Projet VHDL : jeu Tron

NUME3 - Conception VHDL/FPGA

7 janvier 2026

Auteurs :

Mattéo BINET

Aurélien Barthere

Sommaire

Introduction

Le jeu TRON est un jeu vidéo sorti en 1982, dont le but d'un des mini-jeux est de détruire la moto de son adversaire grâce à sa traîné de lumière laissée derrière soi. L'objectif est ici de recréer ce jeu vidéo sur une carte FPGA en utilisant le langage de description VHDL.

Le système est implémenté sur une carte NEXYS4 de Digilent, équipée d'un FPGA Xilinx Artix-7. Il comprend plusieurs modules fonctionnels, notamment la gestion des signaux d'horloge, le système de déplacement des personnages et l'implémentation sur écran VGA. Chaque module est implémenté et testé grâce à l'environnement Vivado avant d'être intégré sur le FPGA.

1 Cahier des Charges et Matériel Utilisé

1.1 Comportement Fonctionnel du Système

Le système démarre après l'implémentation du Bitstream. Une réinitialisation est possible en activant le `reset à '1'` (`rst='1'` en mettant `J15` en position haute) ce qui remet toutes les FSM dans leur état d'initialisation.

1.2 Contraintes de Cadencement

La fréquence d'horloge de base est de **100 MHz**, fournie par le quartz de la carte. un signal de type `Clock Enable` (ce) permet d'adapter cette cadence aux différentes parties du système :

- ce_fsm : commande le cadencement des blocs `fsm_rw` et des `fsm_pos` et par conséquent la vitesse du jeu. Il est calibré par défaut à $200Hz$ ce qui signifie que les joueurs avancent de 50 $pixels/sec$.

1.3 Interface d'Entrées/Sorties

- **Entrées :**
  - `horloge` : signal d'horloge principal (100 MHz)
  - `Interrupteur` : remise à zéro asynchrone
  - `Boutons` : commandes des déplacements du J1
  - `PMOD` : commandes des déplacements du J2
- **Sorties :**
  - `VGA` : Affichage de la partie sur un écran.
  - `LED` : Affichage des états des `FSM` pour les débogages.

1.4 Matériel Utilisé

- **Carte de développement :** Digilent NEXYS4 A7.
- **FPGA :** Xilinx Artix-7 (référence XC7A100T).
- **Périphériques intégrés :**

  - 4 afficheurs 7 segments à anode commune.
  - 16 LEDs, boutons poussoirs, interrupteurs.
  - Ports PMOD.
  - quartz d'horloge à 100 MHz.
- **Périphériques utilisés :**
  - Boutons intégrés.
  - LEDs.
  - Ports PMOD.
  - Port VGA.

2 Jeu Tron

Le principe est simple, les deux joueurs disposent de 4 boutons directionnels qui permettent de diriger leur personnage, ici représenté par un pixel de couleur. Le déplacement et la vitesse constante et automatique donc seule la direction est à contrôler. Ce qui rend le jeu intéressant est le fait que les joueurs laissent derrière eux une traînée de leur couleur bloquant les déplacement de l'adversaire mais aussi de-même. En effet si la tête d'un joueur touche la traînée de n'importe quel joueur ou les bords de l'écran, alors celui perd la partie.

3 Architecture

3.1 Ressources utilisées

Table 1 – Rapport d'utilisation des ressources (Resource Utilization)

Resource	Utilization	Available	Utilization %
LUT	403	63400	0.64
LUTRAM	3	19000	0.02
FF	274	126800	0.22
BRAM	5	135	3.70
IO	31	210	14.76
BUFG	1	32	3.13

La ressources utilisées sont cohérentes avec ce à quoi nous nous attendions. En effet la plus grande partie des ressources dont la majorité des Flip-Flop, des LUTs et tous les Blocs-RAM sont utilisés par le bloc d'affichage VGA fourni par l'équipe enseignante. Le reste des Flip-Flop et des LUTs sont utilisés par le reste de l'architecture. Ce "reste" reste néanmoins assez important puisque la quasi intégralité des blocs sont des machines d'état qui nécessitent beaucoup de ressources.Le BUFG est utilisé pour distribuer l'horloge système de 100 MHz sur l'ensemble du réseau d'horloge du FPGA et les 31 sorties sont cohérentes avec les Boutons, LEDs et Switchs utilisés.

L'indicateur le plus déterminant de ce rapport est le **Worst Negative Slack (WNS)**, qui évalue la marge temporelle sur le « chemin critique ». Le Slack correspond à la différence entre le temps imparti par l'horloge et le temps réel de calcul nécessaire au signal pour traverser la logique. Dans notre cas, avec une horloge de 100 MHz (période $T = 10$ ns), le rapport indique un WNS positif de **3.480 ns**. Cela signifie que le signal arrive à destination avec 3.48 ns d'avance sur le front d'horloge suivant. On peut ainsi déduire le temps de propagation effectif du chemin le plus long du circuit :

$$\text{Temps critique} = T_{\text{horloge}} - \text{WNS} = 10\,\text{ns} - 3.48\,\text{ns} = 6.52\,\text{ns}$$

Une valeur de WNS négative aurait indiqué une violation des contraintes temporelles (le circuit aurait été trop lent). Ici, la valeur positive confirme que le design est parfaitement synchronisé et dispose même d'une marge de sécurité confortable.

3.2 Architecture globale (top_level.vhd)

TABLE 2 – Entrées et Sorties du module `top_level`

Nom du Signal	Direction	Type
`clk`	IN	`STD_LOGIC`
`rst`	IN	`STD_LOGIC`
`J1_up/down/left/right`	IN	`STD_LOGIC`
`J2_up/down/left/right`	IN	`STD_LOGIC`
`vga_hs`	OUT	`STD_LOGIC`
`vga_vs`	OUT	`STD_LOGIC`
`vga_color`	OUT	`STD_LOGIC_VECTOR(11 DOWNTO 0)`
`s_J1_loses (test)`	OUT	`STD_LOGIC`
`s_J2_loses (test)`	OUT	`STD_LOGIC`
`s_end_init (test)`	OUT	`STD_LOGIC`
`s_state_fsm_rw (test)`	OUT	`STD_LOGIC_VECTOR(2 DOWNTO 0)`

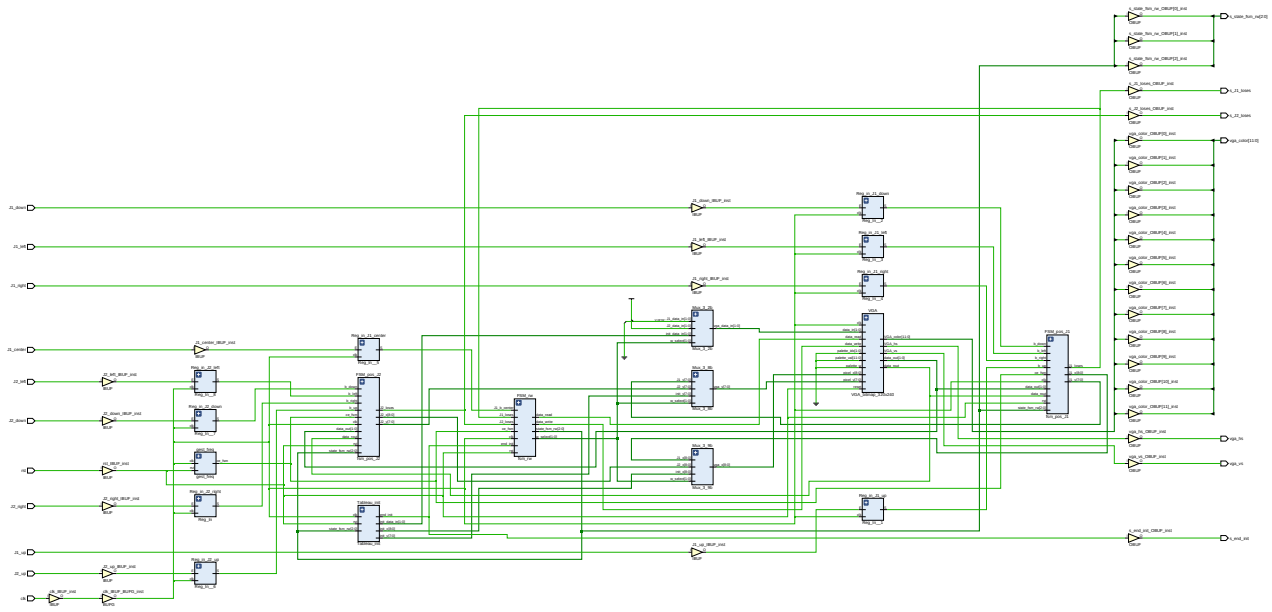

FIGURE 1 – Schématique du Top_Level

Les signaux marqués comme (test) dans le tableau sont des sorties de débogage connectées aux LEDs de la carte, permettant de visualiser l'état interne du système en temps réel.

3.3 Bloc d'initialisation (Tableau_init.vhd)

TABLE 3 – Entrées et Sorties du module `Tableau_init`

Nom du Signal	Direction	Type
`rst`	IN	`STD_LOGIC`
`clk`	IN	`STD_LOGIC`
`state_fsm_rw`	IN	`STD_LOGIC_VECTOR(2 DOWNTO 0)`
`end_init`	OUT	`STD_LOGIC`
`init_x`	OUT	`STD_LOGIC_VECTOR(8 DOWNTO 0)`
`init_y`	OUT	`STD_LOGIC_VECTOR(7 DOWNTO 0)`
`init_data_in`	OUT	`STD_LOGIC_VECTOR(1 DOWNTO 0)`

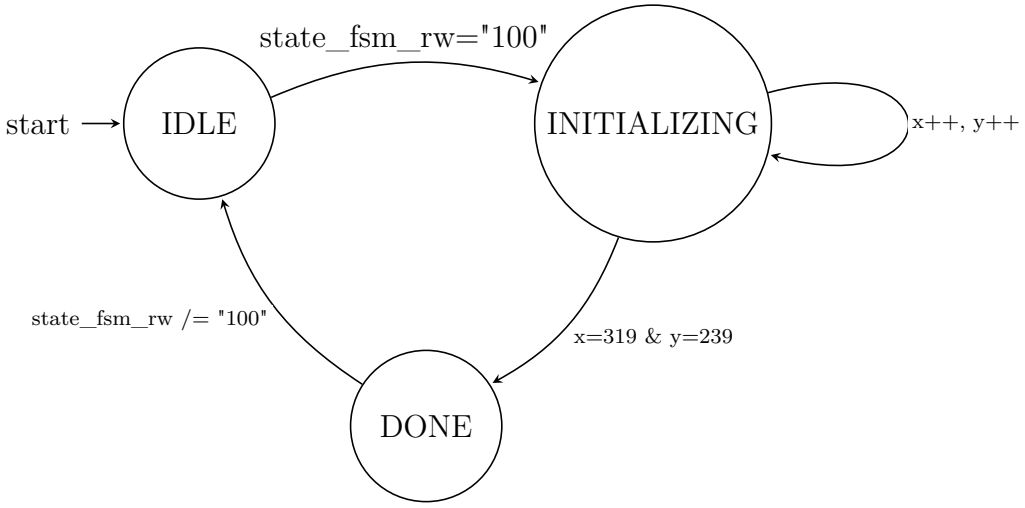

FIGURE 2 – Machine d'état de `Tableau_init`

Ce bloc effectue, à chaque "reset" et à chaque nouvelle partie, l'initialisation de l'affichage du jeu. Pour ce faire il vient remplir l'écran de pixels noirs excepté ceux des bords qu'il remplit en vert. Cette initialisation est réalisée à l'aide de deux compteurs imbriqués et de conditions de couleur sur la position. Le processus est synchrone et implicite. Le bloc fonctionne sur clk (à $100MHz$), l'initialisation ne devant être vu par les joueurs. Il est possible de le faire fonctionner à cette vitesse car le block d'affichage VGA fonctionne aussi à $100MHz$.

3.4 Bloc Buffer (Reg_In.vhd)

TABLE 4 – Entrées et Sorties du module `Reg_In`

Nom du Signal	Direction	Type
`clk`	IN	`STD_LOGIC`
`E`	IN	`STD_LOGIC`
`S`	OUT	`STD_LOGIC`

Ce bloc buffer, décrit dans le cadre des TPs de VHDL du S7, a été réutilisé ici afin de rendre synchrones les entrées des boutons qui ne le sont pas. En effet, nous utilisons 9 boutons qui doivent tous être traités avant d'être envoyés dans le circuit.

3.5 Bloc de gestion de fréquence (Gest_freq.vhd)

TABLE 5 – Entrées et Sorties du module `gest_freq`

Nom du Signal	Direction	Type
`clk`	IN	`STD_LOGIC`
`rst`	IN	`STD_LOGIC`
`ce_fsm`	OUT	`STD_LOGIC`

Ce Bloc sert à l'obtention d'un signal "enable" ici "ce_rw" servant à contrôler la vitesse du bloc de lecture et écriture. Il s'agit d'un simple compteur émettant une impulsion régulière de fréquence $200Hz$.

3.6 Bloc de position (fsm_pos.vhd)

TABLE 6 – Entrées et Sorties du module `fsm_pos_J1`

Nom du Signal	Direction	Type
rst	IN	STD_LOGIC
clk	IN	STD_LOGIC
ce_fsm	IN	STD_LOGIC
b_up	IN	STD_LOGIC
b_down	IN	STD_LOGIC
b_left	IN	STD_LOGIC
b_right	IN	STD_LOGIC
data_out	IN	STD_LOGIC_VECTOR(1 DOWNTO 0)
data_rout	IN	STD_LOGIC
state_fsm_rw	IN	STD_LOGIC_VECTOR(2 DOWNTO 0)
J1_x	OUT	STD_LOGIC_VECTOR(8 DOWNTO 0)
J1_y	OUT	STD_LOGIC_VECTOR(7 DOWNTO 0)
J1_data_in	OUT	STD_LOGIC_VECTOR(1 DOWNTO 0)
J1_loses	OUT	STD_LOGIC

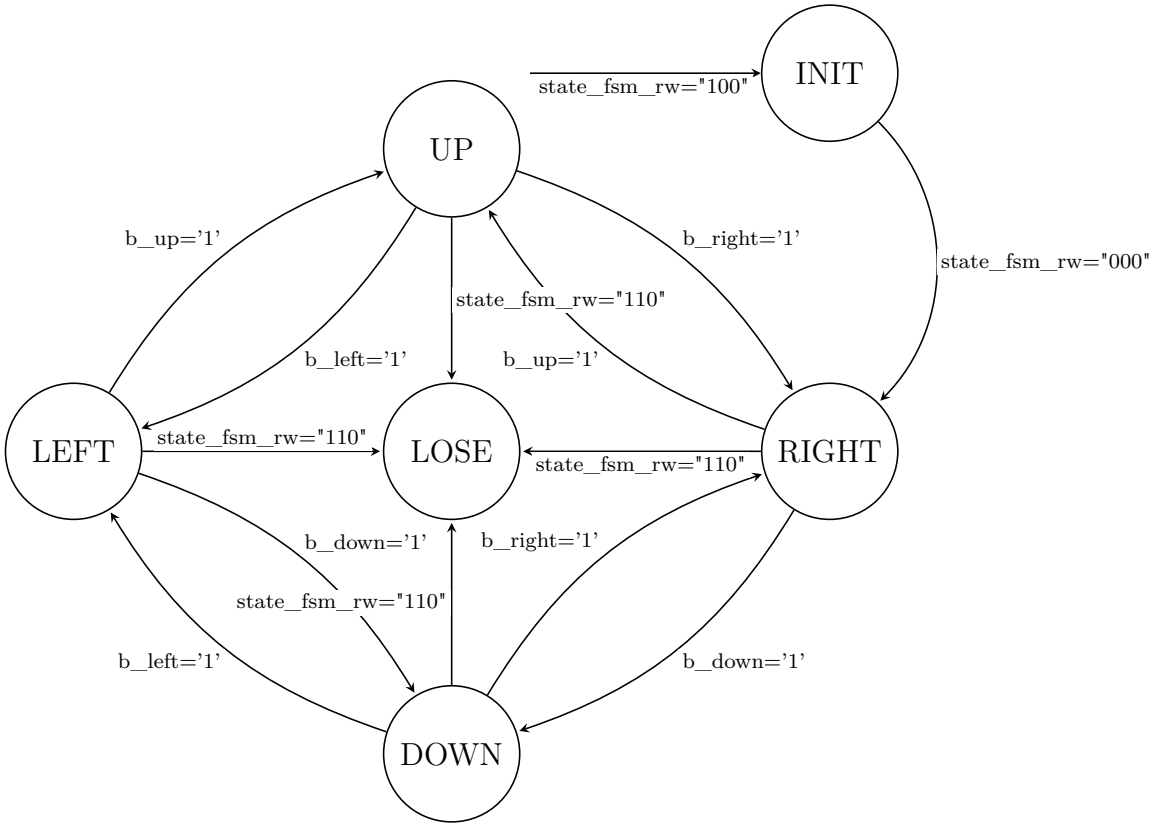

FIGURE 3 – Machine d'état de `fsm_pos_J1`

Ce bloc est chargé de garder en mémoire la position d'un joueur et de le faire se déplacer. Il est également là pour comparer la couleur de la case sur laquelle le joueur est présent afin de déterminer s'il a perdu. En effet, si la couleur renvoyée par le bloc d'affichage est différente du noir, alors le joueur a perdu. Le bloc de position envoie donc le signal de fin de partie au bloc de lecture et d'écriture, qui s'occupera d'arrêter la partie. Ce bloc existe deux fois dans la description puisque chaque joueur possède le sien.

3.7 Bloc d'écriture et de lecture (fsm_rw.vhd)

TABLE 7 – Entrées et Sorties du module `fsm_rw`

Nom du Signal	Direction	Type
clk	IN	STD_LOGIC
ce_fsm	IN	STD_LOGIC
rst	IN	STD_LOGIC
J1_loses	IN	STD_LOGIC
J2_loses	IN	STD_LOGIC
end_init	IN	STD_LOGIC
J1_b_center	IN	STD_LOGIC
state_fsm_rw	OUT	STD_LOGIC_VECTOR(2 DOWNTO 0)
w_select	OUT	STD_LOGIC_VECTOR(1 DOWNTO 0)
data_read	OUT	STD_LOGIC
data_write	OUT	STD_LOGIC

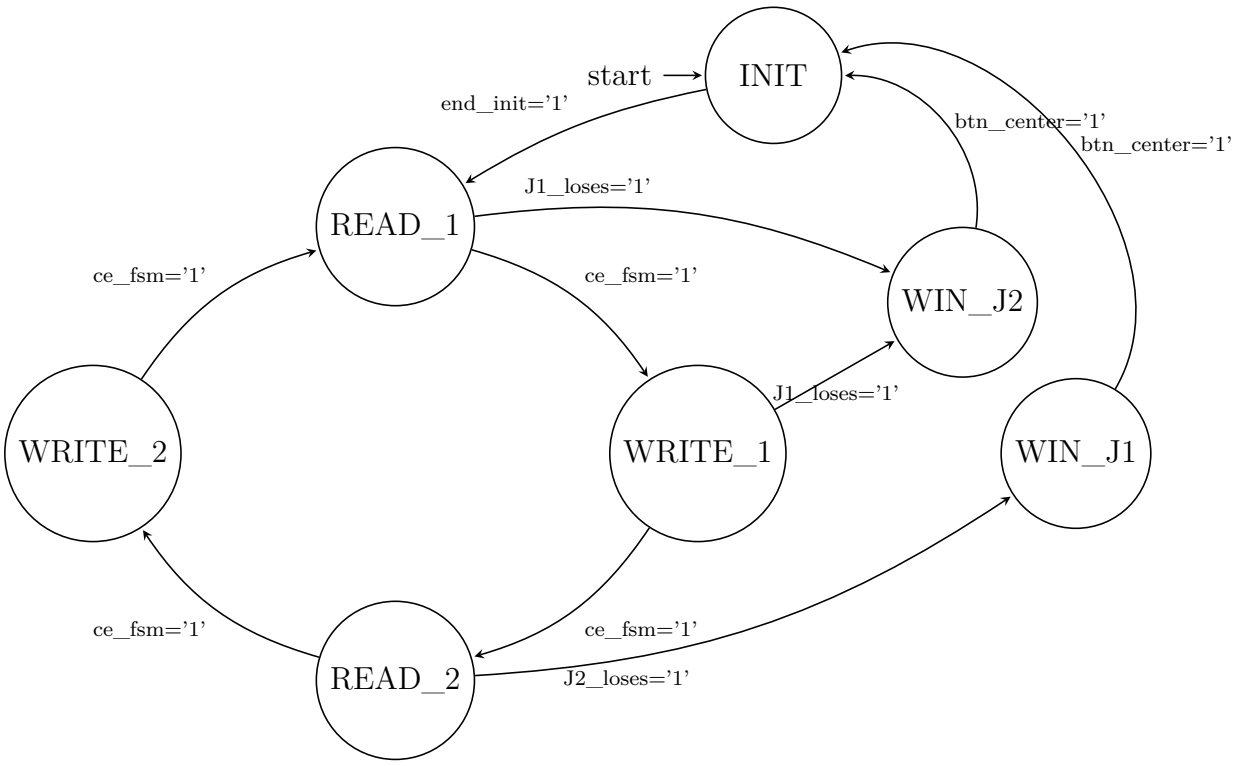

FIGURE 4 – Machine d'état de `fsm_rw`

Ce bloc est point central de l'architecture. En effet, c'est ce bloc qui, par le biais d'une machine d'état, vient commander les bloc d'affichage. Les états sont au nombre de 7 : `init`, `read_j1`, `read_j2`, `write_j1`, `write_j2`, `win_j1`, `win_j1`. La FSM cycle

automatiquement entre les `read` et les `write` quand `ce_fsm='1'`. La FSM partage aussi son état actuel aux autres FSM grâce à `state_fsm_rw`.

TABLE 8 – Encodage des états du signal `state_fsm_rw`

Valeur Binaire	État (VHDL)	Description
"000"	READ_1	Lecture position Joueur 1
"001"	WRITE_1	Écriture nouvelle position J1
"010"	READ_2	Lecture position Joueur 2
"011"	WRITE_2	Écriture nouvelle position J2
"100"	INIT	Initialisation du jeu
"101"	WIN_J1	Victoire Joueur 1
"110"	WIN_J2	Victoire Joueur 2
"111"	∅	État indéfini

3.8 Muxs (Mux_3.vhd)

TABLE 9 – Entrées et Sorties du module `Mux_3_9b`

Nom du Signal	Direction	Type
w_select	IN	STD_LOGIC_VECTOR(1 DOWNTO 0)
init_x	IN	STD_LOGIC_VECTOR(8 DOWNTO 0)
J1_x	IN	STD_LOGIC_VECTOR(8 DOWNTO 0)
J2_x	IN	STD_LOGIC_VECTOR(8 DOWNTO 0)
vga_x	OUT	STD_LOGIC_VECTOR(8 DOWNTO 0)

Ces 3 blocs servent à choisir les données transmises au module d'affichage VGA. Ils sont contrôlés par le bloc d'écriture et de lecture (`w_select`), qui va leur ordonner d'envoyer les données venant du bloc d'initialisation lors de l'initialisation et, périodiquement, les données des deux joueurs sinon.

3.9 Bloc d'affichage VGA (VGA_bitmap_320x240.vhd)

TABLE 10 – Entrées et Sorties du module `vga_bitmap_320x240`

Nom du Signal	Direction	Type
clk	IN	STD_LOGIC
reset	IN	STD_LOGIC
VGA_hs	OUT	STD_LOGIC
VGA_vs	OUT	STD_LOGIC
VGA_color	OUT	STD_LOGIC_VECTOR(11 DOWNTO 0)
pixel_x	IN	STD_LOGIC_VECTOR(8 DOWNTO 0)
pixel_y	IN	STD_LOGIC_VECTOR(7 DOWNTO 0)
data_in	IN	STD_LOGIC_VECTOR(1 DOWNTO 0)
data_write	IN	STD_LOGIC
data_read	IN	STD_LOGIC
data_rout	OUT	STD_LOGIC
data_out	OUT	STD_LOGIC_VECTOR(1 DOWNTO 0)
end_of_frame	OUT	STD_LOGIC

Le bloc d'affichage n'a pas été décrit par nos soins mais il semblait important de l'inclure. Son rôle est de formater les données à transmettre à l'écran VGA pour que tout s'affiche correctement. Il possède un mode de lecture et un mode d'écriture que nous utilisons périodiquement (commandé par le bloc de lecture et d'écriture). Le Mode lecture a un petit délai que l'on doit prendre en compte, cependant, la vitesse de fonctionnement de notre machine d'état étant très faible ($200Hz$) devant l'horloge ($100MHz$), il n'est pas nécessaire de le prendre en compte.

4 Axes d'amélioration

4.1 Difficultés rencontrées

Le projet devait initialement consister en un jeu de Snake à deux joueurs. Nous avions prévu de commencer par réaliser un jeu de Tron, puis de basculer ensuite vers le Snake. Cependant, la première version de la description ne fonctionnait pas correctement. En effet, bien que tous les blocs aient été testés individuellement et qu'ils fonctionnaient comme prévu, l'ensemble ne marchait pas une fois assemblé. Après avoir passé beaucoup de temps à essayer de comprendre l'origine du problème, nous avons décidé de réécrire la quasi-totalité des blocs. L'obtention d'un Tron pleinement fonctionnel n'a donc été que très récente.

4.2 Ajouts envisagés

— Passage du Tron au Snake avec l'ajout d'un mémoire pour la position des pixels des joueurs et d'un système d'apparition aléatoire de pommes.

— Ajout d'un écran de fin en cas de victoire d'un joueur.

— Marqueur de score afin de permettre les revanches.

— Ajouts de plus de joueurs.

Conclusion

L'objectif de ce projet était avant tout de réaliser un système complet en VHDL. Le jeu Tron a été choisi comme support afin de structurer ce travail autour d'une application concrète intégrant des machines à états.

Malgré des difficultés lors de l'intégration des différents blocs, le projet a abouti à une version fonctionnelle. Cette expérience a été particulièrement formatrice et a permis de renforcer notre compréhension de la conception de systèmes numériques sur FPGA.