      Tmp_Player(to_integer(unsigned(OK)))<= Player;
68                      Ok_Load_sync <= '1';
69                   else
70                      Ok_Load_sync <= '0';
71                   end if;
72
73                end if;
74
75             end if;
76
77          end if;
78
79       end process;
80    end Behavioral;
81
82
```

```
1   ----------------------------------------------------------------------------------
2   -- Company: ENSEIRB-MATMECA
3   -- Engineer: Sylvain MARIEL (sylvain.mariel@otmax.fr)
4   -- Engineer: Thomas MOREAU  (thomas.moreau-33@hotmail.fr)
5
6   -- Create Date:    21/05/2013
7   -- Design Name:
8   -- Module Name:    win_manager - Behavioral
9   -- Project Name:
10  -- Target Devices:
11  -- Tool versions:
12  -- Description:
13  --
14  -- Dependencies:
15  --
16  -- Revision:
17  -- Revision 0.01 - File Created
18  -- Additional Comments:
19  --
20  ----------------------------------------------------------------------------------
21  library IEEE;
22  use IEEE.STD_LOGIC_1164.ALL;
23
24  -- Uncomment the following library declaration if using
25  -- arithmetic functions with Signed or Unsigned values
26  --use IEEE.NUMERIC_STD.ALL;
27
28  -- Uncomment the following library declaration if instantiating
29  -- any Xilinx primitives in this code.
30  --library UNISIM;
31  --use UNISIM.VComponents.all;
32
33  entity win_manager is
34     Port( Grid_State     : IN STD_LOGIC_VECTOR(8 downto 0);
35           Grid_Player    : IN STD_LOGIC_VECTOR(8 downto 0);
36           Enable_win_J1  : OUT STD_LOGIC;
37           Enable_win_J2  : OUT STD_LOGIC);
38  end win_manager;
39
40  architecture Behavioral of win_manager is
41
42  signal J1 : std_logic_vector (8 downto 0);
43  signal J2 : std_logic_vector (8 downto 0);
44
45  -- 0 => "xxxxxx111",
46  -- 1 => "xxx111xxx",
47  -- 2 => "111xxxxxx",
48  -- 3 => "xx1xx1xx1",
49  -- 4 => "1xx1xx1xx",
50  -- 5 => "x1xx1xx1x",
51  -- 6 => "1xxx1xxx1",
52  -- 7 => "xx1x1x1xx";
53
54  begin
55
56     J1 <= Grid_State and (not Grid_Player);
57     J2 <= Grid_State and Grid_Player;
```

```
58      Enable_win_J1<=   '1' when J1(0)='1' and J1(1)='1' and J1(2)='1' else
59                        '1' when J1(3)='1' and J1(4)='1' and J1(5)='1' else
60                        '1' when J1(8)='1' and J1(7)='1' and J1(6)='1' else
61                        '1' when J1(0)='1' and J1(3)='1' and J1(6)='1' else
62                        '1' when J1(2)='1' and J1(5)='1' and J1(8)='1' else
63                        '1' when J1(1)='1' and J1(4)='1' and J1(7)='1' else
64                        '1' when J1(0)='1' and J1(4)='1' and J1(8)='1' else
65                        '1' when J1(2)='1' and J1(4)='1' and J1(6)='1' else
66                        '0';
67
68      Enable_win_J2<=   '1' when J2(0)='1' and J2(1)='1' and J2(2)='1' else
69                        '1' when J2(3)='1' and J2(4)='1' and J2(5)='1' else
70                        '1' when J2(8)='1' and J2(7)='1' and J2(6)='1' else
71                        '1' when J2(0)='1' and J2(3)='1' and J2(6)='1' else
72                        '1' when J2(2)='1' and J2(5)='1' and J2(8)='1' else
73                        '1' when J2(1)='1' and J2(4)='1' and J2(7)='1' else
74                        '1' when J2(0)='1' and J2(4)='1' and J2(8)='1' else
75                        '1' when J2(2)='1' and J2(4)='1' and J2(6)='1' else
76                        '0';
77
78
79
80
81    end Behavioral;
82
83
```

```
1  ----------------------------------------------------------------------------------
2  -- Company: ENSEIRB-MATMECA
3  -- Engineer: Sylvain MARIEL (sylvain.mariel@otmax.fr)
4  -- Engineer: Thomas MOREAU  (thomas.moreau-33@hotmail.fr)
5
6  -- Create Date:    21/05/2013
7  -- Design Name:
8  -- Module Name:    Disp_ImgGen - Behavioral
9  -- Project Name:
10 -- Target Devices:
11 -- Tool versions:
12 -- Description:
13 --
14 -- Dependencies:
15 --
16 -- Revision:
17 -- Revision 0.01 - File Created
18 -- Additional Comments:
19 --
20 ----------------------------------------------------------------------------------
21 library IEEE;
22 use IEEE.STD_LOGIC_1164.ALL;
23
24 -- Uncomment the following library declaration if using
25 -- arithmetic functions with Signed or Unsigned values
26 --use IEEE.NUMERIC_STD.ALL;
27
28 -- Uncomment the following library declaration if instantiating
29 -- any Xilinx primitives in this code.
30 --library UNISIM;
31 --use UNISIM.VComponents.all;
32
33 entity Disp_ImgGen is
34    port (Clk            : IN  STD_LOGIC;
35          Ce             : IN  STD_LOGIC;
36          Reset          : IN  STD_LOGIC;
37          Pos_load       : IN  STD_LOGIC;
38          Ok_load        : IN  STD_LOGIC;
39          Player_load    : IN  STD_LOGIC;
40          OldPos_val     : IN  std_logic_vector(7 downto 0);
41          Pos_val        : IN  STD_LOGIC_VECTOR(7 downto 0);
42          Ok_val         : IN  STD_LOGIC_VECTOR(7 downto 0);
43          Player_val     : IN  STD_LOGIC;
44          VRAM_AddrW     : OUT STD_LOGIC_VECTOR(18 downto 0);
45          VRAM_DataW     : OUT STD_LOGIC_VECTOR(7 downto 0);
46          VRAM_EnableW   : OUT STD_LOGIC);
47 end Disp_ImgGen;
48
49 architecture Behavioral of Disp_ImgGen is
50
51 component Disp_ImgGen_GridManager
52    Port( Clk         : IN  STD_LOGIC;
53          Rst         : IN  STD_LOGIC;
54          CE          : IN  STD_LOGIC;
55          Ok_Load     : IN  STD_LOGIC;
56          Player      : IN  STD_LOGIC;
57          OK          : IN  STD_LOGIC_VECTOR(7 downto 0);
```

```
58            Ok_Load_sync: OUT STD_LOGIC;
59            Grid_State  : OUT STD_LOGIC_VECTOR(8 downto 0);
60            Grid_Player : OUT STD_LOGIC_VECTOR(8 downto 0));
61    end component;
62
63    component Disp_ImgGen_ShapeDetermination
64       Port( Clk         : IN  STD_LOGIC;
65             Rst         : IN  STD_LOGIC;
66             CE          : IN  STD_LOGIC;
67             Busy        : IN  STD_LOGIC;
68             Pos_Load    : IN  STD_LOGIC;
69             Ok_Load     : IN  STD_LOGIC;
70             Player_Load : IN  STD_LOGIC;
71             Ok          : IN  STD_LOGIC_VECTOR(7 downto 0);
72             Pos         : IN  STD_LOGIC_VECTOR(7 downto 0);
73             Player      : IN  STD_LOGIC;
74             OldPos      : IN  STD_LOGIC_VECTOR(7 downto 0);
75             Grid_State  : IN  STD_LOGIC_VECTOR(8 downto 0);
76             Grid_Player : IN  STD_LOGIC_VECTOR(8 downto 0);
77             Win_J1      : IN  STD_LOGIC;
78             Win_J2      : IN  STD_LOGIC;
79             Shape_Load  : OUT STD_LOGIC;
80             Shape_Numb  : OUT STD_LOGIC_VECTOR(2 downto 0);
81             Area_Numb   : OUT STD_LOGIC_VECTOR(7 downto 0));
82    end component;
83
84
85    component Disp_ImgGen_ShapeGenerator
86       port (
87          Clk         : IN  STD_LOGIC;
88          Reset       : IN  STD_LOGIC;
89          Ce          : IN  STD_LOGIC;
90          Shape_Load  : IN  STD_LOGIC;
91          Shape_Numb  : IN  STD_LOGIC_VECTOR(2  downto 0);
92          Area_Numb   : IN  STD_LOGIC_VECTOR(7 downto 0);
93          VRAM_data   : OUT STD_LOGIC_VECTOR(7  downto 0);
94          VRAM_addr   : OUT STD_LOGIC_VECTOR(18 downto 0);
95          VRAM_enable : OUT STD_LOGIC);
96    end component;
97
98    component win_manager is
99       Port( Grid_State     : IN STD_LOGIC_VECTOR(8 downto 0);
100            Grid_Player    : IN STD_LOGIC_VECTOR(8 downto 0);
101            Enable_win_J1  : OUT STD_LOGIC;
102            Enable_win_J2  : OUT STD_LOGIC);
103   end component;
104
105   signal Grid_state : std_logic_vector(8 downto 0);
106   signal Grid_Player : std_logic_vector(8 downto 0);
107   signal busy : std_logic;
108   signal Shape_Load : std_logic;
109   signal Shape_Numb : std_logic_vector(2 downto 0);
110   signal area_numb : std_logic_vector(7 downto 0);
111   signal OK_Load_sync : std_logic;
112   signal Win_J1 : STD_LOGIC;
113   signal Win_J2 : STD_LOGIC;
114
```

```
115
116   begin
117
118
119      VRAM_EnableW <= Busy;
120
121      Win : Win_Manager
122      port map(
123      Grid_State     => Grid_State,
124      Grid_Player    => Grid_Player,
125      Enable_Win_J1  => Win_J1,
126      Enable_Win_J2  => Win_J2);
127
128      Grid_Manager : Disp_ImgGen_GridManager
129      port map(
130         Clk         => Clk,
131         Rst         => Reset,
132         CE          => Ce,
133         Ok_Load     => Ok_load,
134         Player      => Player_val,
135         OK          => Pos_val,      -- MODIFIER NOM ENTREE
136         Ok_Load_sync=> OK_Load_sync,
137         Grid_State  => Grid_state,
138         Grid_Player => Grid_Player);
139
140      Shape_Determination : Disp_ImgGen_ShapeDetermination
141      port map(
142         Clk         => Clk,
143         Rst         => Reset,
144         CE          => CE,
145         Busy        => busy,
146         Pos_Load    => Pos_load,
147         Ok_Load     => OK_Load_sync,
148         Player_Load => Player_Load,
149         Ok          => Ok_val,
150         Pos         => Pos_val,
151         Player      => Player_val,
152         OldPos      => OldPos_val,
153         Grid_State  => Grid_state,
154         Grid_Player => Grid_player,
155         Win_J1      => Win_J1,
156         Win_J2      => Win_J2,
157         Shape_Load  => Shape_Load,
158         Shape_Numb  => Shape_Numb,
159         Area_Numb   => Area_Numb);
160
161
162      Shape_Generator : Disp_ImgGen_ShapeGenerator
163      port map(
164         Clk         => Clk,
165         Reset       => Reset,
166         Ce          => Ce,
167         Shape_Load  => Shape_Load,
168         Shape_Numb  => Shape_Numb,
169         Area_Numb   => Area_Numb,
170         VRAM_data   => VRAM_DataW,
171         VRAM_addr   => VRAM_AddrW,
```

```
172         VRAM_enable => Busy);
173
174
175   end Behavioral;
176
177
```

```
1  ----------------------------------------------------------------------------------
2  -- Company: ENSEIRB-MATMECA
3  -- Engineer: Sylvain MARIEL (sylvain.mariel@otmax.fr)
4  -- Engineer: Thomas MOREAU  (thomas.moreau-33@hotmail.fr)
5
6  -- Create Date:    21/05/2013
7  -- Design Name:
8  -- Module Name:    DISP_BUSINT_Decode - Behavioral
9  -- Project Name:
10 -- Target Devices:
11 -- Tool versions:
12 -- Description:
13 --
14 -- Dependencies:
15 --
16 -- Revision:
17 -- Revision 0.01 - File Created
18 -- Additional Comments:
19 --
20 ----------------------------------------------------------------------------------
21 library IEEE;
22 use IEEE.STD_LOGIC_1164.ALL;
23
24 -- Uncomment the following library declaration if using
25 -- arithmetic functions with Signed or Unsigned values
26 --use IEEE.NUMERIC_STD.ALL;
27
28 -- Uncomment the following library declaration if instantiating
29 -- any Xilinx primitives in this code.
30 --library UNISIM;
31 --use UNISIM.VComponents.all;
32
33 entity DISP_BUSINT_Decode is
34     Port(Data_In        : in  STD_LOGIC_VECTOR(5 downto 0);
35          Enable_Disp    : in  STD_LOGIC;
36          RW             : in STD_LOGIC;
37          POS_Load       : out STD_LOGIC;
38          OK_Load        : out STD_LOGIC;
39          PLAYER_Load    : out STD_LOGIC);
40 end DISP_BUSINT_Decode;
41
42 architecture Behavioral of DISP_BUSINT_Decode is
43
44 begin
45
46    POS_Load    <= '1' WHEN Data_In  ="111010" and RW = '1' and Enable_Disp = '1' ELSE
47                   '0';
48    OK_Load     <= '1' WHEN Data_In  ="111011" and RW = '1' and Enable_Disp = '1' ELSE
49                   '0';
50    PLAYER_Load <= '1' WHEN Data_In  ="111001" and RW = '1' and Enable_Disp = '1' ELSE
51                   '0';
52
53 end Behavioral;
54
55
```

```
 1  ----------------------------------------------------------------------------------
 2  -- Company: ENSEIRB-MATMECA
 3  -- Engineer: Sylvain MARIEL (sylvain.mariel@otmax.fr)
 4  -- Engineer: Thomas MOREAU  (thomas.moreau-33@hotmail.fr)
 5
 6  -- Create Date:    21/05/2013
 7  -- Design Name:
 8  -- Module Name:    Reg8bits - Behavioral
 9  -- Project Name:
10  -- Target Devices:
11  -- Tool versions:
12  -- Description:
13  --
14  -- Dependencies:
15  --
16  -- Revision:
17  -- Revision 0.01 - File Created
18  -- Additional Comments:
19  --
20  ----------------------------------------------------------------------------------
21  library IEEE;
22  use IEEE.STD_LOGIC_1164.ALL;
23
24  -- Uncomment the following library declaration if using
25  -- arithmetic functions with Signed or Unsigned values
26  --use IEEE.NUMERIC_STD.ALL;
27
28  -- Uncomment the following library declaration if instantiating
29  -- any Xilinx primitives in this code.
30  --library UNISIM;
31  --use UNISIM.VComponents.all;
32
33  entity Reg8bits is
34     Port (   H        : in  STD_LOGIC;                    -- Horloge
35              RST      : in  STD_LOGIC;                    -- Reset asynchrone du
    composant
36              CE       : in  STD_LOGIC;                    -- Clock Enable, actication
    du composant
37              Load     : in  STD_LOGIC;                    -- Chargement de la valeur
    d'entrée
38              Data_In  : in  STD_LOGIC_VECTOR(7 downto 0); -- Valeur d'entrée
39              Data_Out : out STD_LOGIC_VECTOR(7 downto 0));-- Valeur de sortie
40  end Reg8bits;
41
42  architecture Behavioral of Reg8bits is
43
44  begin
45
46     process (H,RST)
47     begin
48
49        if (RST = '1') then
50           Data_Out <= "00000000";
51
52        elsif (H'event AND H = '1') then
53           if (CE = '1') then
54              if (Load = '1') then
```

```
55                  Data_Out <= Data_In;
56
57              end if;
58
59           end if;
60
61        end if;
62
63     end process;
64
65  end Behavioral;
66
67
```

```
1   ----------------------------------------------------------------------------------
2   -- Company: ENSEIRB-MATMECA
3   -- Engineer: Sylvain MARIEL (sylvain.mariel@otmax.fr)
4   -- Engineer: Thomas MOREAU  (thomas.moreau-33@hotmail.fr)
5
6   -- Create Date:    21/05/2013
7   -- Design Name:
8   -- Module Name:    DISP_BUSINT - Behavioral
9   -- Project Name:
10  -- Target Devices:
11  -- Tool versions:
12  -- Description:
13  --
14  -- Dependencies:
15  --
16  -- Revision:
17  -- Revision 0.01 - File Created
18  -- Additional Comments:
19  --
20  ----------------------------------------------------------------------------------
21  library IEEE;
22  use IEEE.STD_LOGIC_1164.ALL;
23
24  -- Uncomment the following library declaration if using
25  -- arithmetic functions with Signed or Unsigned values
26  --use IEEE.NUMERIC_STD.ALL;
27
28  -- Uncomment the following library declaration if instantiating
29  -- any Xilinx primitives in this code.
30  --library UNISIM;
31  --use UNISIM.VComponents.all;
32
33  entity DISP_BUSINT is
34     Port (Clk            : in  STD_LOGIC;
35           CE             : in  STD_LOGIC;
36           Reset          : in  STD_LOGIC;
37           AddrBus        : in  STD_LOGIC_VECTOR (5 DOWNTO 0);
38           DataBus_fromCPU : in  STD_LOGIC_VECTOR (7 DOWNTO 0);
39           Enable_Img     : in  STD_LOGIC;
40           RW             : in  STD_LOGIC;
41           Player_Out     : out STD_LOGIC_VECTOR (7 DOWNTO 0);
42           OK_Out         : out STD_LOGIC_VECTOR (7 DOWNTO 0);
43           Pos_Out        : out STD_LOGIC_VECTOR (7 DOWNTO 0);
44           OldPos_Out     : out STD_LOGIC_VECTOR (7 DOWNTO 0);
45           OK_Load        : out STD_LOGIC;
46           Pos_Load       : out STD_LOGIC;
47           Player_Load    : out STD_LOGIC);
48  end DISP_BUSINT;
49
50  architecture Behavioral of DISP_BUSINT is
51
52     component DISP_BUSINT_Decode
53           Port(    Data_In        : in  STD_LOGIC_VECTOR(5 downto 0);
54                    Enable_Disp    : in  STD_LOGIC;
55                    RW             : in STD_LOGIC;
56                    POS_Load       : out STD_LOGIC;
57                    OK_Load        : out STD_LOGIC;
```

```
58                    PLAYER_Load    : out STD_LOGIC);
59       end component;
60
61       component REG8bits
62             Port (   H        : in  STD_LOGIC;                    -- Horloge
63                      RST      : in  STD_LOGIC;                    -- Reset asynchrone du
     composant
64                      CE       : in  STD_LOGIC;                    -- Clock Enable,
     actication du composant
65                      Load     : in  STD_LOGIC;                    -- Chargement de la
     valeur d'entrée
66                      Data_In  : in  STD_LOGIC_VECTOR(7 downto 0); -- Valeur d'entrée
67                      Data_Out : out STD_LOGIC_VECTOR(7 downto 0));-- Valeur de sortie
68       end component;
69
70          signal s_Pos_Load    : STD_LOGIC;
71          signal s_OK_Load     : STD_LOGIC;
72          signal s_J_Load      : STD_LOGIC;
73          signal s_Pos_val     : STD_LOGIC_VECTOR(7 downto 0);
74
75    begin
76
77       Player_Load <= s_J_Load;
78       Pos_Load    <= s_Pos_Load;
79       OK_Load     <= s_OK_Load;
80       Pos_Out     <= s_Pos_val;
81
82       REG_J: REG8bits port map (
83          Clk,
84          Reset,
85          CE,
86          s_J_Load,
87          DataBus_fromCPU (7 downto 0),
88          Player_Out (7 downto 0));
89
90       REG_OK: REG8bits port map (
91          Clk,
92          Reset,
93          CE,
94          s_OK_Load,
95          DataBus_fromCPU (7 downto 0),
96          OK_Out (7 downto 0));
97
98       REG_Pos: REG8bits port map (
99          Clk,
100         Reset,
101         CE,
102         s_Pos_Load,
103         DataBus_fromCPU (7 downto 0),
104         s_Pos_val);
105
106      REG_OldPos: REG8bits port map (
107         Clk,
108         Reset,
109         CE,
110         s_Pos_Load,
111         s_Pos_val,
```

```
112         OldPos_Out (7 downto 0));
113
114      Decode: DISP_BUSINT_Decode port map (
115         AddrBus (5 downto 0),
116         Enable_Img,
117         RW,
118         s_Pos_Load,
119         s_OK_Load,
120         s_J_Load);
121
122
123   end Behavioral;
124
125
```

```
  1  ----------------------------------------------------------------------------------
  2  -- Company: ENSEIRB-MATMECA
  3  -- Engineer: Sylvain MARIEL (sylvain.mariel@otmax.fr)
  4  -- Engineer: Thomas MOREAU  (thomas.moreau-33@hotmail.fr)
  5
  6  -- Create Date:    21/05/2013
  7  -- Design Name:
  8  -- Module Name:    Display - Behavioral
  9  -- Project Name:
 10  -- Target Devices:
 11  -- Tool versions:
 12  -- Description:
 13  --
 14  -- Dependencies:
 15  --
 16  -- Revision:
 17  -- Revision 0.01 - File Created
 18  -- Additional Comments:
 19  --
 20  ----------------------------------------------------------------------------------
 21  library IEEE;
 22  use IEEE.STD_LOGIC_1164.ALL;
 23
 24  -- Uncomment the following library declaration if using
 25  -- arithmetic functions with Signed or Unsigned values
 26  --use IEEE.NUMERIC_STD.ALL;
 27
 28  -- Uncomment the following library declaration if instantiating
 29  -- any Xilinx primitives in this code.
 30  --library UNISIM;
 31  --use UNISIM.VComponents.all;
 32
 33  entity Display is
 34     Port (Clk               : in  STD_LOGIC;
 35           CE                : in  STD_LOGIC;
 36           Reset             : in  STD_LOGIC;
 37           Enable            : IN  STD_LOGIC;
 38           RW                : IN  STD_LOGIC;
 39           AddrBus           : IN  STD_LOGIC_VECTOR (5 downto 0);
 40           DataBus_fromCPU   : IN  STD_LOGIC_VECTOR (7 downto 0);
 41           VGA_HS            : OUT STD_LOGIC;
 42           VGA_VS            : OUT STD_LOGIC;
 43           VGA_Red           : OUT STD_LOGIC_VECTOR (2 downto 0);
 44           VGA_Green         : OUT STD_LOGIC_VECTOR (2 downto 0);
 45           VGA_Blue          : OUT STD_LOGIC_VECTOR (1 downto 0));
 46  end Display;
 47
 48  architecture Behavioral of Display is
 49
 50     component Disp_BusInt
 51        Port (Clk            : in  STD_LOGIC;
 52              CE               : in  STD_LOGIC;
 53              Reset            : in  STD_LOGIC;
 54              AddrBus          : in  STD_LOGIC_VECTOR (5 DOWNTO 0);
 55              DataBus_fromCPU  : in  STD_LOGIC_VECTOR (7 DOWNTO 0);
 56              Enable_Img       : in  STD_LOGIC;
 57              RW               : in  STD_LOGIC;
```

```
58                Player_Out        : out STD_LOGIC_VECTOR (7 DOWNTO 0);
59                OK_Out            : out STD_LOGIC_VECTOR (7 DOWNTO 0);
60                Pos_Out           : out STD_LOGIC_VECTOR (7 DOWNTO 0);
61                OldPos_Out        : out STD_LOGIC_VECTOR (7 DOWNTO 0);
62                OK_Load           : out STD_LOGIC;
63                Pos_Load          : out STD_LOGIC;
64                Player_Load       : out STD_LOGIC);
65      end component;
66
67
68      component Disp_ImgGen
69      port (Clk             : IN  STD_LOGIC;
70            Ce              : IN  STD_LOGIC;
71            Reset           : IN  STD_LOGIC;
72            Pos_load        : IN  STD_LOGIC;
73            Ok_load         : IN  STD_LOGIC;
74            Player_load     : IN  STD_LOGIC;
75            OldPos_val      : IN  STD_LOGIC_VECTOR(7 downto 0);
76            Pos_val         : IN  STD_LOGIC_VECTOR(7 downto 0);
77            Ok_val          : IN  STD_LOGIC_VECTOR(7 downto 0);
78            Player_val      : IN  STD_LOGIC;
79            VRAM_AddrW      : OUT STD_LOGIC_VECTOR(18 downto 0);
80            VRAM_DataW      : OUT STD_LOGIC_VECTOR(7 downto 0);
81            VRAM_EnableW    : OUT STD_LOGIC);
82      end component;
83
84      component Disp_VGAinterface
85         Port (Clk             : in  STD_LOGIC;
86               CE              : in  STD_LOGIC;
87               Reset           : in  STD_LOGIC;
88               VRAM_enableW    : in  STD_LOGIC;
89               VRAM_addrW      : in  STD_LOGIC_VECTOR(18 downto 0);
90               VRAM_dataIn     : in  STD_LOGIC_VECTOR(7 downto 0);
91               HS              : OUT STD_LOGIC;
92               VS              : OUT STD_LOGIC;
93               VRAM_dataOut    : OUT STD_LOGIC_VECTOR(7 downto 0));
94      end component;
95
96
97      signal Player_val     : STD_LOGIC_VECTOR(7 downto 0);
98      signal OK_val         : STD_LOGIC_VECTOR(7 downto 0);
99      signal Pos_val        : STD_LOGIC_VECTOR(7 downto 0);
100     signal OldPos_val     : STD_LOGIC_VECTOR(7 downto 0);
101     signal OK_Load        : STD_LOGIC;
102     signal Pos_Load       : STD_LOGIC;
103     signal Player_Load    : STD_LOGIC;
104     signal VRAM_addrW     : STD_LOGIC_VECTOR(18 downto 0);
105     signal VRAM_dataIn    : STD_LOGIC_VECTOR(7 downto 0);
106     signal VRAM_pixel     : STD_LOGIC_VECTOR(7 downto 0);
107     signal VRAM_enableW   : STD_LOGIC;
108
109   begin
110
111     --  Décomposition du pixel en couleurs
112     VGA_Red     <= VRAM_pixel(2 downto 0);
113     VGA_Green   <= VRAM_pixel(5 downto 3);
114     VGA_Blue    <= VRAM_pixel(7 downto 6);
```

```
115
116
117      Disp_Bus_Interface: DISP_BUSINT port map (
118         Clk               => Clk,
119         CE                => Ce,
120         Reset             => Reset,
121         AddrBus           => AddrBus,
122         DataBus_fromCPU   => DataBus_fromCPU,
123         Enable_Img        => Enable,
124         RW                => RW,
125         Player_Out        => Player_val,
126         OK_Out            => OK_val,
127         Pos_Out           => Pos_val,
128         OldPos_Out        => OldPos_val,
129         OK_Load           => OK_Load,
130         Pos_Load          => Pos_Load,
131         Player_Load       => Player_Load);
132
133
134      Disp_Img_Generation: Disp_ImgGen port map(
135         Clk            => CLK,
136         Ce             => Ce,
137         Reset          => Reset,
138         Pos_load       => Pos_load,
139         Ok_load        => OK_load,
140         Player_load    => Player_load,
141         OldPos_val     => OldPos_val,
142         Pos_val        => Pos_val,
143         Ok_val         => OK_val,
144         Player_val     => Player_val(0),
145         VRAM_AddrW     => VRAM_addrW,
146         VRAM_DataW     => VRAM_dataIn,
147         VRAM_EnableW   => VRAM_enableW);
148
149      Disp_VGA_Interface : Disp_VGAinterface port map (
150         Clk            => Clk,
151         CE             => Ce,
152         Reset          => Reset,
153         VRAM_enableW   => VRAM_enableW,
154         VRAM_addrW     => VRAM_addrW,
155         VRAM_dataIn    => VRAM_dataIn,
156         HS             => VGA_HS,
157         VS             => VGA_VS,
158         VRAM_dataOut   => VRAM_pixel);
159
160   end Behavioral;
161
162
```

```
1   ----------------------------------------------------------------------------------
2   -- Company: ENSEIRB-MATMECA
3   -- Engineer: Sylvain MARIEL (sylvain.mariel@otmax.fr)
4   -- Engineer: Thomas MOREAU  (thomas.moreau-33@hotmail.fr)
5
6   -- Create Date:    21/05/2013
7   -- Design Name:
8   -- Module Name:    RAM_56 - Behavioral
9   -- Project Name:
10  -- Target Devices:
11  -- Tool versions:
12  -- Description:
13  --
14  -- Dependencies:
15  --
16  -- Revision:
17  -- Revision 0.01 - File Created
18  -- Additional Comments:
19  --
20  ----------------------------------------------------------------------------------
21  library IEEE;
22  use IEEE.STD_LOGIC_1164.ALL;
23
24  -- Uncomment the following library declaration if using
25  -- arithmetic functions with Signed or Unsigned values
26  use IEEE.NUMERIC_STD.ALL;
27
28  -- Uncomment the following library declaration if instantiating
29  -- any Xilinx primitives in this code.
30  --library UNISIM;
31  --use UNISIM.VComponents.all;
32
33  entity RAM_56 is
34     Port (AddrBus           : in  STD_LOGIC_VECTOR (5 downto 0);
35           DataBus_fromCPU   : in  STD_LOGIC_VECTOR (7 downto 0);
36           RW                : in  STD_LOGIC;
37           ENABLE            : in  STD_LOGIC;
38           clk               : in  STD_LOGIC;
39           Ce                : in  STD_LOGIC;
40           DataBus_toCPU     : out STD_LOGIC_VECTOR (7 downto 0));
41  end RAM_56;
42
43  architecture Behavioral of RAM_56 is
44
45  type tab56 is array (integer range 0 to 55) of STD_LOGIC_VECTOR(7 downto 0);
46  signal memoire : tab56 := (
47  -- INIT
48  0 => x"2F",    -- NOR ALLONE  (CLR)
49  1 => x"B4",    -- STA POS
50  2 => x"B5",    -- STA JOUEUR
51  3 => x"B9",    -- STA AFF_J
52  -- START
53  4 => x"2F",    -- NOR ALLONE  (CLR)
54  5 => x"78",    -- ADD BP      (LOAD)
55  6 => x"71",    -- ADD ONE
56  7 => x"C9",    -- JCC BPDETECT
57  8 => x"C4",    -- JCC START
```

```
58    -- BPDETECT
59    9 => x"30",     -- NOR ZERO
60    10 => x"6F",    -- ADD ALLONE
61    11 => x"D8",    -- JCC BT_NEXT
62    12 => x"6F",    -- ADD ALLONE
63    13 => x"E0",    -- JCC BT_PREV
64    14 => x"CF",    -- JCC BT_OK
65    -- BT_OK
66    15 => x"2F",    -- NOR ALLONE  (CLR)
67    16 => x"74",    -- ADD POS     (LOAD)
68    17 => x"BB",    -- STA AFF_OK
69    18 => x"2F",    -- NOR ALLONE  (CLR)
70    19 => x"75",    -- ADD JOUEUR  (LOAD)
71    20 => x"30",    -- NOR ZERO
72    21 => x"B5",    -- STA JOUEUR
73    22 => x"B9",    -- STA AFF_J
74    23 => x"C4",    -- JCC START
75    -- BT_NEXT
76    24 => x"2F",    -- NOR ALLONE  (CLR)
77    25 => x"74",    -- ADD POS     (LOAD)
78    26 => x"71",    -- ADD ONE
79    27 => x"B4",    -- STA POS
80    28 => x"72",    -- ADD VAL_247
81    29 => x"EB",    -- JCC SENDPOS
82    30 => x"B4",    -- STA POS
83    31 => x"EB",    -- JCC SENDPOS
84    -- BT_PREC
85    32 => x"29",    -- NOR ALLONE
86    33 => x"74",    -- ADD POS
87    34 => x"6F",    -- ADD ONE
88    35 => x"70",    -- ADD ZERO
89    36 => x"B4",    -- STA POS
90    37 => x"71",    -- ADD ONE
91    38 => x"EB",    -- JCC SENDPO
92    39 => x"73",    -- ADD VAL_8
93    40 => x"B4",    -- STA POS
94    41 => x"EB",    -- JCC SEND POS
95    42 => x"C4",    -- JCC START / Bourage pour pas avoir à tout réécrire ...
96    -- SENDPOS
97    43 => x"2F",    -- NOR ALLONE  (CLR)
98    44 => x"74",    -- ADD POS     (LOAD)
99    45 => x"BA",    -- STA AFF_POS
100   46 => x"C4",    -- JCC START
101   -- VARIABLES
102   47 => x"FF",    -- ALLONE
103   48 => x"00",    -- ZERO
104   49 => x"01",    -- ONE
105   50 => x"F7",    -- VAL_247
106   51 => x"08",    -- VAL_8
107   52 => x"00",    -- POS
108   53 => x"00",    -- JOUEUR
109      others => x"00");
110   begin
111
112      process (clk)
113      begin
114
```

```
115         if (clk'event AND clk = '0') then
116            if (CE = '1') then
117               if (Enable = '1') then
118
119                  -- lecture
120                  if (RW = '0') then
121                     DataBus_toCPU <= memoire(to_integer(unsigned(AddrBus)));
122
123                  -- écriture
124                  elsif (RW = '1') then
125                     memoire(to_integer(unsigned(AddrBus))) <= DataBus_fromCPU;
126
127                  end if;
128               end if;
129            end if;
130         end if;
131
132      end process;
133
134
135   end Behavioral;
136
137
```

```
1  ----------------------------------------------------------------------------------
2  -- Company: ENSEIRB-MATMECA
3  -- Engineer: Sylvain MARIEL (sylvain.mariel@otmax.fr)
4  -- Engineer: Thomas MOREAU  (thomas.moreau-33@hotmail.fr)
5
6  -- Create Date:    21/05/2013
7  -- Design Name:
8  -- Module Name:    busArbiter - Behavioral
9  -- Project Name:
10 -- Target Devices:
11 -- Tool versions:
12 -- Description:
13 --
14 -- Dependencies:
15 --
16 -- Revision:
17 -- Revision 0.01 - File Created
18 -- Additional Comments:
19 --
20 ----------------------------------------------------------------------------------
21 library IEEE;
22 use IEEE.STD_LOGIC_1164.ALL;
23
24 -- Uncomment the following library declaration if using
25 -- arithmetic functions with Signed or Unsigned values
26 --use IEEE.NUMERIC_STD.ALL;
27
28 -- Uncomment the following library declaration if instantiating
29 -- any Xilinx primitives in this code.
30 --library UNISIM;
31 --use UNISIM.VComponents.all;
32
33 entity busArbiter is
34    Port (Enable      : in  STD_LOGIC;
35          AddrBus     : in  STD_LOGIC_VECTOR(5 downto 0);
36          Enable_RAM  : out STD_LOGIC;
37          Enable_BP   : out STD_LOGIC;
38          Enable_DISP : out STD_LOGIC);
39 end busArbiter;
40
41 architecture Behavioral of busArbiter is
42
43 begin
44    Enable_RAM  <= Enable AND (NOT(AddrBus(5)) OR (AddrBus(5) AND NOT(AddrBus(4))) OR (
   AddrBus(5) AND AddrBus(4) AND NOT(AddrBus(3))));
45    Enable_BP   <= Enable AND (AddrBus(5) AND AddrBus(4) AND AddrBus(3) AND NOT(AddrBus
   (2)) AND NOT(AddrBus(1)) AND NOT(AddrBus(0)));
46    Enable_DISP <= Enable AND ((AddrBus(5) AND AddrBus(4) AND AddrBus(3) AND NOT(
   AddrBus(2)) AND NOT(AddrBus(1)) AND(AddrBus(0))) OR (AddrBus(5) AND AddrBus(4) AND
   AddrBus(3) AND NOT(AddrBus(2)) AND AddrBus(1)));
47 end Behavioral;
48
49
```

```
1   ----------------------------------------------------------------------------------
2   -- Company: ENSEIRB-MATMECA
3   -- Engineer: Sylvain MARIEL (sylvain.mariel@otmax.fr)
4   -- Engineer: Thomas MOREAU  (thomas.moreau-33@hotmail.fr)
5
6   -- Create Date:    21/05/2013
7   ----------------------------------------------------------------------------------
8
9   library IEEE;
10  use IEEE.STD_LOGIC_1164.ALL;
11
12  entity BP_REG8bits is
13     Port (   Clk      : in  STD_LOGIC;                    -- Horloge
14              Rst      : in  STD_LOGIC;                    -- Reset asynchrone du
    composant
15              CE       : in  STD_LOGIC;                    -- Clock Enable, actication
    du composant
16              Load     : in  STD_LOGIC;                    -- Chargement de la valeur
    d'entrée
17              CLR      : in  STD_LOGIC;                    -- RAZ de Data_Out
18              Data_In  : in  STD_LOGIC_VECTOR(7 downto 0); -- Valeur d'entrée
19              Data_Out : out STD_LOGIC_VECTOR(7 downto 0));-- Valeur de sortie
20  end BP_REG8bits;
21
22  architecture Behavioral of BP_REG8bits is
23
24  begin
25
26     process (Clk,Rst)
27     begin
28
29        if (Rst = '1') then
30           Data_out <= "00000000";
31
32        elsif (Clk'event AND Clk = '1') then
33           if (CLR = '1') then
34                 Data_out <= "00000000";
35           elsif (CE = '1') then
36              if (Load = '1') then
37                 Data_out <= Data_In;
38              end if;
39           end if;
40
41        end if;
42
43     end process;
44
45  end Behavioral;
46
47
48
```

```
1  ----------------------------------------------------------------------------------
2  -- Company: ENSEIRB-MATMECA
3  -- Engineer: Sylvain MARIEL (sylvain.mariel@otmax.fr)
4  -- Engineer: Thomas MOREAU  (thomas.moreau-33@hotmail.fr)
5
6  -- Create Date:    21/05/2013
7  -- Design Name:
8  -- Module Name:    FSM - Behavioral
9  -- Project Name:
10 -- Target Devices:
11 -- Tool versions:
12 -- Description:
13 --
14 -- Dependencies:
15 --
16 -- Revision:
17 -- Revision 0.01 - File Created
18 -- Additional Comments:
19 --
20 ----------------------------------------------------------------------------------
21 library IEEE;
22 use IEEE.STD_LOGIC_1164.ALL;
23
24 -- Uncomment the following library declaration if using
25 -- arithmetic functions with Signed or Unsigned values
26 --use IEEE.NUMERIC_STD.ALL;
27
28 -- Uncomment the following library declaration if instantiating
29 -- any Xilinx primitives in this code.
30 --library UNISIM;
31 --use UNISIM.VComponents.all;
32
33 ENTITY BP_FSM IS
34     PORT ( Clk          : IN  STD_LOGIC;
35            Rst          : IN  STD_LOGIC;
36            CE           : IN  STD_LOGIC;
37            BP_NEXT      : IN  STD_LOGIC;
38            BP_PREV      : IN  STD_LOGIC;
39            BP_OK        : IN  STD_LOGIC;
40            BP_ENABLE    : IN  STD_LOGIC;
41            RegClear     : OUT  STD_LOGIC;
42            RegLoad      : OUT  STD_LOGIC;
43            Data_toReg   : OUT  STD_LOGIC_VECTOR (7 DOWNTO 0));
44 END BP_FSM;
45
46 ARCHITECTURE Behavioral OF BP_FSM IS
47
48 -- Types et signaux correspondant aux états de la FSM
49 TYPE etats IS (WAIT_RELEASE, WAIT_PRESS, BP_DETECT, WAIT_ENABLE);
50 SIGNAL etat_present_BP  :etats := WAIT_RELEASE;
51 SIGNAL etat_futur_BP    :etats := WAIT_RELEASE;
52
53 signal cond_wait_release : std_logic;
54 signal cond_wait_press : std_logic;
55
56 BEGIN
57
```

```
58      cond_wait_release <=    '1' WHEN BP_NEXT ='0' AND BP_PREV ='0' AND BP_OK ='0' ELSE
59                              '0';
60      cond_wait_press   <=    '1' WHEN BP_NEXT ='1' OR BP_PREV ='1' OR BP_OK ='1' ELSE
61                              '0';
62
63      -- Actualisation des etats presents à chaque coup d'horloge et gestion du reset
64      PROCESS (Clk, Rst)
65      BEGIN
66
67         IF (Rst = '1') THEN
68               etat_present_BP <= WAIT_RELEASE; -- En cas de reset, initialisation état A
69
70         ELSIF (Clk'event AND Clk = '1') THEN
71
72            IF(CE = '1') THEN
73               etat_present_BP <= etat_futur_BP; -- Actualisation des états
74            ELSE
75               etat_present_BP <= etat_present_BP;
76            END IF;
77
78         END IF;
79
80      END PROCESS;
81
82      -- Definition des etats futurs en fonction de la FSM
83      PROCESS (etat_present_BP, BP_NEXT, BP_PREV, BP_OK, BP_ENABLE, cond_wait_release,
    cond_wait_press)
84      BEGIN
85         CASE etat_present_BP IS
86
87            WHEN WAIT_RELEASE =>
88               --IF (BP_NEXT ='0' AND BP_PREV ='0' AND BP_OK ='0') THEN
89               IF (cond_wait_release = '1') THEN
90                  etat_futur_BP <= WAIT_PRESS;
91               ELSE
92                  etat_futur_BP <= WAIT_RELEASE;
93               END IF;
94
95            WHEN WAIT_PRESS   =>
96               --IF (BP_NEXT ='1' OR BP_PREV ='1' OR BP_OK ='1') THEN
97               IF (cond_wait_press = '1') THEN
98                  etat_futur_BP <= BP_DETECT;
99               ELSE
100                  etat_futur_BP <= WAIT_PRESS;
101               END IF;
102
103           WHEN BP_DETECT    =>
104              etat_futur_BP <= WAIT_ENABLE;
105
106           WHEN WAIT_ENABLE  =>
107              IF (BP_ENABLE ='1') THEN
108                 etat_futur_BP <= WAIT_RELEASE;
109              ELSE
110                 etat_futur_BP <= WAIT_ENABLE;
111              END IF;
112
113           WHEN OTHERS       =>
```

```
114               etat_futur_BP <= WAIT_RELEASE;
115
116            END CASE;
117         END PROCESS;
118
119
120         --Affectation des sorties si l'état présent a changé
121         PROCESS (etat_present_BP, BP_NEXT, BP_PREV, BP_OK) BEGIN
122
123            CASE etat_present_BP IS
124               WHEN  WAIT_RELEASE =>
125                  Data_toReg              <= "00000000";
126                  RegLoad                 <= '0';
127                  RegClear                <= '1';
128               WHEN  WAIT_PRESS =>
129                  Data_toReg              <= "00000000";
130                  RegLoad                 <= '0';
131                  RegClear                <= '0';
132               WHEN  BP_DETECT =>
133                  RegLoad                 <= '1';
134                  RegClear                <= '0';
135                  Data_toReg(0)           <= (BP_NEXT);
136                  Data_toReg(1)           <= (BP_PREV);
137                  Data_toReg(2)           <= (BP_OK);
138                  Data_toReg(7 DOWNTO 3)  <= "00000";
139               WHEN  WAIT_ENABLE =>
140                  RegLoad                 <= '0';
141                  RegClear                <= '0';
142                  Data_toReg              <= "00000000";
143               WHEN  OTHERS =>
144                  RegLoad                 <= '0';
145                  RegClear                <= '0';
146                  Data_toReg              <= "00000000";
147
148            END CASE;
149         END PROCESS;
150
151      end Behavioral;
152
153
154
155
156
```

```
1  ----------------------------------------------------------------------------------
2  -- Company: ENSEIRB-MATMECA
3  -- Engineer: Sylvain MARIEL (sylvain.mariel@otmax.fr)
4  -- Engineer: Thomas MOREAU  (thomas.moreau-33@hotmail.fr)
5
6  -- Create Date:    21/05/2013
7  -- Design Name:
8  -- Module Name:    BP - Behavioral
9  -- Project Name:
10 -- Target Devices:
11 -- Tool versions:
12 -- Description:
13 --
14 -- Dependencies:
15 --
16 -- Revision:
17 -- Revision 0.01 - File Created
18 -- Additional Comments:
19 --
20 ----------------------------------------------------------------------------------
21 library IEEE;
22 use IEEE.STD_LOGIC_1164.ALL;
23
24 -- Uncomment the following library declaration if using
25 -- arithmetic functions with Signed or Unsigned values
26 --use IEEE.NUMERIC_STD.ALL;
27
28 -- Uncomment the following library declaration if instantiating
29 -- any Xilinx primitives in this code.
30 --library UNISIM;
31 --use UNISIM.VComponents.all;
32
33 entity BP is
34    Port (Clk            : in  STD_LOGIC;
35          CE             : in  STD_LOGIC;
36          Reset          : in  STD_LOGIC;
37          BP_NEXT        : IN  STD_LOGIC;
38          BP_PREV        : IN  STD_LOGIC;
39          BP_OK          : IN  STD_LOGIC;
40          Enable         : IN  STD_LOGIC;
41          RW             : IN  STD_LOGIC;
42          DataBus_toCPU  : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
43 end BP;
44
45 architecture Behavioral of BP is
46
47    component BP_REG8bits
48          Port (   Clk         : in  STD_LOGIC;                    -- Horloge
49                   Rst         : in  STD_LOGIC;                    -- Reset asynchrone
   du composant
50                   CE          : in  STD_LOGIC;                    -- Clock Enable,
   actication du composant
51                   Load        : in  STD_LOGIC;                    -- Chargement de la
   valeur d'entrée
52                   CLR         : in  STD_LOGIC;                    -- RAZ de Data_Out
53                   Data_In     : in  STD_LOGIC_VECTOR(7 downto 0); -- Valeur d'entrée
54                   Data_Out    : out STD_LOGIC_VECTOR(7 downto 0));-- Valeur de sortie
```

```
55      end component;
56
57      component BP_FSM
58            PORT (   Clk         : IN  STD_LOGIC;
59                     Rst         : IN  STD_LOGIC;
60                     CE          : IN  STD_LOGIC;
61                     BP_NEXT     : IN  STD_LOGIC;
62                     BP_PREV     : IN  STD_LOGIC;
63                     BP_OK       : IN  STD_LOGIC;
64                     BP_ENABLE   : IN  STD_LOGIC;
65                     RegClear    : OUT  STD_LOGIC;
66                     RegLoad     : OUT  STD_LOGIC;
67                     Data_toReg  : OUT  STD_LOGIC_VECTOR (7 DOWNTO 0));
68      end component;
69
70      signal RegClear      : STD_LOGIC;
71      signal RegLoad       : STD_LOGIC;
72      signal Data_fsm2reg  : STD_LOGIC_VECTOR(7 downto 0);
73      signal Data_reg2bus  : STD_LOGIC_VECTOR(7 downto 0);
74      signal Enable_FSM    : STD_LOGIC;
75
76   begin
77      DataBus_toCPU <= not(Data_reg2bus);
78      Enable_FSM <= Enable WHEN RW = '0' ELSE '0';
79
80      BP_FSM_in : BP_FSM port map (
81         Clk,
82         Reset,
83         CE,
84         BP_NEXT,
85         BP_PREV,
86         BP_OK,
87         Enable_FSM,
88         RegClear,
89         RegLoad,
90         Data_fsm2reg (7 downto 0));
91
92      BP_REG_out: BP_REG8bits port map (
93         Clk,
94         Reset,
95         CE,
96         RegLoad,
97         RegClear,
98         Data_fsm2reg (7 downto 0),
99         Data_reg2bus (7 downto 0));
100
101   end Behavioral;
102
103
```