

SPACE INVADERS

Projet intégré - Système numérique

M A I
16
2017



SAMUEL RIEDO & PASCAL ROULIN

Table des matières

1 Introduction	4
1.1 Super Mario Bros	4
1.2 Gameplay	5
1.3 Déroulement d'une partie	6
2 Architecture	7
2.1 alienRocket	8
2.1.1 Entrées & Sorties	9
2.1.2 Problèmes rencontrés	9
2.2 Digital Clock Management	10
2.2.1 Entrées & Sorties	10
2.2.2 Problèmes rencontrés	10
2.3 Display	11
2.3.1 Entrées & Sorties	12
2.4 Input	13
2.4.1 Entrées & Sorties	13
2.4.2 Testbench	14
2.5 rocketManager	15
2.5.1 Entrées & Sorties	15
2.5.2 Testbench	16
2.6 VGA Internal	17
2.6.1 Entrées & Sorties	17
2.7 Top Module	18
2.7.1 Entrées & Sorties	18
2.8 Space Invaders Package	20

3 Annexes	21
3.1 Conversion d'image	22
3.1.1 Script Matlab - Conversion en fichier COE	23
3.1.2 Script Matlab - Conversion en tableau VHDL	24
3.2 Code source	25
3.2.1 alienRocket	25
3.2.2 Digital Clock Management	30
3.2.3 Display	32
3.2.4 Input	38
3.2.5 rocketManager	40
3.2.6 VGA Internal	41
3.2.7 Top Module	43
3.2.8 Package	46

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 premiers *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 plates-formes, 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 dimensions, 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 lasers vers le haut de l'écran. Il est confronté à plusieurs aliens, se déplaçant aléatoirement dans la partie supérieure de l'écran. Ces derniers tirent 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.

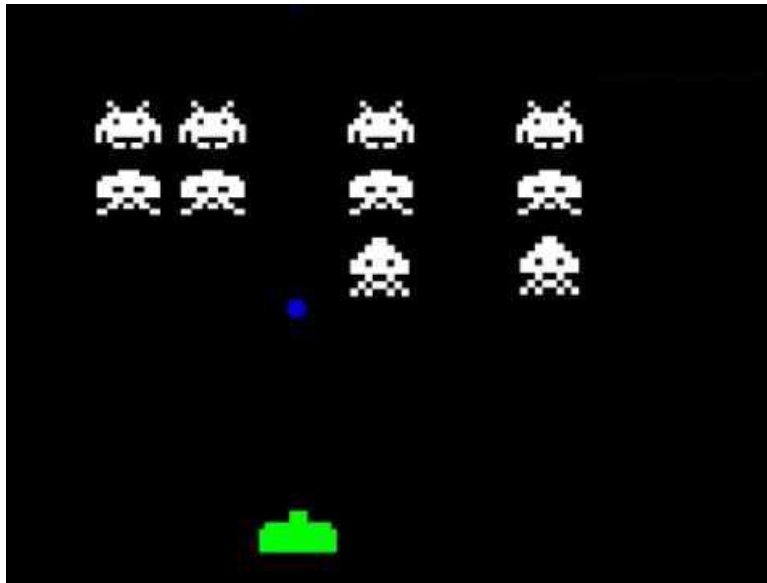


FIG. 1.1 : Space Invaders sur borne arcade

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 dessiner. Au début d'une partie, leur nombre et leur disposition forment une grille rectangulaire complète.

i

Types d'aliens

Bien que les aliens peuvent avoir plusieurs formes différentes (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.

1.3 Déroulement d'une partie



FIG. 1.2 : Ecran d'accueil

Au démarrage du jeu, un écran d'accueil est affiché (figure 1.2). Lorsque le joueur presse sur le bouton central de la croix directionnelle *BtnC*, la partie commence directement. Le vaisseau du joueur est initialement positionné au centre de l'écran sur l'axe *X*, de même que les aliens.

Au moyen des touches gauche *BtnL*, droite *BtnR* et haut *BtnU* de la croix directionnelle, le joueur peut respectivement déplacer son vaisseau à gauche ou à droite, ainsi que tirer des missiles en direction des aliens pour les éliminer. Le jeu reste dans cet état tant que le joueur ne s'est pas fait toucher, ou qu'il y a encore des aliens en vie.

Lorsque la partie se termine, deux fins sont possibles. Dans la première, le joueur a éliminé tous les aliens et un écran "You Win" est affiché, alors que le texte change pour "Game Over" dans la deuxième ou il s'est fait toucher. Le bouton *BtnC*

peut-être utilisé pour relancer une nouvelle partie à ce moment sans revoir l'écran d'accueil. Il est possible, à tout moment, de réinitialiser le jeu avec un reset software en actionnant le switch *SW8*.

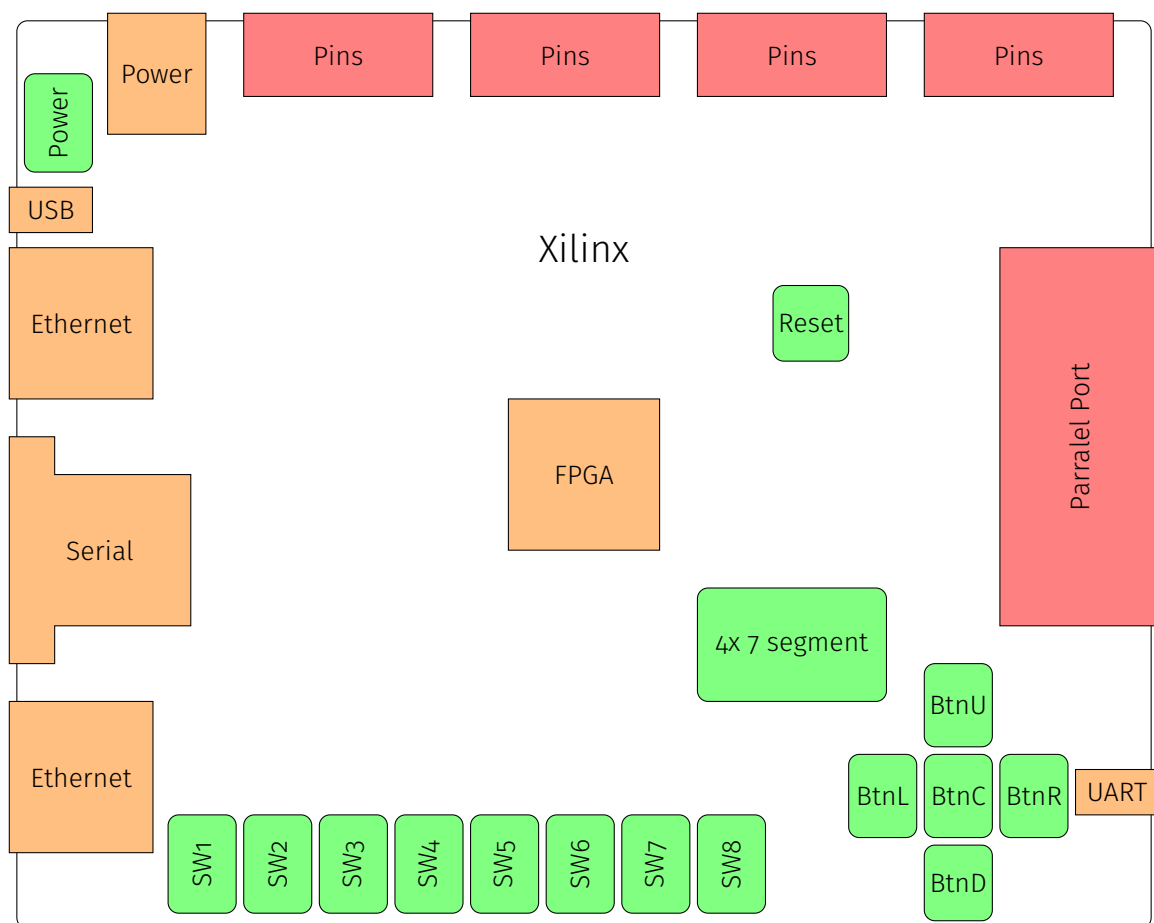


FIG. 1.3 : Digilent NEXYS 3

ARCHITECTURE



La réalisation du projet fut divisée 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 poussés à implémenter le jeu fonction par fonction, puis de tester et déboguer chaque nouvel ajout dès qu'il fut codé. Cette approche présente l'avantage, contrairement à un développement de chaque composant indépendamment les uns des autres, de réduire le risque d'incompatibilité entre deux composants à fin ainsi que le lourd travail de debuggage final. En revanche, cette technique ne permet pas de répartir efficacement le travail dans une équipe constituée de nombreuses personnes.

Bien que nous puissions tester le bon fonctionnement de chaque bloc et fonction 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 testés 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 alien 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 affichée 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 tirée, 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 tirée depuis les aliens 0-4 et 7-9 de la ligne 5 (*alienLine5*), ainsi que depuis les aliens 5 et 6 de la ligne 4 (*alienLine4*).

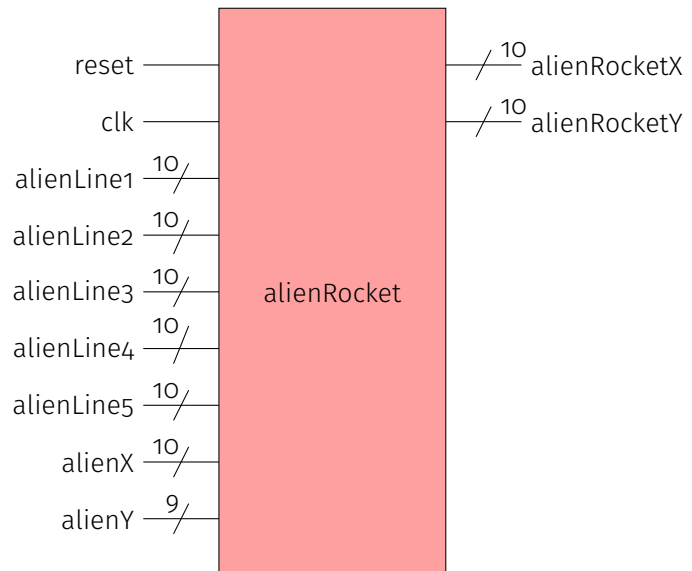


















































FIG. 2.1 : Schéma bloc

	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 supérieure 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 contenus dans *alienLine1*.

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

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

2.1.2 Problèmes rencontrés

```

if shootTimer = rocketFrequency and rocketFinished = 1 then
  rocketLaunched <= 1;
  newShoot <= 0;
  case columnCounter is
    when 0 =>
      if column1 > "00000" then
        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
          -- Rest of the case statement
        end if;
      end case;
    else
      rocketLaunched <= 0;
      newRocketYY <= 0;
      rocketXX <= alienXMargin;
      newShoot <= 0;
    end if;
  end if;

```

— Check which column
— will shoot.
— Check which alien in
— the column will shoot

— Force a new shoot
— as there is no
— alive aliens in
— current column.

Listing 2.1 : Tir de rockets

Les fonctions d'*alienRocket* ont pratiquement toutes fonctionné du premier coup. Néanmoins, il est rapidement apparu que les roquettes tirées par une colonne aléatoire d'aliens suivaient une séquence identique en boucle selon certaines fréquences de tir paramétré dans *SpaceInvadersPackage*. Comme nous n'avons pas pu utiliser une horloge différente pour la gestion de l'aléatoire (voir 2.2.2), la valeur de *rocketFrequency* doit être une valeur impaire afin de résoudre ce problème.

De plus, pour forcer un tir lorsque certaines colonnes ne contiennent plus d'aliens et que l'aléatoire fait qu'elles sont sensées tirer un missile, un nouveau signal *newShoot* est mis à 1 pour forcer un nouveau tir immédiatement sur une autre colonne. Ce processus est répété jusqu'à ce que la colonne sélectionnée aléatoirement contienne au moins un alien et donc soit apte à tirer un missile.

2.2 Digital Clock Management

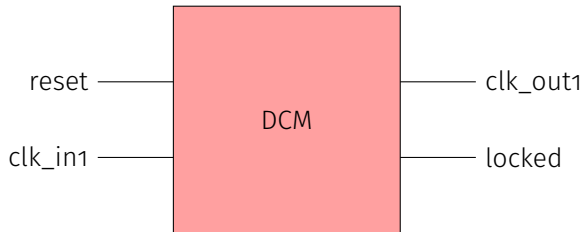


FIG. 2.2 : Schéma bloc

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 tels 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

Toutes les générations d'éléments aléatoires sont basées sur la lecture d'un compteur. Pour éviter que des séquences prédictibles puissent apparaître, nous avons choisi d'utiliser l'horloge à 100MHz de la FPGA en lieu et place de celle à 40MHz générés par le bloc *Digital Clock Management*. Néanmoins, lorsque nous avons affecté le signal *fpga_clk* à l'un de nos composants (excepté le bloc *DCM*), le code n'était plus synthétisable.

Pour contourner cet étrange problème, nous avons tenté d'ajouter une sortie au composant *DCM* se contentant de copier l'horloge d'entrée sur une deuxième horloge de sortie. Pour des raisons inconnues, le problème a persisté et le code n'était à nouveau plus synthétisable. De ce fait, la génération d'éléments aléatoire a, à partir de ce moment, été générée par des compteurs utilisant l'horloge de 40MHz, mais étant réinitialisé à leur valeur maximale modulo *shipPosition*. Ce dernier signal étant la position entre le bord de l'écran et le vaisseau du joueur, il n'est pas possible de prédire sa valeur et cela permet d'éviter que l'aléatoire du jeu soit biaisé.

```

signal counter : integer range 0 to counterMaxValue := 0;

process
begin
  if reset = '1' then
    counter <= counterMaxValue mod shipPosition;
  elsif rising_edge(clk) then
    if counter = counterMaxValue then
      counter <= counterMaxValue mod shipPosition;
    else
      counter <= counter + 1;
    end if;
  end if;
end process;

-- Later in the code
case counter is
  when 0 =>
    ...

```

Listing 2.2 : Génération d'aléatoire

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 bleue 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 coordonnée 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 une défaite du joueur) ainsi que la gestion des collisions entre les roquettes et les aliens ou le joueur. Le choix d'avoir fait ses tests à l'intérieur de *Display* plutôt que de créer un nouveau bloc spécifique à ce cas fut dirigé par le fait que tous les signaux nécessaires à ces deux tests étaient déjà présents dans ce bloc, et qu'il aurait été redondant de les ajouter ailleurs.

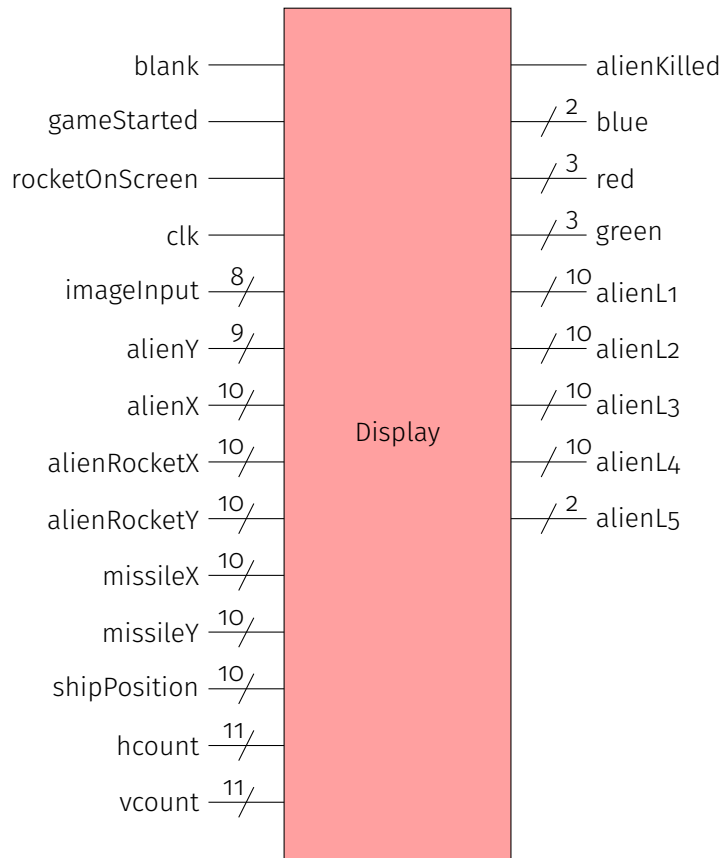


FIG. 2.3 : Schéma bloc

```

— 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(alienRockety)) < 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 2.3 : Détection de la mort du joueur.

Lorsque la partie est en cours, les aliens sont affichés 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).

Ci-contre se trouve le code permettant de détecter la collision entre un tir alien et le spationef. Ce dernier a une hauteur de 30 pixels, pour une largeur de 62 pixels. De ce fait, lorsqu'une roquette a sa position en Y entre 570 et 600 et que sa position en X est entre la position du vaisseau et cette dernière +56, le joueur s'est fait toucher.

Le range X du vaisseau est calculer entre *shipPosition + 6* et *shipPosition + 56* pour supprimer de la détection les 6 pixels noirs présent de chaque côté du spationef.

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 nul.
- gameStarted** Indique par une valeur à 1 que le jeu a débuté.
- rocketOnScreen** Indique par une valeur à 1 qu'une roquet 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 roquet tirée par les aliens.
- alienRocketY** Nombre de pixels entre le le haut de l'écran et le haut de la rockette tirée par les aliens.
- missileX** Nombre de pixels entre le bord gauche de l'écran et la roquet lancée par les aliens.
- missileY** Nombre de pixels entre le haut de l'écran et le haut de la rockette 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 rockette 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.

2.4 Input

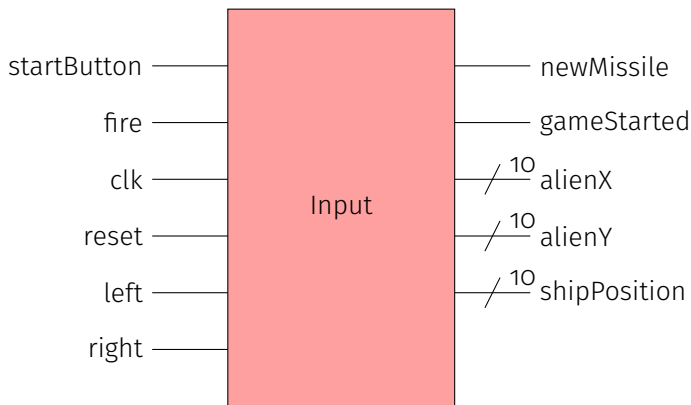


FIG. 2.4 : Schéma bloc

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 maximale 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;

```

Listing 2.4 : Déplacement des aliens

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 a débuté.

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

alienY Nombre de pixels entre le haut de l'écran et le bord supérieur des aliens contenus 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 utilisée pour générer de l'aléatoire.

2.4.2 Testbench

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 signalé au bloc *Display* afin que celui-ci sache qu'un missile doit être affiché.

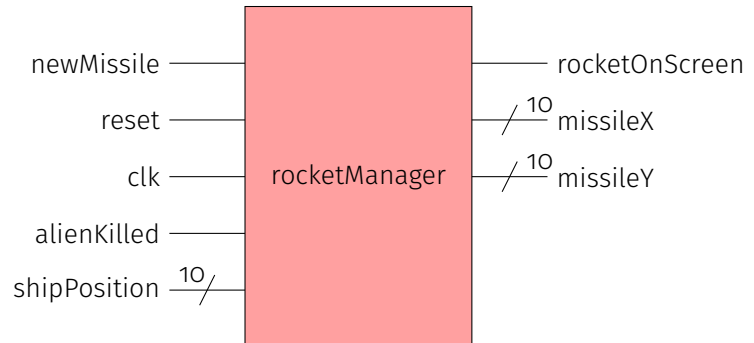


FIG. 2.5 : Schéma bloc

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 horizontal 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 décrémentée 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 roquette 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 roquette doit être affichée à l'écran.

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

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

2.5.2 Testbench

2.6 VGA Internal

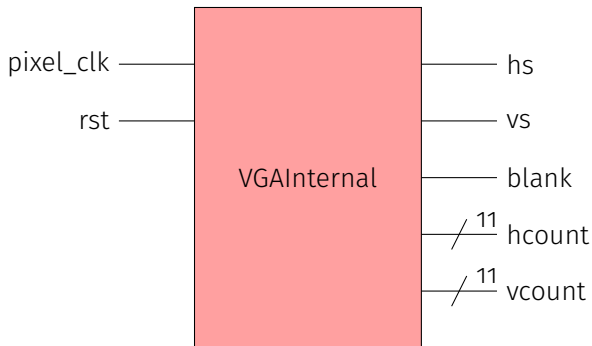


FIG. 2.6 : Schéma bloc

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 nul.

hcount Coordonnée X du balayage.

vcount Coordonnée Y du balayage.

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.

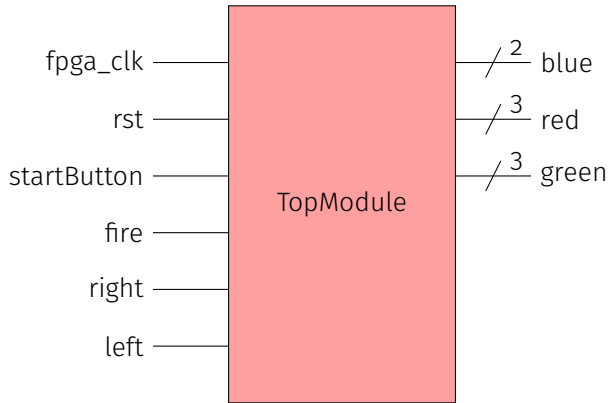


FIG. 2.7 : Schéma bloc

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 commencé 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 cherché 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 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.

2.8 Space Invaders Package

Généralement, il est recommandé de programmer en utilisant des variables et non des *Magic Numbers* dans les langages de programmation dit “haut-niveau”. Les performances du code ne seront que peu, voir pas du tout impacté selon la qualité du compilateur. En revanche, le programme sera plus compréhensible dans le cas d’un débogage ou d’un changement de programmeur.

Autre avantage, modifier une variable à un seul endroit fera le changement dans l’ensemble du code sans qu’il n’y est des risques d’erreurs humaines, tel que modifier le mauvais nombre car deux *Magic Numbers* différents avaient la même valeur.

En VHDL, il est également recommandé de programmer selon ce principe. Néanmoins, un composant VHDL ne peut transmettre d’information à un autre sans qu’un signal contenant les données à partager soit implémenté entre les deux blocs. Lorsqu’un programme contient de nombreuses variables dit génériques, ces dernières peuvent rapidement rendre le design schématique du projet incompréhensible en ajoutant énormément de connexion entre les blocs.

Une solution à cette problématique est l’utilisation de *package*. Un *package* peut être lu et modifié par n’importe quel composant l’ayant importé.

```
library work;
use work.SpaceInvadersPackage.all;
```

Listing 2.5 : Importation d’un package

Par défaut, un package est contenu dans la librairie *work*, qui doit donc également être importée. Notre jeu est codé en utilisant au maximum des signaux contenus dans un package, qui peuvent être modifiés pour changeant certain paramètres du jeu.

```
— Inputs
constant fireSpeed      : integer      := 600000;
constant shipSpeed      : integer      := 100000;
constant alienSpeed     : integer      := 6000000;
constant maxShipPosValue : integer      := 736; — must be pair
constant shipMargin     : integer      := 50; — minimum space between side screen and ship
constant alienXMargin   : integer      := 50; — minimum space between side screen and aliens
constant alienYUpMargin : integer      := 50; — minimum space between top screen and aliens
constant alienYDownMargin : integer    := 200; — maximum space between top screen and aliens
constant maxAlienJump   : integer      := 10; — max pixel
aliens can shift in a single time

— rocketManager
constant missileSpeed    : integer      := 60000; — missile speed
constant rocketLength    : integer      := 10; — rocket length in pixel
constant rocketColor     : std_logic_vector(7 downto 0) := "11111111";
```

Listing 2.6 : Signaux utilisés par *Input* & *rocketManager*

Admettons que l’on souhaite modifier la couleur des roquette pour les rendre plus visibles. Il suffit de changer la valeur de *rocketColor* pour que ce changement soit effectué dans tous les composants nécessaires.

ANNEXES

3

3.1 Conversion d'image

Le bloc *Display* affiche des données à l'écran en affectant une valeur aux trois signaux *red*, *green* et *blue*. Les deux premiers sont contenus sur trois bits, alors que le dernier est uniquement sur deux. Cela signifie que le jeu 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 apparaît alors que pour afficher une image, il faut extraire le code couleur de chacun de ses pixels, puis les stocker d'une des deux façons 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 décimal.

- 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 stockée 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 *sdl_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

Seules les images au format "JPG" sont supportées 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:  Digilent 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 <= alienLines(3) & alienLine4(3) & alienLine3(3) & alienLine2(3) & alienLine1(3);
column5 <= alienLines(4) & alienLine4(4) & alienLine3(4) & alienLine2(4) & alienLine1(4);
column6 <= alienLines(5) & alienLine4(5) & alienLine3(5) & alienLine2(5) & alienLine1(5);
column7 <= alienLines(6) & alienLine4(6) & alienLine3(6) & alienLine2(6) & alienLine1(6);
column8 <= alienLines(7) & alienLine4(7) & alienLine3(7) & alienLine2(7) & alienLine1(7);
column9 <= alienLines(8) & alienLine4(8) & alienLine3(8) & alienLine2(8) & alienLine1(8);
column10 <= alienLines(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 -- Force a new shoot
          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);
          end if;
        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

```

— Company:      HES-SO
— Engineer:     Samuel Riedo & Pascal Roulin
— Create Date:  3/03/2017
— Design Name:  DCM.vhd
— Project Name: Space Invaders — FPGA Edition
— Target Devices:  Digilent NEXYS 3 (Xilinx Spartan 6 XC6LX16—CS324)
— Description:   Create a 40MHz clock with a 100MHz clock
— Revision 0.01 — File Created
—              1.00 — Implemented with IP Core

```

```

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

library unisim;
use unisim.vcomponents.all;

entity DCM is
  port(
    — Clock in ports
    CLK_IN1 : in std_logic;
    — Clock out ports
    CLK_OUT1 : out std_logic;
    — Status and control signals
    RESET : in std_logic;
    LOCKED : out std_logic
  );
end DCM;

architecture xilinx of DCM is
  attribute CORE_GENERATION_INFO : string;
  attribute CORE_GENERATION_INFO of xilinx : architecture is "DCM,clk_wiz_v3_6,{component_name=DCM,
    use_phase_alignment=true,use_min_o_jitter=false,use_max_i_jitter=false,use_dyn_phase_shift=false,
    use_inclk_switchover=false,use_dyn_reconfig=false,feedback_source=FDBK_AUTO,prmttype_sel=DCM_SP,
    num_out_clk=1,clk_in1_period=10.0,clk_in2_period=10.0,use_power_down=false,use_reset=true,use_locked=true,
    use_inclk_stopped=false,use_status=false,use_freeze=false,use_clk_valid=false,feedback_type=SINGLE,
    clock_mgr_type=AUTO>manual_override=false}";
  — Input clock buffering / unused connectors
  signal clk_in1 : std_logic;
  — Output clock buffering
  signal clkfb : std_logic;
  signal clko : std_logic;
  signal clkfx : std_logic;
  signal clkfbout : std_logic;
  signal locked_internal : std_logic;
  signal status_internal : std_logic_vector(7 downto 0);
begin

  — Input buffering
  clk_in1_buf : IBUFG
  port map
  (O => clk_in1,
   I => CLK_IN1);

  — Clocking primitive
  — Instantiation of the DCM primitive
  — * Unused inputs are tied off
  — * Unused outputs are labeled unused
  dcm_sp_inst : DCM_SP
  generic map
  (CLKDV_DIVIDE => 2.500,
   CLKFX_DIVIDE => 5,
   CLKFX_MULTIPLY => 2,
   CLKIN_DIVIDE_BY_2 => false,
   CLKIN_PERIOD => 10.0,
   CLKOUT_PHASE_SHIFT => "NONE",
   CLK_FEEDBACK => "1X",
   DESKEW_ADJUST => "SYSTEM_SYNCHRONOUS",
   PHASE_SHIFT => 0,
   STARTUP_WAIT => false)
  port map
  — Input clock
  (CLKIN => clk_in1,
   CLKFB => clkfb,
   — Output clocks
   CLKO => clko,

```

```

CLK90    => open,
CLK180   => open,
CLK270   => open,
CLK2X    => open,
CLK2X180 => open,
CLKFX    => clkfx,
CLKFX180 => open,
CLKDV    => open,
-- Ports for dynamic phase shift
PSCLK    => '0',
PSEN     => '0',
PSINCDEC => '0',
PSDONE   => open,
-- Other control and status signals
LOCKED   => locked_internal,
STATUS   => status_internal,
RST      => RESET,
-- Unused pin, tie low
DSSSEN   => '0');

LOCKED <= locked_internal;

-- Output buffering
clkf_buf : BUFG
port map
(O => clkfb,
 I => clko);

clkout1_buf : BUFG
port map
(O => CLK_OUT1,
 I => clkfx);
end xilinx;
```

3.2.3 Display

```

— Company:      HES-SO
— Engineer:     Samuel Riedo & Pascal Roulin
— Create Date:  09:20:02 03/02/2017
— Design Name:  Display.vhd
— Project Name:  Space Invaders — FPGA Edition
— Target Devices: Digilent NEXYS 3 (Xilinx Spartan 6 XC6LX16-CS324)
— Description:   Display pixel at vga coordinates using ROM data and package tables
— Revision 0.01 — File Created
—      1.00 — First fonctionnal version, display a ship at the bottom of the screen
—      1.1  — Ship can be moved using arrows buttons, display start screen
—             before playing
—      1.2  — Display moving aliens
—      1.3  — Display ship rockets
—      1.4  — Aliens can be killed
—      1.5  — Display "game over" or "you win" screen
—             Ship can be killed by aliens
—             Display aliens rockets

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

library work;
use work.SpaceInvadersPackage.all;

entity Display is
  port(
    blank          : in  std_logic;           — If 1, video output must be null
    gameStarted    : in  std_logic;           — When 0, show start screen
    rocketOnScreen : in  std_logic;           — If 1, display a rocket
    clk            : in  std_logic;           — 40MHz
    imageInput     : in  std_logic_vector(7 downto 0); — data from rom
    alienY         : in  std_logic_vector(8 downto 0); — first alien position from top screen
    alienX         : in  std_logic_vector(9 downto 0); — first alien position from left screen
    alienRocketx   : in  std_logic_vector(9 downto 0); — Alien rocket x position
    alienRockety   : in  std_logic_vector(9 downto 0); — Alien rocket y position
    missileX       : in  std_logic_vector(9 downto 0); — Missile x coordinate
    missileY       : in  std_logic_vector(9 downto 0); — Pixels between top screen and top missile position
    shipPosition   : in  std_logic_vector(9 downto 0); — Ship x coordinate
    hcount         : in  std_logic_vector(10 downto 0); — Pixel x coordinate
    vcount         : in  std_logic_vector(10 downto 0); — Pixel y coordinate
    alienKilled    : out std_logic;           — 1 if alien killed
    blue           : out std_logic_vector(1 downto 0); — Blue color output
    red            : out std_logic_vector(2 downto 0); — Red color output
    green          : out std_logic_vector(2 downto 0); — Green color output
    alienL1        : out std_logic_vector(0 to 9);    — Same value as alienLine1
    alienL2        : out std_logic_vector(0 to 9);    — Same value as alienLine2
    alienL3        : out std_logic_vector(0 to 9);    — Same value as alienLine3
    alienL4        : out std_logic_vector(0 to 9);    — Same value as alienLine4
    alienL5        : out std_logic_vector(0 to 9);    — Same value as alienLine5
  );
end entity Display;

architecture logic of Display is
  signal color          : std_logic_vector(7 downto 0); — Color output
  signal hcounter       : integer range 0 to 2047;      — Integer value of hcount
  signal vcounter       : integer range 0 to 2047;      — Integer value of vcount
  signal shipPos        : integer range 0 to maxShipPosValue; — x position of the ship
  signal alienXX        : integer range 0 to 1000;     — Integer value of alienX
  signal alienYY        : integer range 0 to 1000;     — Integer value of alienY
  signal missileYY      : integer range 0 to 1023;     — Integer value of missileY
  signal alienLine      : integer range 0 to 4         := 0; — Current alien line displayed
  signal alienIndex     : integer range 0 to 9         := 0; — Current alien in the displayed line
  signal gameWin        : std_logic                   := '0'; — If 1, game win
  signal gameOver       : std_logic                   := '0'; — If 1, game loose
  signal missileXX      : integer range shipMargin to (HLINES-shipMargin) := shipMargin; — Missile X value from left screen

  signal alienLine1 : std_logic_vector(0 to 9) := "111111111"; — 1 bit by alien; 1=alien alive, 0=dead alien
  signal alienLine2 : std_logic_vector(0 to 9) := "111111111"; — 1 bit by alien; 1=alien alive, 0=dead alien
  signal alienLine3 : std_logic_vector(0 to 9) := "111111111"; — 1 bit by alien; 1=alien alive, 0=dead alien
  signal alienLine4 : std_logic_vector(0 to 9) := "111111111"; — 1 bit by alien; 1=alien alive, 0=dead alien
  signal alienLine5 : std_logic_vector(0 to 9) := "111111111"; — 1 bit by alien; 1=alien alive, 0=dead alien

  signal touched : integer range 0 to 1 := 0; — if 1, an alien is killed

  — Integer value of alienLine 1–5
  signal ali1 : integer range 0 to 1023 := 1023;
  signal ali2 : integer range 0 to 1023 := 1023;
  signal ali3 : integer range 0 to 1023 := 1023;

```



```

signal ali4 : integer range 0 to 1023 := 1023;
signal ali5 : integer range 0 to 1023 := 1023;

-- Temp signals used for alienLine computation
signal temp : integer range 0 to 1023 := 0;
signal temp2 : integer range 0 to 1023 := 0;
signal temp3 : integer range 0 to 1023 := 0;

begin

gameWin <= '1' when (alienLine1 = "000000000" and alienLine2 = "000000000"
                    and alienLine3 = "000000000" and alienLine4 = "000000000"
                    and alienLine5 = "000000000")
                else '0';

alienL1 <= alienLine1;
alienL2 <= alienLine2;
alienL3 <= alienLine3;
alienL4 <= alienLine4;
alienL5 <= alienLine5;

hcounter <= to_integer(unsigned(hcount));
vcounter <= to_integer(unsigned(vcount));
shipPos <= to_integer(unsigned(shipPosition));
alienXX <= to_integer(unsigned(alienX));
alienYY <= to_integer(unsigned(alienY));
missileYY <= to_integer(unsigned(missileY));
missileXX <= to_integer(unsigned(missileX));

alienLine1 <= std_logic_vector(to_unsigned(ali1, 10));
alienLine2 <= std_logic_vector(to_unsigned(ali2, 10));
alienLine3 <= std_logic_vector(to_unsigned(ali3, 10));
alienLine4 <= std_logic_vector(to_unsigned(ali4, 10));
alienLine5 <= std_logic_vector(to_unsigned(ali5, 10));

alienIndex <= (((hcounter-alienXX) / 30) mod 10) when (hcounter-alienXX) >= 0 else 0;

-- Can't do the following operations in a single line without warnings, so we did it
-- with temp signals.
temp <= (vcounter-alienYY) when (vcounter-alienYY) >= 0 else 0;
temp2 <= temp / 30;
temp3 <= temp2 mod 5;
alienLine <= temp3;

-- Outputs must be 0 is blank = 0, this happen
-- when hcount and vcount are higher than 800x600.
red <= color(7 downto 5) when blank = '0' else "000";
green <= color(4 downto 2) when blank = '0' else "000";
blue <= color(1 downto 0) when blank = '0' else "00";

-- Main display process
process(hcounter, vcounter, shipPos, gameStarted, ImageInput, alienXX, alienYY, rocketOnScreen, missileYY,
        missileXX, alienLine1, alienLine2, alienLine3, alienLine4, alienLine5, alienLine, alienIndex,
        alienrocketx, alienrockety, gameWin, gameOver)
begin
    -- Show home screen
    if gameStarted = '0' then
        color <= ImageInput;
    -- Show "you win" screen
    elsif gameWin = '1' then
        if hcounter >= 250 and hcounter < 550 and vcounter >= 273 and vcounter < 327 then
            color <= std_logic_vector(to_unsigned(win(hcounter-250, vcounter-273), 8));
        else
            color <= "00000000";
        end if;
    -- Show "game over" screen
    elsif gameOver = '1' then
        if hcounter >= 310 and hcounter < 490 and vcounter >= 245 and vcounter < 354 then
            color <= std_logic_vector(to_unsigned(gameOverTable(hcounter-310, vcounter-245), 8));
        else
            color <= "00000000";
        end if;
    -- Show the game
    else
        -- Display the ship
        if hcounter >= shipPos and hcounter < (shipPos+62) and vcounter > 570 then
            color <= std_logic_vector(to_unsigned(ship((hcounter-shipPos), (vcounter-570)), 8));
        else
            color <= "00000000";
        end if;

        -- Display aliens

```

```

    if hcounter >= alienXX and hcounter < (alienXX + 300) and vcounter >= alienYY and vcounter < (alienYY + 150)
    then
        case alienLine is
            when 0 =>
                if alienLine1(alienIndex) = '1' then
                    color <= std_logic_vector(to_unsigned(blueAlien(((hcounter-alienXX) mod 30), ((vcounter-alienYY)mod
30)), 8));
                else
                    color <= "00000000";
                end if;
            when 1 =>
                if alienLine2(alienIndex) = '1' then
                    color <= std_logic_vector(to_unsigned(DarkBlueAlien(((hcounter-alienXX) mod 30), ((vcounter-alienYY)
mod 30)), 8));
                else
                    color <= "00000000";
                end if;
            when 2 =>
                if alienLine3(alienIndex) = '1' then
                    color <= std_logic_vector(to_unsigned(greenAlien(((hcounter-alienXX) mod 30), ((vcounter-alienYY)mod
30)), 8));
                else
                    color <= "00000000";
                end if;
            when 3 =>
                if alienLine4(alienIndex) = '1' then
                    color <= std_logic_vector(to_unsigned(yellowAlien(((hcounter-alienXX) mod 30), ((vcounter-alienYY)mod
30)), 8));
                else
                    color <= "00000000";
                end if;
            when others =>
                if alienLine5(alienIndex) = '1' then
                    color <= std_logic_vector(to_unsigned(purpleAlien(((hcounter-alienXX) mod 30), ((vcounter-alienYY)mod
30)), 8));
                else
                    color <= "00000000";
                end if;
            end case;
        end if;

        -- Ship missile
        if rocketOnScreen = '1' then
            if hcounter = missileXX and vcounter > missileYY and vcounter < (missileYY+rocketLength) then
                color <= rocketColor;
            end if;
        end if;

        -- Alien missile
        if hcounter = to_integer(unsigned(alienRocketx)) and vcounter > to_integer(unsigned(alienRockety))
            and vcounter < (to_integer(unsigned(alienRockety))+rocketLength) then
            color <= rocketColor;
        end if;
    end if;
end process;

-- When a rocket killed an alien, stop display the current rocket
process(touched)
begin
    if touched = 1 then
        alienKilled <= '1';
    else
        alienKilled <= '0';
    end if;
end process;

-- Alien rocket collision
process(gameStarted, clk)
begin
    if gameStarted = '0' then
        ali1 <= 1023;
        ali2 <= 1023;
        ali3 <= 1023;
        ali4 <= 1023;
        ali5 <= 1023;
        touched <= 0;
    elsif rising_edge(clk) then
        if missileYY >= alienYY and missileYY < (alienYY+150) and missileXX >= alienXX and missileXX < (alienXX+300)
        then

            -- last line
            if ((missileYY-alienYY)/30) = 4 and touched = 0 then

                if alienLine5(0) = '1' and ((missileXX-alienXX)/30) = 0 then

```

```

    ali5    <= ali5 - 512;
    touched <= 1;
  elseif alienLine5(1) = '1' and ((missileXX-alienXX)/30) = 1 then
    ali5    <= ali5 - 256;
    touched <= 1;
  elseif alienLine5(2) = '1' and ((missileXX-alienXX)/30) = 2 then
    ali5    <= ali5 - 128;
    touched <= 1;
  elseif alienLine5(3) = '1' and ((missileXX-alienXX)/30) = 3 then
    ali5    <= ali5 - 64;
    touched <= 1;
  elseif alienLine5(4) = '1' and ((missileXX-alienXX)/30) = 4 then
    ali5    <= ali5 - 32;
    touched <= 1;
  elseif alienLine5(5) = '1' and ((missileXX-alienXX)/30) = 5 then
    ali5    <= ali5 - 16;
    touched <= 1;
  elseif alienLine5(6) = '1' and ((missileXX-alienXX)/30) = 6 then
    ali5    <= ali5 - 8;
    touched <= 1;
  elseif alienLine5(7) = '1' and ((missileXX-alienXX)/30) = 7 then
    ali5    <= ali5 - 4;
    touched <= 1;
  elseif alienLine5(8) = '1' and ((missileXX-alienXX)/30) = 8 then
    ali5    <= ali5 - 2;
    touched <= 1;
  elseif alienLine5(9) = '1' and ((missileXX-alienXX)/30) = 9 then
    ali5    <= ali5 - 1;
    touched <= 1;
  end if;

— line 4
elseif ((missileYY-alienYY)/30) = 3 and touched = 0 then

  if alienLine4(0) = '1' and ((missileXX-alienXX)/30) = 0 then
    ali4    <= ali4 - 512;
    touched <= 1;
  elseif alienLine4(1) = '1' and ((missileXX-alienXX)/30) = 1 then
    ali4    <= ali4 - 256;
    touched <= 1;
  elseif alienLine4(2) = '1' and ((missileXX-alienXX)/30) = 2 then
    ali4    <= ali4 - 128;
    touched <= 1;
  elseif alienLine4(3) = '1' and ((missileXX-alienXX)/30) = 3 then
    ali4    <= ali4 - 64;
    touched <= 1;
  elseif alienLine4(4) = '1' and ((missileXX-alienXX)/30) = 4 then
    ali4    <= ali4 - 32;
    touched <= 1;
  elseif alienLine4(5) = '1' and ((missileXX-alienXX)/30) = 5 then
    ali4    <= ali4 - 16;
    touched <= 1;
  elseif alienLine4(6) = '1' and ((missileXX-alienXX)/30) = 6 then
    ali4    <= ali4 - 8;
    touched <= 1;
  elseif alienLine4(7) = '1' and ((missileXX-alienXX)/30) = 7 then
    ali4    <= ali4 - 4;
    touched <= 1;
  elseif alienLine4(8) = '1' and ((missileXX-alienXX)/30) = 8 then
    ali4    <= ali4 - 2;
    touched <= 1;
  elseif alienLine4(9) = '1' and ((missileXX-alienXX)/30) = 9 then
    ali4    <= ali4 - 1;
    touched <= 1;
  end if;

— line 3
elseif ((missileYY-alienYY)/30) = 2 and touched = 0 then

  if alienLine3(0) = '1' and ((missileXX-alienXX)/30) = 0 then
    ali3    <= ali3 - 512;
    touched <= 1;
  elseif alienLine3(1) = '1' and ((missileXX-alienXX)/30) = 1 then
    ali3    <= ali3 - 256;
    touched <= 1;
  elseif alienLine3(2) = '1' and ((missileXX-alienXX)/30) = 2 then
    ali3    <= ali3 - 128;
    touched <= 1;
  elseif alienLine3(3) = '1' and ((missileXX-alienXX)/30) = 3 then
    ali3    <= ali3 - 64;
    touched <= 1;
  elseif alienLine3(4) = '1' and ((missileXX-alienXX)/30) = 4 then
    ali3    <= ali3 - 32;
    touched <= 1;
  elseif alienLine3(5) = '1' and ((missileXX-alienXX)/30) = 5 then

```

```

    ali3    <= ali3 - 16;
    touched <= 1;
  elsif alienLine3(6) = '1' and ((missileXX-alienXX)/30) = 6 then
    ali3    <= ali3 - 8;
    touched <= 1;
  elsif alienLine3(7) = '1' and ((missileXX-alienXX)/30) = 7 then
    ali3    <= ali3 - 4;
    touched <= 1;
  elsif alienLine3(8) = '1' and ((missileXX-alienXX)/30) = 8 then
    ali3    <= ali3 - 2;
    touched <= 1;
  elsif alienLine3(9) = '1' and ((missileXX-alienXX)/30) = 9 then
    ali3    <= ali3 - 1;
    touched <= 1;
  end if;

— line 2
elsif ((missileYY-alienYY)/30) = 1 and touched = 0 then

  if alienLine2(0) = '1' and ((missileXX-alienXX)/30) = 0 then
    ali2    <= ali2 - 512;
    touched <= 1;
  elsif alienLine2(1) = '1' and ((missileXX-alienXX)/30) = 1 then
    ali2    <= ali2 - 256;
    touched <= 1;
  elsif alienLine2(2) = '1' and ((missileXX-alienXX)/30) = 2 then
    ali2    <= ali2 - 128;
    touched <= 1;
  elsif alienLine2(3) = '1' and ((missileXX-alienXX)/30) = 3 then
    ali2    <= ali2 - 64;
    touched <= 1;
  elsif alienLine2(4) = '1' and ((missileXX-alienXX)/30) = 4 then
    ali2    <= ali2 - 32;
    touched <= 1;
  elsif alienLine2(5) = '1' and ((missileXX-alienXX)/30) = 5 then
    ali2    <= ali2 - 16;
    touched <= 1;
  elsif alienLine2(6) = '1' and ((missileXX-alienXX)/30) = 6 then
    ali2    <= ali2 - 8;
    touched <= 1;
  elsif alienLine2(7) = '1' and ((missileXX-alienXX)/30) = 7 then
    ali2    <= ali2 - 4;
    touched <= 1;
  elsif alienLine2(8) = '1' and ((missileXX-alienXX)/30) = 8 then
    ali2    <= ali2 - 2;
    touched <= 1;
  elsif alienLine2(9) = '1' and ((missileXX-alienXX)/30) = 9 then
    ali2    <= ali2 - 1;
    touched <= 1;
  end if;

— line 1
elsif ((missileYY-alienYY)/30) = 0 and touched = 0 then

  if alienLine1(0) = '1' and ((missileXX-alienXX)/30) = 0 then
    ali1    <= ali1 - 512;
    touched <= 1;
  elsif alienLine1(1) = '1' and ((missileXX-alienXX)/30) = 1 then
    ali1    <= ali1 - 256;
    touched <= 1;
  elsif alienLine1(2) = '1' and ((missileXX-alienXX)/30) = 2 then
    ali1    <= ali1 - 128;
    touched <= 1;
  elsif alienLine1(3) = '1' and ((missileXX-alienXX)/30) = 3 then
    ali1    <= ali1 - 64;
    touched <= 1;
  elsif alienLine1(4) = '1' and ((missileXX-alienXX)/30) = 4 then
    ali1    <= ali1 - 32;
    touched <= 1;
  elsif alienLine1(5) = '1' and ((missileXX-alienXX)/30) = 5 then
    ali1    <= ali1 - 16;
    touched <= 1;
  elsif alienLine1(6) = '1' and ((missileXX-alienXX)/30) = 6 then
    ali1    <= ali1 - 8;
    touched <= 1;
  elsif alienLine1(7) = '1' and ((missileXX-alienXX)/30) = 7 then
    ali1    <= ali1 - 4;
    touched <= 1;
  elsif alienLine1(8) = '1' and ((missileXX-alienXX)/30) = 8 then
    ali1    <= ali1 - 2;
    touched <= 1;
  elsif alienLine1(9) = '1' and ((missileXX-alienXX)/30) = 9 then
    ali1    <= ali1 - 1;
    touched <= 1;
  end if;

```

```

        end if;
    else
        touched <= 0;
    end if;
end if;
end process;

-- 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(alienRockety)) < 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;
end architecture;
```

3.2.4 Input

```

— Company:      HES-SO
— Engineer:     Samuel Riedo & Pascal Roulin
— Create Date:  13/04/2017
— Design Name:  Input.vhd
— Project Name: Space Invaders — FPGA Edition
— Target Devices: Digilent NEXYS 3 (Xilinx Spartan 6 XC6LX16—CS324)
— Description:   Slow inputs
— Revision 0.01 — File Created
—               1.00 — Fire, left and write implemented
—               1.1  — Ship and aliens movements implemented
—               1.2  — New buttons for skipping start screen: startButton

```

```

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

library work;
use work.SpaceInvadersPackage.all;

entity Input is
  port(
    startButton : in std_logic;           — When 1, start game
    fire        : in std_logic;           — When 1, shoot a rocket
    clk         : in std_logic;           — 40MHz
    reset       : in std_logic;           — Active high
    left        : in std_logic;           — Left arrow button
    right       : in std_logic;           — Right arrow button
    newMissile   : out std_logic;          — If 1, new missile launched
    gameStarted : out std_logic;          — When 0, show start screen
    alienX       : out std_logic_vector(9 downto 0); — first alien position from left screen
    alienY       : out std_logic_vector(8 downto 0); — first alien position from top screen
    shipPosition : out std_logic_vector(9 downto 0); — Ship x coordinate
  );
end Input;

architecture Behavioral of Input is

  signal start          : integer range 0 to 1           := 0; — Integer of gameStarted
  signal fireTimer      : integer range 0 to fireSpeed   := 0; — Slow fire rate
  signal shipTimer      : integer range 0 to shipSpeed   := 0; — Slow ship speed
  signal alienTimer     : integer range 0 to alienSpeed   := 0; — Slow alien speed
  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; — Alien pixels mouvement value
  signal alienXX        : integer range alienXMargin to (500—alienXMargin) := 250; — alienX in integer
  signal alienYY        : integer range alienYUpMargin to alienYDownMargin := 100; — alienY in integer
  signal shipPos        : integer range 0 to 737         := (maxShipPosValue/2); — Ship X
  — position
  — from left
  — screen

begin

  — Update outputs according to their integer equivalent
  gameStarted <= '1' when start = 1 else '0';
  shipPosition <= std_logic_vector(to_unsigned(shipPos, 10));
  alienX <= std_logic_vector(to_unsigned(alienXX, 10));
  alienY <= std_logic_vector(to_unsigned(alienYY, 9));

  process(reset, clk)
  begin
    if reset = '1' then
      alienYY <= 100;
      alienXX <= 250;
      shipPos <= (maxShipPosValue/2);
      alienTimer <= 0;
      shipTimer <= 0;
      fireTimer <= 0;
      start <= 0;
      newMissile <= '0';
    elsif rising_edge(clk) then
      — fire
      if fireTimer >= fireSpeed then
        fireTimer <= 0;
        if startButton = '1' then
          start <= 1;
        end if;
        if fire = '1' and start = 1 then
          newMissile <= '1';
        else
          newMissile <= '0';
        end if;
      end if;
    end if;
  end process;

```

```

end if;
else
  fireTimer <= fireTimer + 1;
  newMissile <= '0';
end if;

-- left and right
if start = 1 then
  if shipTimer >= shipSpeed then
    shipTimer <= 0;
    if left = '1' then
      if shipPos > (o+shipMargin) then
        shipPos <= shipPos - 1;
      end if;
    elsif right = '1' then
      if shipPos < (maxShipPosValue-shipMargin) then
        shipPos <= shipPos + 1;
      end if;
    end if;
  else
    shipTimer <= shipTimer + 1;
  end if;
else
  shipTimer <= 0;
end if;

-- aliens
if alienTimer >= alienSpeed then
  alienTimer <= 0;
  case alienDirection is
    when 0 => -- go left
      if alienXX > alienXMargin then
        alienXX <= alienXX -alienJump;
        alienTimer <= alienXX + fireTimer;
      end if;
    when 1 => -- go up left
      if alienXX > alienXMargin and alienYY > alienYUpMargin then
        alienXX <= alienXX -alienJump;
        alienYY <= alienYY -alienJump;
        alienTimer <= alienYY + shipTimer;
      end if;
    when 2 => -- go up
      if alienYY > alienYUpMargin then
        alienYY <= alienYY -alienJump;
        alienTimer <= alienYY + shipTimer;
      end if;
    when 3 => -- go up right
      if alienXX < (500-alienXMargin) and alienYY > alienYUpMargin then
        alienXX <= alienXX +alienJump;
        alienYY <= alienYY -alienJump;
        alienTimer <= alienXX + fireTimer;
      end if;
    when 4 => -- go right
      if alienXX < (500-alienXMargin) then
        alienXX <= alienXX +alienJump;
        alienTimer <= alienXX + shipTimer;
      end if;
    when 5 => -- go right down
      if alienXX < (500-alienXMargin) and alienYY < alienYDownMargin then
        alienXX <= alienXX +alienJump;
        alienYY <= alienYY +alienJump;
        alienTimer <= alienYY + fireTimer;
      end if;
    when 6 => -- go down
      if alienYY < alienYDownMargin then
        alienYY <= alienYY +alienJump;
        alienTimer <= alienXX + shipTimer;
      end if;
    when others => -- go down left
      if alienXX > alienXMargin and alienYY < alienYDownMargin then
        alienXX <= alienXX -alienJump;
        alienYY <= alienYY +alienJump;
        alienTimer <= alienYY + fireTimer;
      end if;
  end case;
else
  alienTimer <= alienTimer + 1;
end if;
end if;
end process;

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;

end Behavioral;

```

3.2.5 rocketManager

```

-- Company:      HES-SO
-- Engineer:     Samuel Riedo & Pascal Roulin
-- Create Date:  13/04/2017
-- Design Name:  rocketManager.vhd
-- Project Name:  Space Invaders — FPGA Edition
-- Target Devices:  Digilent NEXYS 3 (Xilinx Spartan 6 XC6LX16-CS324)
-- Description:   Manage rocket shoot by the ship
-- Revision 0.01 -- File Created
--               1.00 -- Fire, left and write implemented
--               1.1  -- Ship and aliens movements implemented
--               1.2  -- Ship rocket implemented
--               1.3  -- Rocket can be stopped by Display when an alien is killed

```

```

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

library work;
use work.SpaceInvadersPackage.all;

entity rocketManager is
    port(
        newMissile      : in std_logic;           -- If 1, new missile launched
        reset           : in std_logic;           -- Active high
        clk             : in std_logic;           -- 40MHz
        alienKilled     : in std_logic;           -- 1 if alien killed
        shipPosition    : in std_logic_vector(9 downto 0); -- Ship x coordinate
        rocketOnScreen  : out std_logic;          -- If 1, display a rocket
        missileY        : out std_logic_vector(9 downto 0); -- Pixels between top screen and top missile position
        MissileX        : out std_logic_vector(9 downto 0); -- Missile x coordinate
    );
end rocketManager;

architecture Behavioral of rocketManager is

    signal rocketDisplayed : std_logic;           -- Integer of rocketOnScreen
    signal rocketY        : integer range 0 to (VLINES-30); -- Pixels between top screen and top missile
    signal missileTimer    : integer range 0 to missileSpeed; -- Slow down fire rate
    signal shootFinished   : integer range 0 to 1;           -- If 1, current missile is off screen
    signal MissileXX       : integer range shipMargin to (HLINES-shipMargin) := shipMargin; -- Missile x value
                                                         -- from left screen

begin

    rocketOnScreen <= rocketDisplayed;
    missileY      <= std_logic_vector(to_unsigned(rocketY, 10));
    MissileX      <= std_logic_vector(to_unsigned(MissileXX, 10));

    -- Update rocketY
    process(reset, clk, alienKilled)
    begin
        if reset = '1' or alienKilled = '1' then
            missileTimer <= 0;
            rocketY      <= 0;
            MissileXX    <= shipMargin;
        elsif rising_edge(clk) then
            if rocketDisplayed = '1' then
                if rocketY = 0 then
                    rocketY <= (VLINES-30);
                    MissileXX <= (to_integer(unsigned(shipPosition))+31);
                end if;
            end if;
        end if;
    end process;

```



```

        if missileTimer = missileSpeed then
            missileTimer <= 0;
            rocketY <= rocketY - 1;
            MissileXX <= MissileXX;
        else
            missileTimer <= missileTimer + 1;
        end if;
    else
        rocketY <= 0;
        MissileXX <= shipMargin;
    end if;
end if;
end process;

-- Update rocketDisplayed
process(newMissile, shootFinished)
begin
    rocketDisplayed <= '0';
    if newMissile = '1' then
        rocketDisplayed <= '1';
    elsif shootFinished = 0 then
        rocketDisplayed <= '1';
    end if;
end process;

-- Update shootFinished
process(rocketY, shootFinished)
begin
    if rocketY = 0 then
        shootFinished <= 1;
    else
        shootFinished <= 0;
    end if;
end process;

end Behavioral;

```

3.2.6 VGA Internal

```

-- Company:      HES-SO
-- Engineer:     Samuel Riedo & Pascal Roulin
-- Create Date:  09:20:02 03/02/2017
-- Design Name:  vga_internal.vhd
-- Project Name:  Space Invaders - FPGA Edition
-- Target Devices: Digilent NEXYS 3 (Xilinx Spartan 6 XC6LX16-CS324)
-- Description:   Video Graphics Array.
-- Revision 0.01 - File Created
--              1.00 - First functional version

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

use work.SpaceInvadersPackage.all;

entity VGA_Internal is
    port (
        pixel_clk : in std_logic;           -- 40MHz
        rst       : in std_logic;           -- active high
        hs        : out std_logic;           -- Horizontale synchronization impulsion
        vs        : out std_logic;           -- Vertical synchronization impulsion
        blank     : out std_logic;           -- If 1, video output must be null
        hcount    : out std_logic_vector(10 downto 0); -- Pixel x coordinate
        vcount    : out std_logic_vector(10 downto 0); -- Pixel y coordinate
    end entity VGA_Internal;

architecture logic of VGA_Internal is

    signal hcounter : integer range 0 to HMAX; -- integer version of hcount
    signal vcounter : integer range 0 to VMAX; -- integer version of vcount
    signal endOfLine : std_logic;

begin

    hcount <= std_logic_vector(to_unsigned(hcounter, 11));
    vcount <= std_logic_vector(to_unsigned(vcounter, 11));
    endOfLine <= '1' when (hcounter = HMAX) else '0'; -- 1 if hcount = HMAX

    -- Processus: Columns Counter, update hcounter.
    process(rst, pixel_clk)
    begin
        if rst = '1' then

```

```

    hcounter <= 0;
    elsif rising_edge(pixel_clk) then
        if hcounter = HMAX then
            hcounter <= 0;
        else
            hcounter <= hcounter + 1;
        end if;
    end if;
end process;

-- Processus: Lines Counter, update vcounter.
process(rst, pixel_clk, endOfLine)
begin
    if rst = '1' then
        vcounter <= 0;
    elsif rising_edge(pixel_clk) then
        if endOfLine = '1' then
            if vcounter = VMAX then
                vcounter <= 0;
            else
                vcounter <= vcounter + 1;
            end if;
        end if;
    end if;
end process;

-- Update hs.
process (pixel_clk, rst) is
begin
    if rst = '1' then
        hs <= '0';
    elsif rising_edge(pixel_clk) then
        if (hcounter >= HFP and hcounter < HSP) then
            hs <= '1';
        else
            hs <= '0';
        end if;
    end if;
end process;

-- Update vs.
process (pixel_clk, rst) is
begin
    if rst = '1' then
        vs <= '0';
    elsif rising_edge(pixel_clk) then
        if (vcounter >= VFP and vcounter < VSP) then
            vs <= '1';
        else
            vs <= '0';
        end if;
    end if;
end process;

-- Update blank.
process (pixel_clk, rst) is
begin
    if rst = '1' then
        blank <= '1';
    elsif rising_edge(pixel_clk) then
        if (hcounter < H LINES and vcounter < V LINES) then
            blank <= '0';
        else
            blank <= '1';
        end if;
    end if;
end process;
end architecture logic;

```

3.2.7 Top Module

```

— Company:      HES-SO
— Engineer:     Samuel Riedo & Pascal Roulin
— Create Date:  1/03/2017
— Design Name:  TopModule.vhd
— Project Name:  Space Invaders — FPGA Edition
— Target Devices:  Digilent NEXYS 3 (Xilinx Spartan 6 XC6LX16-CS324)
— Description:   Project top module
— Revision 0.01 — File Created
—               1.00 — VGA_internal, Display, DCM and StartScreenROM added
—               1.1  — Ship and aliens movements implemented
—               1.2  — Inputs added
—               1.3  — Display now have a clk
—               1.4  — Add alienRocket

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

entity TopModule is
  port(
    fpga_clk      : in  std_logic;      — 100MHz
    rst           : in  std_logic;      — Active high
    startButton   : in  std_logic;      — Start button
    fire          : in  std_logic;      — Fire button
    right         : in  std_logic;      — Right arrow button
    left          : in  std_logic;      — Left arrow button
    HS            : out std_logic;       — VGA horizontal synchronization
    VS            : out std_logic;       — VGA vertical synchronization
    red           : out std_logic_vector(2 downto 0); — VGA red bus
    green         : out std_logic_vector(2 downto 0); — VGA green bus
    blue          : out std_logic_vector(1 downto 0); — VGA blue bus
  );
end entity;

architecture Behavioral of TopModule is

  signal pixel_clk      : std_logic; — 40MHz clock
  signal blank          : std_logic; — When 1, VGA color outputs must be 0
  signal locked         : std_logic; — Unused
  signal gameStarted    : std_logic; — When 0, show start screen
  signal newMissile     : std_logic; — When 1, new missile launched
  signal rocketOnScreen : std_logic; — If 1, display a rocket
  signal alienKilled    : std_logic; — If 1, the rocket killed an alien
  signal startScreenROMOut : std_logic_vector(7 downto 0); — Used as ImageInput in display
  signal alienY         : std_logic_vector(8 downto 0); — first alien position from top screen
  signal shipPosition   : std_logic_vector(9 downto 0); — From left screen
  signal alienX         : std_logic_vector(9 downto 0); — first alien position from left screen
  signal missileY       : std_logic_vector(9 downto 0); — Pixels between top screen and top missile position
  signal hcount         : std_logic_vector(10 downto 0); — VGA horizontal synchronization
  signal vcount         : std_logic_vector(10 downto 0); — VGA vertical synchronization
  signal romAddress     : std_logic_vector(14 downto 0); — Combination of hcount and vcount
  signal MissileX       : std_logic_vector(9 downto 0); — Missile x coordinate
  signal alienRocketx   : std_logic_vector(9 downto 0); — Alien rocket x position
  signal alienRockety   : std_logic_vector(9 downto 0); — Alien rocket y position
  signal alienL1        : std_logic_vector(0 to 9); — Same value as alienLine2
  signal alienL2        : std_logic_vector(0 to 9); — Same value as alienLine2
  signal alienL3        : std_logic_vector(0 to 9); — Same value as alienLine3
  signal alienL4        : std_logic_vector(0 to 9); — Same value as alienLine4
  signal alienL5        : std_logic_vector(0 to 9); — Same value as alienLine5

  component display is
    port(
      blank          : in  std_logic; — If 1, video output must be null
      gameStarted    : in  std_logic; — When 0, show start screen
      rocketOnScreen : in  std_logic; — If 1, display a rocket
      clk            : in  std_logic; — 40MHz
      alienRocketx   : in  std_logic_vector(9 downto 0); — Alien rocket x position
      alienRockety   : in  std_logic_vector(9 downto 0); — Alien rocket y position
      missileY       : in  std_logic_vector(9 downto 0); — Pixels between top screen and top missile position
      shipPosition   : in  std_logic_vector(9 downto 0); — Ship x coordinate
      MissileX       : in  std_logic_vector(9 downto 0); — Missile x coordinate
      hcount         : in  std_logic_vector(10 downto 0); — Pixel x coordinate
      vcount         : in  std_logic_vector(10 downto 0); — Pixel y coordinate
      imageInput     : in  std_logic_vector(7 downto 0); — data from rom
      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
      alienKilled    : out std_logic; — 1 if alien killed
      red            : out std_logic_vector(2 downto 0); — Red color output
      green          : out std_logic_vector(2 downto 0); — Green color output
      blue           : out std_logic_vector(1 downto 0); — Blue color output
      alienL1        : out std_logic_vector(0 to 9); — Same value as alienLine1
    );
  end component;

  pixel_clk <= fpga_clk;
  blank <= 0;
  locked <= 0;
  gameStarted <= 0;
  newMissile <= 0;
  rocketOnScreen <= 0;
  alienKilled <= 0;
  startScreenROMOut <= (others => 0);
  alienY <= (others => 0);
  shipPosition <= (others => 0);
  alienX <= (others => 0);
  missileY <= (others => 0);
  hcount <= (others => 0);
  vcount <= (others => 0);
  romAddress <= (others => 0);
  MissileX <= (others => 0);
  alienRocketx <= (others => 0);
  alienRockety <= (others => 0);
  alienL1 <= (others => 0);
  alienL2 <= (others => 0);
  alienL3 <= (others => 0);
  alienL4 <= (others => 0);
  alienL5 <= (others => 0);

  display : display
    port map (
      blank          => blank,
      gameStarted    => gameStarted,
      rocketOnScreen => rocketOnScreen,
      clk            => pixel_clk,
      alienRocketx   => alienRocketx,
      alienRockety   => alienRockety,
      missileY       => missileY,
      shipPosition   => shipPosition,
      MissileX       => MissileX,
      hcount         => hcount,
      vcount         => vcount,
      imageInput     => startScreenROMOut,
      alienX         => alienX,
      alienY         => alienY,
      alienKilled    => alienKilled,
      red            => red,
      green          => green,
      blue           => blue,
      alienL1        => alienL1
    );
end architecture;

```

```

    alienL2      : out std_logic_vector(0 to 9);      — Same value as alienLine2
    alienL3      : out std_logic_vector(0 to 9);      — Same value as alienLine3
    alienL4      : out std_logic_vector(0 to 9);      — Same value as alienLine4
    alienL5      : out std_logic_vector(0 to 9);      — Same value as alienLine5
  );
end component;

component dcm is
  port(
    CLK_IN1      : in  std_logic;      — 100MHz
    RESET        : in  std_logic;      — Active high
    CLK_OUT1     : out std_logic;      — 40MHz
    LOCKED       : out std_logic;      — Unused
  );
end component;

component vga_internal is
  port(
    pixel_clk    : in  std_logic;      — 40MHz
    rst          : in  std_logic;      — active low
    hs           : out std_logic;      — Horizontale synchronization impulsion
    vs           : out std_logic;      — Vertical synchronization impulsion
    blank        : out std_logic;      — If 1, video output must be null
    hcount       : out std_logic_vector(10 downto 0); — Pixel x coordinate
    vcount       : out std_logic_vector(10 downto 0); — Pixel y coordinate
  );
end component;

component StartScreenRom is
  port(
    clka         : in  std_logic;      — 40MHz
    addra        : in  std_logic_vector(14 downto 0); — Combinaison of hcount and vcount
    douta        : out std_logic_vector(7 downto 0);  — ROM output
  );
end component;

component Input is
  port(
    startButton  : in  std_logic;      — When 1, start game
    fire         : in  std_logic;      — When 1, shoot a rocket
    clk          : in  std_logic;      — 40MHz
    reset        : in  std_logic;      — Active high
    left         : in  std_logic;      — Left arrow button
    right        : in  std_logic;      — Right arrow button
    newMissile   : out std_logic;      — If 1, new missile launched
    gameStarted  : out std_logic;      — When 0, show start screen
    alienX       : out std_logic_vector(9 downto 0); — first alien position from left screen
    alienY       : out std_logic_vector(8 downto 0); — first alien position from top screen
    shipPosition : out std_logic_vector(9 downto 0); — Ship x coordinate
  );
end component;

component rocketManager is
  port(
    newMissile    : in  std_logic;      — If 1, new missile launched
    reset         : in  std_logic;      — Active high
    clk           : in  std_logic;      — 40MHz
    alienKilled   : in  std_logic;      — 1 if alien killed
    shipPosition  : in  std_logic_vector(9 downto 0); — Ship x coordinate
    rocketOnScreen : out std_logic;      — If 1, display a rocket
    missileY      : out std_logic_vector(9 downto 0); — Pixels between top screen and top missile position
    missileX      : out std_logic_vector(9 downto 0); — Missile x coordinate
  );
end component;

component 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 component;

begin
  — Convert hcount and vcount in an address usable by a ROM block.

```

```

romAddress <= std_logic_vector(to_unsigned((150*(to_integer(unsigned(hcount))/4) + (to_integer(unsigned(vcount))
/4), 15));

StartScreenRom_map : StartScreenRom
  port map(
    clka => pixel_clk,
    addra => romAddress,
    douta => startScreenROMOut
  );

DCM_map : DCM
  port map(
    CLK_IN1 => fpga_clk,
    CLK_OUT1 => pixel_clk,
    RESET => rst,
    LOCKED => LOCKED
  );

VGA : VGA_Internal
  port map(
    pixel_clk => pixel_clk,
    rst => rst,
    hs => hs,
    vs => vs,
    blank => blank,
    hcount => hcount,
    vcount => vcount
  );

Display_Map : Display
  port map(
    blank => blank,
    gameStarted => gameStarted,
    rocketOnScreen => rocketOnScreen,
    clk => pixel_clk,
    missileY => missileY,
    hcount => hcount,
    vcount => vcount,
    alienRocketx => alienRocketx,
    alienRockety => alienRockety,
    alienKilled => alienKilled,
    MissileX => MissileX,
    shipPosition => shipPosition,
    alienX => alienX,
    alienY => alienY,
    red => red,
    green => green,
    imageInput => startScreenROMOut,
    blue => blue,
    alienL1 => alienL1,
    alienL2 => alienL2,
    alienL3 => alienL3,
    alienL4 => alienL4,
    alienL5 => alienL5
  );

Input_Map : Input
  port map(
    startButton => startButton,
    fire => fire,
    clk => pixel_clk,
    reset => rst,
    right => right,
    left => left,
    newMissile => newMissile,
    shipPosition => shipPosition,
    gameStarted => gameStarted,
    alienX => alienX,
    alienY => alienY
  );

rocketManager_Map : rocketManager
  port map(
    newMissile => newMissile,
    reset => rst,
    clk => pixel_clk,
    shipPosition => shipPosition,
    rocketOnScreen => rocketOnScreen,
    alienKilled => alienKilled,
    missileY => missileY,
    MissileX => missileX
  );

alienRocketMap : alienRocket
  port map(

```

```

reset      => rst,
clk        => pixel_clk,
alienLine1 => alienL1,
alienLine2 => alienL2,
alienLine3 => alienL3,
alienLine4 => alienL4,
alienLine5 => alienL5,
alienX     => alienX,
alienY     => alienY,
alienRocketx => alienRocketx,
alienRockety => alienRockety
);

```

```
end architecture;
```

3.2.8 Package

```

— Company:      HES—SO
— Engineer:     Samuel Riedo & Pascal Roulin
— Create Date:  11:24:52 03/02/2017
— Design Name:  SpaceInvadersPackage.vhd
— Project Name: Space Invaders — FPGA Edition
— Target Devices: Digilent NEXYS 3 (Xilinx Spartan 6 XC6LX16—CS324)
— Description:   Contain all constant.
— Revision 0.01 — File Created
—               1.00 — Add VGA constants
—               1.1  — Add Inputs constants
—               1.2  — Add RocketManager constants
—               1.3  — Add alienRocket constants

```

```

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

```

```
package SpaceInvadersPackage is
```

```

— VGA
constant HMAX      : integer := 1056;
constant VMAX      : integer := 628;
constant HLines    : integer := 800;
constant VLines    : integer := 600;
constant HSP       : integer := 968;
constant HFP       : integer := 840;
constant VFP       : integer := 601;
constant VSP       : integer := 605;

```

```

— Inputs
constant fireSpeed : integer := 600000;
constant shipSpeed : integer := 100000;
constant alienSpeed : integer := 6000000;
constant maxShipPosValue : integer := 736; — must be pair
constant shipMargin : integer := 50; — minimum space between side screen and ship
constant alienXMargin : integer := 50; — minimum space between side screen and aliens
constant alienYUpMargin : integer := 50; — minimum space between top screen and aliens
constant alienYDownMargin : integer := 200; — maximum space between top screen and aliens
constant maxAlienJump : integer := 10; — max pixel aliens can shift in a single time

```

```

— rocketManager
constant missileSpeed : integer := 60000; — missile speed
constant rocketLength : integer := 10; — rocket length in pixel
constant rocketColor : std_logic_vector(7 downto 0) := "11111111";

```

```

— alienRocket
constant rocketFrequency : integer := 6000000; — the lower the number,
— the faster the rocket launched

```

```
— Tables of aliens and ship
```

```

— 0 = no alien
— 1 = blue, 3 = dark blue, 5 = green, 7 = purple, 9 = yellow
— 2 = blue alien killed, 4 = dark blue alien killed, etc ...
type aliensArray is array(9 downto 0, 4 downto 0) of integer range 0 to 10;
signal aliens : aliensArray := (others => (others => 1)); — Initialized to 0

```

```

— All aliens are 30x30
type alienPicture is array(0 to 29, 0 to 29) of integer;

```

```

constant blueAlien : alienPicture := (
  (16#0#, 16#0#, 16#0#, 16#0#, 16#0#, 16#0#, 16#0#, 16#0#, 16#20#, 16#20#, 16#0#, 16#0#, 16#4#, 16#4#, 16#0#,
  16#0#, 16#0#, 16#0#, 16#0#, 16#0#, 16#20#, 16#20#, 16#21#, 16#21#, 16#4#, 16#4#, 16#4#, 16#4#, 16#0#, 16#0#),

```

[illegible]