POLYTECH LILLE

Département : Systèmes embarqués

Spécialité : Internet Of Things

Professeur : Monsieur N.Wichmann

RAPPORT TP CNSPS

Réalisé par :

CRUCHET Simon – DUHR Romain

Année universitaire : 2023-2024

Sommaire :

Introduction :

Ce rapport présente un projet de développement en VHDL visant à créer un afficheur 7 segments et un afficheur VGA, chacun étant divisé en trois sections distinctes : le but du projet, les I/O utilisées et les résultats obtenus. L'ensemble du code source, des images, des vidéos ainsi que les ressources associées, sont disponibles sur un dépôt Git dédié.

1. But du Projet

La première partie de ce rapport expose en détail l'objectif principal du projet, mettant en lumière les fonctionnalités attendues et les avantages attendus de la mise en œuvre de ces deux afficheurs. Cette section définit clairement le contexte du projet et son importance dans le domaine de l'affichage électronique.

2. I/O Utilisées

La deuxième partie se penche sur les entrées/sorties (I/O) essentielles utilisées dans le développement de l'afficheur 7 segments et de l'afficheur VGA. Elle détaille les connexions matérielles nécessaires, les broches FPGA impliquées, ainsi que toute autre interférence avec les composants externes, le tout dans le but d'assurer une compréhension complète de l'architecture

3. Résultats

La dernière section de ce rapport est dédiée à l'examen des résultats obtenus à la suite du développement du code VHDL pour les afficheurs 7 segments et VGA. Elle inclut une analyse des performances, des captures d'écran, des données de test et d'autres informations pertinentes, démontrant ainsi la conformité aux objectifs du projet.

L'afficheur 7 segments :

But du projet :

Nous avons décidé de nous intéresser à l'affichage 7 segments grâce au but suivant :

Créer un compteur de 0 à 9999 à l'aide des 4 afficheurs. En cas de dépassement ou d'appuyé sur un bouton reset le compteur redémarrera à 0.

I/O utilisées :

Tout d'abord, voyons les entrées/sortie que nous avons utilisé.

```
entity display is
     port {
     clk_fpga : in STD_LOGIC;
     reset : in STD_LOGIC;
     aff : out STD_LOGIC_VECTOR{7 downto 0};
     an : out STD_LOGIC_VECTOR{3 downto 0}
     };
end display;
```

- clk_fpga : le signal de clock du fpga

Dans le fichier de contrainte :

```
   ## Clock signal
   set_property -dict { PACKAGE_PIN W5   IOSTANDARD LVCMOS33 } [get_ports
clk_fpga]
   create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5}
[get_ports clk]
```

- reset : le signal permettant de remettre le compteur à zéro grâce à un bouton

Dans le fichier de contrainte :

```
## Switches
  set_property -dict { PACKAGE_PIN R2    IOSTANDARD LVCMOS33 } [get_ports
reset]
```

- aff : un tableau de 8 bits relier à chaque led d'un segment à afficher (en ajoutant le point de l'afficheur)

```
##7 Segment Display
   set_property -dict { PACKAGE_PIN W7   IOSTANDARD LVCMOS33 } [get_ports
{aff[0]}]
   set_property -dict { PACKAGE_PIN W6   IOSTANDARD LVCMOS33 } [get_ports
{aff[1]}]
   set_property -dict { PACKAGE_PIN U8   IOSTANDARD LVCMOS33 } [get_ports
{aff[2]}]
   set_property -dict { PACKAGE_PIN V8   IOSTANDARD LVCMOS33 } [get_ports
{aff[3]}]
   set_property -dict { PACKAGE_PIN U5   IOSTANDARD LVCMOS33 } [get_ports
{aff[4]}]
   set_property -dict { PACKAGE_PIN V5   IOSTANDARD LVCMOS33 } [get_ports
{aff[5]}]
   set_property -dict { PACKAGE_PIN U7   IOSTANDARD LVCMOS33 } [get_ports
{aff[6]}]

   set_property -dict { PACKAGE_PIN V7   IOSTANDARD LVCMOS33 } [get_ports
{aff[7]}]
```

Exemple : aff à "11111110" allumera la led du point de l'afficheur séléctionné.

- an : un tableau comprenant 4 bits permettant de sélectionner l'un des 4 afficheurs 7 segments

Dans le fichier de contrainte :

```
   set_property -dict { PACKAGE_PIN U2   IOSTANDARD LVCMOS33 } [get_ports
{an[0]}]
   set_property -dict { PACKAGE_PIN U4   IOSTANDARD LVCMOS33 } [get_ports
{an[1]}]
   set_property -dict { PACKAGE_PIN V4   IOSTANDARD LVCMOS33 } [get_ports
{an[2]}]
   set_property -dict { PACKAGE_PIN W4   IOSTANDARD LVCMOS33 } [get_ports
{an[3]}]
```

Exemple : an à "1110" sélectionnera l'afficheur le plus à droite.

Explications de l'algorithme :

```
   architecture Behavioral of display is

   signal count_an : integer range 3 downto 0 := 0;
   signal clk_enable : integer range 4999 downto 0 := 0;
   signal clk_counter : integer range 2999999 downto 0 := 0;
   constant nb0 : std_logic_vector(7 downto 0) := "11000000";
   constant nb1 : std_logic_vector(7 downto 0) := "11111001";
   constant nb2 : std_logic_vector(7 downto 0) := "10100100";
   constant nb3 : std_logic_vector(7 downto 0) := "10110000";
   constant nb4 : std_logic_vector(7 downto 0) := "10011001";
   constant nb5 : std_logic_vector(7 downto 0) := "10010010";
   constant nb6 : std_logic_vector(7 downto 0) := "10000010";
   constant nb7 : std_logic_vector(7 downto 0) := "11111000";
   constant nb8 : std_logic_vector(7 downto 0) := "10000000";
   constant nb9 : std_logic_vector(7 downto 0) := "10010000";
   constant seg0 : std_logic_vector(3 downto 0) := "1110";
   constant seg1 : std_logic_vector(3 downto 0) := "1101";
   constant seg2 : std_logic_vector(3 downto 0) := "1011";
   constant seg3 : std_logic_vector(3 downto 0) := "0111";
   signal chiffre4 : integer range 9 downto 0 := 0;
   signal chiffre3 : integer range 9 downto 0 := 0;
   signal chiffre2 : integer range 9 downto 0 := 0;
   signal chiffre1 : integer range 9 downto 0 := 0;

   type mynumbers is array(9 downto 0) of std_logic_vector(7 downto 0);
   signal numbers : mynumbers :=
(nb9,nb8,nb7,nb6,nb5,nb4,nb3,nb2,nb1,nb0);
```

On a:

- count_an : compte pour effectuer les opérations sur les afficheurs un par un.
- clk_enable : diviser la clock pour l'affichage et évite les recouvrements.
- clk_counter : divise la clock pour le compteur et lui évite de compter bien trop rapidement.
- nb* : représentation des chiffres par leurs segments.
- seg* : attribue chaque segment à une valeur constante.
- chiffre* : variable représentant le chiffre en * position
- numbers : associe la représentation logique d'un chiffre à un entier.

Les process

- 1er process :

```
  Begin

  -- display process

  process(clk_fpga)

begin
     if clk_fpga'event and clk_fpga = '1' then
        if clk_enable = 4999 then
           clk_enable <= 0;
           if count_an = 0 then
              aff <= numbers(chiffre4);
              an <= seg0;
              count_an <= count_an + 1;
           elsif count_an = 1 then
              aff <= numbers(chiffre3);
              an <= seg1;
              count_an <= count_an + 1;
           elsif count_an = 2 then
              aff <= numbers(chiffre2);
              an <= seg2;
              count_an <= count_an + 1;
           elsif count_an = 3 then
              aff <= numbers(chiffre1);
              an <= seg3;
              count_an <= 0;
           end if;
        else
           clk_enable <= clk_enable + 1;

       end if;

     end if;

  end process;
```

Ce premier process permet à chaque tick de clock + diviseur d'afficher un chiffre sur le premier afficheur. Au prochain tick sur le second puis le 3ème et enfin le 4ème. Ensuite l'algorithme boucle

- le diviseur (clk_enable) ralentit la cadence de switch entre afficheur. Si l'on ne divise du fait de la vitesse les chiffres ne s'affichent pas correctement.

Ainsi la logique et l'électronique étant ici très rapide devant la fréquence de perception de l'oeil humain, l'utilisateur à l'impression que l'affichage et statique et que rien ne clignote.

- 2ème processus :

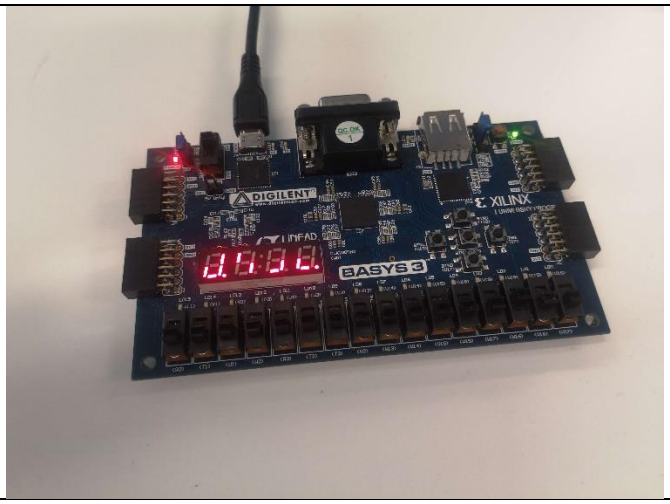
```
-- process counter
process(clk_fpga)
begin
   if clk_fpga'event and clk_fpga = '1' then
      if clk_counter = 2999999 then
         clk_counter <= 0;
         if reset = '1' then
            chiffre1 <= 0;
            chiffre2 <= 0;
            chiffre3 <= 0;
            chiffre4 <= 0;
         else if chiffre4 = 9 then
            chiffre4 <= 0;
            if chiffre3 = 9 then
               chiffre3 <= 0;
               if chiffre2 = 9 then
                  chiffre2 <= 0;
                     if chiffre1 = 9 then
                        chiffre1 <= 0;
                     else
                        chiffre1 <= chiffre1 + 1;
                     end if;
               else
                  chiffre2 <= chiffre2 + 1;
               end if;
            else
               chiffre3 <= chiffre3 + 1;
            end if;
         else
            chiffre4 <= chiffre4 + 1;
         end if;
      else
         clk_counter <= clk_counter + 1;
      end if;
   end if;
end process;
end Behavioral;
```

Ce processus permet de compter avec un diviseur de la clock de base.

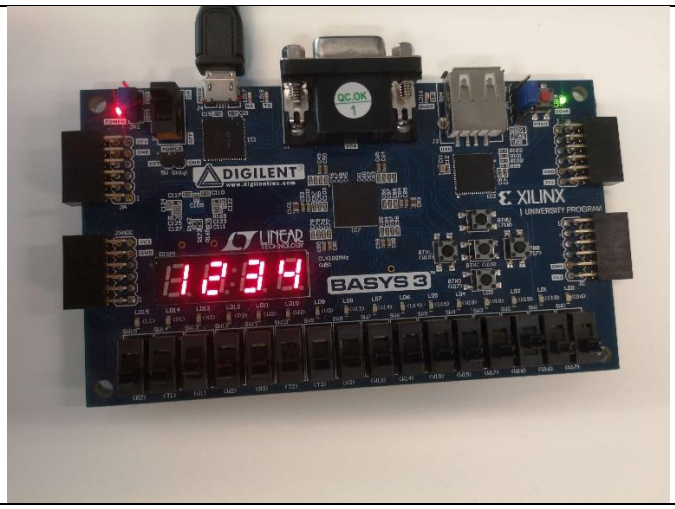
Lorsque le chiffre tout à droite et à 9 et qu'il doit être incrémenter alors il passe à 0. Si le chiffre à sa gauche n'est pas à 9 alors il est incrémenté sinon il passe à 0 et on regarde le chiffre de gauche en suivant le même algorithme.

Si on a 9999 ou un 1 logique sur reset alors tous les chiffres repassent à 0.

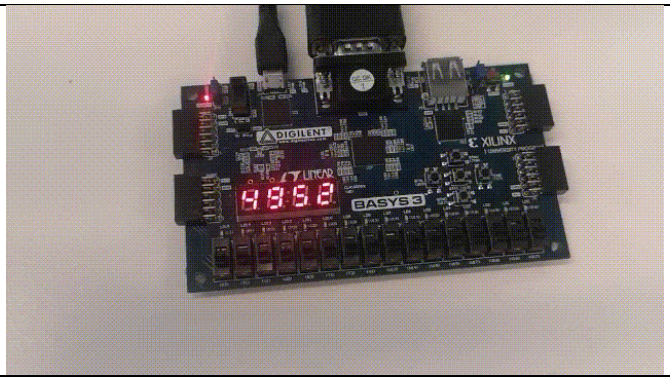
Résultats :



Nous avons dans un premier temps essayer d'afficher sur les 4 afficheurs sans recouvrement. C'est pour cela qu'il a fallu baisser la clock.



Ensuite il a fallu donner du sens à ce que nous affichions. C'est pourquoi nous avons tenté d'afficher 1 2 3 4.



Enfin il a fallu implémenter un simple algorithme pour le compteur.

L'afficheur VGA :

But du projet :

Nous avons eu comme projet d'afficher différentes choses sur un écran en passant par son port VGA. Nous allons commencer à faire un écran de couleur puis afficher une forme puis la faire bouger.

I/O utilisées :

Tout d'abord, voyons les entrées/sortie que nous avons utilisées.

```
entity vga_controller is
    Port (clk_fpga : in std_logic;
          sw : in std_logic_vector(15 downto 0);
          vgaRed : out std_logic_vector(3 downto 0);
          vgaGreen : out std_logic_vector(3 downto 0);
          vgaBlue : out std_logic_vector(3 downto 0);
          Hsync : out std_logic;
          Vsync : out std_logic);
end vga_controller;
```

- clk_fpga : le signal de clock du fpga
- sw : interrupteurs pour commander les couleurs
- vgaRed/Green/Blue : permet de commander la couleur à afficher
- Hsyn : synchronisation horizontale
- Vsyn : synchronisation verticale

Dans le fichier de contrainte :

Cela permet associer les interrupteurs au tableau sw.

```
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports
{sw[0]}]
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports
{sw[1]}]
set_property -dict { PACKAGE_PIN W16   IOSTANDARD LVCMOS33 } [get_ports
{sw[2]}]
set_property -dict { PACKAGE_PIN W17   IOSTANDARD LVCMOS33 } [get_ports
{sw[3]}]
set_property -dict { PACKAGE_PIN W15   IOSTANDARD LVCMOS33 } [get_ports
{sw[4]}]
set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports
{sw[5]}]
set_property -dict { PACKAGE_PIN W14   IOSTANDARD LVCMOS33 } [get_ports
{sw[6]}]
set_property -dict { PACKAGE_PIN W13   IOSTANDARD LVCMOS33 } [get_ports
{sw[7]}]
set_property -dict { PACKAGE_PIN V2    IOSTANDARD LVCMOS33 } [get_ports
{sw[8]}]
```

```
set_property -dict { PACKAGE_PIN T3    IOSTANDARD LVCMOS33 } [get_ports
{sw[9]}]
set_property -dict { PACKAGE_PIN T2    IOSTANDARD LVCMOS33 } [get_ports
{sw[10]}]
set_property -dict { PACKAGE_PIN R3    IOSTANDARD LVCMOS33 } [get_ports
{sw[11]}]
set_property -dict { PACKAGE_PIN W2    IOSTANDARD LVCMOS33 } [get_ports
{sw[12]}]
set_property -dict { PACKAGE_PIN U1    IOSTANDARD LVCMOS33 } [get_ports
{sw[13]}]
set_property -dict { PACKAGE_PIN T1    IOSTANDARD LVCMOS33 } [get_ports
{sw[14]}]
set_property -dict { PACKAGE_PIN R2    IOSTANDARD LVCMOS33 } [get_ports
{sw[14]}]
```

On associe tous les pins du connecteur VGA avec nos signaux pour gérer l'affichage

```
set_property -dict { PACKAGE_PIN G19   IOSTANDARD LVCMOS33 } [get_ports
{vgaRed[0]}]
set_property -dict { PACKAGE_PIN H19   IOSTANDARD LVCMOS33 } [get_ports
{vgaRed[1]}]
set_property -dict { PACKAGE_PIN J19   IOSTANDARD LVCMOS33 } [get_ports
{vgaRed[2]}]
set_property -dict { PACKAGE_PIN N19   IOSTANDARD LVCMOS33 } [get_ports
{vgaRed[3]}]
set_property -dict { PACKAGE_PIN N18   IOSTANDARD LVCMOS33 } [get_ports
{vgaBlue[0]}]
set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33 } [get_ports
{vgaBlue[1]}]
set_property -dict { PACKAGE_PIN K18   IOSTANDARD LVCMOS33 } [get_ports
{vgaBlue[2]}]
set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 } [get_ports
{vgaBlue[3]}]
set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33 } [get_ports
{vgaGreen[0]}]
set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports
{vgaGreen[1]}]
set_property -dict { PACKAGE_PIN G17   IOSTANDARD LVCMOS33 } [get_ports
{vgaGreen[2]}]
set_property -dict { PACKAGE_PIN D17   IOSTANDARD LVCMOS33 } [get_ports
{vgaGreen[3]}]
set_property -dict { PACKAGE_PIN P19   IOSTANDARD LVCMOS33 } [get_ports
Hsync]
set_property -dict { PACKAGE_PIN R19   IOSTANDARD LVCMOS33 } [get_ports
Vsync]
```

Explications de l'algorithme:

Pour que le code soit compréhensible facilement, nous avons créé plusieurs process pour les différentes utilités.

Les signaux :

```
signal clk_65MHz : std_logic;
signal Hcount : integer range 1344 downto 1 := 1;
signal Vcount : integer range 1083264 downto 1 := 1;
signal myRed: std_logic_vector(3 downto 0);
signal myBlue: std_logic_vector(3 downto 0);
signal myGreen: std_logic_vector(3 downto 0);

signal leftLimit: integer range 1025 downto 0:= 0;
signal rightLimit: integer range 1025 downto 0:= 100;
signal upLimit: integer range 768 downto 0:= 0;
signal downLimit: integer range 768 downto 0:= 100;
signal pixelX: integer range 1344 downto 1:= 1;
signal pixelY: integer range 768 downto 1;
```

- *Limit : Permet de délimiter les coordonnées du carré
- pixel* : Désigne la coordonnée du pixel que nous affichons

Les process :

Ce process permet de suivre les différents signaux de synchronisation verticale et horizontale grâce à une horloge 65MHz. Les signaux suivent le graphique vu en tp.

```
process(clk_65MHz)
begin
if clk_65MHz'event and clk_65MHz = '1' then

-- Gestion du vga

    -- Syncro horizontale
    if Hcount < 1025 then
        -- affichage
        Hsync <= '1';
        Hcount <= Hcount + 1;
    elsif Hcount  < 1049 then
        -- Fp
        Hsync <= '1';
        Hcount <= Hcount + 1;
    elsif Hcount < 1185 then
        -- Pw
        Hsync <= '0';
        Hcount <= Hcount + 1;
    elsif Hcount < 1344 then
        -- Bp
        Hsync <= '1';
        Hcount <= Hcount + 1;
    else
```

```
        Hcount <= 1;
    end if;

    -- Synchro vertical
    if Vcount < 1032193 then
        -- Affichage de toutes les lignes
        Vsync <= '1';
        Vcount <= Vcount + 1;
    elsif Vcount < (1032193 + 4032) then
        --Fp
        Vsync <= '1';
        Vcount <= Vcount + 1;
    elsif Vcount < (1032193 + 4032 + 8064) then
        --Pw
        Vsync <= '0';
        Vcount <= Vcount + 1;
    elsif Vcount < (1032192 + 4032 + 8064 + 38976) then
        --Bp
        Vsync <= '1';
        Vcount <= Vcount + 1;
    else
        Vcount <= 1;
    end if;
end if;
end process;
```

Ce process prend en charge l'affichage du fond de l'écran et ainsi que l'affichage du carré. En fonction de la position du carré, les couleurs vont changer avec un blanc pour le fond et la couleur sélectionnée pour le carré.

```
process(clk_65MHz)
-- Gestion de l'affichage
begin
if clk_65MHz'event and clk_65MHz = '1' then
    if  pixelX > leftLimit and pixelX < rightLimit and pixelY > upLimit
and pixelY < downLimit then
    -- Carré
        vgaRed <= myRed;
        vgaBlue <= myBlue;
        vgaGreen <= myGreen;
    elsif Hcount < 1025 and pixelY < 769 then
    -- Background
        vgaRed <= (others => '1');
        vgaBlue <= (others => '1');
        vgaGreen <= (others => '1');
    else
    -- Hors pixel
        vgaRed <= (others => '0');
        vgaBlue <= (others => '0');
        vgaGreen <= (others => '0');
    end if;
end if;
end process;
```

Ce process permet de gérer le mouvement de notre carré grâce à ces coordonnées. Il suit se déplace vers la droite puis dès que le carré touche le bord droit, retourne tout à gauche et est décalé vers le bas puis continue son chemin. Sinon le carré retourne à sa position initiale.

```
process(clk_65MHz)
begin
if clk_65MHz'event and clk_65MHz = '1' then
    if Vcount = 1032193 then
        if rightLimit > 1024 and downLimit > 768 then
            leftLimit <= 0;
            rightLimit <= 100;
            upLimit <= 0;
            downLimit <= 100;
        elsif rightLimit > 1024 then
            leftLimit <= 0;
            rightLimit <= 100;
            upLimit <= upLimit + 1;
            downLimit <= downLimit + 1;
        else
            leftLimit <= leftLimit + 1;
            rightLimit <= rightLimit +1;
        end if;
    end if;
end if;
end process;
```

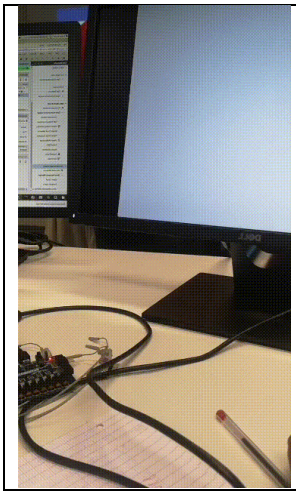
Ce process permet de simplifier des valeurs pour la gestion d'écran.

```
process(clk_65MHz)
-- Simplification des valeurs d'écran
begin
if clk_65MHz'event and clk_65MHz = '1' then
    if Vcount = 1 then
        pixelY <= 1;
    elsif Hcount = 1025 then
        pixelY <= pixelY + 1;
    end if;
end if;
end process;
```

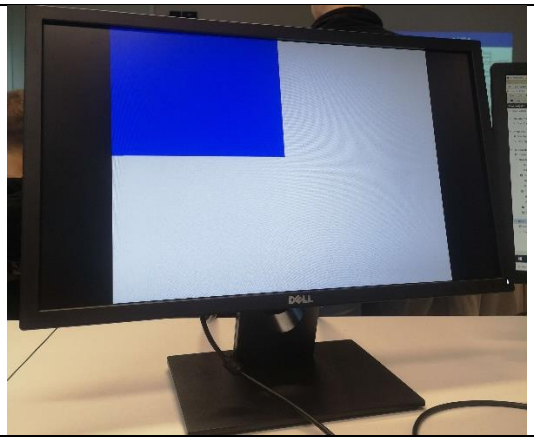
Résultats

- Nous avons mis du temps pour afficher notre première image de couleur car nous n'avions pas vu que les vecteurs RGB devaient être à 0 en dehors de l'affichage à l'écran.

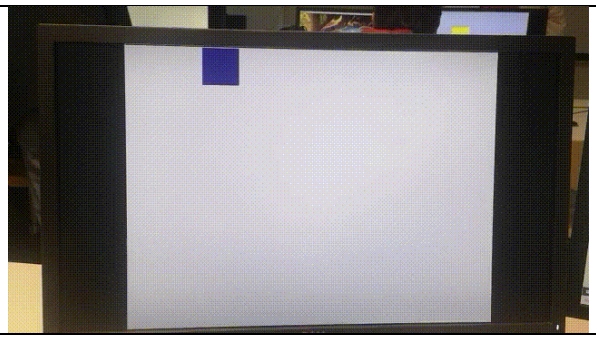
- Nous avons dans un premier temps réussi à afficher un écran dont la couleur est commandée par 12 interrupteurs (4 interrupteurs par couleurs)



Puis nous avons réussi à afficher un carré dans un coin de l'écran (toujours commandé en couleurs par les interrupteurs)



Finalement avec la séparation du code en process et en ajoutant des fonctionnalités, nous avons réussi à animer le carré automatiquement qui se déplace de droite à gauche et revient à gauche dès qu'il touche le coin droit tout en descendant.



Résultat final.

Conclusion :

Au terme de ce TP, nous pouvons tirer de nombreuses conclusions positives qui témoignent de notre progression dans le domaine de la conception VHDL et de l'utilisation des FPGA. Cette expérience enrichissante nous a offert une série d'enseignements précieux qui contribuent de manière significative à notre formation.

Tout d'abord, ce TP nous a ouvert les portes de l'application pratique du langage VHDL et de la programmation FPGA. Grâce à ce projet, nous avons pu passer de la théorie à la pratique, en utilisant ces outils pour utiliser des afficheurs 7 segments et port VGA. Cette expérience concrète nous a permis de mieux comprendre comment les concepts théoriques se traduisent dans des applications réelles, ce qui est essentiel pour quiconque souhaite travailler dans le domaine de la conception électronique.

Enfin, ce TP a renforcé notre compréhension des concepts du VHDL et des FPGA. En manipulant directement ces technologies, nous avons pu approfondir nos connaissances et résoudre des problèmes réels, ce qui a eu un impact positif sur notre maîtrise de ces sujets.

En somme, ce TP a été une étape cruciale dans notre parcours d'apprentissage. Nous avons acquis des compétences pratiques, amélioré notre compréhension des concepts clés, et pris de l'assurance dans l'utilisation du VHDL et des FPGA. Cette expérience constitue un jalon important vers notre capacité à relever des défis plus complexes dans le futur, et elle nous a apporté une base solide pour notre croissance continue dans le domaine de la conception électronique.