Groupe 1

Jean-Baptiste Mattei

Clément Veith

Valentin Scias

Lucas Ribeiro

Paul François

Rapport VHDL

ISEN MÉDITERRANÉE | L'ÉCOLE DES INGÉNIEURS DU NUMÉRIQUE

Introduction

Ce projet VHDL, réalisé sur la carte Digilent Basys3, a pour objectif de mettre en œuvre une série de modules logiques décrits en VHDL, en allant de simples clignotements de LED jusqu'à l'affichage de jeux interactifs sur écran VGA comme Pong et Snake. Chaque sujet aborde une nouvelle fonctionnalité ou complexité, permettant de développer une compréhension progressive de la conception matérielle. L'accent est mis sur la gestion des entrées/sorties, les contraintes temporelles, le respect des standards de communication visuelle (VGA), et la structuration modulaire du code VHDL. Ce rapport présente successivement les différentes réalisations et les problèmes rencontrés, ainsi que les solutions apportées.

Sujet 1 : Clignotement d'une LED

Pour ce sujet, nous devions allumer une LED sur la carte. Il fallait donc, dans le fichier de contraintes .xdc, activer le pin de la LED, que l'on associera dans le fichier VHDL à une variable (0 ou 1) issue de l'entité led. Il fallait également activer la clock et le RESET de la carte pour pouvoir l'utiliser correctement (associés respectivement à CLK pour l'horloge et RST pour le RESET).

Concernant la carte, son horloge est à 100 MHz. Or, si l'on utilise directement cette horloge, la LED clignotera trop vite pour que cela soit perceptible à l'œil nu. Il faut donc faire clignoter la LED tous les 2^{24} front montants, ce qui donne 2^{24} / 100 000 000 = 0,167 s, grâce à un compteur.

Enfin, il ne reste plus qu'à faire en sorte que, lorsque le bouton RST est appuyé, la valeur du compteur revienne à 0.

Sujet 2 : Chenillard à LED

Le principe est assez similaire au sujet 1 à l'exception du nombre de led qui passe de 1 à 16, et de la manière dont on va les gérer. Le compteur reste toujours présent étant donné que même si plusieurs leds sont utilisées, si la fréquence est trop élevée on verra toutes les leds allumées malgré que le code allume réellement 1 led à la fois. On utilise dans l'entité led non pas 1 bit pour la led mais autant de bits que de led présente, donc 16 bits dans un vecteur (chaque bit est associé à une seule et unique LED). On va traiter les 4 octets qui nous fournira des valeurs allant de 0 jusqu'à 15 avec une conversion binaire -> décimal, et on associe chaque valeur à

une led (0 pour la led0, 1 pour la led1 etc...). Après la mise en place des variables nécessaire pour le sujet on procède d'abord par incrémenter le compteur à chaque front d'horloge. Seul uniquement les 4 bits de poids fort sont intéressants pour déterminer la led à allumer. On éteint donc toutes les leds puis on allume celle qui est concerné par les 4 bits de points fort (exemple : si on a 0010 = 2 on va donc mettre à 1 le 2eme bits, 0000 0000 0000 0010, ce qui correspond bien à la 2eme led à allumer). On rajoute le reset du sujet 1 et le sujet 2 est complété. Enfin dans le cas où on est à la dernière led, si on ajoute à 15 (1111 en binaire) on revient à 0 car on est limité à 4 bits, la boucle est bouclée.

Sujet 3 : Compteurs et décompteur incrémental à LEDs

Le principe reste basé sur un compteur piloté par l'horloge, mais on ajoute une gestion de sens : le compteur peut augmenter (compteur) ou diminuer (décompteur). Un bouton ou une entrée de contrôle détermine le sens. L'affichage se fait toujours sur un vecteur de LEDs, chaque bit du vecteur représentant une LED (1 = allumée, 0 = éteinte). À chaque évènement (fréquence divisée pour être visible), la valeur du compteur est traduite en binaire sur les LEDs : si le compteur vaut 5, alors la LED 5 s'allume ; en mode décompteur, le compteur décrémente et la LED correspondante s'allume en sens inverse. On utilise typiquement un registre pour stocker la valeur courante et une logique pour détecter les bornes (0 et maximum), permettant de reboucler ou d'inverser le sens. Si la valeur atteint la limite haute, elle revient à 0 (ou inversement si décomptage et 0 atteint, retour à la valeur haute). La remise à zéro du compteur s'effectue avec un signal de reset comme pour les sujets précédents.

On manipule donc la direction du comptage, le rebouclage et l'affichage binaire direct sur les LEDs, synchronisés sur l'horloge divisée, pour garantir un affichage visible.

Sujet 4: Affichage de 4 chiffres identiques sur les afficheurs

Dans ce projet, on cherche à réaliser l'affichage du même chiffre sur les quatre digits d'un afficheur 7 segments, en utilisant le langage VHDL. Pour cela, nous avons choisi d'afficher le chiffre 1 sur tous les afficheurs, de manière synchronisée et fluide. Comme tous les digits partagent les mêmes lignes de segments, on utilise la technique du multiplexage temporel. Ce principe permet d'activer les digits un par un, très rapidement, de façon à ce que l'œil humain perçoive les quatre chiffres comme étant affichés en même temps.

Le système repose sur une seule entrée, CLK, qui est l'horloge du système, typiquement cadencée à 100 MHz sur la carte Basys 3. En sortie, on utilise deux

vecteurs : SEG, qui contient 7 bits pour piloter les segments (de a à g) de l'afficheur, et AN, un vecteur de 4 bits servant à sélectionner quel digit est activé. Les anodes sont actives à l'état bas, ce qui signifie qu'un 0 logique sur une ligne AN active le digit correspondant.

Pour afficher le chiffre 1 sur tous les digits, on utilise un compteur de 16 bits qui sert à diviser la fréquence de l'horloge. Ce compteur s'incrémente à chaque front montant du signal CLK. On récupère ses deux bits les plus significatifs (les bits 15 et 14) pour créer un signal sel, qui va nous servir à faire tourner l'activation des quatre afficheurs.

En fonction de la valeur de sel, on active à tour de rôle une seule anode parmi les quatre. À chaque étape, on assigne à SEG la configuration binaire correspondant au chiffre 1. De cette manière, bien qu'un seul afficheur soit physiquement activé à la fois, on a l'impression que tous les afficheurs affichent le chiffre 1 en même temps grâce à la rapidité du multiplexage.

Sujet 5: Affichage de 4 chiffres différent sur les afficheurs

Dans ce projet, notre objectif était d'afficher quatre chiffres différents — 4, 3, 2 et 1 — sur un afficheur 7 segments à 4 digits en utilisant la technique du multiplexage. Nous nous sommes appuyés sur un premier code qui affichait le chiffre 1 sur les quatre digits en alternant rapidement leur activation pour créer l'illusion d'un affichage simultané. Plutôt que de repartir de zéro, nous avons conservé cette structure et modifié uniquement ce qui était nécessaire pour afficher des chiffres différents sur chaque digit.

Les entrées et sorties du module sont restées identiques à la première version. L'horloge CLK rythme l'ensemble du système, les sorties SEG pilotent les segments communs (a à g) et le vecteur AN active successivement les afficheurs. Comme la logique de multiplexage est inchangée, il n'a pas été nécessaire de modifier les ports ni le fichier de contraintes .xdc.

En interne, la logique utilise toujours un compteur 16 bits s'incrémentant à chaque front montant de l'horloge. Les deux bits de poids fort du compteur génèrent le signal sel, qui sélectionne successivement chaque afficheur. Ce signal active les anodes via AN et détermine la valeur affichée via SEG.

La principale amélioration est dans le contenu affiché. Alors que le premier code envoyait toujours le code binaire du chiffre 1, nous avons créé un tableau de constantes contenant les motifs binaires des chiffres 4, 3, 2 et 1. À chaque cycle de rafraîchissement, sel permet de sélectionner dynamiquement l'un de ces chiffres, assurant ainsi un affichage différent sur chaque digit.

Nous avons aussi pris en compte le câblage des afficheurs, connectés physiquement de droite à gauche. Pour respecter l'ordre naturel de lecture (gauche à droite), l'ordre d'activation dans le code a été inversé : sel = "00" active le digit de gauche avec le chiffre 4, jusqu'à sel = "11" affichant le chiffre 1 à droite.

Ces ajustements assurent un affichage fluide et lisible, où chaque chiffre apparaît à la bonne place, donnant l'illusion d'un affichage simultané malgré le multiplexage.

Sujet 6: Chronomètre

Nous avons conçu un système d'affichage VGA dont le fonctionnement repose sur un principe proche de celui des projets précédents de type LED clignotante : un signal d'horloge, un découpage temporel via compteur, et une sortie visuelle synchronisée. Ici, au lieu d'allumer séquentiellement des LEDs, nous balayeons ligne par ligne une matrice de pixels sur un écran VGA pour afficher une image fixe monochrome.

Le cœur du système est le contrôleur VGA, qui gère le balayage des coordonnées X et Y selon les timings imposés par le standard VGA 640×480 @ 60 Hz. Ce contrôleur génère les signaux de synchronisation horizontale (Hsync) et verticale (Vsync), mais également les signaux video_on, X et Y qui permettent de savoir si la position courante se situe dans la zone visible de l'écran.

Comme pour les projets à LEDs, une horloge rapide doit être ralentie pour que l'affichage soit perceptible. On utilise un diviseur d'horloge pour passer de 100 MHz (horloge du Basys3) à 25 MHz, fréquence adaptée au protocole VGA. Sans cette division, les transitions seraient trop rapides pour être visibles ou interprétables par un écran.

L'image monochrome est codée sous forme d'une mémoire ROM 2D dans le fichier image.vhd, où chaque bit correspond à un pixel (0 = noir, 1 = blanc). La sortie vidéo est un signal RGB simplifié : seul le canal rouge est activé en fonction de la valeur binaire du pixel à afficher. Les coordonnées (X,Y) générées par le contrôleur servent à adresser cette mémoire.

Le module image_top.vhd réalise l'interconnexion des sous-modules. Il reçoit les coordonnées VGA, interroge la ROM image, et transmet le bit courant au signal rouge uniquement lorsque video_on est actif. Le système prend aussi en compte les phases invisibles du balayage : front porch, sync pulse, back porch, afin que l'affichage reste stable et conforme aux normes VGA.

Les contraintes physiques sont définies dans le fichier .xdc, associant chaque signal (horloge, VGA R/G/B, Hsync, Vsync) à une broche spécifique de la carte Basys3.

Le fonctionnement global est donc analogue à une animation LED pilotée par un compteur : ici, c'est la position (X,Y) qui joue le rôle de l'index, balayant tous les

pixels à 25 MHz, et la mémoire image remplace le vecteur LED. Le signal video_on agit comme une condition d'activation, empêchant l'affichage dans les zones de synchronisation. Le résultat est une image fixe, stable, affichée intégralement ligne par ligne, avec une précision binaire à chaque pixel.

Sujet 7 : Image fixe monochrome sur port VGA:

Nous avons conçu un système d'affichage VGA qui transpose les principes des projets LED clignotantes au domaine du balayage vidéo. L'objectif est d'afficher une image fixe monochrome sur un écran VGA à partir d'une mémoire ROM, en utilisant la carte Basys3.

Le cœur du système repose sur un contrôleur VGA conforme à la norme 640×480 à 60 Hz. Cette norme impose une fréquence de pixel de 25 MHz. La carte Basys3 fournissant une horloge à 100 MHz, nous utilisons un diviseur pour réduire cette fréquence à 25 MHz. Comme dans les projets à LED, une fréquence trop rapide rendrait l'affichage imperceptible ; ici aussi, on temporise pour rendre l'affichage stable et visible.

Le contrôleur VGA génère les signaux de synchronisation horizontale (Hsync) et verticale (Vsync), les coordonnées de balayage X et Y, ainsi qu'un signal video_on qui indique si la position courante est dans la zone visible de l'écran. Le balayage de l'image suit un schéma temporel précis constitué de quatre phases successives.

La première phase est l'affichage actif, où les pixels sont visibles sur l'écran. Elle couvre une zone de 640 colonnes en horizontal et 480 lignes en vertical. Ensuite vient le front porch, qui correspond à une courte période après la fin de la zone visible, avant l'impulsion de synchronisation. Cette phase sert à stabiliser le signal. Elle dure 16 pixels horizontalement et 10 lignes verticalement.

La phase suivante est le sync pulse, qui correspond à l'impulsion de synchronisation proprement dite. Le signal Hsync passe à l'état bas pendant 96 pixels pour signaler la fin d'une ligne, tandis que Vsync passe à l'état bas pendant 2 lignes pour signaler la fin d'une trame complète.

Enfin, le back porch suit l'impulsion et sert à repositionner le faisceau avant de reprendre l'affichage actif. Cette phase dure 48 pixels horizontalement et 33 lignes verticalement.

Ainsi, une ligne complète dure 800 cycles de pixel (640 + 16 + 96 + 48), et une trame complète dure 525 lignes (480 + 10 + 2 + 33). Le signal video_on est actif uniquement pendant les 640x480 pixels d'affichage, ce qui permet de limiter l'accès à la mémoire image uniquement à la zone visible.

L'image à afficher est définie dans une mémoire ROM 2D codée en VHDL, où chaque bit correspond à un pixel monochrome, 1 pour un pixel blanc, 0 pour un pixel

noir. Les coordonnées X et Y permettent d'adresser directement le pixel courant. Le module principal interroge la ROM uniquement quand video_on est actif, récupère le bit associé au pixel, et active le signal rouge en conséquence. Les canaux vert et bleu restent à zéro, ce qui donne un affichage en noir et blanc.

Les broches physiques de la carte sont configurées via un fichier de contraintes .xdc qui définit les pins pour l'horloge, les signaux VGA, et les sorties RGB. L'image s'affiche ligne par ligne, de manière stable, sans scintillement, grâce au respect strict des timings VGA. Ce fonctionnement est analogue à une animation LED cyclique, transposée à une matrice de pixels avec balayage synchronisé.

Sujet 8 :

Le système repose sur le même contrôleur VGA conforme à la norme 640×480 à 60 Hz. La fréquence pixel exigée est de 25 MHz, obtenue par division de l'horloge 100 MHz via le module div_25MHz. La temporalité VGA (affichage, front porch, sync pulse, back porch) reste inchangée : 800 pixels par ligne, 525 lignes par image, signal video_on actif sur 640×480.

Différence fondamentale : l'image stockée dans la ROM n'est plus monochrome (1 bit par pixel), mais encode trois couleurs distinctes. Implémentation standard : chaque pixel de la ROM (voir image.vhd) code 2 bits, 00 pour noir, 01 pour rouge, 10 pour vert, 11 pour bleu (voir schéma dans le fichier image).

Processus de balayage : les coordonnées pixel_x et pixel_y issues du contrôleur VGA indexent la ROM image uniquement si video_on est actif. Le module principal image_top.vhd récupère le mot associé (2 bits), le décode et alimente sélectivement les signaux RGB du port VGA Basys3.

ROM(x, y) égal à 00 donne aucun canal activé, pixel noir
ROM(x, y) égal à 01 donne canal rouge activé, pixel rouge
ROM(x, y) égal à 10 donne canal vert activé, pixel vert
ROM(x, y) égal à 11 donne canal bleu activé, pixel bleu

Aucune combinaison simultanée, un seul canal actif à la fois, pas de cyan, magenta, jaune ou blanc. Contrôle strict des timings et multiplexage des canaux RGB synchrone au balayage.

Les sorties VGA (R, G, B, Hsync, Vsync) et l'horloge 100 MHz sont assignées aux broches physiques via le fichier .xdc Basys3_afficheur_CLK.xdc. Les bits de couleur alimentent les pins correspondantes, la structure générale du fichier de contraintes reste identique avec un périmètre modifié pour les signaux RGB supplémentaires.

Synthèse fonctionnelle : le système affiche une image fixe de 640×480, chaque pixel adressé depuis la ROM et décodé en l'une des 3 couleurs rouge, vert ou bleu selon la valeur des 2 bits lus. Aucun artefact, aucune temporisation visuelle, l'affichage est

stable, non scintillant, chaque ligne générée selon les timings VGA imposés. Le principe LED clignotante est généralisé à une matrice RGB synchrone, extension immédiate du monochrome au polychrome binaire trois couleurs.

Comparaison et extension, seul le décodage couleur change. Balayage, synchronisation, temporisation, adressage mémoire : identiques à la version monochrome. L'analogie d'une animation LED se transpose ici à une animation couleur discrète, balayant la mémoire pixel par pixel selon le schéma VGA.

Sujet 9 : Pong

Ce projet implémente le jeu Pong en VHDL sur la carte Digilent Basys3, avec un affichage VGA en 640x480 à 60 Hz. L'architecture est modulaire et sépare clairement la gestion vidéo, la logique de jeu et l'affichage graphique.

Le fichier de contraintes pong.xdc permet d'associer les signaux logiques VHDL (horloge, boutons, VGA, LEDs) aux broches physiques de la carte. L'horloge de 100 MHz est divisée en 25 MHz grâce au module div_25MHz.vhd, nécessaire pour le standard VGA. Le module vga_controller_640_60.vhd génère les signaux de synchronisation (HSYNC et VSYNC), les coordonnées des pixels (pixel_x, pixel_y), et le signal video_on, définissant la zone visible à l'écran.

La logique du jeu est intégrée dans le module principal top.vhd. Elle gère la position des deux raquettes et de la balle, les déplacements en réponse aux boutons, et les rebonds. Chaque bouton commande un mouvement vertical d'une raquette. La balle suit une trajectoire qui est recalculée en continu. En cas de collision avec une raquette ou un mur, sa direction est inversée. Si la balle franchit un bord, elle est replacée au centre. Cette logique peut être étendue à un système de score.

Le rendu graphique est assuré par le module pong_display.vhd, qui, à chaque pixel affiché, reçoit les coordonnées actuelles ainsi que les positions des éléments du jeu. Il détermine si le pixel appartient à la balle, à une raquette, à la ligne centrale ou au fond, et affecte la couleur RGB correspondante. L'affichage donne la priorité à la balle, suivie des raquettes, de la ligne centrale, puis du fond. Cette approche combinatoire garantit un rendu fluide et immédiat.

Le module top.vhd coordonne l'ensemble : il instancie le diviseur d'horloge, le contrôleur VGA et l'unité de rendu, tout en centralisant les entrées utilisateurs et la logique de déplacement. La séparation des fonctions permet une organisation claire et facilement extensible.

Sujet 10 : Snake

Ce projet a pour but de concevoir un jeu Snake interactif entièrement implémenté en VHDL, sans processeur embarqué, et affiché sur un écran VGA en 640×480 à 60 Hz. L'objectif est de produire un affichage fluide, réactif et autonome, tout en respectant les contraintes strictes de synchronisation imposées par le standard VGA. Ce choix permet de mettre en valeur les capacités du langage VHDL pour modéliser un système matériel complet, en temps réel.

Le jeu est piloté par une horloge principale clk à 100 MHz, générée par la carte Basys 3. Un module diviseur d'horloge transforme ce signal en une horloge de 25 MHz (pixel_clk), compatible avec la fréquence VGA. Le signal rst permet de réinitialiser la partie à tout moment, en restaurant la position initiale du serpent, sa taille et la position de la nourriture. Le joueur contrôle le serpent à l'aide de quatre boutons (btnU, btnD, btnL, btnR) correspondant aux directions haut, bas, gauche et droite.

En sortie, on génère les signaux standards VGA : HS et VS pour la synchronisation horizontale et verticale, ainsi que trois bus de 4 bits (RED, GREEN, BLUE) pour produire les couleurs à l'écran. Ces signaux permettent d'afficher dynamiquement la grille de jeu, la position du serpent et celle de la nourriture, avec un fond animé en damier.

La logique interne repose sur deux modules principaux. Le premier, vga_controller_640_60, génère les signaux de synchronisation ainsi que les coordonnées hcount et vcount représentant la position actuelle du pixel à l'écran. Le second, snake_display, contient la logique du jeu. La grille est composée de 20 colonnes et 15 lignes, chaque case occupant 32×32 pixels. La tête du serpent est repérée par head_x et head_y, et sa direction par dir_x et dir_y. Le corps du serpent est stocké dans un tableau body_mem, dont la longueur est variable.

Le serpent avance à chaque mise à jour, sa vitesse étant régulée par un compteur move_tick. À chaque déplacement, on vérifie les collisions avec les murs ou le corps. En cas de contact, le jeu redémarre. Lorsque la tête atteint la nourriture, le serpent s'allonge et un nouvel aliment apparaît à une position calculée pseudo-aléatoirement.

L'affichage final est généré en fonction des coordonnées VGA : la tête est rouge, le corps orange, la nourriture bleue, et le fond alterne entre deux verts pour un effet visuel de damier. Le signal blank évite tout affichage hors de la zone visible.

Problèmes rencontrés :

Pour la gestion des LEDs, des problèmes de double-allumage ou d'extinction simultanée de plusieurs LEDs sont apparus, dus à l'absence d'anti-rebond sur les boutons. L'ajout d'un système anti-rebond avec temporisation a permis de stabiliser la détection des appuis. Certains boutons n'étaient pas reconnus à cause d'une mauvaise configuration des broches dans le fichier de contraintes ; ce problème a été résolu en activant explicitement les broches correspondantes dans le .xdc.

Pour les afficheurs 7 segments, l'affichage était initialement illisible à cause d'un mauvais mapping des segments dans le fichier de contraintes (erreurs sur les broches V6, U7, U8). La correction s'est faite en vérifiant et remplaçant progressivement les broches jusqu'à obtenir un affichage correct. Le point décimal s'allumait à tort car un bit de commande était actif par erreur : forcer la valeur concernée à '1' a corrigé ce comportement. Certains chiffres comme le 9 ou le 5 étaient mal affichés, nécessitant un recalibrage de la table de vérité des segments après détection via des tests manuels.

Pour la partie VGA et affichage d'image fixe, plusieurs difficultés ont été relevées. Un décalage des bandes de couleur ou une mauvaise répartition provenaient d'une mauvaise gestion des limites horizontales dans le code, résolue par un calcul précis des valeurs de hc pour chaque bande. L'écran restait noir lorsque le signal blank était mal utilisé, problème réglé en respectant l'inversion de ce signal pour afficher les couleurs seulement dans la zone active. Enfin, une fréquence d'horloge incorrecte a été corrigée en ajoutant un diviseur d'horloge pour fournir le signal pixel_clk à 25 MHz.

Pour le module Pong, plusieurs problèmes spécifiques ont été rencontrés. Les raquettes étaient invisibles, ce qui a été corrigé par l'ajustement des coordonnées de dessin dans le module d'affichage. La balle sortait de l'écran sans jamais revenir, nécessitant une logique de réinitialisation automatique avec inversion de la direction horizontale. L'absence de rebond sur les bords haut et bas a été corrigée par l'implémentation d'un test sur la position verticale de la balle et l'inversion de sa direction verticale. La vitesse de la balle était trop rapide pour être jouable : l'ajout d'un signal de tick pour ralentir les mises à jour de position a permis d'obtenir une animation adaptée.

Enfin, pour le Snake, la taille de la grille était trop grande initialement, ce qui a été réglé en augmentant la taille des cellules pour adapter la résolution à l'écran VGA. Un espace visuel apparaissait entre la tête et le corps du serpent : il a été supprimé en mettant à jour la position de la tête avant le corps et en stockant les positions précédentes. Le déplacement du serpent était trop rapide ou bruité, corrigé en

ajoutant une temporisation similaire à un anti-rebond logiciel. Des valeurs de position hors limites provoquaient des erreurs d'affichage ou des bugs de collision : elles ont été fixées par un encadrement strict des bornes logiques de la grille. Chaque problème rencontré a ainsi été identifié, analysé, puis résolu par une modification structurée du code, des fichiers de contraintes ou de l'architecture logique des modules, garantissant la fiabilité et la stabilité du système final.

Conclusion

Ce projet nous a permis d'explorer concrètement les possibilités offertes par le langage VHDL dans le cadre du développement matériel sur FPGA. À travers des sujets de difficulté croissante, nous avons acquis une compréhension approfondie de la gestion du temps, du multiplexage, de l'affichage VGA et de la synchronisation des signaux. L'approche modulaire a facilité la conception, le test et la résolution des erreurs. Ce travail représente une base solide pour des projets futurs plus complexes, où la rigueur de la conception matérielle est essentielle.