HEIA-FR
HTA-FR

Systèmes Numériques 2
Projet VHDL

Date :	20.06.2016	Etudiants :	Cyril Vallélian
Classe :	T-2a		Gilles Waeber

Projet VHDL – Flappy Bird

Table des matières

1. Introduction

1.1 Projet

Le projet consiste à réaliser un jeu graphique sur un écran VGA avec une FPGA Xilinx Spartan 6.

Ce document décrit dans un premier temps les fonctionnalités de base et optionnelles du jeu. Des maquettes illustrent ensuite le déroulement d'une partie.

Le chapitre de conception décrit l'architecture du système ainsi que le fonctionnement des différents composants. Une partie de test valide finalement le fonctionnement du jeu.

1.2 Description du jeu

Flappy Bird est un jeu d'obstacle qui a pour but de faire avancer un oiseau, tout en évitant les tuyaux qui se présentent devant lui. Un point est gagné à chaque tuyau passé. La partie se termine lorsque l'oiseau touche le sol ou un tuyau. Le but est donc de passer le plus de tuyaux possibles.

L'oiseau se déplace « par à-coups ». A chaque clic, l'oiseau fait un petit saut, puis redescend. La difficulté réside à ajuster le nombre de sauts pour parvenir à passer entre les deux tuyaux.

L'image ci-dessous illustre le jeu original :

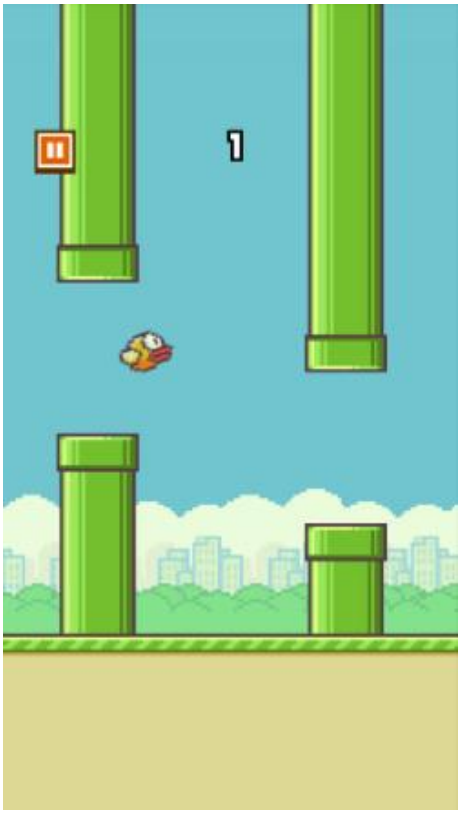


Figure 1 - Capture du jeu Flappy Bird original

2. Analyse

Ce chapitre décrit les fonctionnalités du jeu ainsi que le déroulement d'une partie sous forme de maquettes.

2.1 Fonctionnalités

Cette partie liste les fonctionnalités de notre jeu qui sont séparées en deux catégories :

- Les fonctionnalités de bases, qu'il est impératif d'implémenter
- Les fonctionnalités « nice to have », qui représentent des fonctionnalités optionnelles

2.1.1 Fonctionnalités de base

Ces fonctionnalités doivent être implémentées pour que le jeu fonctionne correctement :

- Affichage de l'écran d'accueil du jeu
- Implémentation de deux boutons de contrôle
- Affichage de l'oiseau, des tuyaux et du score
- Défilement des tuyaux selon une séquence définie (emplacements et taille des trous fixes)
- Saut de l'oiseau lorsque l'on clique sur un bouton
- Détection du contact entre l'oiseau et les tuyaux/le sol
- Affichage d'un écran différent lorsque la partie est perdue, avec :
    - Affichage du score
    - Possibilités de recommencer la partie ou de retourner à l'écran d'accueil

2.1.2 Fonctionnalités « nice to have »

Une fois les fonctionnalités de base implémentées, les fonctionnalités ci-dessous peuvent apporter une plus-value au jeu :

- Enregistrement du score maximum
- Amélioration du graphisme, couleurs
- Sons lors du saut de l'oiseau, du passage d'un tuyau et lorsque la partie et perdue
    - Réglage pour activer / désactiver le son
- Taille aléatoire des tuyaux
- Défilement des tuyaux aléatoire (pas toujours la même séquence)
- Plusieurs niveaux de difficulté (augmentation au fil du score)
    - Augmentation de la vitesse du défilement
    - Variation de l'espace entre les tuyaux (vertical et horizontal)
- Mettre une partie en pause
- Cheat mode / « easter egg »

2.2 Déroulement d'une partie

Cette partie décrit le déroulement d'une partie de Flappy Bird que nous allons implémenter à l'aide de maquettes.

Les maquettes de ce chapitre illustrent le fonctionnement du jeu et ne sont pas représentatives du graphisme final du jeu ainsi que des informations (texte, score) qui seront affichées.

2.2.1 Boutons de contrôle

Pour interagir avec le jeu, il est nécessaire de définir des boutons de contrôle. Notre jeu disposera de 3 boutons physiques différents :

- Le bouton gauche permet de démarrer une partie et de faire sauter Flappy.
- Le bouton droit permet de recommencer une partie.
- Le bouton de reset permet de réinitialiser le jeu.

2.2.2 Ecran d'accueil

Lorsque le jeu démarre, un écran d'accueil est affiché avec le nom du jeu ainsi que les instructions pour jouer et pour commencer une partie. Un clic sur le bouton gauche démarre une nouvelle partie tandis qu'un clic sur le bouton droit ne fait rien.

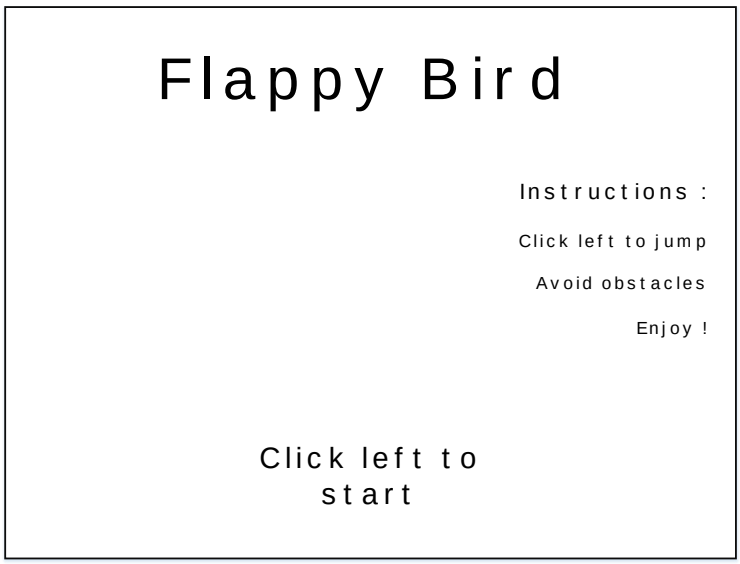


Figure 2 - Ecran d'accueil (maquette)

2.2.3 Démarrage du jeu

Lorsque le jeu démarre, Flappy se trouve en l'air et est figé. La partie débute réellement lorsque le joueur clique une deuxième fois sur le bouton gauche.

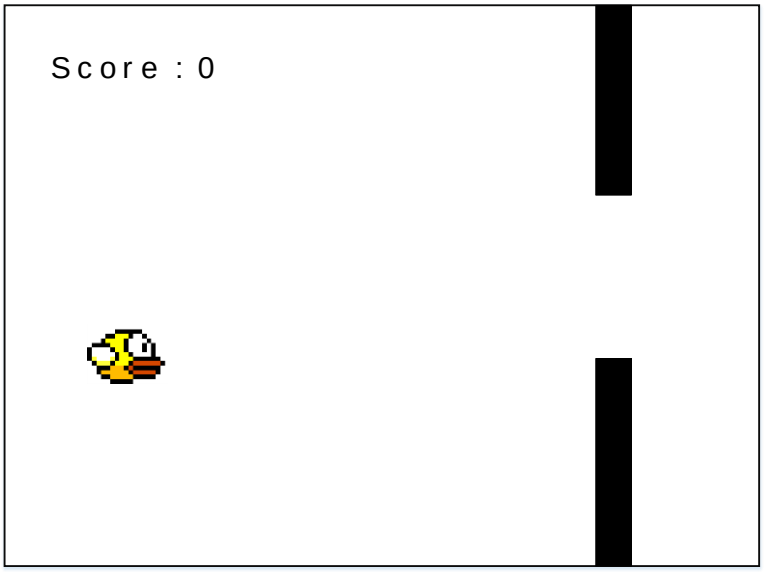


Figure 3 - Démarrage du jeu (maquette)

2.2.4 Durant la partie

Le bouton gauche permet de faire sauter Flappy. Une fois le saut terminé, on peut le faire sauter à nouveau, mais il n'est pas possible d'effectuer un « double saut ». A chaque passage d'un couple de tuyaux, Flappy marque un point. Durant le jeu, une pression sur le bouton droit redémarre la partie (chapitre 2.2.3).

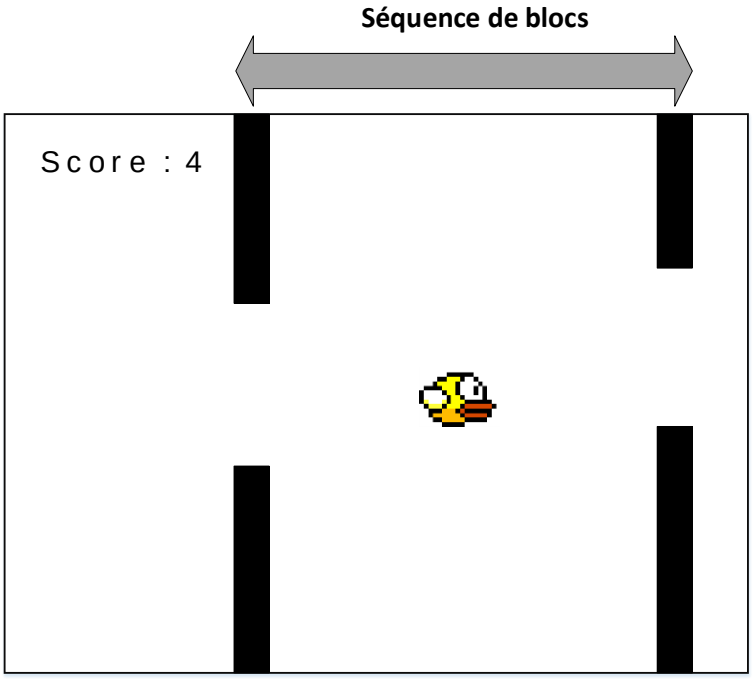


Figure 4 - En jeu (maquette)

2.2.5 Fin de partie

La partie se termine lorsque Flappy touche un des tuyaux ou lorsqu'il touche le sol. Un écran de fin de partie est ensuite affiché avec le score de la partie ainsi que le score maximum (optionnel). Un clic sur le bouton gauche permet de recommencer une partie (chapitre 2.2.3) tandis qu'un clic sur le bouton droit permet de revenir à l'écran d'accueil (chapitre 2.2.2).

Figure 5 - Fin de partie (maquette)

3. Conception

Une fois le fonctionnement de notre jeu défini, nous avons imaginé l'architecture de notre jeu. Nous avons conçu l'architecture en séparant les différentes fonctions que le jeu doit fournir pour fonctionner correctement dans différents composants.

La description de certaines entrées/sorties redondantes est parfois omise (clk, rst par ex.).

3.1 Composants

Dans cette partie, nous détaillons le fonctionnement de chaque composant décrit dans notre schéma-bloc (en annexe). Les fonctions de chaque composant sont décrites de manière à pouvoir facilement comprendre leur utilité. Les entrées et sorties sont également listées avec leur type, une description ainsi que leur plage de valeur.

3.1.1 DCM

Fonctions

Le DCM (Digital Clock Management) est un composant intégré dans la FPGA qui permet de convertir l'horloge de 100MHz en 40MHz (fréquence de l'écran VGA).

Entrées

Nom	Type	Description / Plage de valeur
fpga_clk	std_logic	Horloge générée par la FPGA (100MHz)
rst	std_logic	Reset

Sorties

Nom	Type	Description / Plage de valeur
clk	std_logic	Clock utilisée dans le circuit (40MHz)

3.1.2 VGA

Fonctions

Ce composant est le contrôleur VGA. Il a pour fonction de balayer l'écran ligne par ligne et de fournir en sortie les compteurs représentant le balayage (coordonnées horizontale et verticale).

Entrées

Nom	Type	Description / Plage de valeur
clk	std_logic	Clock utilisée dans le circuit (40MHz)
rst	std_logic	Reset

Sorties

Nom	Type	Description / Plage de valeur
enable	std_logic	Signaler que le balayage de l'écran est terminé
HS	std_logic	Impulsion de synchronisation horizontale
VS	std_logic	Impulsion de synchronisation verticale
hcount	Bus (11)	Valeur actuelle du balayage horizontal
vcount	Bus (10)	Valeur actuelle du balayage vertical
blank	std_logic	Sortie vidéo nulle lors blank = 1

3.1.3 Btns_synchro

Fonctions

Synchronise tous les boutons d'entrée avec le signal d'horloge.

Entrées

Nom	Type	Description / Plage de valeur
btn_left_unsync	std_logic	Bouton gauche non synchronisé
btn_right_unsync	std_logic	Bouton droit non synchronisé
clk	std_logic	-
rst	std_logic	-

Sorties

Nom	Type	Description / Plage de valeur
btn_left	std_logic	Bouton gauche, valeur : impulsion si pressé, sinon 0
btn_right	std_logic	Bouton droite, valeur : impulsion si pressé, sinon 0

3.1.4 Ctrl_game

Fonctions

Le contrôleur du jeu a pour but de gérer dans quel état le jeu se trouve. Nous avons défini 4 états différents :

- Start menu L'écran d'accueil est affiché.
- Start game Une partie démarre, mais elle n'a pas encore commencé (affichage figé).
- Game Une partie est en cours.
- End game L'écran de fin de partie est affiché.

La machine d'état ci-dessous représente les transitions entre les différents états. Ces différentes transitions représentent les actions du joueur décrit dans le déroulement d'une partie (chapitre 2.2).

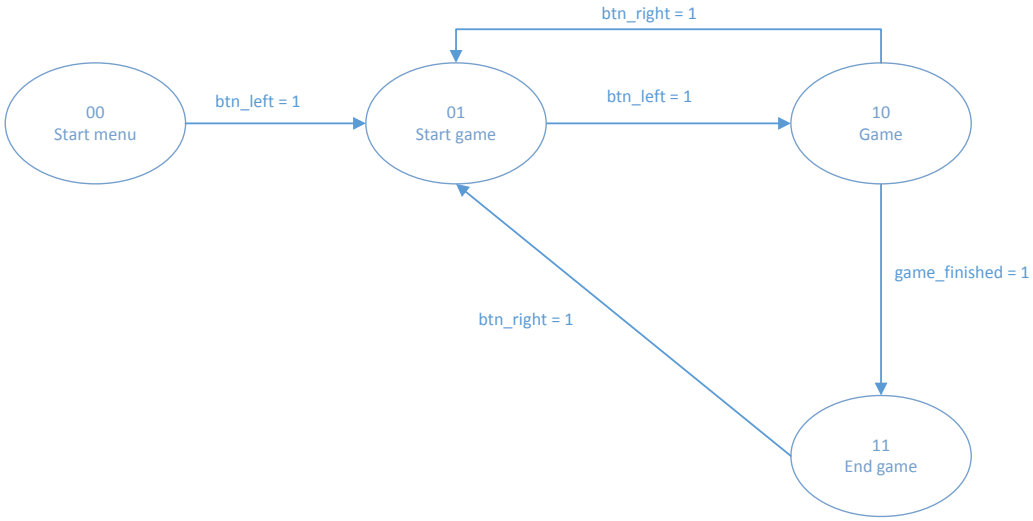


Figure 6 - Machine d'état du contrôleur du jeu

Entrées

Nom	Type	Description / Plage de valeur
btn_left	std_logic	Impulsion du bouton gauche
btn_right	std_logic	Impulsion du bouton droit
game_finished	std_logic	Partie perdue lorsque game_finished = 1
clk	std_logic	-
rst	std_logic	-

Sorties

Nom	Type	Description / Plage de valeur
game_state	Bus (2)	Etat dans lequel le jeu se trouve

3.1.5 Flappy_pos

Fonctions

Ce composant permet de calculer la position de Flappy sur le jeu. Les caractéristiques de Flappy sont ses coordonnées en x et y ainsi que sa taille (largeur/hauteur).

Sa coordonnée en x est fixée (environ 280 sur l'écran 800x600) et sa largeur/hauteur est également fixe (40 pixels environ). Seule la coordonnée y varie.

Lors du démarrage du jeu, Flappy est placé au milieu de l'écran (y = 300). Lors de chaque « incrément » du jeu, Flappy descend/tombe d'un nombre n de pixels en y. Si l'utilisateur appuie sur le bouton gauche, Flappy effectue au saut.

Un saut correspond à un incrément de la coordonnée y de Flappy. Le saut est réalisé au moyen d'un compteur (0 à n incréments) qui incrémente à chaque fois la coordonnée y de n pixels.

Pseudo algorithme de calcul de la coordonnée y :

```
if (btn_left == 1){
      for(int i = 0; i < nbr_incr; i++){
            y++;
      }
}else{
      y--;
}
```

Les valeurs pour le nombre d'incréments du compteur ainsi que la taille d'un déplacement restent encore à définir.

Pour améliorer l'affichage du Flappy, une sortie flappy_state indique l'état dans lequel se trouve le flappy : 00 pour décrément, 01 pour normal, 10 pour incrément et 11 n'est pas utilisé.

Entrées

Nom	Type	Description / Plage de valeur
game_state	Bus (2)	Etat dans lequel le jeu se trouve
enable	std_logic	Indique que le balayage de l'écran est terminé
btn_left	std_logic	Impulsion du bouton gauche
clk	std_logic	-
rst	std_logic	-

Sorties

Nom	Type	Description / Plage de valeur
flappy_y	Bus (10)	Coordonnée y du centre de Flappy
flappy_state	Bus (2)	Etat du Flappy (falling, normal, rising)

3.1.6 Pipes_pos

Fonctions

Un maximum de 2 couples de tuyaux est toujours affiché en même temps sur l'écran. Le couple se trouvant à gauche est appelé A tandis que celui se trouvant à droite de l'écran est appelé B. Les caractéristiques (coordonnées x, taille tuyau du haut et taille tuyau du bas) des 2 couples sont passées en sortie du calculateur de position des tuyaux et seront utilisées par le composant d'affichage.

La coordonnée x du couple/tuyau correspond à la position de son côté le plus à droite et non le centre du couple/tuyau.

Pour créer l'environnement du jeu, une séquence de tuyaux est nécessaire. Nous allons donc créer un tableau contenant le numéro du couple, sa coordonnée x et la taille du tuyau haut et du tuyau bas.

Chaque couple de cette séquence aura une coordonnée x de départ prédéfinie. Le but est qu'à chaque coup du jeu, la coordonnée x de chaque couple est décrémentée. Les 2 couples avec la coordonnée x la plus petite et donc qui se trouvent au début de la séquence, seront affichés au travers des tuyaux A et B.

Un compteur sera utilisé (comme index) pour savoir quel couple est à sortir comme couple A. Le couple B sera simplement le couple avec l'index suivant (compteur+1). Une fois que le couple concerné par l'index voit sa coordonnée x arriver à zéro :

- Le compteur/index est incrémenté d'un couple
- Ce couple est « placé » en fin de séquence au travers d'une mise à jour de sa coordonnée x.

Lorsque l'index atteint le dernier couple de tuyaux dans la séquence, sa prochaine valeur sera zéro.

Le mécanisme réalisé peut s'apparenter à une liste en tourniquet dans lequel les couples « utilisés » (coordonnée x < 0) sont replacés en fin de tourniquet.

Le principe de fonctionnement du composant Pipes_pos est illustré par la figure ci-dessous.

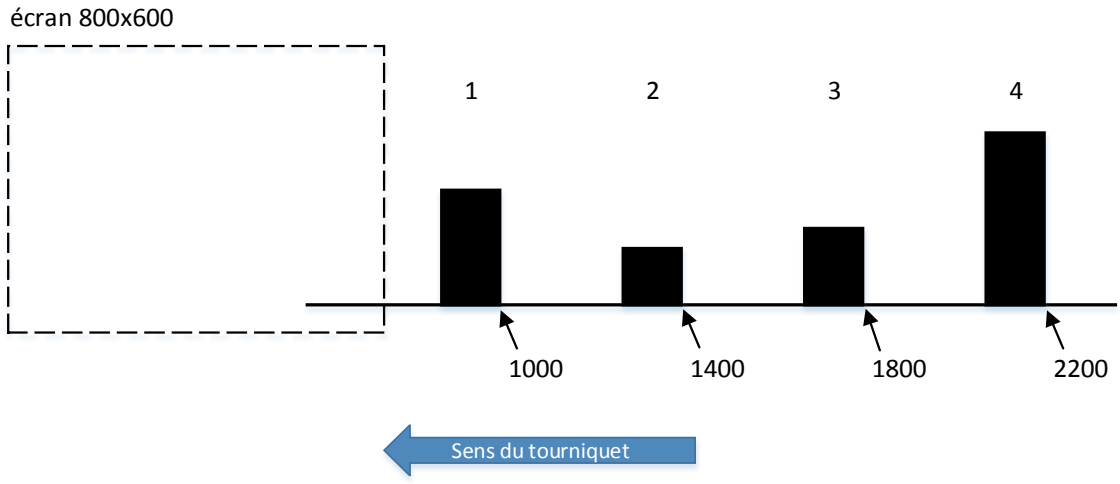


Figure 7 - Principe de fonctionnement du tourniquet de Pipes_pos

La largeur de chaque tuyau est la même et est définie avec une constante fixe de 100 pixels.

Le tableau ci-dessous représente la structure utilisée pour gérer le positionnement et les dimensions des tuyaux avec des valeurs d'exemple factices (lors du démarrage du jeu).

id	x	h_top	h_bottom
1	1500	200	200
2	2000	150	250
3	2500	300	100

Pour avoir un effet aléatoire, seule la taille des 3 premiers couples de tuyaux est fixe. Ensuite à partir du 4ème couple, la taille des deux tuyaux est calculée aléatoirement au moyen d'un compteur (random). Lorsque le couple est remis en fin de liste, la valeur du compteur est utilisée pour la taille du tuyau du haut. La taille du tuyau bas est calculée avec :

$$pipe_{bottom} = HAUTEUR\_ECRAN - (random + space);$$

Le signal space correspond à l'espacement entre les deux tuyaux. Pour augmenter la difficulté du jeu, il est décrémenté au fur et à mesure que le score augmente.

Le compteur random qui représente la taille du tuyau du haut varie entre 100 et 400 pixels et est incrémenté à chaque coup d'horloge. Pour améliorer l'effet aléatoire, la valeur du compteur est changée (par exemple : random = (random * 13) mod 18) lorsque le joueur clique sur le bouton gauche.

Entrées

Nom	Type	Description / Plage de valeur
game_state	Bus (2)	Etat dans lequel le jeu se trouve
enable	std_logic	Indique que le balayage de l'écran est terminé
score	Bus (10)	Score de la partie en cours
btn_left	std_logic	Bouton gauche
clk	std_logic	-
rst	std_logic	-

Sorties

Nom	Type	Description / Plage de valeur
pipe_a_x	Bus (10)	Coordonnée x du 1er couple de tuyaux
pipe_a_h_top	Bus (9)	Hauteur du tuyau du haut du 1er couple
pipe_a_h_bottom	Bus (9)	Hauteur du tuyau du bas du 1er couple
pipe_b_x	Bus (10)	Coordonnée x du 2ème couple de tuyaux
pipe_b_h_top	Bus (9)	Hauteur du tuyau du haut du 2ème couple
pipe_b_h_bottom	Bus (9)	Hauteur du tuyau du bas du 2ème couple

3.1.7 Score

Fonctions

Ce composant a pour fonction de calculer le score. Lorsque la partie début le score à est zéro et à chaque fois que Flappy passe un couple de tuyaux sans accrochage, le score augmente (incrément de 1). Le test permettant de mettre à jour le score est le suivant :

$$A_x == \left(flappy\_x \ - \frac{flappy\_size}{2}\right) + 1$$

Une sortie score_record permet d'enregistrer le meilleur score toutes parties confondues. Le meilleur score est mis à jour à la fin de la partie si le score de la partie en cours était meilleur.

Entrées

Nom	Type	Description / Plage de valeur
game_state	Bus (2)	Etat dans lequel le jeu se trouve
enable	std_logic	Indique que le balayage de l'écran est terminé
pipe_a_x	Bus (10)	Coordonnée x du 1er couple de tuyaux
clk	std_logic	-
rst	std_logic	-

Sorties

Nom	Type	Description / Plage de valeur
score	Bus (10)	Score de la partie en cours
score_record	Bus (10)	Meilleur score

3.1.8 Crashing_test

Fonctions

Ce composant teste si le Flappy se crashe contre un tuyau ou touche le sol, ce qui est synonyme de fin de partie.

Contact avec le sol

Le test ci-dessous permet de test si Flappy entre en contact avec le sol :

$$\left(flappy\_y \; - \frac{flappy\_size}{2}\right) == 0$$

Contact avec un tuyau

Flappy peut entrer en collision avec un tuyau à différents endroits qui sont représentés dans la figure ci-dessous. Pour chacune des différentes zones de collision, un test est effectué.

Pour le tuyau du haut, toutes les zones de collisions (au travers de point précis) sont représentées tandis que pour le tuyau du bas, seuls deux points sont utilisés pour représenter les zones.

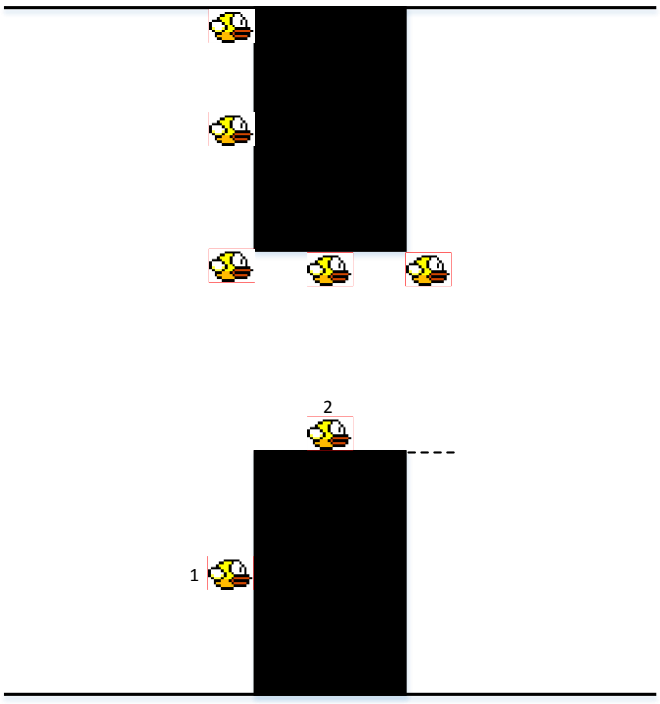


Figure 8 - Possibilités de collision entre Flappy et les tuyaux

Zones

1. Le côté droit de Flappy entre en collision avec le côté gauche du tuyau. Pour ce test, on utilise le coin en bas à droite de Flappy ainsi que la hauteur du tuyau.
2. Le bas de Flappy entre en collision avec le haut du tuyau. Pour ce test, on utilise également le coin en bas à droite du tuyau ainsi que la largeur du tuyau.
   Flappy peut toucher le haut du tuyau avec un de ses coins sans que l'autre coin le touche (voir le tuyau du haut dans la figure ci-dessous). Au lieu d'effectuer un test pour chacun des 2 coins, on ne testera que le coin en bas à droite en agrandissant la zone à tester sur la droite du tuyau de la largeur de Flappy (trait tillé dans la figure ci-dessus).

Equations pour les différents tests (tuyau du bas) :

Test 1

Coordonnée x : $\left(flappy\_x + \frac{flappy\_size}{2}\right) == (pipe\_a\_x - pipe\_width)$ AND

Coordonnée y : $\left(flappy\_y - \frac{flappy\_size}{2}\right) <= pipe\_a\_h\_bottom$

Test 2

Coordonnée x : $\left(flappy\_x + \frac{flappy\_size}{2}\right) >= (pipe\_a\_x - pipe\_width)$ AND

$\left(flappy\_x + \frac{flappy\_size}{2}\right) <= (pipe\_a\_x + flappy\_size)$ AND

Coordonnée y : $\left(flappy\_y - \frac{flappy\_size}{2}\right) == pipe\_a\_h\_bottom$

Le test de collision avec le tuyau du haut reprend les mêmes principes et équations que ceux expliqués pour le tuyau du bas.

Entrées

Nom	Type	Description / Plage de valeur
game_state	Bus (2)	Etat dans lequel le jeu se trouve
enable	std_logic	Indique que le balayage de l'écran est terminé
flappy_y	Bus (10)	Coordonnée y du centre de Flappy
pipe_a_x	Bus (10)	Coordonnée x du 1er couple de tuyaux
pipe_a_h_top	Bus (9)	Hauteur du tuyau du haut du 1er couple
pipe_a_h_bottom	Bus (9)	Hauteur du tuyau du bas du 1er couple
clk	std_logic	-
rst	std_logic	-

Sorties

Nom	Type	Description / Plage de valeur
game_finished	std_logic	Indique que la partie est terminée car Flappy s'est crashé

3.1.9 Display

Fonctions

Suivant l'état du jeu, le composant Display affichera soit les menus (menu d'accueil et de fin de partie) soit le jeu en cours. Pour le menu de fin de partie, le score est affiché et optionnellement le score maximum.

Pour le jeu en cours, le Flappy, 2 à 4 tuyaux (couples de tuyaux A et B), le score actuel et le décor du jeu sont affichés.

Pour le Flappy, sa coordonnée y ainsi que son état (fallin ou rising) sont reçus e. Sa taille ainsi que sa coordonnée x sont récupérer dans les constantes.

Pour les couples de tuyaux (A et B), le composant reçoit en entrée la coordonnée x qui correspond au côté le plus à droite des tuyaux ainsi que la hauteur de chaque tuyau (celui du haut et celui du bas).

Entrées

Nom	Type	Description / Plage de valeur
game_state	Bus (2)	Etat dans lequel le jeu se trouve
hcount	Bus(11)	Valeur actuelle du balayage horizontal
vcount	Bus (10)	Valeur actuelle du balayage vertical
blank	std_logic	Sortie vidéo nulle
score	Bus (10)	Score de la partie en cours
score_record	Bus (10)	Meilleur score
flappy_y	Bus (10)	Coordonnée y du centre de Flappy
flappy_state	Bus (2)	Etat du Flappy
pipe_a_x	Bus (10)	Coordonnée x du 1er couple de tuyaux
pipe_a_h_top	Bus (9)	Hauteur du tuyau du haut du 1er couple
pipe_a_h_bottom	Bus (9)	Hauteur du tuyau du bas du 1er couple
pipe_b_x	Bus (10)	Coordonnée x du 2ème couple de tuyaux
pipe_b_h_top	Bus (9)	Hauteur du tuyau du haut du 2ème couple
pipe_b_h_bottom	Bus (9)	Hauteur du tuyau du bas du 2ème couple
clk	std_logic	-
rst	std_logic	-

Sorties

Nom	Type	Description / Plage de valeur
red	Bus (3)	Valeur du rouge pour le VGA
green	Bus (3)	Valeur du vert pour le VGA
blue	Bus (2)	Valeur du blue pour le VGA

3.2 Composant Top Level

Le composant top level permet de mettre ensemble et d'interconnecter tous les composants du jeu. Le schéma top level se trouve en annexe.

Entrées

Nom	Type	Description / Plage de valeur
btn_left_unsync	std_logic	Bouton gauche non synchronisé
btn_right_unsync	std_logic	Bouton droit non synchronisé
fpga_clk	std_logic	Horloge générée par la FPGA (100MHz)
rst	std_logic	Reset

Sorties

Nom	Type	Description / Plage de valeur
HS	std_logic	Impulsion de synchronisation horizontale
VS	std_logic	Impulsion de synchronisation verticale
red	Bus (3)	Valeur du rouge pour le VGA
green	Bus (3)	Valeur du vert pour le VGA
blue	Bus (2)	Valeur du blue pour le VGA

3.3 Package pour les constantes

Les différentes constantes utilisées par les composants sont définies dans un package « local ». Ce package peut ensuite être importé par les composants qui en ont besoin.

Dans le package local, on retrouve les constantes pour :

- Le driver VGA (largeur de l'écran, hauteur de l'écran, etc.)
- La position du Flappy (position x du Flappy, taille du Flappy, position y initiale du Flappy, etc.)
- Le calcul du score (limite du score maximal)
- La position des tuyaux (nombre de couples, largeur des tuyaux, limites pour le random, espacement minimal et maximal entre les tuyaux, etc.)
- Le composant de test de crash (ajustements pour les tests)
- L'affichage (taille et emplacement des différents éléments graphiques (textes, logo, scores, etc.), constantes pour les couleurs)

Un deuxième package est également créé pour les constantes (de matrices) utilisées pour l'affichage des images. Ce package contient les constantes pour :

- Le texte du score et du best score
- Les digits (0 à 9) du score
- Les différents Flappy (incrément, décrément, partie perdue)
- Texte de démarrage du jeu

4. Réalisation

Cette partie décrit la démarche utilisée pour la réalisation de ce projet. Elle comprend également une explication de la façon dont les images sont affichées. Le code en annexe constitue le reste de la réalisation et se base sur la conception précédemment présentée. Le dernier chapitre est un commentaire sur l'utilisation des ressources de la FPGA.

4.1 Démarche

Pour la réalisation du jeu, les composants ont dans un premier temps été implémentés de manière individuelle suivant la conception réalisée précédemment. Au fur et à mesure de l'avancement de la réalisation, les composants ont été attachés au top level pour apporter une fonctionnalité supplémentaire au jeu. Comme les fonctionnalités ont été mises en place une par une, cela a permis de facilement identifier les bugs tout au long du développement.

4.2 Affichage des images

Pour afficher des images dans le composant Display, deux moyens ont été utilisés.

Constantes tableau 2D

Les plus petites images sont stockées sous forme de constantes dans le package local. Pour chaque image, une nouveau type (tableau 2D) de STD_LOGIC_VECTOR(8) est créé. La constante prend ensuite la valeur de chaque pixel au format RGB en hexadécimal.

Exemple ci-dessous avec la déclaration de la constante du Flappy :

```
type memory_m_40_40 is array(0 to 39,  0 to 39) of std_logic_vector(7 downto 0);
constant d_flappy : memory_m_40_40 :=
       ((x"5b", x"5b", x"5b", x"5b", x"5b", x"5b", ...),
       (...));
```

La conversion d'une image JPG en hexadécimal ainsi que le formatage de la constante au format VHDL est effectuée au moyen d'un script Python (script réalisé par Bruno Produit) disponible en annexe.

L'accès à l'image dans le composant display se fait au travers des deux index du tableau 2D. La position actuelle du balayage de l'écran (hcount et vcount) est utilisée comme paramètre. Comme les images sont plus petites que la taille de l'écran, des constantes sont utilisées pour ajuster la valeur d'hcount et vcount pour matcher l'index de l'image. Exemple ci-dessous :

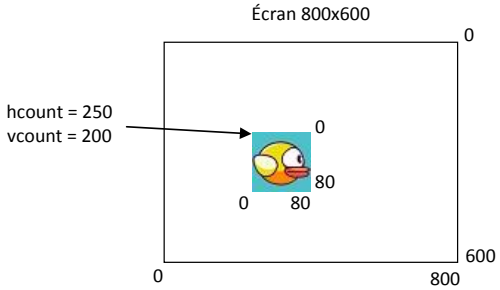


Figure 9 - Position des index de l'écran et de l'image

Affectation de la constante en VHDL : `color_s <= d_flappy((hcount_s - FLAPPY_OFFSET_H_c), (vcount_s - FLAPPY_OFFSET_V_c));`

Block ROM

Les plus grosses images sont stockées sous la forme d'un block / composant dans la ROM.

La démarche pour créer un block ROM est la suivante :

1. Choisir un « Block Memory Generator » dans les IP Core d'ISE
2. Choisir comme type de mémoire une « Single Port ROM »
3. La taille de la mémoire est ensuite configurée avec :
    a. Largeur : 8 bits
    b. Longueur : n bits (avec n = largeur*hauteur de l'image à stocker)
4. Initialiser la mémoire avec l'image au format .coe

Pour effectuer la conversion d'une image JPG en .coe, un script Matlab (disponible en annexe) est utilisé.

ISE crée ensuite un composant avec l'entité suivante :

```
COMPONENT d_comp_name
  PORT (
    clka  : IN STD_LOGIC;
    addra : IN STD_LOGIC_VECTOR(nn DOWNTO 0);
    douta : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
  );
END COMPONENT;
```

Ce composant est ensuite intégré au composant Display. Ci-dessous le schéma d'un exemple d'intégration.

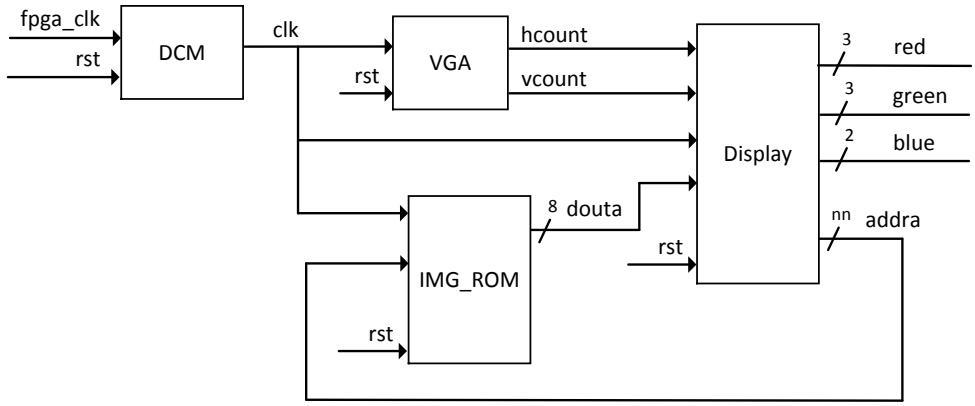
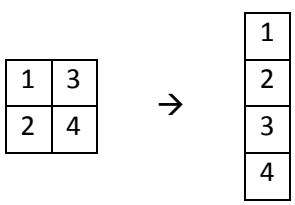


Figure 10 - Intégration d'un block ROM au composant Display pour l'affichage d'une image

L'entrée « addra » correspond au « numéro » / à l'adresse du byte que l'on veut accéder. Comme les dimensions de l'image ont été transformées de « largeur x hauteur x 8 bits » à « 8 bits x (largeur*hauteur) », les index hcount et vcount doivent être transformés pour être passés en paramètre au composant. Ci-dessous le format de stockage des bytes d'une image dans un block ROM.



4.3 Utilisation des ressources de la FPGA

L'affichage d'images sur la FPGA utilise beaucoup de ressources, et spécialement des LUTs. Le tableau ci-dessous illustre les principales ressources utilisées.

Ressource	Utilisé	Disponible	Utilisation
Slice Registers	232	18 224	1 %
Slice LUTs	7 224	9 112	79 %
Occupied Slices	2 235	2 278	98 %
RAMB16BWERs	24	32	75 %
RAMB8BWERs	16	64	25 %

Pour illustrer l'impact de la partie graphique du jeu sur les LUTs, nous avons analysé le « Post place & route report » avant et après ajout d'une fonctionnalité (affichage d'un Flappy « mort » en fin de partie).

On voit dans les captures ci-dessous que l'ajout de cette fonctionnalité augmente de 9% le nombre de LUTs utilisées.

```
Number of Slice LUTs:                  6,392 out of   9,112   70%
  Number used as logic:                6,389 out of   9,112   70%
```

Figure 11 - Utilisation des LUTs dans la FPGA avant ajout fonctionnalité

```
Number of Slice LUTs:                  7,224 out of   9,112   79%
  Number used as logic:                7,221 out of   9,112   79%
```

Figure 12 - Utilisation des LUTs dans la FPGA après ajout fonctionnalité

5. Validation

La phase de tests est une étape obligatoire permettant le fonctionnement final du projet. Certains composants ont tout d'abord été simulés. La deuxième partie décrit les tests effectués pour valider le comportement du jeu. Finalement, des photos illustrent le jeu réalisé.

5.1 Simulation

Pour valider le fonctionnement de certains composants, des macros ont été réalisées. Les composants testés avec macros sont :

- **Btns_synchro** pour vérifier que les boutons sont bien synchronisés avec l'horloge.
- **Flappy_pos** pour vérifier que la position verticale du Flappy est bien décrémentée et qu'elle est incrémentée lorsque le bouton gauche est pressé.
- **Pipes_pos** pour vérifier que la position horizontale des tuyaux (A et B) est décrémentée.

Ci-dessous un extrait de la macro utilisée pour vérifier la fonctionnalité du composant Flappy_pos :

```
force -freeze sim:/Flappy_pos/game_state 10 0
force -freeze sim:/flappy_pos/clk 1 0, 0 {50 ns} -r 100
force -freeze sim:/Flappy_pos/rst 0 0
force -freeze sim:/Flappy_pos/enable 0 0
run
...
run
force -freeze sim:/Flappy_pos/enable 1 0
force -freeze sim:/Flappy_pos/btn_left 0 0
run
...
run
force -freeze sim:/Flappy_pos/btn_left 1 0
run
...
```

5.2 Tests du jeu

Pour tester notre jeu, nous avons utilisé plusieurs cas de tests imaginés à partir du fonctionnement du jeu. Chaque cas test est décrit de la manière suivante :

Etape du jeu	N°	Objectif, Description, Entrées et Sorties	Résultat
	O		
	D		
	E		
	S		

Figure 13- Structure des cas de test

Les champs suivants sont utilisés :

Champ	Description
Etape du jeu	Etape du jeu à tester
Numéro	Numéro du test
O (objectif)	Qu'est-ce qui est testé ?
D (description)	Comment est-ce que l'objectif est testé ?
E (entrées)	Actions à réaliser pour effectuer le test (entrées du test)
S (sorties)	Valeurs attendues en sortie du test
Résultat	Résultat du test (OK / NOK) qui dépend des sorties

L'ensemble des cas de tests effectués sont disponibles en annexe.

5.3 Etat de fonctionnement final

Tous les tests effectués sont validés. Nous n'avons pas trouvé de bugs et pouvons valider que le jeu correspond à nos attentes.

Au niveau des fonctionnalités implémentées, les tableaux ci-dessous résument les fonctionnalités initialement prévues (de base et « nice to have ») et indique si elles ont été réalisées ou non.

Fonctionnalités de base

Fonctionnalité	Réalisé
Affichage de l'écran d'accueil du jeu	OK
Implémentation de deux boutons de contrôle	OK
Affichage de l'oiseau, des tuyaux et du score	OK
Défilement des tuyaux selon une séquence définie (emplacements et taille des trous fixes)	OK
Saut de l'oiseau lorsque l'on clique sur un bouton	OK
Détection du contact entre l'oiseau et les tuyaux/le sol	OK
Affichage d'un écran différent lorsque la partie est perdue, avec : • Affichage du score • Possibilités de recommencer la partie ou de retourner à l'écran d'accueil	NOK

Figure 14 - Récapitulatif des fonctionnalités de base implémentées

Toutes les fonctionnalités de base ont été implémentées, sauf l'écran de fin de partie. En effet, nous n'avons pas réussi à mettre en place le menu de fin de partie à cause du manque de ressource de la FPGA.

Lorsque la partie est perdue, Flappy apparait avec une croix sur les yeux. Il est possible de recommencer la partie en pressant sur le bouton droit.

Fonctionnalités « nice to have »

Fonctionnalité	Réalisé
Enregistrement du score maximum	OK
Amélioration du graphisme, couleurs	OK
Sons lors du saut de l'oiseau, du passage d'un tuyau et lorsque la partie et perdue	NOK
Taille aléatoire des tuyaux	OK
Défilement des tuyaux aléatoire (pas toujours la même séquence)	OK
Plusieurs niveaux de difficulté (augmentation au fil du score)	OK
Mettre une partie en pause	NOK
Cheat mode / « easter egg »	NOK

Figure 15 - Récapitulatif des fonctionnalités de base implémentées

La plupart des fonctionnalités optionnelles ont été implémentées. Nous n'avons pas trouvé que d'ajouter du son ajoutait vraiment une plus-value au jeu. Les niveaux de difficulté ont été implémentés avec la diminution de l'espace entre le tuyau du haut et du bas en fonction de l'avancement du score.

La pause d'une partie, le cheat mode et le easter egg n'ont pas été implémentés.

5.4 Photos du jeu

Les photos suivantes illustrent le jeu réalisé. Elles représentent le résultat final sur l'écran.

Figure 16 - Ecran d'accueil (photo)

Figure 17 - Début de partie (photo)

Figure 18 - En cours de partie (photo)

Figure 19 - Fin de partie (photo)

6. Conclusion

Nous avons beaucoup apprécié travailler sur ce projet. En effet, cela nous a permis d'exercer toutes les notions vues en cours dans un projet plus conséquent. Le fait d'avoir un résultat visuel et de pouvoir interagir avec le système est un point très positif. Une fois la base de l'application réalisée, il est très satisfaisant de faire évoluer le système.

Nous avons tout d'abord réfléchi aux fonctionnalités absolument nécessaires à notre jeu, puis aux fonctionnalités optionnelles. Les maquettes ont ensuite permis de visualiser notre représentation du jeu. La phase de conception fut très importante. Elle a permis de concevoir tous les éléments du système. La phase d'implémentation s'est résumée à l'application des concepts réfléchis lors de la conception et à beaucoup de tests avec l'écran.

Au final, nous sommes satisfaits du résultat obtenu, autant au niveau visuel qu'au niveau « gameplay ». Nous avons également développé nos compétences et appris de nouvelles choses.

7. Références et Annexes

7.1 Sources

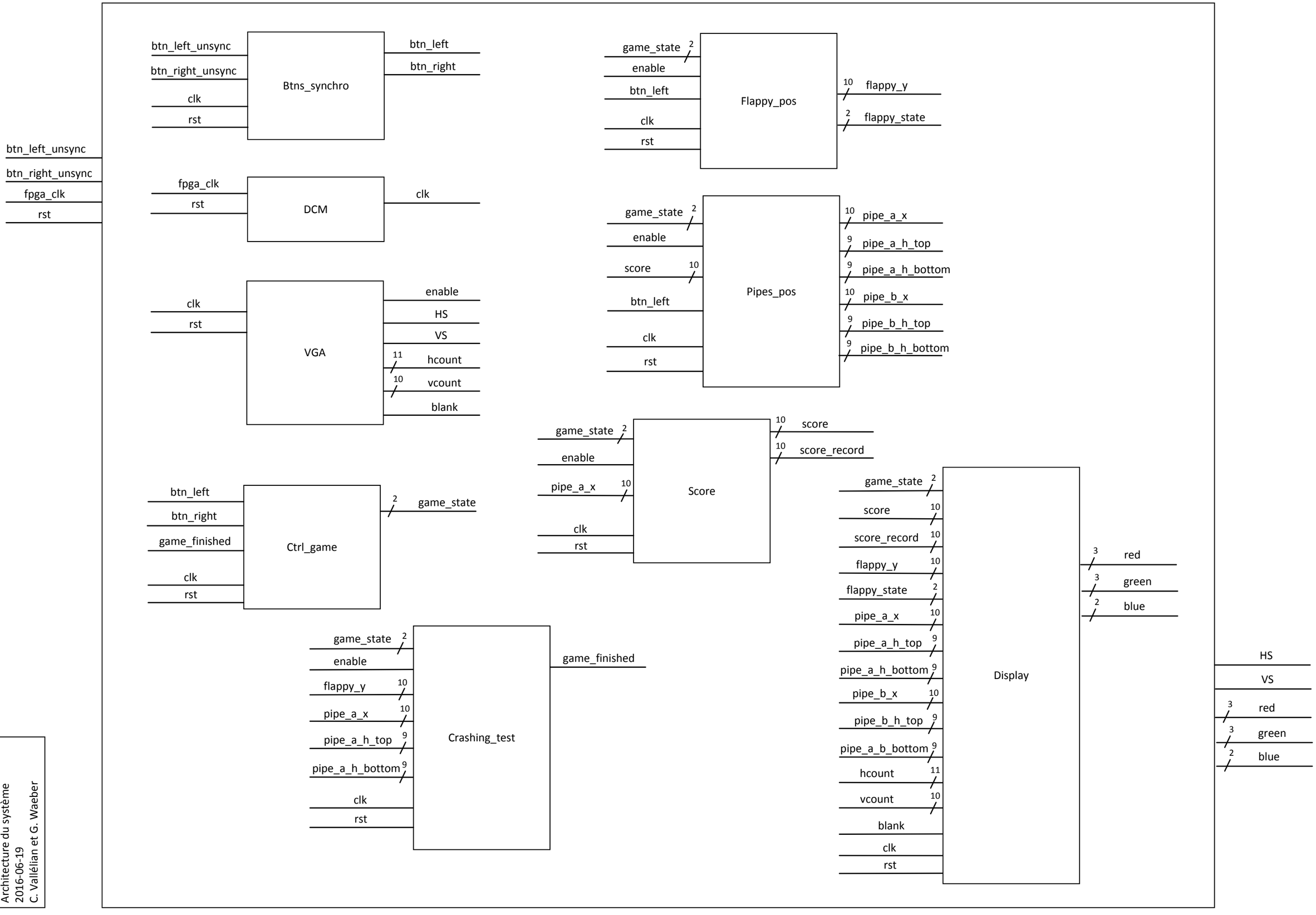
LBE Books, http://www.lbebooks.com/downloads/exportal/vhdl_nexys_example24.pdf

Bruno Produit, 2016, Script Python « JPG image to VHDL array »

7.2 Table des figures

7.3 Annexes

- Schéma des composants
- Cas de tests
- Script Python « JPG image to VHDL array »
- Script Matlab « IMG to .coe »



Sys Num 2 - Projet Flappy Bird
Architecture du système
2016-06-19
C. Vallélian et G. Waeber

Etape du jeu	N°	Objectif, Description, Entrées et Sorties	Résultat
Accueil du jeu	A1	O Affichage de l'écran d'accueil D Valider que l'image prévue pour l'écran d'accueil s'affiche correctement E Téléchargement du .bit sur la FPGA S Affichage sur l'écran VGA	OK
	A2	O Test de la machine d'état du jeu pour l'écran d'accueil D Valider que la réaction du jeu est correcte par rapport aux boutons pressés E Presser le bouton gauche S L'écran de démarrage du jeu (Flappy figé) doit s'afficher	OK
	A3	O Test de la machine d'état du jeu pour l'écran d'accueil D Valider que la réaction du jeu est correcte par rapport aux boutons pressés E Presser le bouton droit S Aucune réaction, l'écran accueil doit toujours être affiché	OK
Démarrage du jeu	D1	O Affichage de l'écran de démarrage du jeu D Valider que les éléments de départ (Flappy, tuyaux, score) se trouvent au bon endroit E - S Le score doit être égal à 0, le flappy et les tuyaux figés à l'endroit prévu	OK
	D2	O Démarrer une partie D Valider que la partie démarre E Presser le bouton gauche S La partie commence, flappy descend et les tuyaux défilent	OK
	D3	O Retourner au menu d'accueil D Valider la réaction du jeu par rapport au bouton pressé E Presser sur le bouton droit S Affichage de l'écran d'accueil	OK
Partie en cours	P1	O Affichage de Flappy D Valider l'affichage de Flappy E Ne pas presser sur le bouton gauche (saut) S Flappy doit être dans l'état "falling" et avoir le bec dirigé vers le sol	OK
	P2	O Affichage de Flappy (2) D Valider l'affichage de Flappy E Presser à plusieurs reprises sur le bouton gauche S Flappy doit être dans l'état "rising" et avoir le bec dirigé vers le haut	OK
	P3	O Recommencer une partie D Valider la réaction du jeu par rapport au bouton pressé E Commencer une partie (bouton gauche), puis presser sur le bouton droit pour recommencer la partie S L'écran de démarrage du jeu (Flappy et tuyaux figés) doit s'afficher	OK
	P4	O Affichage des tuyaux D Valider l'affichage et le défilement des tuyaux E Passer à travers plusieurs tuyaux et répéter sur plusieurs parties S La position des tuyaux et l'espacement entre celui du haut et du bas doivent être aléatoires (dès le 4ème tuyau). L'espace entre les tuyaux du haut et du bas doit diminuer au fur et à mesure de la partie.	OK
	P5	O Affichage du score correct D Valider l'affichage et l'évolution du score E Passer à travers plusieurs tuyaux S Le score est à chaque fois incrémenté de 1	OK
	P6	O Affichage du score correct (2) D Valider l'affichage et l'évolution du score E Passer à travers plusieurs tuyaux, puis recommencer une partie (bouton droit) S Le score doit être revenu à 0	OK
	P7	O Détection de la collision avec les tuyaux et le sol D Valider le test de fin de partie E Faire plusieurs parties en essayant de tester les cas suivants : - Entrer en collision avec le tuyau du bas - Toucher le sol - Entrer en collision avec le tuyau du haut - Entrer en collision avec les coins des tuyaux S La partie doit être perdue et Flappy doit s'afficher avec une croix noire sur les yeux	OK
	P8	O Affichage du best score D Valider l'affichage et l'évolution du best score E Faire une première partie (best score à 0) en passant à travers plusieurs tuyaux, perdre et noter le score final. Recommencer une partie. S Le best score doit avoir repris la valeur de la partie précédente	OK
	P9	O Affichage du background D Vérifier l'affichage du background E Jouer une partie et laisser descendre Flappy dans la zone du background S Un background doit s'afficher derrière les tuyaux sur toute la largeur. Le fond bleu autour de Flappy ne doit pas s'afficher (transparence)	OK

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

__author__  = "Bruno Produit"


import sys
import PIL
from PIL import Image

im = Image.open(sys.argv[1])    #Can be in different formats

# rescale if asked
if len(sys.argv) > 2:
    size = (int(sys.argv[2]), int(sys.argv[3]))
    im = im .resize(size, Image.ANTIALIAS)

# load image and create output file with 4 last characters stripped
output = open("".join([sys.argv[1][:-4], ".txt"]) , 'w')
pix = im.load()

#print (im.size) #Get the width and hight of the image for iterating over
#print (pix[5,5]) #Get the RGBA Value of the a pixel of an image

output.write("type memory_m is array(0 to ")
output.write(str(im.size[0]-1))
output.write(", 0 to ")
output.write(str(im.size[1]-1))
output.write(") of std_logic_vector(7 downto 0);\nconstant mem : memory_m :=\n(")
output.write("(")

# iterate over image
for i in range(0, im.size[0]):
  if (i != 0):
     output.write("), \n(")
  for j in range(0, im.size[1]):

      # take out and convert each pixel to it's binary value
      Blue =  bin(round(pix[i,j][0] >> 5))[2:].zfill(3) # bit shift of 5, stripped "0b", do
      not stip zeros
      Green = bin(round(pix[i,j][1] >> 5))[2:].zfill(3) # bit shift of 5, stripped "0b", do
      not stip zeros
      Red =   bin(round(pix[i,j][2] >> 6))[2:].zfill(2) # bit shift of 6 (only 2 bits for
      blue), stripped "0b", do not stip zeros
      data ='{:02x}'.format(int(''.join([Blue, Green, Red]), 2)) # concatenate and convert
      to hex
      output.write("x\"")
      output.write(data)
      output.write("\"")
      if (j != im.size[1]-1):
         output.write(", ")

output.write(")")
output.write(");")
```

```
function img2 = IMG2coe8(imgfile, outfile)
% Create .coe file from .jpg image
% .coe file contains 8-bit words (bytes)
% each byte contains one 8-bit pixel
% color byte: [R2,R1,R0,G2,G1,G0,B1,B0]
% img2 = IMG2coe8(imgfile, outfile)
% img2 is 8-bit color image
% imgfile = input .jpg file
% outfile = output .coe file
% Example:
% img2 = IMG2coe8('loons240x160.jpg', 'loons240x160.coe');

img = imread(imgfile);
height = size(img, 1);
width = size(img, 2);
s = fopen(outfile,'wb'); %opens the output file
fprintf(s,'%s\n','; VGA Memory Map ');
fprintf(s,'%s\n','; .COE file with hex coefficients ');
fprintf(s,'; Height: %d, Width: %d\n\n', height, width);
fprintf(s,'%s\n','memory_initialization_radix=16;');
fprintf(s,'%s\n','memory_initialization_vector=');
cnt = 0;
img2 = img;
for r=1:height
    for c=1:width
        cnt = cnt + 1;
        R = img(r,c,1);
        G = img(r,c,2);
        B = img(r,c,3);
        Rb = dec2bin(R,8);
        Gb = dec2bin(G,8);
        Bb = dec2bin(B,8);
        img2(r,c,1) = bin2dec([Rb(1:3) '00000']);
        img2(r,c,2) = bin2dec([Gb(1:3) '00000']);
        img2(r,c,3) = bin2dec([Bb(1:2) '000000']);
        Outbyte = [ Rb(1:3) Gb(1:3) Bb(1:2) ];
        if (Outbyte(1:4) == '0000')
            fprintf(s,'0%X',bin2dec(Outbyte));
        else
            fprintf(s,'%X',bin2dec(Outbyte));
        end
        if ((c == width) && (r == height))
            fprintf(s,'%c',';');
        else
            if (mod(cnt,32) == 0)
                fprintf(s,'%c\n',',');
            else
                fprintf(s,'%c',',');
            end
        end
    end
end
fclose(s);
```