Rapport Projet

VHDL/FPGA

Introduction :

ABBES Ayoub, Développeur Firmware
Utilisation de la carte Basys3 de Xilinx et développement de code dans l'environnement de développement Vivado.

SUJET 1 : Clignotement d'une LED.

Ce sujet a pour objectif de faire clignoter la LED à intervalle régulier.

- Définir les ports entrée/sortie (I/O) de la LED pour communiquer avec les
- périphériques externes.

Créer l'architecture qui permettra de faire clignoter la LED.

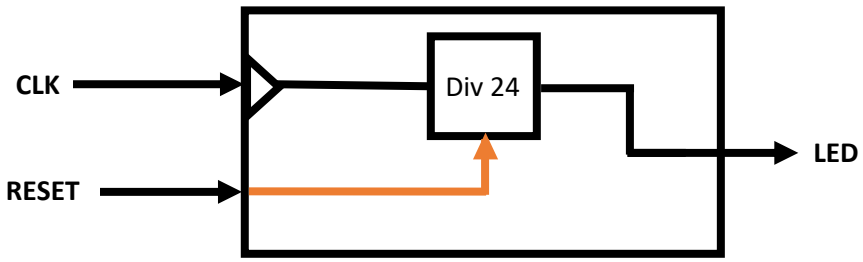
L'architecture doit inclure :

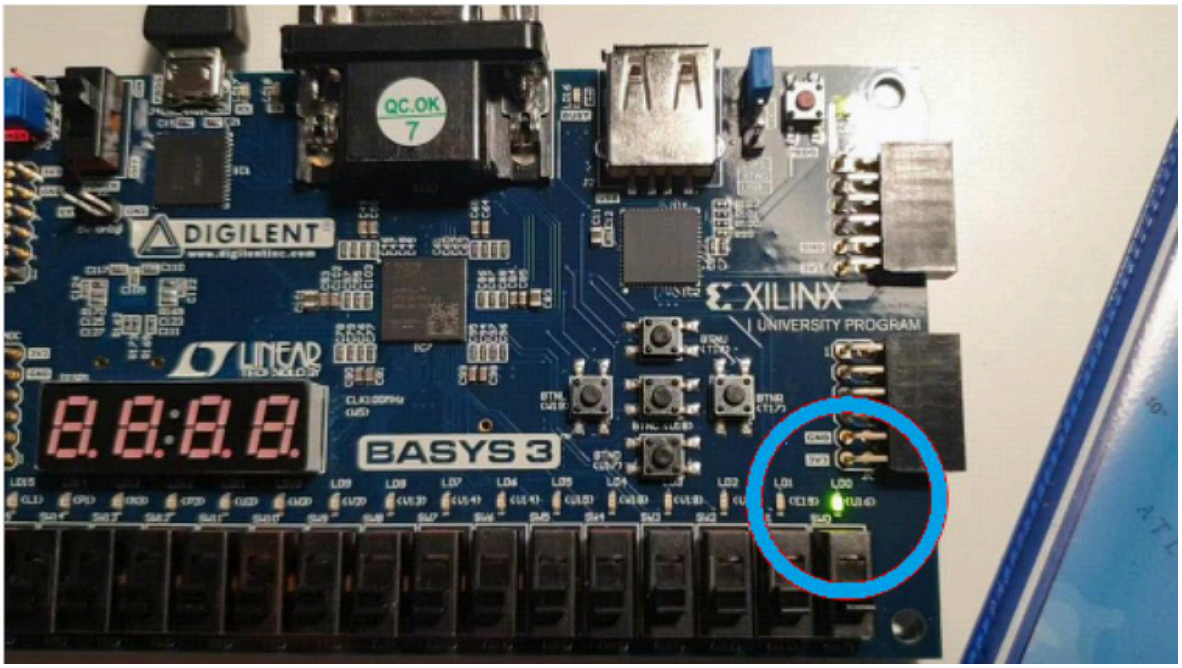
a. Un signal diviseur d'horloge. Ce signal d'horloge 24 bits sera utilisé pour synchroniser le fonctionnement du code avec le temps réel.
Les 24 bits comprennent le bit de poids fort égal à 1 (pour allumer la LED).

- Instruction de temporisation qui permettra de définir l'intervalle de temps entre
- chaque état de la LED (ON / OFF).

Le code qui va faire clignoter la LED. On utilise "if/elseif" pour changer l'état de la LED entre "allumé" et "éteint" à chaque intervalle de temps défini par la temporisation.

LED allumée → 1
LED éteinte → 0





Sujet 2 : Chenillard à LEDs.

En se basant sur le sujet 1, notre objectif est de d'allumer plusieurs LEDs à la suite.
2 Processus vont être nécessaire :

- Diviser la fréquence du signal d'horloge
- Décaler les bits

On va tout d'abord définir les entées :

- CLK → Signal d'horloge
- RST → Signal de reset

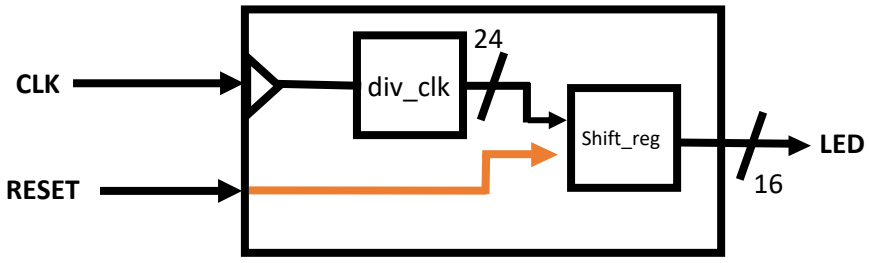
Ainsi que les 16 bits de sortie des LEDs

Dans ce code, un compteur d'horloge est utilisé pour déclencher le décalage des LED. Ce compteur est un mot de 24 bits (de 0 à 23) qui est incrémenté à chaque coup d'horloge. Lorsque le 24ème bit est à l'état logique "1", chaque LED va prendre l'état de la LED située à côté d'elle. Ainsi, le décalage crée un effet de chenillard, en faisant défiler les LED d'une position à la fois. Au démarrage du circuit, la LED0 est allumée de base pour permettre le cycle. En revanche, si l'utilisateur appuie sur le bouton reset, la LED0 est allumée et le cycle recommence.

Comme énoncé précédemment, le programme se base sur 2 processus :

1. La division de fréquence du signal d'horloge, qui utilise **clock_div** pour compter les cycles d'horloge. Le compteur incrémente à chaque front montant de CLK. Lorsque le signal RST est actif, le compteur est réinitialisé à 0.

2. Le 2ème processus implémente le décalage des bits, compté sur 16 bits nommé **shift_reg**. Le processus vérifie si le bit de poids fort de **clock_div** a changé pour détecter les cycles d'horloge. Lorsque cela se produit, les bits sont décalés d'une position vers la gauche. Le bit de poids fort est déplacé vers le bit de poids faible, et les autres bits sont déplacés de 1 vers la gauche. Le bit de poids faible est copié du bit de poids fort. Le bit de poids faible est donc décalé vers la gauche, ce qui crée « l'effet de chenillard ». Lorsque le signal RST est actif, les bits sont réinitialisé à la valeur x"0001", qui représente la LED la plus à droite allumée.



Sujet 3 : Compteur et décompteur incrémental à LEDs.

L'objectif de ce sujet est de pouvoir incrémenter ou décrémenter une LED en fonction du bouton sur lequel on appuis (haut ou bas).

Le code ci-après implémente un compteur binaire 4 bits en utilisant un registre à décalage pour générer une temporisation et deux boutons pour incrémenter ou décrémenter l'état. Les 16 états possibles sont utilisés pour afficher la valeur du compteur sur une rangée de 16 LED.

« **incre** » décrit les ports d'entrée et de sortie du module. Elle prend 4 entrées :

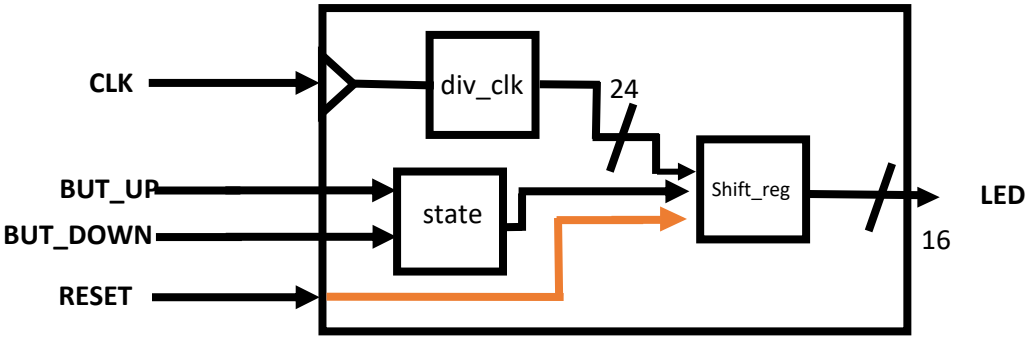
- **RST** et **CLK** initialisent le signal qui est un registre à décalage de 24 bits.
  Initialisé à 0 si **RST = 1**
  Incrémenté de 1 à chaque front montant de **CLK**
- **BUT_LEFT :**
- **BUT_RIGHT :**

et une sortie (LED) qui est un vecteur de 16 bits.

Les signaux **RST**, **BUT_LEFT**, **BUT_RIGHT**, et **compteur(23)** servent à initialiser le signal **state**, qui est un vecteur de 4 bits. **state** représente l'état actuel du compteur.
Si :

- **compteur(23)** est égal à 1 (ce qui se produit après 16 cycles de **CLK)**,
  les boutons **BUT_LEFT** et **BUT_RIGHT** sont utilisés pour incrémenter ou décrémenter l'état.
  state est initialisé à zéro lorsque **RST** est égal à 1.
- Si **state** est égal à 1111, il ne peut plus être incrémenté, et si **state** est égal à 0000, il ne peut plus être décrémenté.

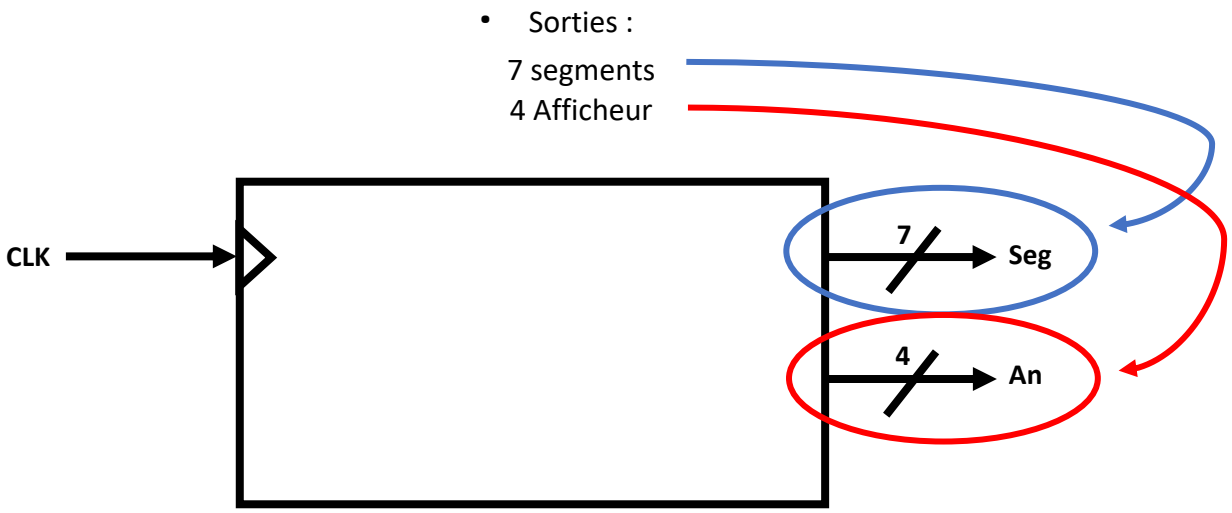
Le dernier processus utilise le signal **state** pour mettre à jour le signal LED. Le processus utilise une instruction case pour assigner la valeur appropriée à LED en fonction de la valeur de **state**. Si **state** est égal à 1 des 16 états possibles de 0000 à 1111, le signal LED correspondant est assigné. Si **state** est différent de ces états, LED est initialisé à zéro.

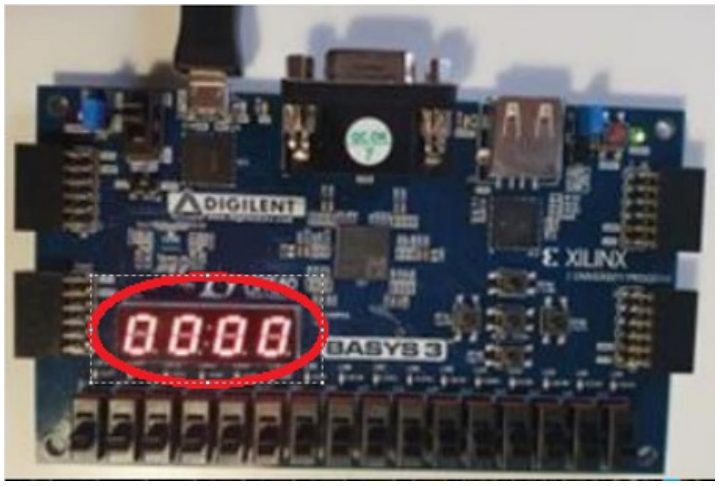


Sujet 4 : Affichage de 4 chiffres identiques sur les afficheurs.

Le code ci-après présente un afficheur à sept segments qui affiche seulement le chiffre 0 continuellement. Il est composé de deux processus :

- Le premier processus implémente un diviseur de fréquence. Il prend en entrée le signal d'horloge **CLK** et génère un signal **clk_div** qui est un compteur binaire de 4 bits. Chaque fois que le front montant de CLK est détecté, le compteur est incrémenté de 1. Ainsi, la fréquence du signal **clk_div** est divisée par 16 par rapport à celle du signal **CLK**.
- Le deuxième processus est un compteur qui prend en entrée le bit de poids fort du signal **clk_div** (clk_div(3)). Le signal de sortie an (qui contrôle les afficheurs) est initialisé à "0000", ce qui signifie que tous les afficheurs sont allumés. Le signal de sortie seg (qui contrôle les segments) est initialisé à "1000000", ce qui correspond à l'affichage du chiffre 0 sur un afficheur à sept segments.





Sujet 5 : Contrôle d'Afficheurs à 4 Chiffres

Ce module vise à contrôler un afficheur à 4 chiffres à 7 segments via un signal d'horloge (`CLK`). Il produit des signaux de contrôle pour les segments (`seg`) et chiffres (`an`) de l'afficheur.

Dans l'architecture `rtl`, `clk_div` divise `CLK` en quatre, permettant d'afficher un chiffre différent par phase. `clk_lent` ralentit `CLK`, assurant un affichage suffisamment long pour chaque chiffre.

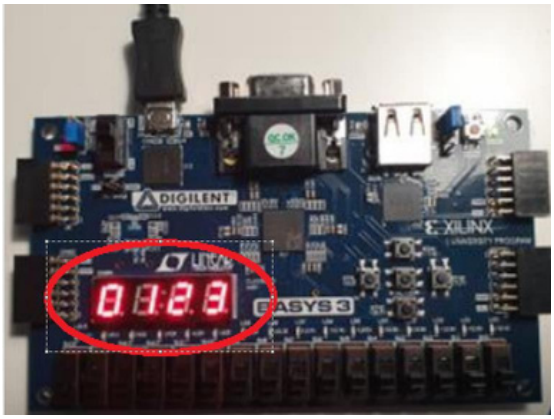
Processus :

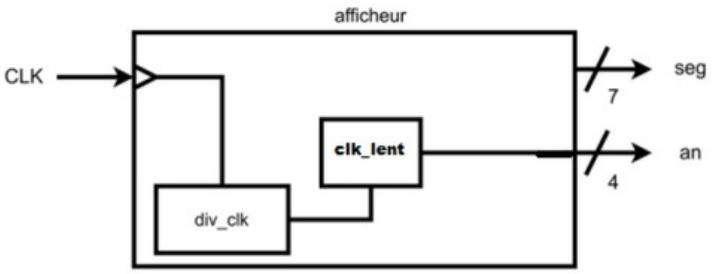
Premièrement, le signal `CLK` génère `clk_lent` et `clk_compte`, incrémenté à chaque cycle d'horloge jusqu'à 50 000 puis réinitialisé. Le signal `clk_lent` est défini sur '0' pendant la première moitié de chaque cycle de `clk_compte` et sur '1' pendant la seconde moitié.

Deuxièmement, `clk_lent` génère `clk_div`, qui compte jusqu'à 3 puis se réinitialise.

Finalement, `clk_div` contrôle `seg` et `an` via `case`, attribuant à `seg` et `an` des valeurs basées sur `clk_div`. Si `clk_div` ne correspond pas à "00" à "11", `an` est réinitialisé à "1111".

Schéma block :





Sujet 6 : Chronomètre

Ce module affiche les minutes et les secondes écoulées depuis le lancement sur les afficheurs 7 segment

Processus :

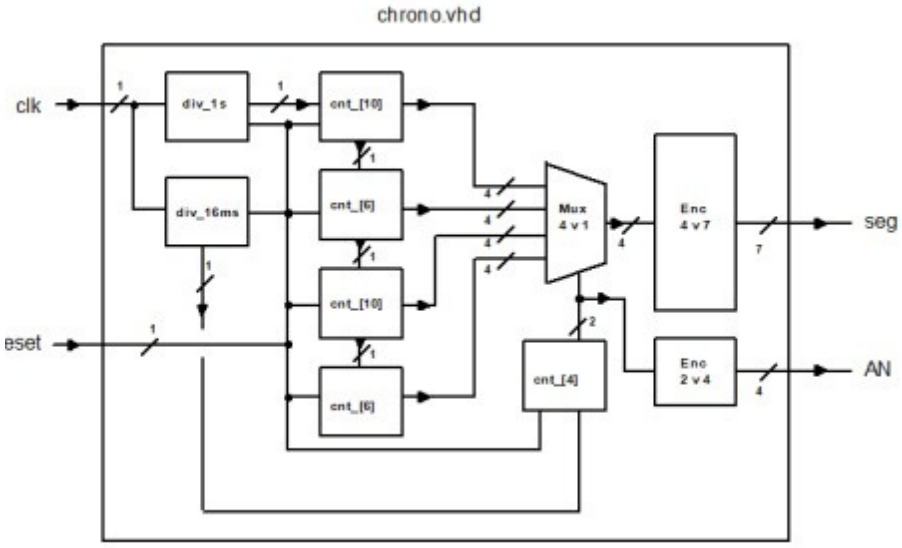
1. Le premier génère `clk_1s` et `clk_16ms` à partir de `CLK` et les réinitialise si `RST` est activé.

2. Les suivants ("clock divider : afficheurs" et "clock divider : chiffres") incrémente les compteurs `clk_div` et `sec_l`, `sec_h`, `min_l`, `min_h` selon `clk_16ms` et `clk_1s`. Ils sont réinitialisés à une certaine valeur ou si reset est activé.

3. Le dernier gère l'affichage sur l'afficheur à 7 segments. Il choisit le chiffre actif selon `clk_div` et allume les segments appropriés en fonction de `sec_h`, `sec_l`, `min_h`, `min_l`.

Pour l'incrémentation des signaux liés au temps, on utilise un diviseur d'horloge qui divise CLK par 50000000 afin d'avoir 1Hz

Chaque état a été codé manuellement pour simplifier le dernier processus.



Sujet 7 : Image fixe monochrome sur port VGA

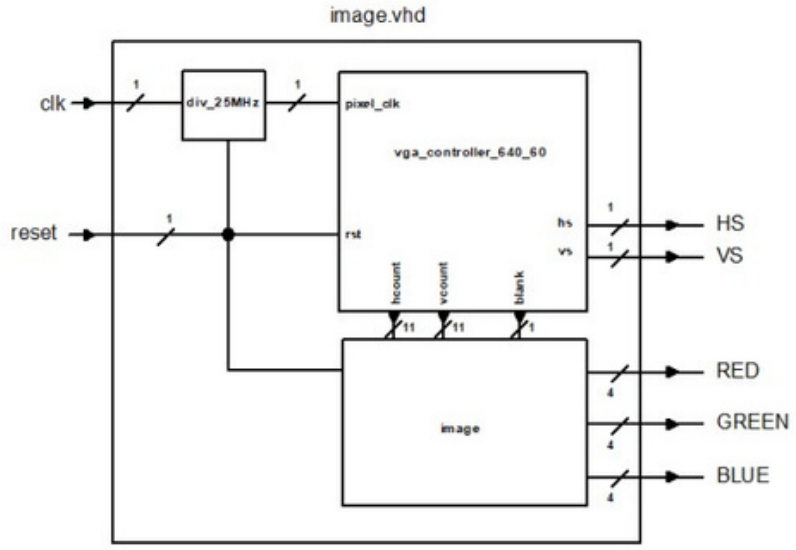
Ce module affiche une image fixe monochrome sur un port VGA

Pour commencer, l'approche nécessite l'usage d'un diviseur d'horloge. Ce dispositif est crucial pour modérer la fréquence d'horloge à 25 Mhz. C'est une étape essentielle, car l'affichage sur un port VGA ne serait pas réalisable avec une fréquence supérieure. Ainsi, le diviseur d'horloge assure que l'horloge opère à une vitesse compatible avec le port VGA.

Ensuite, nous introduisons `hcount`, qui est un compteur horizontal. Il est configuré pour s'incrémenter à chaque cycle de l'horloge réduite à 25 Mhz. Le compteur `hcount` continue à s'incrémenter jusqu'à atteindre la valeur de 640, qui correspond à la largeur de l'image en termes de pixels. Lorsque cette valeur est atteinte, `vcount`, qui est un compteur vertical, s'incrémente à son tour. De cette façon, `hcount` et `vcount` permettent de balayer tous les pixels de l'image, chacun correspondant à une coordonnée spécifique.

Après cette opération, chaque pixel de l'image est prêt à recevoir une couleur. La couleur est déterminée par trois paramètres : RED, GREEN, et BLUE (RGB), qui sont définis explicitement dans le code. En utilisant ces paramètres, nous pouvons attribuer à chaque pixel une couleur spécifique.

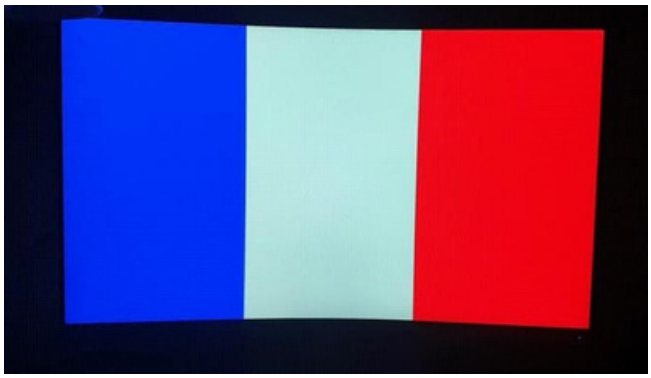
Dans les cas où aucune couleur n'est explicitement attribuée à un pixel, une valeur par défaut est utilisée. Cette valeur par défaut, appelée **"blank"**, donne à chaque pixel sans couleur attribuée la couleur blanche. De plus, en cas de réinitialisation, tous les pixels de l'image sont automatiquement attribués à cette couleur blanche, garantissant ainsi qu'aucun pixel ne reste sans couleur.



Sujet 8 : Image fixe 3 couleurs différents sur port VGA.

Pour ce sujet on va reprendre exactement le même principe que pour le sujet précédent, à l'exception que l'on va rajouter plusieurs conditions sur nos différents signaux **« hcount** » et **« vcount »** .

Ici on représente le drapeau de la France donc il faut séparer noter écran en 3 pour pouvoir afficher une couleur dans chaque partie, ainsi quand **« hcount »** dépasse 213 puis 426 notre écran est divisé on a plus qu'à remplir la partie en réglant RED, BLUE et GREEN.



Conclusion :

Grâce à ce projet j'ai pu apprendre à coder en VHDL et à me servir du logiciel Vivado ainsi que modelsim .

De plus ,j'ai pu appliquer la théorie qu'on a vu en module de logique durant le premier semestre (S5).