



ÉCOLE
CENTRALE LYON

ÉCOLE CENTRALE LYON

UE PRO
PE 23 - RÉTROGAMING
RAPPORT

Rapport final

Elèves :

Fabrice MOENECLAEY
Antoine LÉMERY
Ugo INSALACO
Badis ITIM
Mengqian ZOU
Maxime LAFONT

Tuteur scientifique :
Alberto BOSIO

Conseiller en communication:

Christophe CORRE

Conseiller en gestion de projet:

Gaylord GUILLOUNNEAU

Président de jury :

Laurent BLANC

Résumé

Français

Ce travail a pour but d'émuler matériellement une console Atari 2600 sur un type de carte programmable (FPGA) pour obtenir une console indépendante, la plus fidèle à celle d'origine. Pour émuler l'Atari, une étude approfondie de chaque composant de cette console a été menée. Le comportement de chacun de ces composants a été répliqué à l'aide du logiciel Vivado sur une carte programmable. Les fichiers simulant les différents composants n'ont pas été codés par notre équipe mais ils ont été trouvés sur internet parmi tout les travaux précédemment menés. En revanche, un travail de liaison de ces pièces a été effectué et des problèmes de compatibilité ont été résolus. Une émulation complète de la console a enfin été réalisée avec succès sur le logiciel Vivado, en revanche l'implémentation complète de l'Atari sur la carte n'a pas été mené à son terme. L'entièreté des codes utilisés ont été déposé en libre accès sur internet, pour permettre à d'autres personnes intéressées de reprendre le travail effectué.

English

This project aims to realize a hardware emulation of the Atari 2600 console via implementations on a programmable board (FPGA), in the very manner which is the most faithful to the original console. To simulate the Atari, deep research for each component was carried out, and behaviors of each component have been successfully replicated on the programmable board through the software Vivado. Those programming files emulating different components were not coded by our group, but were found among previous projects on the internet. Connections between components and compatibility problems were resolved on the other hand. A complete emulation of the console has finally been successfully achieved through the Vivado software, however, the whole implementation of the Atari has not been completed. All of the codes used in this project have been posted on the internet, allowing other persons interested in it to retrieve the realized work.

Remerciements

Nous tenons à remercier toutes les personnes qui ont contribué à ce projet d'étude.

Nous voudrions dans un premier temps remercier, notre tuteur scientifique M.Bosio, professeur à l'École Centrale de Lyon au département EEA (électronique, électrotechnique et automatique), pour sa disponibilité et surtout ses judicieux conseils, qui ont contribué à l'avancement du projet.

Nous remercions également nos conseillers en communication et en gestion de projet, qui nous ont permis de recentrer les objectifs du projet et de lui donner une direction et une méthodologie.

Nous tenons à témoigner toute notre reconnaissance aux personnes qui ont travaillé sur le projet A2601 du site retromaster.wordpress.com [1], qui ont travaillé sur les fichiers nécessaires à l'émulation hardware de l'Atari 2600 et qui les ont partagés.

Enfin, nous voudrions remercier l'Ecole Centrale de Lyon et son département EEA (électronique, électrotechnique et automatique), qui nous ont donné la possibilité et les moyens techniques de réaliser ce projet.

Table des matières

1	Introduction	5
2	Objectifs et approche	7
3	Rappels théoriques	8
3.1	Rappels d'électronique	8
3.2	FPGA	8
3.3	VHDL	9
3.4	Structure globale de l'Atari	10
4	Analyse des constituants de l'Atari	11
4.1	RIOT	11
4.2	CPU	13
4.2.1	Fonctionnement général du CPU	13
4.2.2	Du CPU 6502 au 6507	16
4.3	TIA	17
4.3.1	Description Générale	17
4.3.2	Aspect Visuel	18
4.3.3	Contrôles	19
5	Simulation des trois pièces principales	20
5.1	Recherche des pièces	20
5.2	Implémentation des pièces principales	20
5.3	Simulation et débogage	23
6	Reconstituer l'Atari en rassemblant les pièces	24
6.1	Lien entre les pièces	24
6.2	Adapter le code à notre carte	24
6.3	HDMI - VGA	25
7	Test de la console	26
7.1	Simulation de la console	26
7.2	Implémentation sur la carte FPGA	28
8	Conclusion	28
9	Bibliographie	29
10	Annexe	30
10.1	Lien du github	30
10.2	Signaux du RIOT	30
10.3	Table de vérité du RIOT	31
10.4	Schéma bloc du RIOT	31
10.5	Diagramme de GANTT modifié	32
10.6	Check-list	33

Table des figures

1	Atari 2600 (Evan Amos) ©Atari SA	5
2	NES mini ©Nintendo	6
3	Cahier des charges fonctionnel	7
4	Matrice FPGA ©Eduardo Sanchez, École polytechnique fédérale de Lausanne [4]	9
5	Carte FPGA ZYNQ-7010	9
6	Structure de l'Atari 2600	10
7	Description des bits du port A	12
8	Description des bits du port B	12
9	Exemple simple d'architecture de processeur [7]	13
10	Premier bloc de l'architecture	14
11	Une ALU à deux entrées [8]	14
12	Brochages des processeurs	16
13	Architecture de la sous-pièce TIA	17
14	Format de l'affichage de l'Atari	18
15	Schéma de l'implémentation du CPU	21
16	Lien entre les différents pièces de l'Atari	24
17	Test du convertisseur HDMI-VGA	25
18	Test de différents jeux	27
19	Signaux du RIOT	30
20	Table de vérité du RIOT	31
21	Schéma bloc du RIOT	31
22	Nouveau diagramme de GANTT	32

1 Introduction

En 1977, la société Atari lance sa console Atari 2600 (Figure 1), une console qui dans son architecture matérielle révolutionna l'industrie du jeu vidéo. Cette console rencontra un franc succès avec plus de 30 millions d'unités vendues à travers le monde et propulsa le marché du jeu vidéo dans une nouvelle ère. Aujourd'hui ce marché ne cesse de se développer et d'attirer de nouveaux joueurs. Néanmoins, on peut observer chez de nombreux utilisateurs une certaine nostalgie qui les pousse à vouloir rejouer aux jeux de leur enfance. C'est alors que de nombreux constructeurs de jeux vidéo remettent sur le marché leurs anciennes consoles à l'instar de Nintendo en 2016 avec sa NES ressortie en version mini (Figure 2) puis avec la SNES Mini. Tous les constructeurs de l'industrie s'y sont essayés, et on peut même remarquer qu'un des principaux arguments de vente de la prochaine console de Microsoft (Xbox Series X) est qu'elle peut aussi émuler les jeux des anciennes Xbox. Ainsi le marché du rétrogaming est en plein essor et c'est de ce constat qu'est né ce projet consistant à réaliser une émulation matérielle de l'Atari 2600 sur une carte programmable achetabile aujourd'hui dans le commerce.



FIGURE 1 – Atari 2600 (Evan Amos) ©Atari SA

Cette carte est une carte FPGA, choisie pour deux raisons. La première est qu'elle permet d'avoir une structure matérielle pour implémenter l'émulateur (émulation hardware), qui contrairement à l'émulation software (implémenté comme programme interne à un ordinateur), permet de retrouver l'esprit d'une console de salon, qui est primordiale pour les joueurs. La seconde est son utilisation. En effet, des composants de beaucoup de systèmes électroniques du passé sont désormais en fin de vie (comme notamment des processeurs), et trouver leurs pièces de rechange (qui ne sont plus commercialisées) est très difficile. Pour remédier à ce problème on peut soit utiliser un modèle différent (cependant des problèmes de compatibilité peuvent apparaître avec les autres composants), soit réaliser un émulateur dit 'hardware'. Pour cela, on choisit d'utiliser une carte FPGA : en effet avec cette carte on peut répliquer le fonctionnement du composant manquant à l'identique sans problème de compatibilité. Ce projet s'inscrit donc aussi dans une problématique plus générale à celle du jeu-vidéo pour venir appréhender la résolution de problèmes concernant plus globalement les systèmes électroniques de toutes sortes.



FIGURE 2 – NES mini ©Nintendo

L'objectif global est donc d'avoir une émulation hardware de l'Atari 2600 sur une carte FPGA. Cette émulation devrait permettre d'ajouter des manettes et un écran directement reliés à la carte FPGA pour une expérience de jeu optimale.

Pour décrire ce projet, nous allons d'abord devoir présenter quelques éléments de théorie de l'électronique ainsi que décrire les éléments techniques essentiels au projet. Ensuite, nous détaillerons la structure et le fonctionnement des pièces constituants l'Atari 2600. Puis, nous évoquerons la simulation des pièces constituants la console, partie allant de la recherche des pièces jusqu'au débogage. Enfin, nous expliciterons la reconstitution de l'Atari à partir des pièces la constituant ainsi que les tests effectués sur la console.

2 Objectifs et approche

Comme énoncé en introduction, l'objectif est d'avoir une émulation hardware de l'Atari 2600 sur une carte FPGA. Le but est de reproduire le plus fidèlement possible l'expérience de jeu que l'on peut ressentir sur une vraie Atari 2600. Un autre objectif est de pouvoir déplacer aisément la carte sans avoir l'ordinateur à côté de soi pour jouer, ce qui implique que les jeux soient stockés dans la carte elle même. Un cahier des charges fonctionnel résume ces objectifs dans un tableau.

Priorité	Fonction Principale	Fonction Secondaire	Critère
F0	Emuler l'Atari 2600		Utilisation d'une carte FPGA
F1	Avoir un système autonome	Stocker des jeux Stocker les programmes	Utilisation d'une carte FPGA
F2	Avoir une émulation proche de l'expérience de jeu originale	Avoir des graphismes identiques	Gestion des 128 couleurs Résolution : 160x192 pixels
		Emuler avec une fluidité adaptée	60Hz (norme NTSC)
		Retransmettre les sons de la console	Plage de fréquence de 937,5Hz à 30kHz Contrôle des fréquences sur 5 bits
		Contrôler le jeu	Utilisation d'un joystick du commerce
F3	Réaliser le projet avec le budget fourni		300 €
F4	Réaliser le projet dans le temps imparti		De septembre 2019 à juin 2020

FIGURE 3 – Cahier des charges fonctionnel

Afin de répondre au problème posé, une étude des 3 composants de l'Atari 2600 sera menée, cette étude permettra de réaliser l'implémentation informatique de ces composants qui une fois réuni ensemble donneront l'émulation de la console Atari sur la carte FPGA.

3 Rappels théoriques

Pour réaliser une émulation hardware de l'Atari 2600 sur une carte FPGA, nous utilisons le logiciel Vivado qui permet à partir d'un programme écrit sur un fichier de type VHDL, d'implémenter sur la carte FPGA ce programme. Le logiciel Vivado va donc régler les portes logiques de la carte FPGA comme indiqué par le programme afin de reproduire le comportement des pièces constituants l'Atari.

Nous allons commencer par des rappels d'électronique ainsi qu'une rapide explication de ce qu'est une carte FPGA, et le language VHD. Puis, nous présenterons l'architecture de l'Atari 2600.

3.1 Rappels d'électronique

Dans toute la suite, nous allons utiliser un vocabulaire de logique combinatoire et séquentielle. En effet, nous allons utiliser une carte FPGA qui est constituée de portes logiques programmables.

La logique peut être combinatoire, c'est-à-dire, une logique qui traite les informations binaires logiques "0" et "1", appelé aussi bit, avec des opérateurs booléens simples. Ces bits peuvent former un bus de données qui permet la circulation des données, y compris les instructions d'un programme entre la mémoire et le processeur. Sa taille est variable, elle dépend de la taille des mots transférés en un cycle. Les portes logiques, quant à elles, sont des circuits électroniques, qui combinent les signaux logiques présentés à leurs entrées sous forme de tensions [2].

La logique peut aussi être séquentielle lorsque la ou les sorties, à un instant t , dépendent des entrées à l'instant t et de l'état des sorties à l'instant $t-1$. Dans les systèmes séquentiels dit synchrone, on applique un signal, synchronisant les portes logiques, que l'on appelle une horloge. Un signal d'horloge est, un signal électrique, souvent carré, qui rythme les actions d'un circuit. Sa période est appelée cycle d'horloge [3].

3.2 FPGA

La carte FPGA, de son nom complet Field-programmable gate array, ou réseau de portes programmables est constitué de circuits composés des cellules logiques élémentaires qui sont connectées par programmation, comme indiqué dans la figure 4. Cette connexion est réversible et ajustable, ce qui permet de réaliser les fonctions numériques voulues.

Utilisés dans diverses applications nécessitant de l'électronique numérique, la carte FPGA possède plusieurs avantages :

- un délai de mise sur le marché plus court, car ce sont des composants standards.
- un temps de conception plus court, car on réutilise des fonctions de base dont la reconfiguration autorise une validation préalable moins stricte.
- un coût inférieur pour de petites séries (moins de 10 000 unités).

Dans notre projet, on choisit la carte FPGA ZYNQ-7010 (Figure 5). En branchant cette carte par interface USB à un ordinateur muni du logiciel Vivado, où nos codes sont préprogrammés, on associe les signaux définis dans les codes aux composants de cette carte, et ainsi on réalise les fonctions voulues. Les composants comme les boutons sur la carte permettent de réaliser les signaux physiques d'entrée, représentant les actions du joueur. C'est de cette façon que l'on passe du logiciel au matériel.

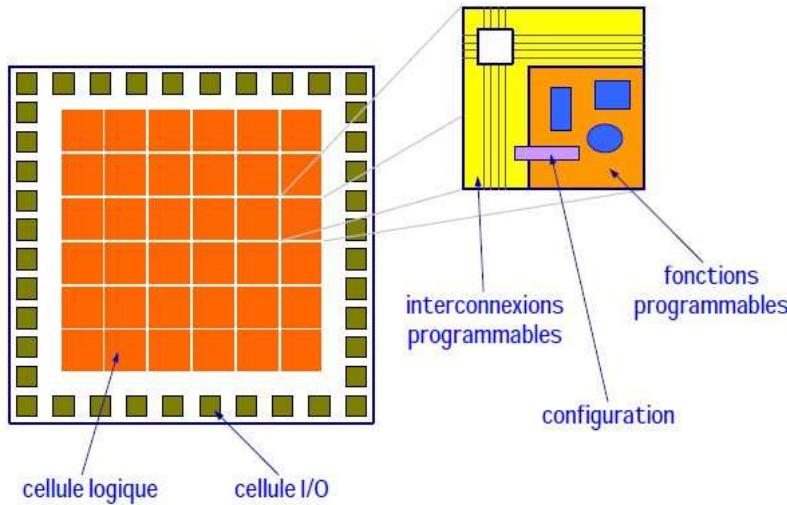


FIGURE 4 – Matrice FPGA ©Eduardo Sanchez, École polytechnique fédérale de Lausanne [4]

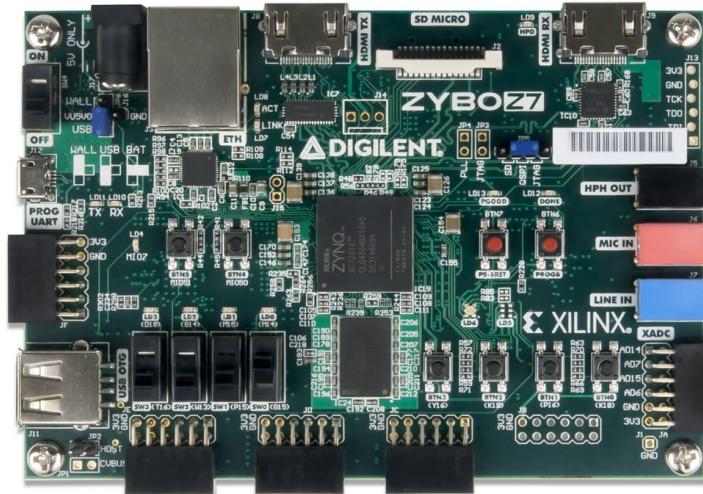


FIGURE 5 – Carte FPGA ZYNQ-7010

3.3 VHDL

Le langage VHDL est un diminutif de VHSIC1 (Very High Speed Integrated Circuit) Hardware Description Language.

Comme son nom l'indique, il s'agit d'un langage destiné à représenter le comportement et l'architecture d'un système matériel. Différent des langages développés pour créer des logiciels, le language VHDL insiste sur le parallélisme à l'intérieur d'un circuit, et contient deux aspects principaux :

- *entity* : C'est l'entête du composant : elle permet de définir les entrées et sorties de la pièce future. L'*entity* permet de résumer quelles seront les moyens d'interagir avec le composant et à travers quelle forme de données (bus de huit bits, signal d'un seul bit etc...).
- *architecture* : il décrit la structure du circuit, son fonctionnement logique interne, les signaux intermédiaires utilisés et si besoin des composants imbriqués à l'intérieur du circuit lui même. L'*architecture* décrit généralement des fonctions, nommées *process*, prenant en entrée certains signaux et en retournant d'autres calculés entre

temps. Un process est alors effectué chaque fois que l'horloge change d'état. Avec les outils de la plate-forme Vivado, la programmation en VHDL nous permet de simuler et vérifier les fonctionnements d'une conception du système électronique avant que cette conception soit réalisé en matériel, ce qui simplifie le processus de fabrication. De plus, le grand avantage du langage VHDL est qu'il permet de passer d'une description fonctionnelle à un schéma en porte logique.

3.4 Structure globale de l'Atari

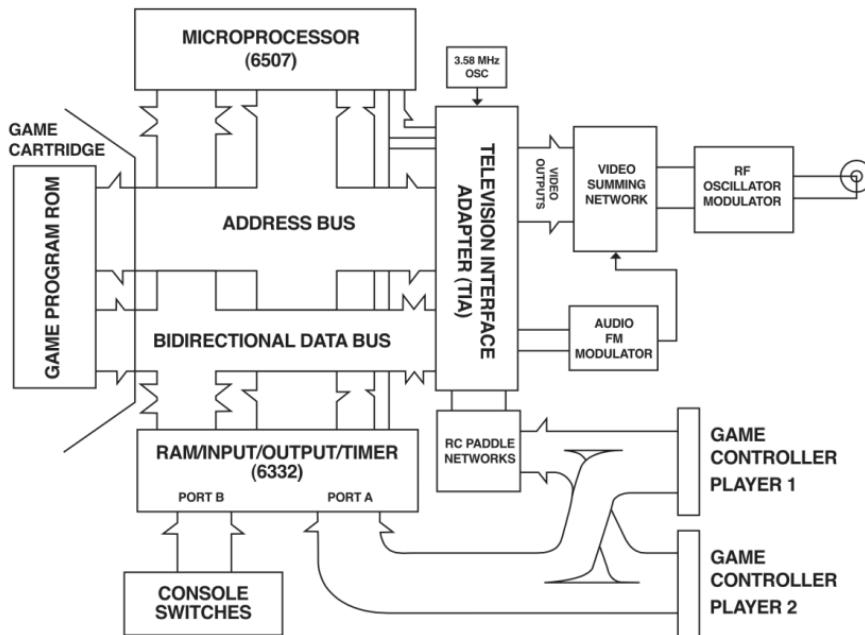


FIGURE 6 – Structure de l'Atari 2600

L'Atari 2600 est constituée de trois composants principaux : le RIOT (RAM/INPUT/OUTPUT/TIMER sur la figure 6), le TIA (Television Interface Adapter) et le CPU (microprocessor). La cartouche de jeu est un composant principal, mais elle ne fera pas partie des fichiers VHDL de la console à proprement parler. Ces trois composants ont été étudiés par groupes de deux personnes afin de comprendre aux mieux les fichiers VHDL qui leur correspondent.

4 Analyse des constituants de l'Atari

La fameuse console de la société Atari, est constitué de trois principaux composants : le RIOT, le CPU, et le TIA. Pour pouvoir implémenter cette console sur la carte FPGA, il nous est nécessaire de comprendre le fonctionnement de l'Atari, et donc, de ces trois principaux composants. Une description de chaque composant est proposée par la suite.

4.1 RIOT

Le RIOT (RAM Input Output Timer) est une puce à 40 branches qui est **en liaison avec le microprocesseur** et qui reçoit toutes les informations venant **des périphériques externes** (manettes et interrupteurs sur la console). Elle est composé d'une RAM de 128 octets (1 octet est égal à 8 bits) et d'une horloge programmable. Le RIOT est en liaison avec le microprocesseur à l'aide de 8 bits bidirectionnels et avec les périphériques externes par deux port de 8 bits en parallèle. L'ensemble de ces informations qui suivent sont tiré des documents [5] et [6].

RAM

La RAM du RIOT est une RAM classique, c'est à dire une **mémoire vive**, les informations dans celle-ci sont stockées temporairement. Sa capacité est de 1024 bits. On peut écrire sur cette RAM à l'aide de 3 bits de contrôle. L'adresse est donnée par 7 bits pour sélectionner l'emplacement dans la mémoire.

Input Output et ses registres

Les 16 bits dédiés aux périphériques sont divisés en deux ports A et B. Ces deux ports peuvent être utilisés soit comme entrée soit comme sortie (c'est pour cela qu'ils sont bidirectionnels), plus précisément chacun des bits du port peut être programmé comme entrée ou sortie . Pour choisir si un bit est une entrée ou une sortie il faut rentrer un octet (8 bits) dans le Data Direction Register (DDR). Ce registre code donc la manière dont chaque bit du port émet ou reçoit de l'information, pour qu'un bit soit considéré comme une entrée il faut mettre un 0 dans le bit correspondant à l'entrée dans l'octet et un 1 pour une sortie. Le port A reçoit les commandes **venant des deux manettes** tandis que le port B reçoit les informations **des boutons sur la console** comme ceux pour modifier la difficulté ou pour mettre en pause la partie. Pour l'Atari 2600 le port B est toujours défini comme une entrée, le port A quant à lui peut être configuré comme une sortie, dans ce cas il permet d'activer ce qu'on appelle des "keyboard controllers" qui sont utilisés pour certains jeux.

Data Bit	Direction	Player
D7	right	P0
D6	left	P0
D5	down	P0
<u>D4</u>	up	P0
D3	right	P1
D2	left	P1
D1	down	P1
D0	up	P1

(P0 = left player, P1 = right player)

FIGURE 7 – Description des bits du port A

Data Bit	Switch	Bit Meaning
D7	P1 difficulty	0 = amateur (B), 1 = pro (A)
D6	P0 difficulty	0 = amateur (B), 1 = pro (A)
D5/D4	(not used)	
D3	color - B/W	0 = B/W, 1 = color
D2	(not used)	
D1	game select	0 = switch pressed
D0	game reset	0 = switch pressed

FIGURE 8 – Description des bits du port B

Timer

Le Timer peut être considéré comme un compteur, il permet de mesurer combien de temps s'est écoulé depuis un certain temps, ou de compter tout simplement, il a donc plusieurs fonctionnalités. L'avantage de celui du RIOT est qu'il est programmable, nous allons voir ce que cela veut dire par la suite. Le compteur du RIOT peut compter jusqu'à 255 intervalles de temps. Ces intervalles de temps peuvent prendre différentes valeurs : 1T, 8T, 64T et 1024T où T est la période de l'horloge du système, c'est pour cela qu'on dit que le compteur est programmable. Une fois que le compteur arrive à 255 intervalles de temps, un signal dit "interrupt flag" est activé avec la valeur 1, à partir de là le Timer se met à compter jusqu'à 255 intervalles de temps avec un intervalle qui est égale à la période T. Ceci permet à l'aide d'une opération Read Timer de savoir depuis quand le signal "interrupt flag" a été activé.

Bits de commande et bits d'adresse

Le RIOT possède 2 bits de commande CS1, $\overline{CS2}$, 2 bits de sélection \overline{RS} , R/ \overline{W} , et 7 bits d'adresse (A0-A7). Ces bits de commande et d'adresse permettent de sélectionner soit la RAM, soit l'intervalle du timer ou d'autres fonctionnalités du RIOT. Ces bits définissent le comportement du RIOT, ils sont émis par le microprocesseur. Le microprocesseur choisit la valeur de ces bits en fonction de l'information dont il a besoin. Ces bits de commandes et d'adresse permettent aussi de programmer le Timer en choisissant à combien de période d'horloge correspond un intervalle de temps.

4.2 CPU

Le CPU (central processing unit), processeur en français, est la partie 'intelligente' de la console : c'est lui qui traite les différentes instructions qu'on demande (les instructions forment un programme décrivant un algorithme). Il est la partie centrale de la console. L'Atari 2600 étant la première console populaire à utiliser un CPU, car ces derniers étaient trop chers auparavant. Son CPU est alors un CPU 6507, version low cost d'un CPU populaire : le 6502. Le 6507 ne contient que peu de documentations contrairement au 6502.

Ces CPU étant des CPU de fonctionnement très classiques, il convient de détailler en premier lieu le fonctionnement général d'un CPU, puis d'étudier les spécificités du 6502 et enfin d'arriver au 6507 de l'Atari 2600.

4.2.1 Fonctionnement général du CPU

Le fonctionnement d'un CPU sera expliqué grâce à l'exemple suivant, à plus petite échelle que le 6507, mais qui toutefois modélise bien son fonctionnement.

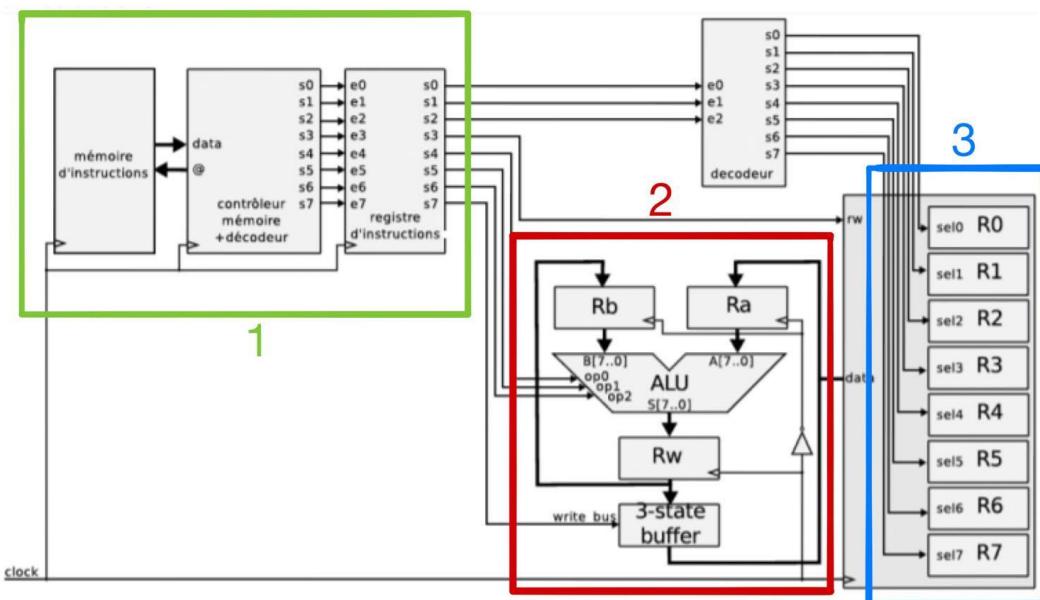


FIGURE 9 – Exemple simple d'architecture de processeur [7]

Le CPU se décompose de manière générale en plusieurs parties qui interagissent entre elles. Il y a tout d'abord une partie qui initialise les instructions (visible avec le cadre vert). Une autre partie réalise les instructions (au centre dans le cadre rouge), son élément le plus essentiel est l'ALU. Enfin, les registres forment le dernier groupement du CPU (visibles à droite dans cet exemple avec 8 registres R0 à R7 dans le cadre bleu).

Décodage des instructions

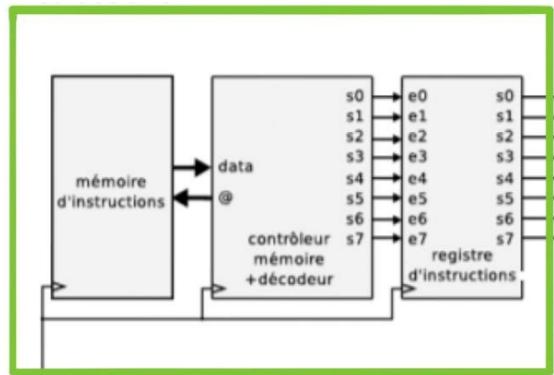


FIGURE 10 – Premier bloc de l'architecture

Ce premier bloc est lié au décodage des instructions. Les instructions sont stockées dans une mémoire d'instructions, qui dans le cas réel n'est pas réellement dans le CPU comme ici mais une entrée de celui-ci. Ensuite, l'instruction à exécuter est décodée. Une machine à états finis est requise ici pour séquencer l'instruction. Elle permet de décrire l'instruction par une successions d'états. Grâce à cette machine, le CPU change d'état en une succession finie, ce qui augmente son efficacité, et la logique du CPU est par la même occasion plus claire. En effet, une instruction étant réalisée en plusieurs fronts montants d'horloge, il faut la décomposer en sous-instructions élémentaires qui elles peuvent être réalisées en un seul front d'horloge. Ces micro instructions sont ensuite envoyées à la suite dans un registre d'exécution, (ie une mémoire de la micro-instruction en cours d'exécution). Celle-ci est alors envoyée pour traitement.

ALU

L'ALU (Arithmetic logic unit), UAL en français, est un composant inclus dans CPU pour effectuer les calculs.

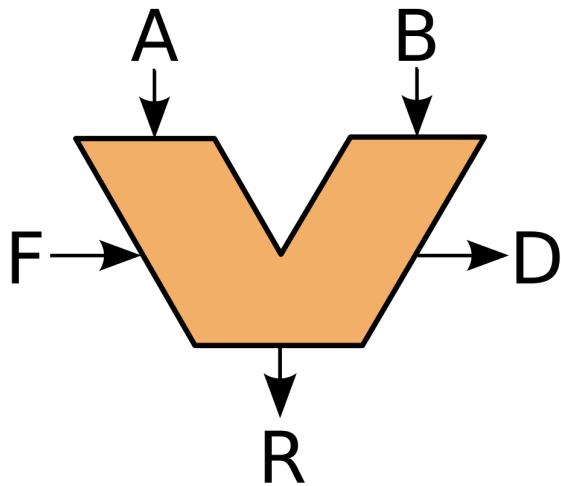


FIGURE 11 – Une ALU à deux entrées [8]

La figure ci-dessus représente un schéma d'un ALU, dans lequel il y a deux entrées A et B recevant les données à traiter, une autre entrée F désignant l'opération à effectuer et

deux sorties R et D. La sortie R est le résultat de l'opération, et la sortie D renvoie elle d'éventuels "drapeaux" qui informe de possibles erreurs.

Dans le CPU, l'ALU est un composant crucial pour exécuter les instructions : en effectuant les calculs des opérations décrites par les instructions, et en envoyant pour stockage les résultats dans les mémoires, il permet d'exécuter les programmes d'instruction, et ainsi réalise à lui seul la partie contrôle.

Registres

Enfin, les registres sont les mémoires des variables utilisées par le CPU, celles sur lesquelles il réalise les calculs. Elles sont donc stockées à proximité immédiate de l'ALU. Il est important de faire attention et à distinguer les registres à d'autres mémoires telles que la ROM présentes dans la console complète.

Cohérence globale du CPU

Ces blocs étant détaillés, il convient d'expliciter comment ils fonctionnent entre eux. Notons tout d'abord que le CPU est cadencé par une horloge en entrée (de fréquence élevée) et fonctionne lors de ses fronts montants (lorsque sa valeur passe de 0 à 1). L'instruction devant être exécutée est choisie et initialisée grâce au premier bloc. En sortie de ce dernier, l'instruction se décompose en deux parties principales et un bit distinct.

La première partie (ici composée des 3 premiers bits pour 8 possibilités de nombres), désigne le registre sur lequel l'instruction doit être exécutée. Un décodeur permet alors de sélectionner ce registre (en passant son bit de sélection à 1 et en mettant les autres à 0). Enfin la seconde partie de l'instruction est elle envoyée à la partie opérande (celle de l'ALU) et désigne l'opération que l'ALU doit effectuer.

Enfin, l'ALU sait si la variable sélectionnée du registre est celle sur laquelle il doit effectuer une opération, ou si il doit envoyer son résultat au registre sélectionné pour l'écrire dessus grâce au bit seul entre les deux parties principales de l'instruction.

Le fonctionnement du CPU étant explicité, la suite est de le transposer à l'Atari 2600.

4.2.2 Du CPU 6502 au 6507

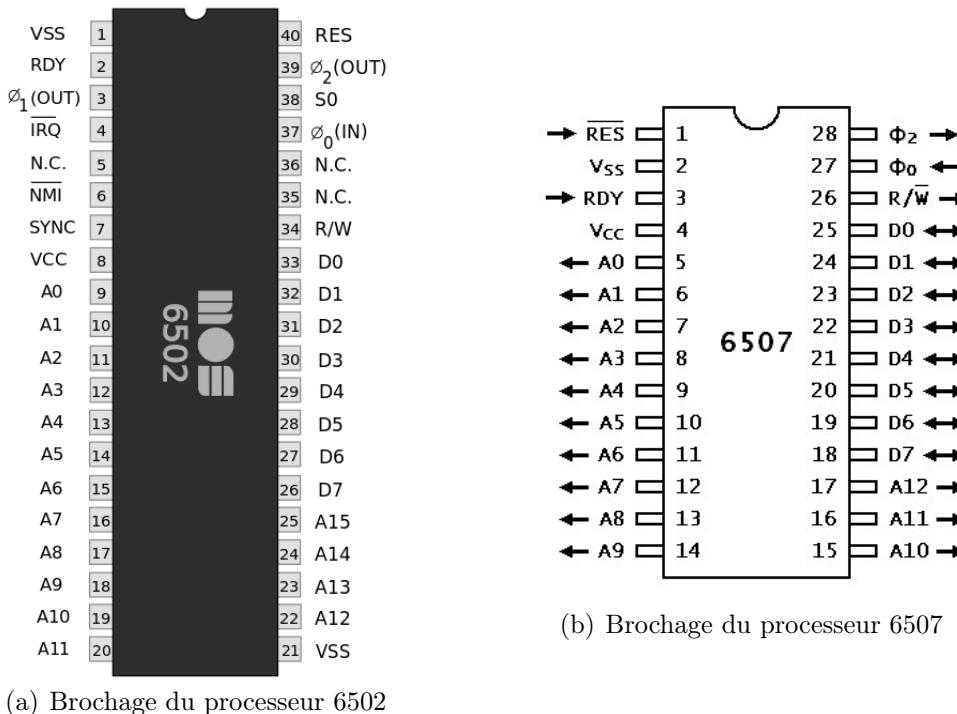


FIGURE 12 – Brochages des processeurs

Description technique du 6502

Il fonctionne sous **8 bits**, représentés par D0 à D7 dans la figure ci-dessus (qui correspondent à l'instruction reçue). Il est cadencé grâce à une **horloge de fréquence 1.19 MHz**. Il peut échanger 64 kbits de mémoires et il est possible de l'interrompre en cas de nécessité. Comme indiqué dans la figure, il contient 16 lignes d'adresse (du A0 au A15, d'où les $2^{16} = 64$ kbits de mémoire disponible). Enfin ses instructions suivent un jeu d'instructions RISC (Reduced instruction set compute), qui correspond à un mode d'architecture matérielle de processeurs avec des instructions de base simples.

Différences du 6507

Le 6507, celui présent dans la console, pour des raisons budgétaires, a seulement **13 lignes d'adresses** (représentés par A0 à A12 dans la figure du brochage du processeur 6507), ce qui limite en même temps **la mémoire que peut échanger le CPU à 8 kbits ($=2^{13}$)**. Il perd également quelques interruptions possibles. Mais il est identique au 6502 pour tout le reste et ils utilisent les mêmes instructions.

Les 8 kbits de mémoire disponible sont utilisés par l'Atari de deux manières différentes. D'une part, il y'a 4 kbits dédiés à la mémoire pour les entrées et sorties de la console et d'autre part 4 kbits réservés aux programmes traités par le CPU. Chacune de ses instructions est réalisée en 1 à 5 cycles (soit périodes) de l'horloge car le processeur peut réaliser à la suite des transferts de données et exécuter des instructions.

4.3 TIA

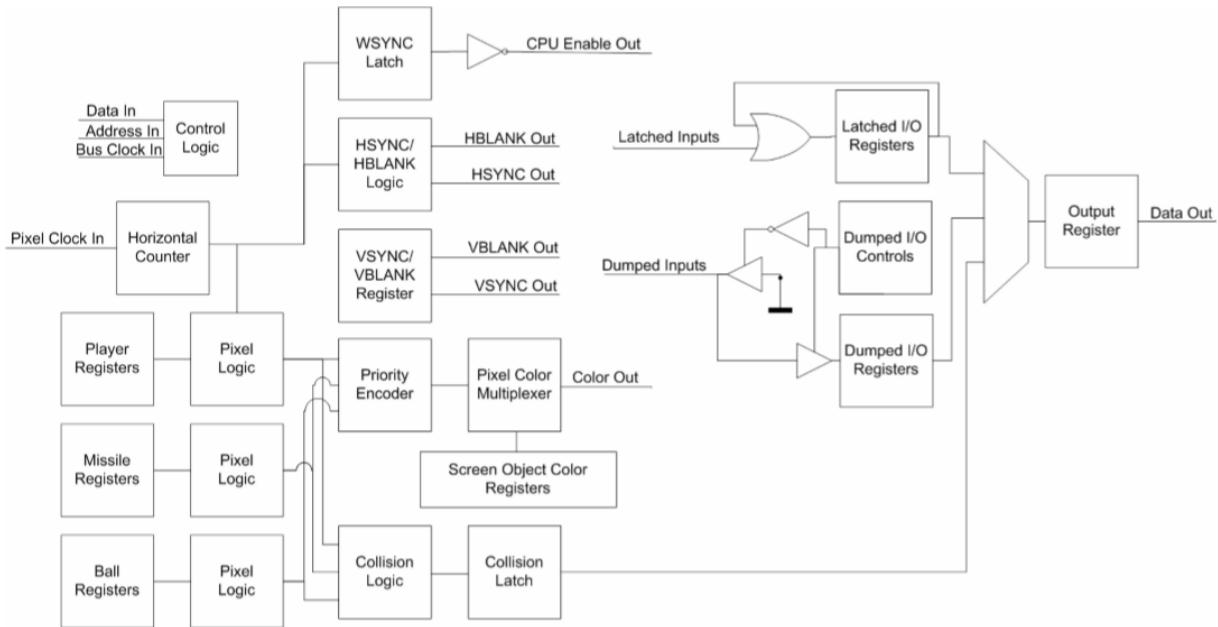


FIGURE 13 – Architecture de la sous-pièce TIA

4.3.1 Description Générale

Le TIA (Television Interface Adaptator) est une puce personnalisée créée par Atari. Celle-ci gère l'affichage de l'écran, les effets sonores et la lecture des commandes sur lesquelles appuient l'utilisateur. Une description générale de cette pièce est donnée par le Stella Programmer's Guide [Steve Wright, 1979]. Nous n'allons pas nous intéresser ici à l'aspect sonore ni à la façon dont celui-ci est géré par le TIA.

Le TIA communique avec le CPU et le RIOT, il reçoit une instruction à effectuer (sous la forme d'un bus de données) et renvoie les données relatives au résultat de cette instruction. Par exemple l'instruction 'CXMOP' envoyée au TIA correspond à la demande du processeur si le missile du joueur 0 (M0) est entré en collision (CX) avec un joueur (P). Le retour de cette instruction est un mot de deux bits correspondant à la collision entre M0 et P0 et la collision entre M0 et P1 (le principe des collisions est expliqué plus bas).

Le Protocole de télévision

Le TIA permet de gérer l'affichage de l'Atari. Cet affichage s'effectue pixel par pixel, ligne par ligne et image par image sur un format de 160x192 plus un certain nombre de pixels "blancs". Ces pixels blancs permettent au TIA d'avoir le temps pour effectuer les calculs quant aux informations des pixels qui seront affichés dans la ligne à venir. On appelle ces périodes d'affichage blanc le "vertical blank" et "horizontal blank".

Ce sont les variables HSYNC (horizontal synchronisation) et VSYNC (vertical synchronisation) qui permettent de définir si l'on se situe en période d'affichage blanc ou non, vertical ou horizontal (pour information le Latch WSYNC signifie wait for synchronisation et défini plus généralement si l'on est en train d'attendre la fin de l'affichage blanc ou non).

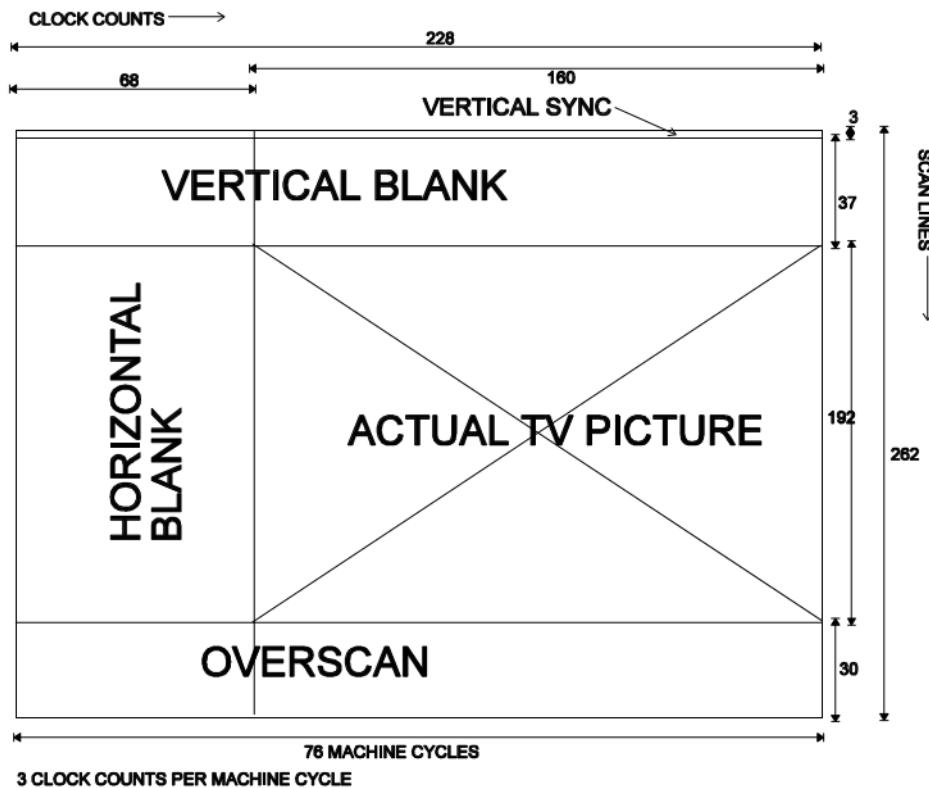


FIGURE 14 – Format de l'affichage de l'Atari

4.3.2 Aspect Visuel

Les objets dynamiques

Le terrain de jeu est constitué d'un fond, de 2 joueurs (ou un seul le cas échéant), de deux missiles et d'une balle. Un terrain de jeu fait de murs, de nuages, de barrières peut être ajouté au dessus du fond uniforme. Chacun de ces objets sont créés et manipulés par le TIA sur des séries de registres sur lesquelles le microprocesseur peut écrire et lire. Ces objets ont tous des caractéristiques différentes. Par exemple le fond ne peut pas bouger à moins d'être totalement réécrit, alors que les joueurs peuvent bouger à l'aide d'une seule instruction.

Format d'affichage

Les couleurs et la luminosité sont assignées au fond et aux 5 objets dynamiques. Elles sont gérées par le bus de données COLU (color luminescence). Chaque objet dynamique possède aussi un registre NUSIZ (number size) qui comme son nom l'indique permet de modifier le nombre de fois où l'on affiche cet objet (si l'on veut pouvoir contrôler plusieurs joueurs simultanément par exemple) et la taille de l'objet à afficher.

Priorité d'affichage et collisions

Certain objets nécessitent d'être affichés devant d'autre (cela semblerait étrange de vouloir afficher un joueur derrière le background par exemple). Au moment de la superposition de deux pixels correspondant à deux objets distincts il faut donc faire un choix entre les deux. Le TIA possède donc une table de priorité qu'il applique afin de résoudre ces conflits (le joueur s'affiche devant le background, le joueur 1 est affiché devant le joueur 2 etc...). C'est grâce au bloc priority encoder que le TIA effectue cette tâche.

Les collisions entre plusieurs objets sont détectées par le TIA (dans le bloc collision logic). Chaque collision entre 2 objet correspond à un bit dans le signal 'CX' du TIA (CX a donc 15 bits) qui vaut 1 s'il y a une collision et 0 sinon. Le contenu du bus CX peut être au besoin accédé en lecture par le CPU.

4.3.3 Contrôles

Il y a six ports d'entrées pour des contrôles dans l'Atari, ceux-ci sont séparés en deux types : les dumped input port et les latch input port. Deux dumped input port sont utilisés pour pouvoir connecter jusqu'à deux manettes. Le joystick de la manette sert à charger une capacité électrique dans l'atari qui est l'élément de base pour pouvoir déterminer la position du joueur dans le jeu. Les deux latch input ports restants sont connectés aux boutons d'activation des manettes.

Pour résumer, l'Atari 2600 est constituée de trois pièces fondamentales. Premièrement, le RIOT qui est une puce recevant toutes les informations venant des périphériques externes, elle gère donc les entrées et les sorties, Input/Output en anglais. De plus, c'est cette pièce qui contient la RAM, c'est-à-dire la mémoire vive de la console, ainsi qu'un horloge programmable. Deuxièmement, le CPU ou processeur, c'est la partie de la console qui traite les instructions et réalise les calculs. Et troisièmement, le TIA (Television Interface Adaptator) qui gère l'affichage de l'écran, les effets sonores et la lecture des commandes sur lesquelles appuient l'utilisateur.

5 Simulation des trois pièces principales

Une fois le fonctionnement des trois composants étudié, il était temps de passer à la partie simulation. Chaque groupe se mit donc à la recherche des fichiers VHDL correspondant à leur pièce respective. Puis, avec les fichiers VHDL obtenus, chaque groupe s'est intéressées à l'implémentation de leur pièce sur le logiciel Vivado. Cette dernière étape a nécessité de passer par une période de débogage et de réaliser de nombreuses simulations.

5.1 Recherche des pièces

Nous devions trouver les fichiers pour simuler les trois composants principaux de l'Atari. Cependant, ces fichiers devaient être compatible avec le logiciel à notre disposition, Vivado, et ils devaient être compatibles entre eux car la finalité du projet est de réunir ces composants pour reconstituer l'Atari 2600. Cependant, il nous était impossible d'imposer à nos fichiers d'être compatible avec notre carte FPGA vu le nombre de carte FPGA différente disponible sur le marché. Ces contraintes ont limité le nombre de fichiers trouvés, en effet, très peu de fichiers VHDL des composants de l'Atari 2600 étaient disponible sur l'Internet. De plus, parmi le peu de fichier présent, certains [9] utilisaient une bibliothèque d'Altera , qui n'existe pas sur Vivado et qui nous est donc inutilisable. Finalement, nous avons trouvé les fichiers VDHL [1] de chaque composants adéquats, ils sont compatibles entre eux et avec le logiciel Vivado . L'étape suivante est donc de tester et d'implémenter ces fichiers.

5.2 Implémentation des pièces principales

Implémentation du RIOT sous Vivado

Il faut désormais implémenter le RIOT sur le logiciel Vivado, il va donc falloir répliquer le comportement du RIOT et créer des composants comme la RAM.

La RAM du RIOT comporte au **128 octets**. Il faut choisir une structure de donnés pour modéliser cette RAM, on choisit d'utiliser **une matrice** de taille 128×1 où chaque élément de la matrice est lui même un vecteur de taille 8×1 . On peut faire deux opérations sur la RAM, on peut soit la lire en donnant une adresse et elle renvoie donc le contenu du registre à cette adresse (Read), soit on peut écrire en donnant une adresse et un contenu à écrire (Write) . Pour l'opération Read il s'agit juste d'affecter aux bus de données le contenu du registre de l'adresse. Pour l'opération Write on change la valeur de l'octet (ici c'est donc un vecteur) correspondant à l'adresse donnée par la valeur appliquée en entrée.

On a vu que le RIOT renvoie des informations au microprocesseur en fonction de ce que celui ci envoie, le microprocesseur peut aussi envoyer des commandes à exécuter par le RIOT comme configurer le port A du RIOT en entrée. On dispose des tables de vérités du RIOT (voir annexe) qui nous permettent d'affecter à la sortie du RIOT les informations demandé par le microprocesseur ou d'exécuter les opérations d'écriture. Ces tables de vérités sont traduites sur Vivado par des instructions if, then et de simples affectations de variables.

Implémentation du CPU sous Vivado

Le CPU ayant été précédemment décrit, il reste à étudier son implémentation sur la carte FPGA.

Sous Vivado, son modèle utilise également la description par étape, comme nous venons de le faire, en passant par le populaire 6502. Les différents programmes sont donc interdépendants et "s'emboîtent" les uns les autres.

Notons également que les signaux ont été définis dessus de la même manière que sur le brochage (par exemple, les 13 lignes d'adresses sont dans un vecteur A de taille 13).

Les fichiers VHDL ou .vhd se schématisent alors de la manière suivantes :

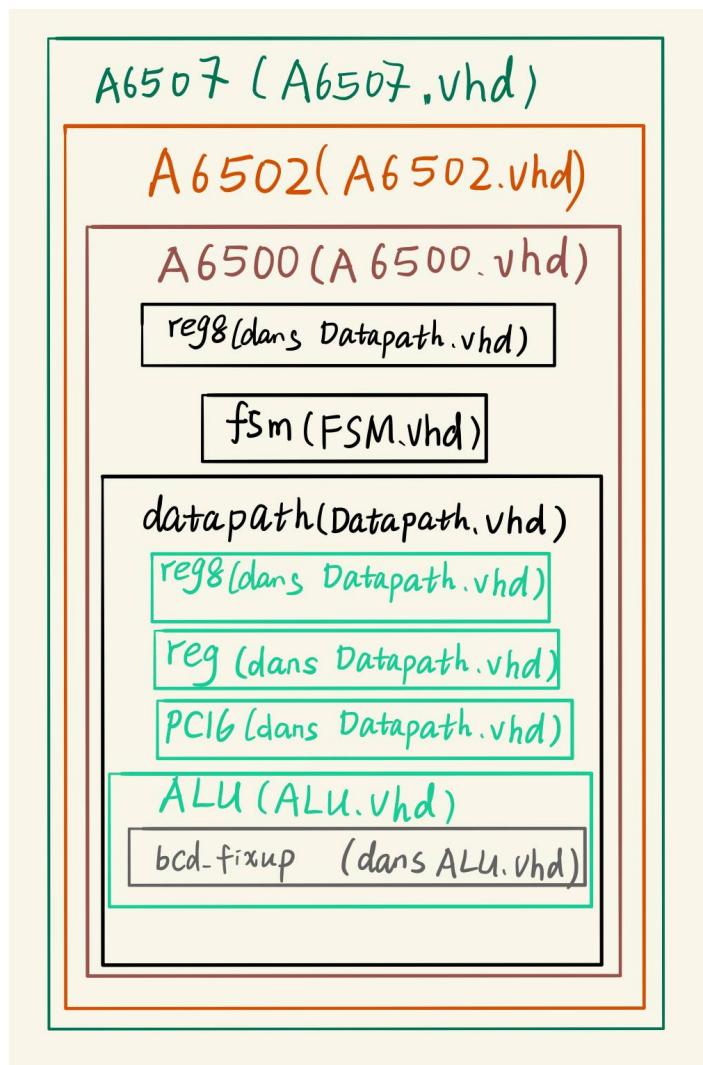


FIGURE 15 – Schéma de l'implémentation du CPU

Décrivons cette implémentation avec une approche top-down en partant du fichier .vhd global pour arriver sur les fichiers les plus précis à la fin.

Le A6507 est défini à partir du processeur à 6502 via les restrictions dont nous avons parlé précédemment. Ainsi le fichier A6507.vhd reprend le A6502.vhd en imposant une réduction de la taille du vecteur A pour baisser le nombre de lignes d'adresse.

Ensuite le A6502 est défini grâce au fichier A6500.vhd, et le fichier A6502.vhd permet d'ajouter la définition d'une variable "stop" d'arrêt en cas de besoin d'arrêter le fonctionnement du CPU.

Ceci mène donc au fichier A6500.vhd, qui lui, est chargé d'une partie de la cohérence glo-

bale du CPU. Il fait le lien entre la partie décodage de l'instruction et une seconde partie composée des deux blocs ALU et registres (mais le lien entre ces deux derniers est réalisé dans le datapath). Il relie ces sous blocs de manière cohérente. Enfin il s'occupe également de gérer les interruptions (que ce soit des interruptions voulues ou bien des interruptions inopinées dues à des erreurs lors des calculs par exemple comme une division par 0).

Ce fichier contient lui 2 sous-fichiers. Le premier est FSM.vhd (FSM pour finite state machine). Ce fichier sert comme son nom l'indique à définir la machine d'états finis évoquée dans le fonctionnement général du CPU. Dans ce fichier est alors défini le séquençage des opérations. Les micro-instructions générées par ce séquençage sont ensuite enregistrées dans les registres d'instructions de reg8 qu'on retrouvera dans le datapath.

Le second sous-fichier est datapath.vhd. Ce sous fichier est essentiel puisqu'il est le 2ème fichier en charge de la cohésion globale du CPU. Il réalise la liaison des blocs 2 et 3 décrits dans le fonctionnement (soit les registres et l'ALU). Selon les différents types d'opérations voulues, il choisit les composants parmi ceux mentionnés précédemment et transmet les données voulues entre eux. C'est donc lui qui permet aux composants de communiquer entre eux.

Les registres qu'ils contient sont à la fois les registres des données (reg) et les registres des instructions élémentaires (reg8). Il contient également la petite architecture Pc16 qui elle est utilisée comme un pointeur d'instruction. Elle désigne en effet le cardinal de l'instruction pointée ; prend donc une incrémentation +1 quand il faut passer à une autre instruction et revient à 0 en cas de reset.

La dernière partie de datapath est l'ALU (de fichier ALU.vhd). Dans celui-ci, les opérations sont réalisées selon les bits de sélection d'opération provenant d'une micro-instruction en entrée. Il contient une sous-architecture bcd_fixup qui permet de convertir des nombres binaires en nombres bcd (binaires codé décimaux) et inversement pour plus de simplicité dans l'écriture des calculs dans son programme.

Implémentation du TIA sur Vivado

l'implémentation du TIA se fait sur plusieurs fichiers :

- Le fichier principal 'TIA.vhd' qui définit tous les composants nécessaires pour le fonctionnement d'un TIA ainsi que l'architecture et la logique des composants
- Le fichier 'Common.vhd' permet de définir les constantes correspondant aux instructions données au TIA pour plus de lisibilité dans le code. Par exemple l'instruction CXM0P correspond au mot binaire '0000'.
- 'NTSC-lookups.vhd' qui sert pour l'affichage des couleurs en sortie du TIA sur l'écran.

Les Objets dynamiques

L'implémentation du TIA faite dans le fichier éponyme repose sur une encapsulation de chaque objet dynamique (player, missile, ball) dans un composant séparé et indépendant du TIA. Dans le schéma de l'architecture du TIA, les blocs correspondants aux registres des objets sont associés à leur 'pixel logic' pour former leur propre composant. Il existe donc une architecture liée aux joueurs, une autre aux missiles et une dernière à la balle. Chaque architecture est ensuite appelée par l'architecture globale du TIA (une pour la balle et deux fois pour les joueurs et missiles) en tant que composant.

La logique du TIA

Toute la logique associée au TIA est décomposée en process effectuant une tâche particulière. Il y a par exemple le 'Collision process', 'Priority process' etc...

Le bloc de 'Control Logic' est aussi un process dont l'utilité est de comparer l'adresse de la commande envoyée pour ensuite la réaliser et renvoyer la bonne donnée en sortie.

5.3 Simulation et débogage

Chaque composant est associé à un fichier "bench". Ces fichiers bench sont des fichiers de simulation, ils permettent de vérifier que le composant modélisé ait le comportement voulu c'est à dire que les signaux de sortie varient bien en fonction de ce qui a été prévu par les tables de vérités. Les fichiers bench que nous avons récupéré comportaient des erreurs et n'étaient pas cohérent avec les variables déclarées dans le fichier du composant. Ainsi nous avons dû modifier ces fichiers pour qu'ils coïncident avec les fichiers de chaque pièce. Enfin, les simulations de chaque pièce ont pu être réalisées et vérifiées.

Après de longues recherches, les fichiers des composants de l'Atari 2600 sont récupérées et analysées. Puis, après une période de débogage et de tests, les fichiers VHDL sont enfin prêts. L'étape suivante, et de relier ces pièces afin de reconstituer la fameuse console des années 80.

6 Reconstituer l'Atari en rassemblant les pièces

Afin de relier les pièces constitutives de l'Atari 2600, il a fallu comprendre et simuler les liens internes de la console. Une fois les liens réalisés, il était nécessaire d'adapter les fichiers de simulation pour qu'ils soient compatibles avec notre carte FPGA.

6.1 Lien entre les pièces

Chaque pièce communique avec les autres selon le schéma de la figure 16. Les pièces s'échangent des informations entre elles via des bus de données et d'adresse. Par exemple le CPU envoie une adresse d'instruction au TIA afin de récupérer les données dans un bus Data.

Chaque pièce est aussi liée aux autres par une horloge commune afin de synchroniser le fonctionnement de la console.

Pour simuler les liens entre les pièces, on utilise donc un nouveau composant nommé 'A2601Core' rassemblant les trois composants principaux de l'Atari et reliant les entrées et sorties du CPU, TIA et RIOT.

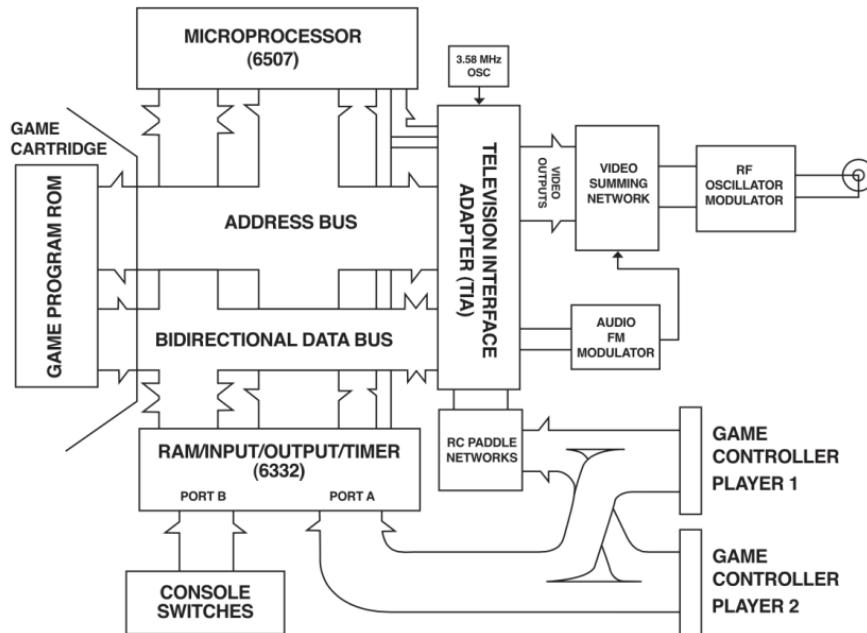


FIGURE 16 – Lien entre les différents pièces de l'Atari

6.2 Adapter le code à notre carte

Le fichier de contrainte pour un programme sur carte FPGA permet de lier les signaux d'entrée et de sortie aux composants physiques de la carte FPGA (boutons, prise HDMI, etc.). Un fichier de contrainte était inclus dans les programmes trouvés sur internet, mais ceux-ci n'étaient pas compatibles avec notre carte. Nous avons donc dû les recréer pour qu'ils soient en accord avec notre modèle de carte. Pour cela il a fallu trouver la signification des différents signaux des fichiers de contraintes à notre disposition, puis retrouver à quoi ces signaux devaient être connectés sur notre carte.

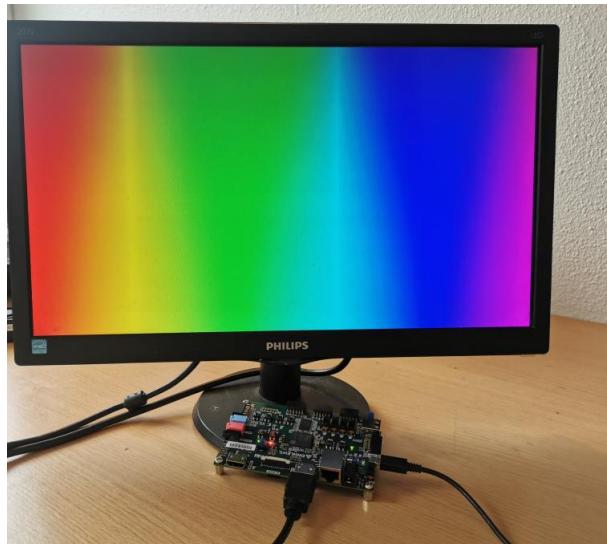


FIGURE 17 – Test du convertisseur HDMI-VGA

6.3 HDMI - VGA

Au niveau de la sortie vidéo sur la carte, un problème s'est posé. La seule carte FPGA que nous possédions pendant le confinement n'avait pas de sortie VGA, il y avait seulement une sortie HDMI. Nous avons donc dû trouver un moyen de convertir cette sortie VGA en sortie HDMI (en effet notre code produit une sortie VGA). En recherchant sur internet, nous avons trouvé des codes qui effectuaient cette conversion, nous avons pu les tester et nous en avons trouvé un que nous avons pu faire marcher sur notre carte en lui faisant afficher une image composé de toute les couleurs de pixels existantes (cf figure 17).

Les liens entre les différents constituants de l'Atari 2600 sont réalisés et vérifiés. Les fichiers de contrainte sont adaptés à notre carte FPGA, ce qui permet de relier les boutons de la carte aux signaux des fichiers de simulations. Enfin, une conversion VGA-HDMI est réalisé au sein de la carte, ce qui rend possible le lien entre la sortie de la console émuler (sortie VGA) et la sortie vidéo de la carte FPGA (sortie HDMI).

7 Test de la console

La dernière étape est de réaliser une simulation complète de la console. Et, si cette simulation est un succès, d'implémenter la console sur la carte FPGA pour la tester.

7.1 Simulation de la console

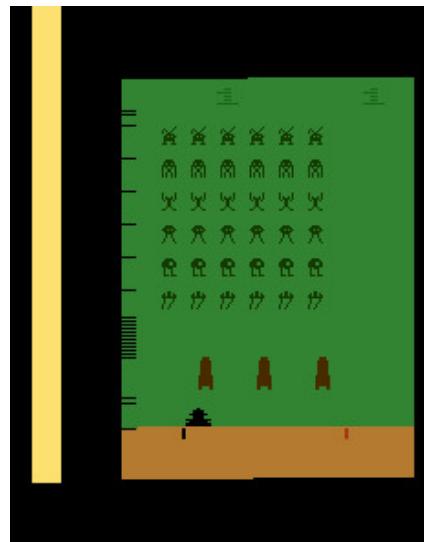
Afin de simuler la console, on utilise à nouveau un fichier 'Bench' pour imposer certaines entrées et sorties à l'Atari, observer les sorties qui sont produites et vérifier qu'elles sont en adéquation avec ce que l'on cherche à obtenir.

Une des entrées de l'Atari fournie par le bench est la lecture d'un fichier ROM qui sera chargée dans le RIOT afin de faire fonctionner le jeu demandé. Ce fichier ROM est obtenu grâce à un fichier binaire converti en fichier texte dont chaque ligne correspond à une instruction en hexadécimal. L'utilisation d'un fichier texte permet de simplifier la lecture du fichier par le logiciel Vivado.

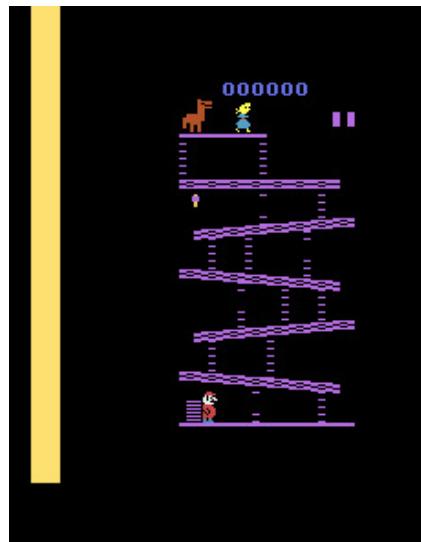
En sortie de la console, on obtient deux nouveaux fichiers textes comprenant des nombres de 0 à 255 sur chaque ligne. Ces deux fichiers correspondent aux sorties vidéo et audio de la console. Pour simplifier l'étude de la console nous nous sommes seulement intéressé à la sortie vidéo.

A partir du fichier vidéo, on peut obtenir une série d'images correspondant à la simulation grâce à un script python. Voici donc en figure 18 quatre exemples d'images, correspondants à quatre jeux différents, que nous obtenons en sortie de la simulation.

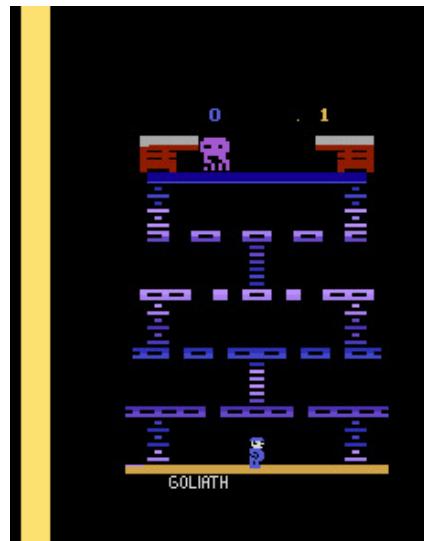
Plusieurs choses sont à remarquer sur ces quatre images : d'une part elle sont entourées d'un bandeau noir large en haut, en bas et à gauche ce qui est caractéristique du format d'affichage présenté dans la figure 14 (format de l'affichage de l'Atari) et montre une cohérence de la simulation. D'autre part on observe une bande jaune présente sur le côté gauche. Nous ne connaissons pas l'origine de cette bande, qui apparaît dans chaque simulation (même lorsqu'aucune ROM n'est donnée en entrée) mais celle-ci ne gène pas le bon fonctionnement des simulations et l'affichage du jeu.



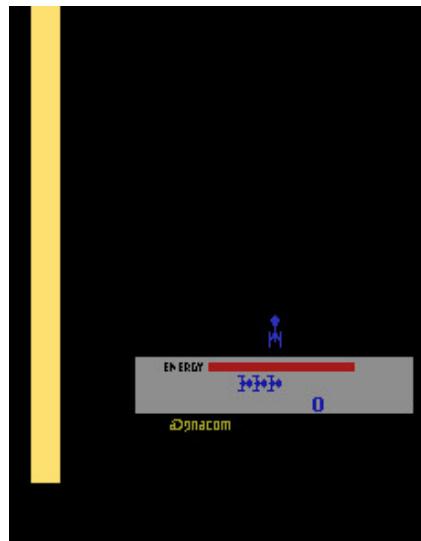
(a) Space_Invaders



(b) Kong



(c) Pac_Kong



(d) Megamania

FIGURE 18 – Test de différents jeux

7.2 Implémentation sur la carte FPGA

L'implémentation sur la carte FPGA constitue la dernière partie de ce travail. La simulation fonctionnant, les seules tâches restantes sont de définir des contraintes permettant d'associer les différentes entrées/sorties de nos codes à celles voulues sur la carte FPGA et d'implémenter tous les fichiers sur la carte. Ce travail final n'a pas encore abouti. En effet un problème est présent ici : la carte FPGA fournie par notre tuteur peut contenir des ROM échangeant des données avec un BUS de 4 bits, tandis que notre Atari fonctionne avec un BUS 8 bits pour ses roms. La carte FPGA dont nous disposons ne semble donc pas convenir pour émuler une Atari 2600 en utilisant une ROM disponible sur la carte elle-même. Pour résoudre ce problème, trois solutions sont possibles. La première peut être d'acheter une carte FPGA convenable, cette solution est la plus simple à réaliser mais également la plus onéreuse. La deuxième solution est de modifier l'architecture de la mémoire ROM dans les codes pour la rendre compatible avec notre carte. Cette modification nécessite de réaliser de nombreuses modifications, et oublier d'en réaliser une seule peut mettre à mal le fonctionnement de la console. Enfin, la dernière solution possible est de créer une nouvelle ROM pour la carte, cette solution est la plus coûteuse en terme d'efforts, moins risquée que la deuxième et ne présente pas d'utilisation de budget réel.

Ainsi la simulation complète de la console fonctionne pour tous les jeux essayés. Cependant, l'implémentation de la console sur la carte FPGA n'est pas réalisée car quelques problèmes sont encore présent.

8 Conclusion

Une simulation concluante de l'Atari complète a été effectué sur le logiciel Vivado, en revanche l'implémentation sur la carte n'a pas été mené à son terme. Celle-ci pourrait être finalisée puis la reconnaissance de manettes pourrait être ajoutée ainsi que la gestion du son via des périphériques audios. L'entièreté des codes a été déposée sur un github en libre accès ([lien ici](#)), ainsi il est accessible à n'importe quelle personne souhaitant travailler sur le sujet.

9 Bibliographie

- [1] «Retromaster's Electronics Projects», *WordPress*,
[<https://retromaster.wordpress.com/a2601>](https://retromaster.wordpress.com/a2601)
- [2] Alberto Bosio, Frédéric Gaffiot, Cédric Marchand, David Navarro, Hai Son Nguyen, Ian O'Connor, Pedro Rojo-Romeo, «Logique combinatoire», *Unité d'Enseignement Science et Technologie de l'Information Systèmes Électroniques*, pp. 113-130.
- [3] Alberto Bosio, Frédéric Gaffiot, Cédric Marchand, David Navarro, Hai Son Nguyen, Ian O'Connor, Pedro Rojo-Romeo, «La logique sequentielle», *Unité d'Enseignement Science et Technologie de l'Information Systèmes Électroniques*, pp. 132-152.
- [4] Eduardo Sanchez, École polytechnique fédérale de Lausanne, *Les circuits FPGA*,
[<http://lslwww.epfl.ch/pages/teaching/cours_lsl/cp_es/fpga.pdf>](http://lslwww.epfl.ch/pages/teaching/cours_lsl/cp_es/fpga.pdf)
- [5] Steve Wright, *Stella Programmer's Guide*, 1979,
[<https://cdn.hackaday.io/files/1646277043401568/stella.pdf>](https://cdn.hackaday.io/files/1646277043401568/stella.pdf)
- [6] Rockwell, *R6500 Microcomputer System Hardware Manual*,
[<https://cdn.hackaday.io/files/1646277043401568/stella.pdf>](https://cdn.hackaday.io/files/1646277043401568/stella.pdf)
- [7] Alberto Bosio, Frédéric Gaffiot, Cédric Marchand, David Navarro, Hai Son Nguyen, Ian O'Connor, Pedro Rojo-Romeo, «Architecture des microprocesseurs», *Unité d'Enseignement Science et Technologie de l'Information Systèmes Électroniques*, pp. 153-160.
- [8] «Arithmetic logic units», *Wikipédia*,
[<https://simple.wikipedia.org/wiki/Arithmetic_logic_unit>](https://simple.wikipedia.org/wiki/Arithmetic_logic_unit)
- [9] «MiSTER – devel/Atari2600_MiSTER », *GitHub*,
[<https://github.com/MiSTER-devel/Atari2600_MiSTER>](https://github.com/MiSTER-devel/Atari2600_MiSTER)

10 Annexe

10.1 Lien du github

Le lien vers le github du projet est disponible à l'adresse :

https://github.com/mengqianzou/Atari_Ecole_Centrale_de_Lyon

10.2 Signaux du RIOT

Voici un schéma récapitulatif du RIOT avec les différents signaux d'entrée ou de sortie ainsi que ses interfaces.

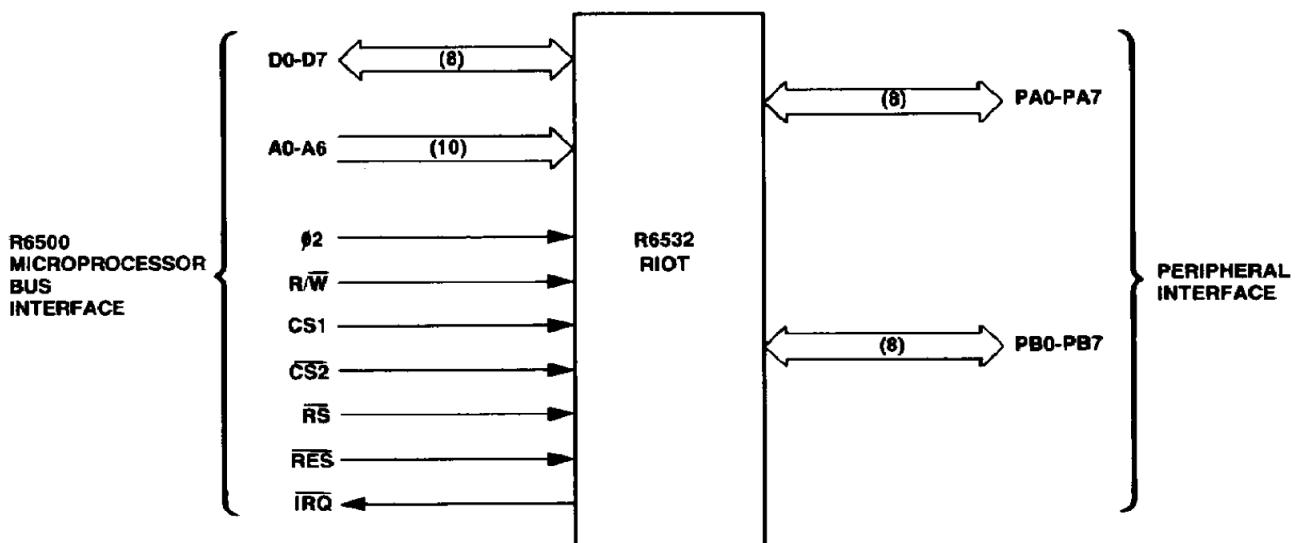


FIGURE 19 – Signaux du RIOT

- D0-D7 : ces bits sont connectés au Bus d'information (interface d'échange d'information entre les différents composants de l'Atari) entre le microprocesseur et le RIOT.
- A0-A6 : bits d'adresse qui dictent en partie ce que le RIOT doit envoyer au microprocesseur.
- ϕ_2 : signal d'horloge du système
- R/ \overline{W} : bit de contrôle, qui gère le transfert de données avec le microprocesseur (R/ \overline{W} = 1 donne une opération Read, R/ \overline{W} = 0 donne une opération Write)
- CS1, $\overline{CS2}$: bits de contrôle
- \overline{RS} : permet de sélectionner la RAM pour la lire (read) ou écrire (write)
- \overline{RES} : Met à zéro les registres de l'interface des périphériques
- \overline{IRQ} : signal d'interruption envoyé par le RIOT, qui interrompt le programme exécuté par le microprocesseur et en lance un programme spécial à la place.
- PA0-PA7 : Port A
- PB0-PB7 : Port B

10.3 Table de vérité du RIOT

OPERATION	RS	R/W	A4	A3	A2	A1	A0
Write RAM	0	0	—	—	—	—	—
Read RAM	0	1	—	—	—	—	—
Write DDRA	1	0	—	—	0	0	1
Read DDRA	1	1	—	—	0	0	1
Write DDRB	1	0	—	—	0	1	1
Read DDRB	1	1	—	—	0	1	1
Write Output Reg A	1	0	—	—	0	0	0
Read Output Reg A	1	1	—	—	0	0	0
Write Output Reg B	1	0	—	—	0	1	0
Read Output Reg B	1	1	—	—	0	1	0
Write Timer							
+ 1T	1	0	1	(a)	1	0	0
+ 8T	1	0	1	(a)	1	0	1
+ 64T	1	0	1	(a)	1	1	0
+ 1024T	1	0	1	(a)	1	1	1
Read Timer	1	1	—	(a)	1	—	1
Read Interrupt Flag(s)	1	1	—	—	1	—	1
Write Edge Detect Control	1	0	0	—	1	(b)	(c)

FIGURE 20 – Table de vérité du RIOT

10.4 Schéma bloc du RIOT

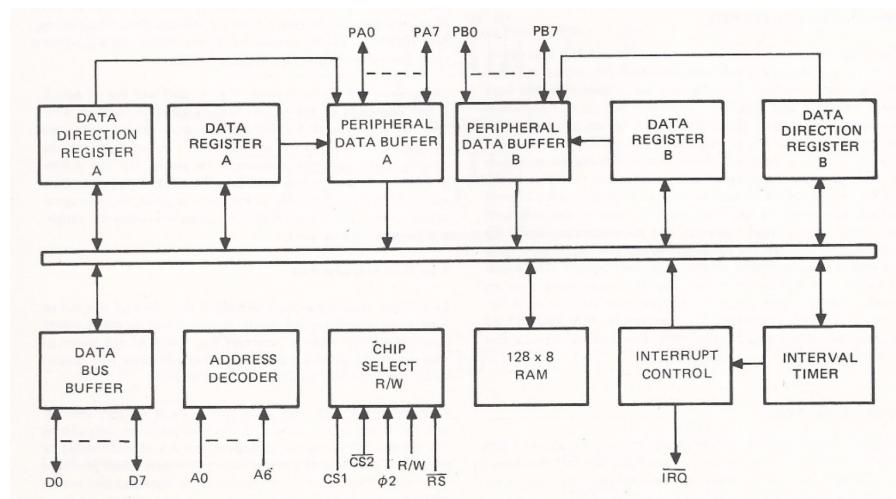


FIGURE 21 – Schéma bloc du RIOT

10.5 Diagramme de GANTT modifié

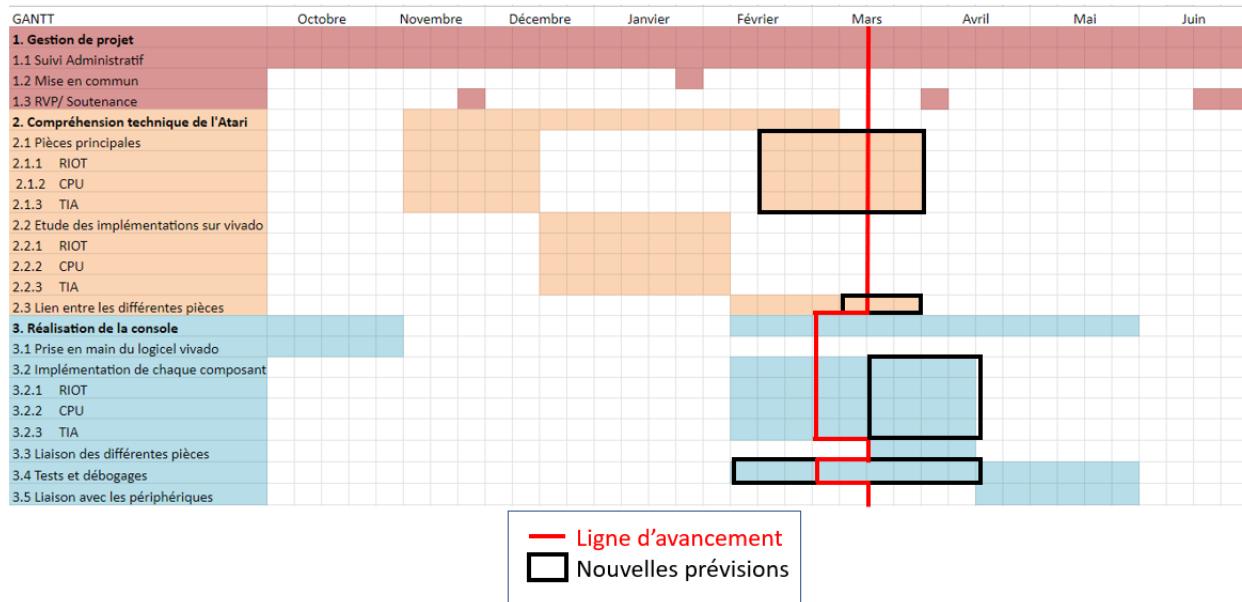


FIGURE 22 – Nouveau diagramme de GANTT

Voici le diagramme de GANTT qui a été modifié suite aux difficultés rencontrées pendant l'année et à la crise sanitaire. En effet nous avons eu du mal à trouver des codes corrects qui fonctionnent sur notre carte FPGA, la crise sanitaire a réduit la communication dans le groupe et celle avec notre tuteur scientifique.

10.6 Check-list

Check-list de rapport de Projet d'Etudes
A remplir par les rédacteurs (élèves)
et à insérer en dernière page du rapport

A développer

Renseigner la case par le nom du responsable, ou la date ou une simple croix lorsque la vérification a été faite.

Contenu	Vérification présence	Vérification qualité
Résumé en français	✓	✓
Résumé en anglais	✓	✓
Table des matières	✓	✓
Table des figures	✓	✓
Introduction	✓	✓
Conclusion générale	✓	✓
Bibliographie	✓	✓
Citation des références dans le texte	✓	✓

Forme

Vérification orthographe	✓	✓
Pagination	✓	✓
Homogénéité de la mise en page	✓	✓
Lisibilité des figures	✓	✓