

SPACE INVADERS

Projet intégré - Système numérique

M A I
11
2017



SAMUEL RIEDO & PASCAL ROULIN

1

INTRODUCTION

Space Invaders est un jeu vidéo d'arcade créé par Tomohiro Nishikado, paru pour la première fois en 1978 au Japon. Il est l'un des tout premier *Shoot 'em up*, c'est-à-dire un type de jeux consistant à abattre un grand nombre d'ennemies en leur tirant dessus. Le principe du jeu consiste en un vaisseau spatial attaqué par des vagues d'aliens qu'il doit détruire en leur tirant dessus sans se faire toucher par les tirs des aliens.

Space Invaders connu rapidement un succès mondial et est aujourd'hui considéré comme un grand classique de l'univers vidéoludique. Il a de ce fait connu de nombreux ports et suites sur un grand nombre de plate-forme, vieille comme récente.

1.1 Super Mario Bros

Dans un premier temps, nous avons souhaité reproduire *Super Mario Bros*. La première tentative pour recréer le monde 1-1 du jeu original fut de créer toute la map en une image, puis de la stocker dans une RAM ou ROM. Un *scalling* de 4 permet de drastiquement réduire le nombre de pixels à stocker, en passant de 6400x800 pour l'image d'origine à 1600x150 pour celle que nous utiliserons. Ceci représentait 240 000 pixels à stocker. En prenant en compte qu'un pixel fait exactement un Byte (deux bits pour la composante bleu, trois pour la rouge et trois pour la verte), les RAM et ROM à disposition de la Spartan 6 XC6LX16-CS324 ne pouvaient pas stocker toutes ses données.

Afin de contourner ce problème, l'image de base a été divisée en 8 images plus petites, faisant chacune 200x150 pixels, soit 30kB. Il était alors possible de stocker une image dans une ROM/RAM, puis une deuxième dans une seconde ROM/RAM, mais il était à nouveau impossible d'enregistrer les six suivantes sans dépasser les capacités de la carte.

Devant ses limitations hardware, la décision fut prise de changer de jeu. Il nous est apparu qu'avoir un fond statique, ou alors une répétition permanente d'un même arrière plan était indispensable pour que le projet soit synthétisable sur notre carte. Un jeu tel que *Super Mario Bros*, avec des mondes très différents et non répétitifs, n'est pas adapté à la programmation VHDL sur un hardware limité. Notre choix s'est alors porté sur *Space Invaders*.

1.2 Gameplay

Space Invaders est un jeu en deux dimension, aussi appelé jeu en 2D ou tout simplement jeu 2D. Le joueur contrôle un vaisseau spatial pouvant se déplacer uniquement sur l'axe X , et tirer des laser vers le haut de l'écran. Il est confronté à plusieurs aliens, se déplaçant aléatoirement dans la partie supérieur de l'écran. Ces derniers tire aléatoirement des lasers vers le bas pour détruire le vaisseau spatial contrôlé par le joueur.

Si le vaisseau du joueur se fait toucher par un laser alien, la partie est perdue. Si, au contraire, le joueur réussit à détruire tous les aliens sans se faire lui-même toucher, il gagne la partie. La figure 1.1 représente une partie typique de *Space Invaders* sur borne arcade tel que le jeu était lors de son lancement initial en 1978.

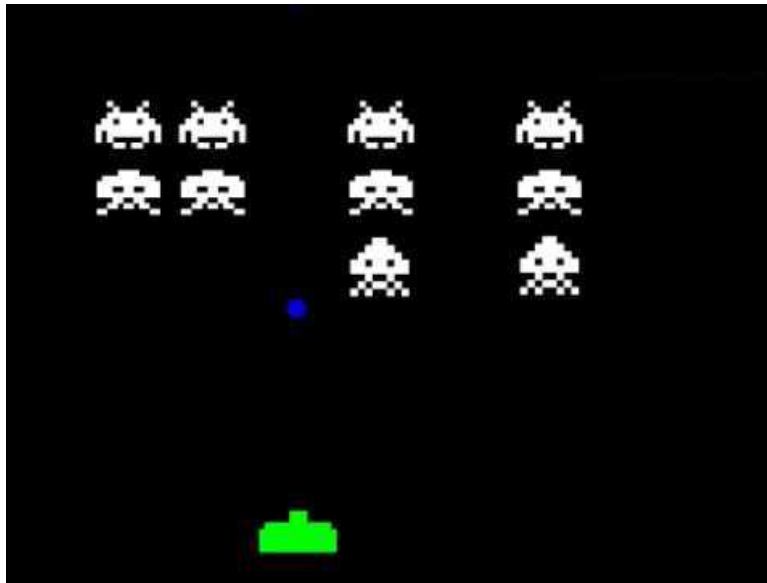


FIG. 1.1 : Space Invaders sur borne arcade

Le vaisseau du joueur est représenté par la forme verte. Ce dernier a tiré un laser, symbolisé par un point bleu. Les aliens, au nombre de 10, ont déjà été partiellement dessinés. Au début d'une partie, leur nombre et leur disposition forme une grille rectangulaire complète.

i

Types d'aliens

Bien que les aliens peuvent avoir plusieurs formes différents (trois sur la figure 1.1), cela n'influence en rien leur comportement. Il ne s'agit ni plus ni moins que d'un skin.

Les versions suivantes de Space Invaders implémenteront de nouvelles fonctionnalités, tel que :

- Score, déterminé par les aliens détruits et le temps pour y arriver.
- Vaisseau alien traversant l'écran horizontalement de façon aléatoire. Le détruire rapporte des points bonus.
- Bouclier pour protéger le vaisseau.

2

ARCHITECTURE

La réalisation du projet fut divisé en 6 principaux blocs, plus un top module et un package. Trois de ces blocs furent repris du travail pratique concernant l'affichage par VGA, alors que les autres ont été spécialement implémentés pour ce projet.

Le fait d'avoir déjà une base de départ nous a poussé à implémentés le jeu fonction par fonction, puis de tester et déboguer chaque nouvel ajout dès qu'il fut coder. Cette approche présente l'avantage, contrairement à un développement de chaque composants indépendamment les uns des autres, de réduire le risque d'incompatibilité entre deux composants à fin ainsi que le lourd travaille de debuggage final. En revanche, cette technique ne permet pas de répartir efficacement le travail dans une équipe constituée de nombreuse personnes.

Bien que nous puissions tester le bon fonctionnement de chaque blocs et fonctions directement en programmant la FPGA pour voir le résultat sur l'écran, chaque bloc sera testé par une macro pour valider tous les cas pouvant intervenir dans le déroulement d'une partie. De plus, deux composants, *Input* et *rocketManager*, seront testé via un testbench.

2.1 alienRocket

Le bloc *alienRocket* gère les roquettes, aussi appelé laser ou missile, tirés par les aliens en direction du spationef. Il est chargé de générer de nouveaux missiles ainsi que de transmettre les informations nécessaires au bloc *Display* pour afficher correctement une roquette à l'écran. Dans notre implémentation du jeu, le joueur fait face à 50 aliens, réparties en une grille de 5 lignes et 10 colonnes (tableau 2.1). Par colonne, chaque aliens le plus proche du bas de l'écran peut tirer une roquette vers le bas pour tenter de détruire le vaisseau du joueur. Une seule roquette peut être affiché à l'écran en même temps (sans compter les tirs du joueur). Cela signifie que tant que le missile n'a pas atteint le bat de l'écran, les aliens ne peuvent pas en tirer un nouveau.

Lorsqu'une roquette peut être tiré, le choix de la colonne d'alien pouvant tirer se fait de manière aléatoire entre toutes les colonnes qui contiennent au moins un alien. Par exemple, soit les aliens encore en vie selon le tableau 2.1. Une roquette peut être tiré depuis les aliens 0-4 et 7-9 de la ligne 5 (*alienLine5*), ainsi que depuis les alien 5 et 6 de la ligne 4 (*alienLine4*).

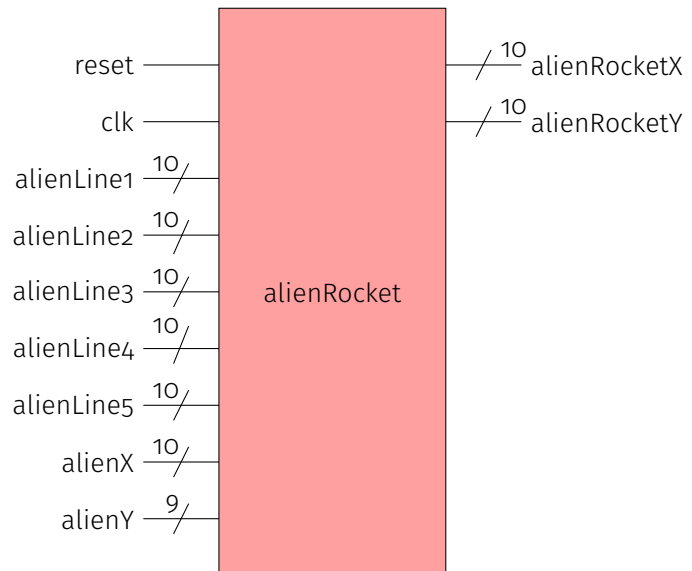


















































FIG. 2.1 : Schéma bloc

	Index									
	0	1	2	3	4	5	6	7	8	9
alienLine1										
alienLine2										
alienLine3										
alienLine4										
alienLine5										

TAB. 2.1 : Gestion des aliens dans le jeux

Les signaux *alienLine1* à *alienLine5* sont des *std_logic_vector* de 10 bits. Chaque bits indique par une valeur à 1 la présence d'un alien à l'index correspondant, alors qu'un 0 indique que l'alien a été tué par le joueur. Par exemple, selon le tableau 2.1, *alienLine5* vaut 111100111.

La fréquence à laquelle les aliens tirent des roquettes peut être modifiée en changeant la constante *rocket-Frequency* dans le package *SpaceInvadersPackage*.

2.1.1 Entrées & Sorties

reset Reset du circuit, actif à l'état haut.

clk Horloge 40MHz, active sur front montant.

alienLine1 Indique, par un 0 un alien vivant, et par 1 un alien mort dans la ligne d'alien de la partie supérieur de l'écran.

alienLine2 Identique à alienLine1, pour la ligne d'alien en dessous.

alienLine3 Identique à alienLine2, pour la ligne d'alien en dessous.

alienLine4 Identique à alienLine3, pour la ligne d'alien en dessous.

alienLine5 Identique à alienLine4, pour la ligne d'alien en dessous.

alienX Nombre de pixels entre le bord gauche de l'écran et le board droit des aliens à l'index 0.

alienY Nombre de pixels entre le haut de l'écran et le bord supérieur des aliens contenues dans *alien-Line1*.

alienRocketX Nombre de pixels entre le bord gauche de l'écran et la rocket tirée par les aliens.

alienRocketY Nombre de pixels entre le le haut de l'écran et le haut de la rocket tirée par les aliens.

2.1.2 Problèmes rencontrés

2.2 Digital Clock Management

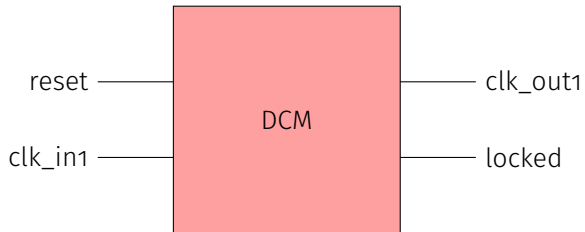


FIG. 2.2 : Schéma bloc

La norme VGA utilise une fréquence de 40MHz pour le balayage de l'écran. Or, l'horloge intégrée à notre carte dispose d'une fréquence de 100MHz. Le bloc DCM crée une horloge de 40MHz grâce à une horloge d'entrée de 100MHz. Ce type de montage étant très courant, il existe des outils, appelé IP Core, pour générer des composants génériques tel qu'un DCM.

2.2.1 Entrées & Sorties

reset Reset du circuit, actif à l'état haut.

clk_in1 Horloge 100MHz, active sur front montant.

clk_out1 Horloge 40MHz.

locked Sortie non utilisée.

2.2.2 Problèmes rencontrés

2.3 Display

Grâce aux signaux généraux par les composants *VGA_Internal* et *DCM*, *Display* est en mesure d'afficher des données à l'écran au moyen des trois sorties *red*, *green* et *blue*. Ses dernières sont codées sur trois bits, à l'exception de la composante bleu qui n'est uniquement constituée de deux bits, en accord avec la norme VGA.

Différent test sont effectué dans *Display* pour distinguer quels éléments doivent être affichés et lesquels ne doivent pas l'être. Le premier d'entre eux est de consulter la valeur de *gameStarted* pour afficher au non l'écran titre du jeu. Si cette valeur vaut 1, le composant détecte si la partie est terminée ou encore en cours. Pour cela, il suffit de regarder si la roquette des aliens est au même coordonnées que le vaisseau du joueur.

De cette manière, *Display* est non seulement utilisé pour afficher des données à l'écran mais également pour faire des tests sur l'état du jeu, comme la détection de fin de partie (que se soit une victoire ou un défaite du joueur) ainsi que la gestion des collisions entre les rockets et les aliens ou le joueur. Le choix d'avoir fait ses tests

à l'intérieur de *Display* plutôt que de créer un nouveau block spécifique à ce cas fut digirer par le fait que tous les signaux nécessaires à ces deux tests étaient déjà présent dans ce blocs, et qu'il aurait été redondant de les ajouter ailleurs.

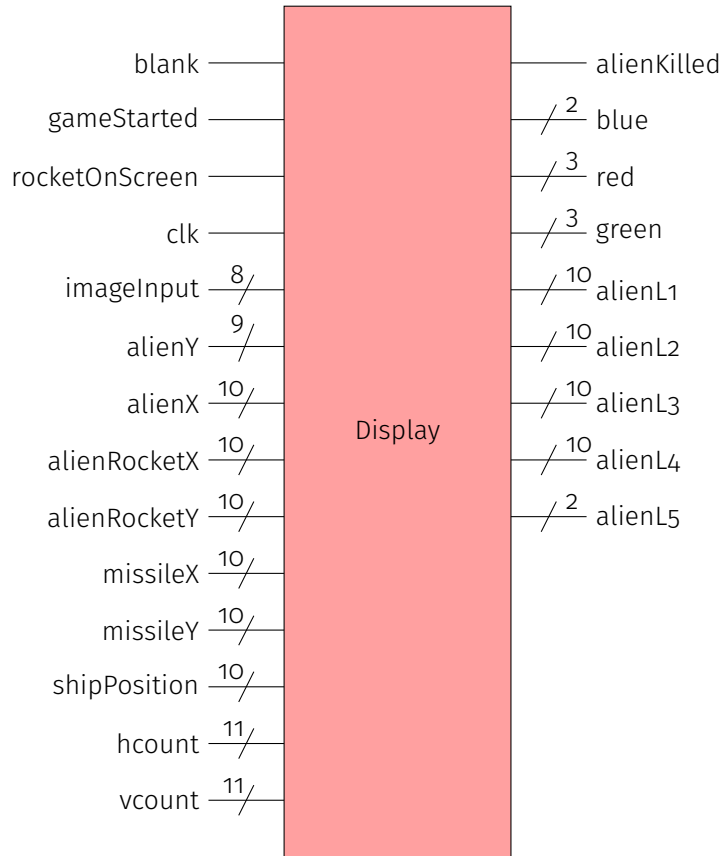


FIG. 2.3 : Schéma bloc

```

— Update gameOver, ship and rocket collision detection
process(gameStarted, clk, shipPos, alienRocketx,
  alienRockety)
begin
  if gameStarted = '0' then
    gameOver <= '0';
  elsif rising_edge(clk) then
    if to_integer(unsigned(alienRockety)) >= 570 and
       to_integer(unsigned(alienRocketx)) < 600 then
      if (to_integer(unsigned(alienRocketx))+6) >=
         shipPos and to_integer(unsigned(alienRocketx)) < (
           shipPos+56) then
        gameOver <= '1';
      end if;
    end if;
  end if;
end process;

```

Listing 2.1 : Détection de la mort du joueur.

Lorsque la partie est en cours, les aliens sont affichées à l'écran selon les signaux internes *alienLine1* à *alienLine5*. Ces derniers fonctionnent de la même manière que dans le bloc *alienRocket* (voir tableau 2.1).

2.3.1 Entrées & Sorties

- blank** Si 1, le balayage est en dehors de l'écran et les composantes RGB, c'est-à-dire les sorties *red*, *blue* et *green* doivent être null.
- gameStarted** Indique par une valeur à 1 que le jeu à débuté.
- rocketOnScreen** Indique par une valeur à 1 qu'une rocket doit être affiché à l'écran.
- clk** Horloge 40MHz, active sur front montant.
- imageInput** Bus de données en provenance de la ROM contenant l'image d'accueil du jeu.
- alienY** Nombre de pixels entre le haut de l'écran et le bord supérieur des aliens contenues dans *alien-Line1*.
- alienX** Nombre de pixels entre le bord gauche de l'écran et le board droit des aliens à l'index 0.
- alienRocketX** Nombre de pixels entre le bord gauche de l'écran et la rocket tirée par les aliens.
- alienRocketY** Nombre de pixels entre le le haut de l'écran et le haut de la rocket tirée par les aliens.
- missileX** Nombre de pixels entre le bord gauche de l'écran et la rocket lancée par les aliens.
- missileY** Nombre de pixels entre le haut de l'écran et le haut de la rocket lancée par les aliens.
- shipPosition** Nombre de pixels entre le bord gauche de l'écran et le bord gauche du vaisseau contrôlé par le joueur.
- hcount** Coordonnée *X* du balayage.
- vcount** Coordonnée *Y* du balayage.
- alienKilled** Indique par une valeur à 1 qu'un alien à été touché par une rocket lancée par le joueur.
- blue** Composante bleu de la sortie VGA.
- red** Composante rouge de la sortie VGA.
- green** Composante verte de la sortie VGA.
- alienL1** Indique par une valeur à 1 la présence d'un alien au même index dans la rangée d'aliens la plus proche du haut de l'écran (voir tableau 2.1).
- alienL2** Identique à *alienL1* pour la rangée d'aliens inférieur.
- alienL3** Identique à *alienL2* pour la rangée d'aliens inférieur.
- alienL4** Identique à *alienL3* pour la rangée d'aliens inférieur.
- alienL5** Identique à *alienL4* pour la rangée d'aliens inférieur.

2.3.2 Problèmes rencontrés

2.4 Input

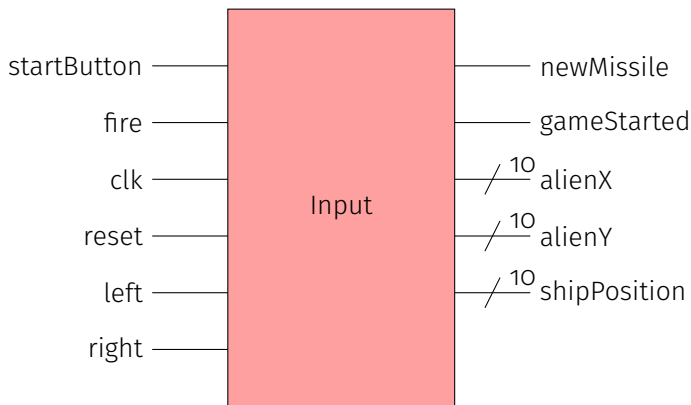


FIG. 2.4 : Schéma bloc

Input est chargé de recevoir et traiter les actions faites par le joueur. Via quatre des cinq boutons de la croix directionnelle, le joueur peut démarrer une partie, déplacer son vaisseau et tirer. Toutes ses actions sont traitées dans *Input*, tout comme le déplacement aléatoire des aliens sur l'écran. Ses derniers peuvent se mouvoir en haut, bas, gauche, droite ainsi qu'en diagonale. La direction qu'ils prennent est aléatoire, en fonction d'un compteur *alienDirection* s'incrémentant à chaque coût d'horloge. De plus, la distance, ou saut, qu'ils parcourent en un déplacement est également aléatoire

via un autre compteur *alienJump*. La taille maximal d'un saut peut être définie en modifiant la valeur *maxAlienJump* dans le package du jeu.

```

signal alienDirection : integer range 0 to 7 := 0; — If 0, aliens move left, 1=up left, 2 = up, ...
signal alienJump      : integer range 1 to maxAlienJump := 1; — Pixels number alien use as unit to move
process(reset, clk)
begin
  if reset = '1' then
    alienDirection <= 0;
    alienJump      <= 1;
  elsif rising_edge(clk) then
    — alien direction
    if alienDirection >= 7 then
      alienDirection <= 0;
    else
      alienDirection <= alienDirection + 1;
    end if;
    — alien jump
    if alienJump >= maxAlienJump then
      alienJump <= 1;
    else
      alienJump <= alienJump + 1;
    end if;
  end if;
end process;

```

2.4.1 Entrées & Sorties

startButton Bouton situé au centre de la croix directionnelle (B8). Utilisé pour démarrer une partie depuis l'écran d'accueil.

fire Bouton supérieur de la croix directionnelle (BTNU, A8). Utilisé pour tirer des lasers depuis le vaisseau vers les aliens pendant une partie.

clk Horloge 40MHz, active sur front montant.

reset Reset du circuit, actif à l'état haut.

left Bouton gauche de la croix directionnelle (BTNL, C4). Utilisé pour déplacer le vaisseau du joueur vers la gauche.

right Bouton right de la croix directionnelle (BTNR, D9). Utilisé pour déplacer le vaisseau du joueur vers la droite.

newMissile Indique par une valeur à 1 qu'un nouveau missile a été tiré par le vaisseau du joueur.

gameStarted Indique par une valeur à 1 que le jeu à débuté.

alienX Nombre de pixels entre le bord gauche de l'écran et le board droit des aliens à l'index 0.

alienY Nombre de pixels entre le haut de l'écran et le bord supérieur des aliens contenues dans *alien-Line1*.

shipPosition Nombre de pixels entre le bord gauche de l'écran et le bord droit du vaisseau de joueur. Cette valeur est utilisé pour générer de l'aléatoire.

2.4.2 Problèmes rencontrés

2.5 rocketManager

Le bloc *rocketManager* a pour rôle de générer les missiles du vaisseau spatial que le joueur pilote. Le bloc *Input* fourni à *rocketManager* une impulsion, via le signal *newMissile*, lorsque celui-ci doit générer un nouveau missile. Cette impulsion apparaît lorsque le joueur clique sur le bouton pour tirer un nouveau missile. Dès qu'un missile est tiré (*fire*), celui-ci doit avoir une coordonnée en X, une coordonnée en Y ainsi qu'être signaler au bloc *Display* afin que celui-ci sache qu'un missile doit être affiché.

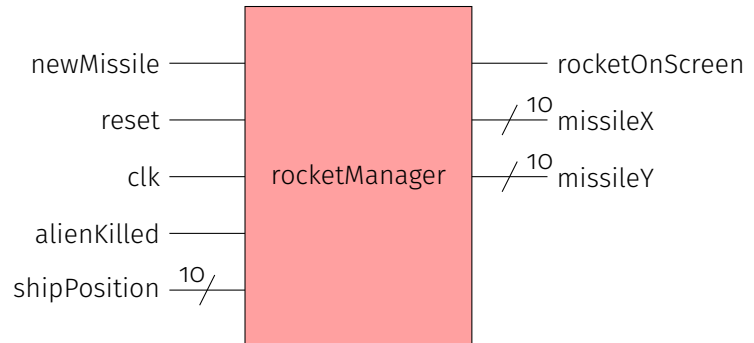


FIG. 2.5 : Schéma bloc

L'entrée *alienKilled* permet au bloc de savoir si un alien a été touché, et dans ce cas le signal de sortie *rocketOnScreen* passera à 0 et le missile n'apparaîtra plus à l'écran. Ce signal passe également à 0 lorsque le missile atteint le haut de l'écran.

RocketManager utilise le signal *shipPosition*, qui correspond à la coordonnée sur l'axe horizontale du vaisseau au moment du tir, afin de fournir en sortie (*missileX*) la position X du missile. Cette position reste la même tant que le missile n'a pas atteint un alien ou le haut de l'écran.

Concernant la position Y du missile, celle-ci se comporte comme un compteur. En effet, cette position est remise à la valeur de 570 (hauteur de l'écran moins la hauteur du vaisseau), afin que le missile parte du vaisseau, et est ensuite décrémenté jusqu'à la valeur minimum de 0 (le missile atteint donc le haut de l'écran).

Les positions X et Y sont fournis au bloc *Display* via leurs signaux respectifs et ce dernier se charge d'afficher le missile.

Le bloc *rocketManager* utilise la clock afin de décrémenter le compteur de la position Y et le signal reset afin de remettre tous les signaux dans leur état d'origine.

2.5.1 Entrées & Sorties

newMissile Indique par une valeur à 1 qu'un nouveau missile a été tiré par le vaisseau du joueur.

reset Reset du circuit, actif à l'état haut.

clk Horloge 40MHz, active sur front montant.

alienKilled Indique par une valeur à 1 qu'un alien a été touché par une rocket lancée par le joueur.

shipPosition Nombre de pixels entre le bord gauche de l'écran et le bord droit du vaisseau de joueur. Cette valeur est utilisée pour générer de l'aléatoire.

rocketOnScreen Indique par une valeur à 1 qu'une rocket doit être affichée à l'écran.

missileX Nombre de pixels entre le bord gauche de l'écran et la rocket lancée par les aliens.

missileY Nombre de pixels entre le haut de l'écran et le haut de la rocket lancée par les aliens.

2.5.2 Problèmes rencontrés

2.6 VGA Internal

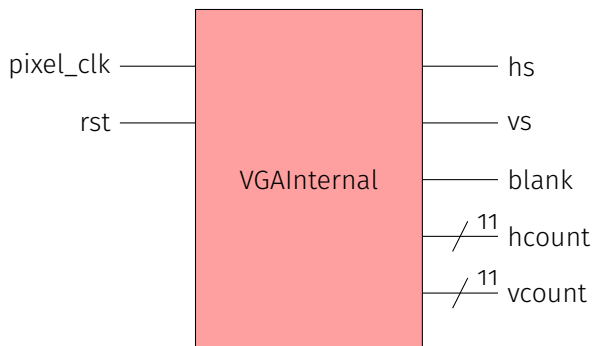


FIG. 2.6 : Schéma bloc

Une interface VGA fonctionne selon les trois composantes RGB ainsi qu'une synchronisation horizontale et verticale. Les signaux RGB décrivent la couleur des pixels composant l'image selon un balayage effectué de gauche à droite, en ligne de haut en bas. L'écran recevant ce flux RGB est capable de savoir à quel pixel il correspond selon l'instant t auquel il lit ses données dans le balayage. Néanmoins, ce n'est pas l'écran qui est chargé de sauter automatiquement à la ligne suivante lorsque chaque pixel de la ligne actuel a été traité. C'est ce à quoi sert le signal *HS*, alors que *VS* indique un retour à la première ligne.

i

Retour à la ligne

Afin d'informer le monitor que le balayage est arrivé à la fin d'une ligne et que le prochain pixel sera le premier de la ligne suivante, le signal *HS* produit une impulsion de synchronisation. De même, lorsque le balayage est arrivé à la fin de la dernière ligne (et donc que toute une image a été transmise), le signal *VS* produit une impulsion pour indiquer un retour à la première ligne (et donc la transmission d'une nouvelle image).

Lorsque le balayage se trouve en dehors de l'écran, le signal *blank* prend comme valeur 0 afin d'indiquer au bloc *Display* de mettre les composantes de sorties RGB à 0. Ce comportement est défini dans la norme VGA et résulte dans une erreur d'affichage "Index out of bound" s'il n'est pas respecté.

2.6.1 Entrées & Sorties

pixel_clk Horloge 40MHz, active sur front montant.

rst Reset du circuit, actif à l'état haut.

hs Impulsion de synchronisation horizontale. Indique par une pulse à l'état haut un retour à la ligne du balayage de l'écran.

vs Impulsion de synchronisation verticale. Indique par une pulse à l'état haut un retour du balayage à la première ligne de l'écran.

blank Si 1, le balayage est en dehors de l'écran et les composantes RGB, c'est-à-dire les sorties *red*, *blue* et *green* doivent être null.

hcount Coordonnée X du balayage.

vcount Coordonnée Y du balayage.

2.6.2 Problèmes rencontrés

2.7 Top Module

Le bloc *topModule* est comme son nom l'indique l'élément tout en haut de notre architecture. C'est lui qui instancie les composants et relie ceux-ci via des signaux intermédiaires.

Concernant ses entrées et sorties, celles-ci sont des éléments physiques de la carte FPGA et sont assignées via le fichier UCF.

Le signal d'entrée *fpga_clk* correspond à la clock de la FPGA. Cette clock fonctionne à une fréquence de 100 MHz. Le *reset* est assigné sur l'un des switches (voir figure **XXXXXXX**) et permet le reset software du jeu.

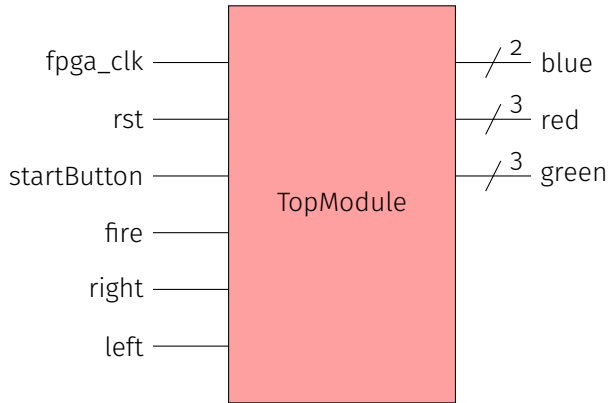


FIG. 2.7 : Schéma bloc

Plusieurs boutons sont utilisés afin de jouer et ont donc été assignés grâce au fichier UCF. Au niveau des signaux, il s'agit de *startButton*, de *fire*, de *right* et de *left*. Le bouton *fire* permet de tirer un missile depuis le vaisseau. Puisqu'il n'est possible que de tirer un seul missile à la fois, celui-ci n'est pas d'effet tant qu'il y a un missile en déplacement. Les boutons *right* et *left* sont explicites. Ils permettent le déplacement latéral du vaisseau que pilote le joueur. Le *startButton* permet de passer de l'écran de démarrage au jeu. Il n'a d'effet que durant cette étape afin qu'en cas d'appui involontaire durant la partie celle-ci ne s'en retrouve pas impactée.

Nous avons commencer par utiliser un seul bouton afin de passer de l'écran de démarrage au jeu et pour tirer des missiles. Cependant, lorsque le jeu commençait, un missile était automatiquement tiré et cela ne correspondait pas à ce que nous voulions. Nous avons chercher un moyen de corriger ce problème mais cela s'est avéré compliqué pour un problème mineur alors qu'une solution simple, et finalement plus pratique, était de séparer ces deux fonctions en deux boutons.

Concernant les sorties *blue*, *red* et *green*, celles-ci sont assignées sur la sortie VGA de la carte. Ces signaux correspondent aux trois couleurs (RGB) utilisé par le VGA afin d'afficher sur un écran. Puisque le VGA utilise 8 bits, les signaux *red* et *green* disposent de 3 bits chacun. Le signal *blue* ne dispose que de 2 bits. Il est tout de même possible de d'afficher 256 couleurs différentes.

2.7.1 Entrées & Sorties

fpga_clk Horloge 100MHz, active sur front montant.

rst Reset du circuit, actif à l'état haut.

startButton Bouton situé au centre de la croix directionnelle (B8). Utilisé pour démarrer une partie depuis l'écran d'accueil.

fire Bouton supérieur de la croix directionnelle (BTNU, A8). Utilisé pour tirer des lasers depuis le vaisseau vers les aliens pendant une partie.

right Bouton right de la croix directionnelle (BTNR, D9). Utilisé pour déplacer le vaisseau du joueur vers la droite.

left Bouton gauche de la croix directionnelle (BTNL, C4). Utilisé pour déplacer le vaisseau du joueur vers la gauche.

blue Composante bleu de la sortie VGA.

red Composante rouge de la sortie VGA.

green Composante verte de la sortie VGA.

2.7.2 Problèmes rencontrés

2.8 Package

ANNEXES

3

3.1 Conversion d'image

Le bloc *Display* affiche des données à l'écran en affectant une valeur au trois signaux *red*, *green* et *blue*. Les deux premiers sont contenues sur trois bits, alors que le dernier est uniquement sur deux. Cela signifie que le jeux de couleurs à disposition vaut :

$$\begin{aligned} n_{colors} &= 2^8 \\ &= 256 \end{aligned}$$

Une couleur est ainsi affectée à chaque pixel, en commençant par celui en haut à gauche de l'écran, puis celui à sa droite et ainsi de suite jusqu'à arriver à la droite de l'écran et commencer la ligne suivante. Il apparait alors que pour afficher une image, il faut extraire le code couleur de chaque'un de ses pixels, puis les stocker d'une des deux façon suivantes :

- Dans une RAM ou une ROM, cela implique de convertir l'image en un fichier *COE* de la forme suivante :

```
memory_initialization_radix=16;
memory_initialization_vector=
43,
26,
2,
6e,
6e,
4a,
d9,
b6,
6e,
dd,
b5,
6a,
dd,
b6,
6a;
```

memory_initialization_radix=16; indique les valeurs sont stockées en hexadécimale. Il est possible de le faire en binaire ou en decimal.

- Dans un tableau 2D :

```
type memoryPicture is array(0 to 4, 0 to 2) of integer;
constant picture : memoryPicture :=(
(16#43#,16#26#,16#2#),
(16#6e#,16#6e#,16#4a#),
(16#d9#,16#b6#,16#6e#),
(16#dd#,16#b5#,16#6a#),
(16#dd#,16#b6#,16#6a#));
```

Avec un COE, chaque valeur est stocké à la suite. En utilisant un tableau VHDL, il est possible de stocker en deux dimensions les valeurs pour avoir une représentation identique à celle de l'écran. La forme *16#<value>#* indique en VHDL que le nombre est sous forme hexadécimale et peut être directement assigné à un signal de type *std_logic_vector*.

Afin de rapidement convertir des images en fichier COE ou en tableau 2D VHDL, nous avons écrit un script Matlab. Le détail de son fonctionnement est inclus dans les commentaires du code.



Format des images

Seul les images au format "JPG" sont supportés par le script.

3.1.1 Script Matlab - Conversion en fichier COE

```

1 %read the image
2 I = imread('yourPicture.jpg');
3 [x,y,z] = size(I); % x = width, y = heigh
4 width = x-1;
5
6 %Extract RED, GREEN and BLUE components from the image
7 R = I(:, :,1);
8 G = I(:, :,2);
9 B = I(:, :,3);
10
11 %make the numbers to be of double format for
12 R = double(R);
13 G = double(G);
14 B = double(B);
15
16 %Raise each member of the component by appropriate value.
17 R = R.^(3/8); % 8 bits -> 3 bits
18 G = G.^(3/8); % 8 bits -> 3 bits
19 B = B.^(1/4); % 8 bits -> 2 bits
20
21 %translate to integer
22 R = uint8(R); % float -> uint8
23 G = uint8(G);
24 B = uint8(B);
25
26 %minus one cause sometimes conversion to integers rounds up the numbers wrongly
27 R = R-1;
28 G = G-1;
29 B = B-1;
30
31 %shift bits and construct one Byte from 3 + 3 + 2 bits
32 G = bitshift(G, 2);
33 R = bitshift(R, 5);
34 COLOR = R+G+B;
35
36 %save variable COLOR to a file in HEX format for the chip to read
37 fileID = fopen('output.coe', 'w');
38 fprintf(fileID, 'memory_initialization_radix=16;\n');
39 fprintf(fileID, 'memory_initialization_vector=\n');
40
41 for i = 1:size(COLOR,:), 1)-1
42     fprintf(fileID, '%x', COLOR(i)); % COLOR (dec) -> print to file (hex)
43     fprintf(fileID, ',\n');
44 end
45 fprintf(fileID, '%x;', COLOR(size(COLOR,:), 1))); % last pixel
46 fclose(fileID);
47
48 %translate to hex to see how many lines
49 COLOR_HEX = dec2hex(COLOR);

```

3.1.2 Script Matlab - Conversion en tableau VHDL

```

1 %read the image
2 I = imread('yourPicture.jpg');
3 [x,y,z] = size(I); % x = width, y = heigh
4 width = x-1;
5
6 %Extract RED, GREEN and BLUE components from the image
7 R = I(:,:,1);
8 G = I(:,:,2);
9 B = I(:,:,3);
10
11 %make the numbers to be of double format for
12 R = double(R);
13 G = double(G);
14 B = double(B);
15
16 %Raise each member of the component by appropriate value.
17 R = R.^(3/8); % 8 bits -> 3 bits
18 G = G.^(3/8); % 8 bits -> 3 bits
19 B = B.^(1/4); % 8 bits -> 2 bits
20
21 %translate to integer
22 R = uint8(R); % float -> uint8
23 G = uint8(G);
24 B = uint8(B);
25
26 %minus one cause sometimes conversion to integers rounds up the numbers wrongly
27 R = R-1;
28 G = G-1;
29 B = B-1;
30
31 %shift bits and construct one Byte from 3 + 3 + 2 bits
32 G = bitshift(G, 2);
33 R = bitshift(R, 5);
34 COLOR = R+G+B;
35
36 %save variable COLOR to a file in HEX format for the chip to read
37 fileID = fopen('output.vhd', 'w');
38 fprintf(fileID, 'type memoryPicture is array(0 to ');
39 fprintf(fileID, '%d', y-1);
40 fprintf(fileID, ', 0 to ');
41 fprintf(fileID, '%d', x-1);
42 fprintf(fileID, ') of integer;\n');
43 fprintf(fileID, 'constant picture : memoryPicture :=(\n(');
44 for i = 1:size(COLOR,:), 1)-1
45     fprintf(fileID, '16#');
46     fprintf(fileID, '%x', COLOR(i)); % COLOR (dec) -> print to file (hex)
47     fprintf(fileID, '#');
48     if width == 0 % line end
49         fprintf(fileID, '),\n(');
50         width=x-1;
51     else % not end of line
52         fprintf(fileID, ',');
53         width=width-1;
54     end
55 end
56 fprintf(fileID, '16#');
57 fprintf(fileID, '%x', COLOR(size(COLOR,:), 1)); % last pixel
58 fprintf(fileID, '#');
59 fprintf(fileID, '));');
60 fclose(fileID);
61
62 %translate to hex to see how many lines
63 COLOR_HEX = dec2hex(COLOR);

```

3.2 Code source

3.2.1 alienRocket

```

— Company :      HES-SO
— Engineer :     Samuel Riedo & Pascal Roulin
— Create Date :  20/04/2017
— Design Name :  alienRocket.vhd
— Project Name :  Space Invaders — FPGA Edition
— Target Devices : Diligent NEXYS 3 (Xilinx Spartan 6 XC6LX16-CS324)
— Description :   Manage rockets shoot by aliens
— Revision 0.1 —  File Created
—              1.0 —  First implementation

```

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.SpaceInvadersPackage.all;

entity alienRocket is
  port(
    reset      : in std_logic;           — Active high
    clk        : in std_logic;           — 40MHz
    alienLine1 : in std_logic_vector(0 to 9); — Top screen alien line
    alienLine2 : in std_logic_vector(0 to 9);
    alienLine3 : in std_logic_vector(0 to 9);
    alienLine4 : in std_logic_vector(0 to 9);
    alienLine5 : in std_logic_vector(0 to 9); — Bottom screen alien line
    alienX      : in std_logic_vector(9 downto 0); — first alien position from left screen
    alienY      : in std_logic_vector(8 downto 0); — first alien position from top screen
    alienRocketx : out std_logic_vector(9 downto 0); — Alien rocket x coordinate from left screen
    alienRockety : out std_logic_vector(9 downto 0); — Alien rocket y coordinate from top screen
  );
end entity;

architecture logic of alienRocket is

  signal column1 : std_logic_vector(4 downto 0); — First column of alien
  signal column2 : std_logic_vector(4 downto 0); — MSB = bottom screen alien
  signal column3 : std_logic_vector(4 downto 0); — LSB = top screen alien
  signal column4 : std_logic_vector(4 downto 0);
  signal column5 : std_logic_vector(4 downto 0);
  signal column6 : std_logic_vector(4 downto 0);
  signal column7 : std_logic_vector(4 downto 0);
  signal column8 : std_logic_vector(4 downto 0);
  signal column9 : std_logic_vector(4 downto 0);
  signal column10 : std_logic_vector(4 downto 0); — Last column of alien

  signal alienXX : integer range 0 to 1000; — Integer value of alienX
  signal alienYY : integer range 0 to 1000; — Integer value of alienY
  signal rocketSpeed : integer range 0 to missileSpeed; — Missile speed
  signal columnCounter : integer range 0 to 9 := 0; — Determine the shooting column
  signal newShoot : integer range 0 to 1 := 0; — Force a new shoot if 1
  signal rocketLaunched : integer range 0 to 1 := 0; — If 1, a rocket is launched
  signal newRocketYY : integer range 0 to VLINES := 0; — Y position of a new rocket launched
  signal rocketFinished : integer range 0 to 1 := 1; — If 1, a new rocket can be launched
  signal shootTimer : integer range 0 to rocketFrequency := 0; — Determine the shooting column

  signal rocketYY : integer range 0 to VLINES := VLINES; — Alien rocket y position from top screen
  signal rocketXX : integer range alienXMargin to (HLINES — alienXMargin) := alienXMargin; — Rocket x position when a new one is launched
  signal rocketXXX : integer range alienXMargin to (HLINES — alienXMargin) := alienXMargin; — Copy rocketXX before it is reset to 0

begin
  alienRocketx <= std_logic_vector(to_unsigned(rocketXXX, 10));
  alienRockety <= std_logic_vector(to_unsigned(rocketYY, 10));

  column1 <= alienLine5(0) & alienLine4(0) & alienLine3(0) & alienLine2(0) & alienLine1(0);
  column2 <= alienLine5(1) & alienLine4(1) & alienLine3(1) & alienLine2(1) & alienLine1(1);
  column3 <= alienLine5(2) & alienLine4(2) & alienLine3(2) & alienLine2(2) & alienLine1(2);

```

```

column4 <= alienLine5(3) & alienLine4(3) & alienLine3(3) & alienLine2(3) & alienLine1(3);
column5 <= alienLine5(4) & alienLine4(4) & alienLine3(4) & alienLine2(4) & alienLine1(4);
column6 <= alienLine5(5) & alienLine4(5) & alienLine3(5) & alienLine2(5) & alienLine1(5);
column7 <= alienLine5(6) & alienLine4(6) & alienLine3(6) & alienLine2(6) & alienLine1(6);
column8 <= alienLine5(7) & alienLine4(7) & alienLine3(7) & alienLine2(7) & alienLine1(7);
column9 <= alienLine5(8) & alienLine4(8) & alienLine3(8) & alienLine2(8) & alienLine1(8);
column10 <= alienLine5(9) & alienLine4(9) & alienLine3(9) & alienLine2(9) & alienLine1(9);

alienXX <= to_integer(unsigned(alienX));
alienYY <= to_integer(unsigned(alienY));

-- Update columnCounter, rocketXXX and shootTimer.
process(reset, clk)
begin
  if reset = '1' then
    columnCounter <= 0;
    shootTimer <= 0;
    rocketXXX <= alienXMargin;
  elsif rising_edge(clk) then
    -- rocketXX
    if rocketXX > alienXMargin then -- If a new rocket has been launched, rocketXX != alienXMargin.
      rocketXXX <= rocketXX;
    else
      rocketXXX <= rocketXXX;
    end if;
    -- columnCounter
    if columnCounter = 9 then
      columnCounter <= 0;
    else
      columnCounter <= columnCounter + 1;
    end if;
    -- shootTimer
    if newShoot = 1 then -- Force a new shoot.
      shootTimer <= rocketFrequency;
    elsif shootTimer = rocketFrequency then
      shootTimer <= 0;
    else
      shootTimer <= shootTimer + 1;
    end if;
  end if;
end process;

-- Launch new alien rocket.
process(shootTimer, columnCounter, column1, column2, column3, column4, column5, column6, column7,
        column8, column9, column10, alienyy, alienxx, rocketFinished)
begin
  if shootTimer = rocketFrequency and rocketFinished = 1 then
    rocketLaunched <= 1;
    newShoot <= 0;
    case columnCounter is
      when 0 =>
        -- Check which column will shoot.
        if column1 > "00000" then
          -- Check which alien in the column will shoot.
          if column1 > "01111" then
            newRocketYY <= (alienYY + 150);
            rocketXX <= (alienXX+15);
          elsif column1 > "00111" then
            newRocketYY <= (alienYY + 120);
            rocketXX <= (alienXX+15);
          elsif column1 > "00011" then
            newRocketYY <= (alienYY + 90);
            rocketXX <= (alienXX+15);
          elsif column1 > "00001" then
            newRocketYY <= (alienYY + 60);
            rocketXX <= (alienXX+15);
          else
            newRocketYY <= (alienYY + 30);
            rocketXX <= (alienXX+15);
          end if;
        else
          newRocketYY <= 0;
          rocketXX <= alienXMargin;
          newShoot <= 1;
        end if;
      when 1 =>
        if column2 > "00000" then
          if column2 > "01111" then
            newRocketYY <= (alienYY + 150);
            rocketXX <= (alienXX+45);
          elsif column2 > "00111" then
            newRocketYY <= (alienYY + 120);
            rocketXX <= (alienXX+45);
          elsif column2 > "00011" then
            newRocketYY <= (alienYY + 90);
            rocketXX <= (alienXX+45);
          elsif column2 > "00001" then
            newRocketYY <= (alienYY + 60);
            rocketXX <= (alienXX+45);
          else
            newRocketYY <= (alienYY + 30);
            rocketXX <= (alienXX+45);
          end if;
        else
          newRocketYY <= 0;
          rocketXX <= alienXMargin;
          newShoot <= 1;
        end if;
      when 2 =>
        if column3 > "00000" then
          if column3 > "01111" then
            newRocketYY <= (alienYY + 150);
            rocketXX <= (alienXX+75);
          elsif column3 > "00111" then
            newRocketYY <= (alienYY + 120);
            rocketXX <= (alienXX+75);
          elsif column3 > "00011" then
            newRocketYY <= (alienYY + 90);
            rocketXX <= (alienXX+75);
          elsif column3 > "00001" then
            newRocketYY <= (alienYY + 60);
            rocketXX <= (alienXX+75);
          else
            newRocketYY <= (alienYY + 30);
            rocketXX <= (alienXX+75);
          end if;
        else
          newRocketYY <= 0;
          rocketXX <= alienXMargin;
          newShoot <= 1;
        end if;
      when 3 =>
        if column4 > "00000" then
          if column4 > "01111" then
            newRocketYY <= (alienYY + 150);
            rocketXX <= (alienXX+105);
          elsif column4 > "00111" then
            newRocketYY <= (alienYY + 120);
            rocketXX <= (alienXX+105);
          elsif column4 > "00011" then
            newRocketYY <= (alienYY + 90);
            rocketXX <= (alienXX+105);
          elsif column4 > "00001" then
            newRocketYY <= (alienYY + 60);
            rocketXX <= (alienXX+105);
          else
            newRocketYY <= (alienYY + 30);
            rocketXX <= (alienXX+105);
          end if;
        else
          newRocketYY <= 0;
          rocketXX <= alienXMargin;
          newShoot <= 1;
        end if;
      when 4 =>
        if column5 > "00000" then
          if column5 > "01111" then
            newRocketYY <= (alienYY + 150);
            rocketXX <= (alienXX+135);
          elsif column5 > "00111" then
            newRocketYY <= (alienYY + 120);
            rocketXX <= (alienXX+135);
          elsif column5 > "00011" then
            newRocketYY <= (alienYY + 90);
            rocketXX <= (alienXX+135);
          elsif column5 > "00001" then
            newRocketYY <= (alienYY + 60);
            rocketXX <= (alienXX+135);
          else
            newRocketYY <= (alienYY + 30);
            rocketXX <= (alienXX+135);
          end if;
        else
          newRocketYY <= 0;
          rocketXX <= alienXMargin;
          newShoot <= 1;
        end if;
      when 5 =>
        if column6 > "00000" then
          if column6 > "01111" then
            newRocketYY <= (alienYY + 150);
            rocketXX <= (alienXX+165);
          elsif column6 > "00111" then
            newRocketYY <= (alienYY + 120);
            rocketXX <= (alienXX+165);
          elsif column6 > "00011" then
            newRocketYY <= (alienYY + 90);
            rocketXX <= (alienXX+165);
          elsif column6 > "00001" then
            newRocketYY <= (alienYY + 60);
            rocketXX <= (alienXX+165);
          else
            newRocketYY <= (alienYY + 30);
            rocketXX <= (alienXX+165);
          end if;
        else
          newRocketYY <= 0;
          rocketXX <= alienXMargin;
          newShoot <= 1;
        end if;
      when 6 =>
        if column7 > "00000" then
          if column7 > "01111" then
            newRocketYY <= (alienYY + 150);
            rocketXX <= (alienXX+195);
          elsif column7 > "00111" then
            newRocketYY <= (alienYY + 120);
            rocketXX <= (alienXX+195);
          elsif column7 > "00011" then
            newRocketYY <= (alienYY + 90);
            rocketXX <= (alienXX+195);
          elsif column7 > "00001" then
            newRocketYY <= (alienYY + 60);
            rocketXX <= (alienXX+195);
          else
            newRocketYY <= (alienYY + 30);
            rocketXX <= (alienXX+195);
          end if;
        else
          newRocketYY <= 0;
          rocketXX <= alienXMargin;
          newShoot <= 1;
        end if;
      when 7 =>
        if column8 > "00000" then
          if column8 > "01111" then
            newRocketYY <= (alienYY + 150);
            rocketXX <= (alienXX+225);
          elsif column8 > "00111" then
            newRocketYY <= (alienYY + 120);
            rocketXX <= (alienXX+225);
          elsif column8 > "00011" then
            newRocketYY <= (alienYY + 90);
            rocketXX <= (alienXX+225);
          elsif column8 > "00001" then
            newRocketYY <= (alienYY + 60);
            rocketXX <= (alienXX+225);
          else
            newRocketYY <= (alienYY + 30);
            rocketXX <= (alienXX+225);
          end if;
        else
          newRocketYY <= 0;
          rocketXX <= alienXMargin;
          newShoot <= 1;
        end if;
      when 8 =>
        if column9 > "00000" then
          if column9 > "01111" then
            newRocketYY <= (alienYY + 150);
            rocketXX <= (alienXX+255);
          elsif column9 > "00111" then
            newRocketYY <= (alienYY + 120);
            rocketXX <= (alienXX+255);
          elsif column9 > "00011" then
            newRocketYY <= (alienYY + 90);
            rocketXX <= (alienXX+255);
          elsif column9 > "00001" then
            newRocketYY <= (alienYY + 60);
            rocketXX <= (alienXX+255);
          else
            newRocketYY <= (alienYY + 30);
            rocketXX <= (alienXX+255);
          end if;
        else
          newRocketYY <= 0;
          rocketXX <= alienXMargin;
          newShoot <= 1;
        end if;
      when 9 =>
        if column10 > "00000" then
          if column10 > "01111" then
            newRocketYY <= (alienYY + 150);
            rocketXX <= (alienXX+285);
          elsif column10 > "00111" then
            newRocketYY <= (alienYY + 120);
            rocketXX <= (alienXX+285);
          elsif column10 > "00011" then
            newRocketYY <= (alienYY + 90);
            rocketXX <= (alienXX+285);
          elsif column10 > "00001" then
            newRocketYY <= (alienYY + 60);
            rocketXX <= (alienXX+285);
          else
            newRocketYY <= (alienYY + 30);
            rocketXX <= (alienXX+285);
          end if;
        else
          newRocketYY <= 0;
          rocketXX <= alienXMargin;
          newShoot <= 1;
        end if;
    end case;
  end if;
end process;

```

```

        newRocketYY <= (alienYY + 60);
        rocketXX    <= (alienXX+45);
    else
        newRocketYY <= (alienYY + 30);
        rocketXX    <= (alienXX+45);
    end if;
else
    newRocketYY <= 0;
    rocketXX    <= alienXMargin;
    newShoot    <= 1;
end if;
when 2 =>
    if column3 > "00000" then
        if column3 > "01111" then
            newRocketYY <= (alienYY + 150);
            rocketXX    <= (alienXX+75);
        elsif column3 > "00111" then
            newRocketYY <= (alienYY + 120);
            rocketXX    <= (alienXX+75);
        elsif column3 > "00011" then
            newRocketYY <= (alienYY + 90);
            rocketXX    <= (alienXX+75);
        elsif column3 > "00001" then
            newRocketYY <= (alienYY + 60);
            rocketXX    <= (alienXX+75);
        else
            newRocketYY <= (alienYY + 30);
            rocketXX    <= (alienXX+75);
        end if;
    else
        newRocketYY <= 0;
        rocketXX    <= alienXMargin;
        newShoot    <= 1;
    end if;
when 3 =>
    if column4 > "00000" then
        if column4 > "01111" then
            newRocketYY <= (alienYY + 150);
            rocketXX    <= (alienXX+105);
        elsif column4 > "00111" then
            newRocketYY <= (alienYY + 120);
            rocketXX    <= (alienXX+105);
        elsif column4 > "00011" then
            newRocketYY <= (alienYY + 90);
            rocketXX    <= (alienXX+105);
        elsif column4 > "00001" then
            newRocketYY <= (alienYY + 60);
            rocketXX    <= (alienXX+105);
        else
            newRocketYY <= (alienYY + 30);
            rocketXX    <= (alienXX+105);
        end if;
    else
        newRocketYY <= 0;
        rocketXX    <= alienXMargin;
        newShoot    <= 1;
    end if;
when 4 =>
    if column5 > "00000" then
        if column5 > "01111" then
            newRocketYY <= (alienYY + 150);
            rocketXX    <= (alienXX+135);
        elsif column5 > "00111" then
            newRocketYY <= (alienYY + 120);
            rocketXX    <= (alienXX+135);
        elsif column5 > "00011" then
            newRocketYY <= (alienYY + 90);
            rocketXX    <= (alienXX+135);
        elsif column5 > "00001" then
            newRocketYY <= (alienYY + 60);
            rocketXX    <= (alienXX+135);
        else
            newRocketYY <= (alienYY + 30);
            rocketXX    <= (alienXX+135);
        end if;
    else
        newRocketYY <= 0;
        rocketXX    <= alienXMargin;
        newShoot    <= 1;
    end if;
when 5 =>
    if column6 > "00000" then
        if column6 > "01111" then
            newRocketYY <= (alienYY + 150);

```

```

        rocketXX    <= (alienXX+165);
    elseif column6 > "00111" then
        newRocketYY <= (alienYY + 120);
        rocketXX    <= (alienXX+165);
    elseif column6 > "00011" then
        newRocketYY <= (alienYY + 90);
        rocketXX    <= (alienXX+165);
    elseif column6 > "00001" then
        newRocketYY <= (alienYY + 60);
        rocketXX    <= (alienXX+165);
    else
        newRocketYY <= (alienYY + 30);
        rocketXX    <= (alienXX+165);
    end if;
else
    newRocketYY <= 0;
    rocketXX    <= alienXMargin;
    newShoot    <= 1;
end if;
when 6 =>
    if column7 > "00000" then
        if column7 > "01111" then
            newRocketYY <= (alienYY + 150);
            rocketXX    <= (alienXX+195);
        elseif column7 > "00111" then
            newRocketYY <= (alienYY + 120);
            rocketXX    <= (alienXX+195);
        elseif column7 > "00011" then
            newRocketYY <= (alienYY + 90);
            rocketXX    <= (alienXX+195);
        elseif column7 > "00001" then
            newRocketYY <= (alienYY + 60);
            rocketXX    <= (alienXX+195);
        else
            newRocketYY <= (alienYY + 30);
            rocketXX    <= (alienXX+195);
        end if;
    else
        newRocketYY <= 0;
        rocketXX    <= alienXMargin;
        newShoot    <= 1;
    end if;
when 7 =>
    if column8 > "00000" then
        if column8 > "01111" then
            newRocketYY <= (alienYY + 150);
            rocketXX    <= (alienXX+225);
        elseif column8 > "00111" then
            newRocketYY <= (alienYY + 120);
            rocketXX    <= (alienXX+225);
        elseif column8 > "00011" then
            newRocketYY <= (alienYY + 90);
            rocketXX    <= (alienXX+225);
        elseif column8 > "00001" then
            newRocketYY <= (alienYY + 60);
            rocketXX    <= (alienXX+225);
        else
            newRocketYY <= (alienYY + 30);
            rocketXX    <= (alienXX+225);
        end if;
    else
        newRocketYY <= 0;
        rocketXX    <= alienXMargin;
        newShoot    <= 1;
    end if;
when 8 =>
    if column9 > "00000" then
        if column9 > "01111" then
            newRocketYY <= (alienYY + 150);
            rocketXX    <= (alienXX+255);
        elseif column9 > "00111" then
            newRocketYY <= (alienYY + 120);
            rocketXX    <= (alienXX+255);
        elseif column9 > "00011" then
            newRocketYY <= (alienYY + 90);
            rocketXX    <= (alienXX+255);
        elseif column9 > "00001" then
            newRocketYY <= (alienYY + 60);
            rocketXX    <= (alienXX+255);
        else
            newRocketYY <= (alienYY + 30);
            rocketXX    <= (alienXX+255);
        end if;
    else
        newRocketYY <= 0;
        rocketXX    <= alienXMargin;
        newShoot    <= 1;
    end if;
else

```



```

    newRocketYY <= 0;
    rocketXX    <= alienXMargin;
    newShoot    <= 1;
  end if;
  when others =>
    if column10 > "00000" then
      if column10 > "01111" then
        newRocketYY <= (alienYY + 150);
        rocketXX    <= (alienXX+285);
      elsif column10 > "00111" then
        newRocketYY <= (alienYY + 120);
        rocketXX    <= (alienXX+285);
      elsif column10 > "00011" then
        newRocketYY <= (alienYY + 90);
        rocketXX    <= (alienXX+285);
      elsif column10 > "00001" then
        newRocketYY <= (alienYY + 60);
        rocketXX    <= (alienXX+285);
      else
        newRocketYY <= (alienYY + 30);
        rocketXX    <= (alienXX+285);
      end if;
    else
      newRocketYY <= 0;
      rocketXX    <= alienXMargin;
      newShoot    <= 1;
    end if;
  end case;
else
  rocketLaunched <= 0;
  newRocketYY    <= 0;
  rocketXX       <= alienXMargin;
  newShoot       <= 0;
end if;
end process;

-- Update rocketYY.
process(reset, clk, newShoot)
begin
  if reset = '1' or newShoot = 1 then
    rocketYY    <= VLINES;
    rocketFinished <= 1;
  elsif rising_edge(clk) then
    if rocketLaunched = 1 then
      rocketYY    <= newRocketYY;
      rocketFinished <= 0;
    end if;

    if rocketSpeed = missileSpeed then
      rocketSpeed <= 0;
      if rocketYY = VLINES then
        rocketYY    <= VLINES;
        rocketFinished <= 1;
      else
        rocketYY    <= rocketYY + 1;
        rocketFinished <= 0;
      end if;
    else
      rocketSpeed <= rocketSpeed + 1;
    end if;
  end if;
end process;
end architecture;
```

3.2.2 Digital Clock Management

3.2.3 Display

3.2.4 Input

3.2.5 rocketManager

3.2.6 VGA Internal

3.2.7 Top Module

3.2.8 Package
