Projet VHDL

ISEN
ALL IS DIGITAL!

yncréa

Theo DE SAUVAGE, Eva GAUTHIER, Anthony PHILIPPE, Leo VILLENEUVE

SOMMAIRE

1 . PRESENTATION GENERALE

a. Présentation

Le VHDL est un langage de description de matériel destiné à représenter le comportement ainsi que l'architecture d'un système numérique.

a. Répartition

Pour réaliser ce projet, nous nous sommes divisé le travail en groupe/seul afin d'arriver au bout du projet. Pour mieux appréhender Vivado, nous avons fait le projet 1 et 2 chacun de son côté afin de bien prendre en main ce nouveau langage. A la suite de ce sujet, nous nous sommes repartit le travail. Ainsi Leo et Anthony ont travaillé ensemble. Anthony s'est concentré sur le sujet 5 tandis que Léo s'est concentré sur le sujet 6. L'autre équipe était composé de Théo et Eva. Théo a réalisé les sujets 3 et 4 tandis que Eva a fait les sujets 7 et 8. Le prjet 9 étant plus complexe, nous l'avons réalisé tous les 4.

b. Information

Tout d'abord, dans tous les projets nous avons utilisés les mêmes librairies qui sont :

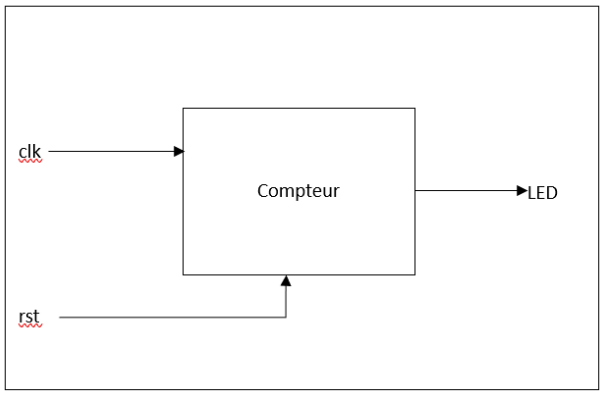
```
Library IEEE ;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL ;
use IEEE.STD_LOGIC_UNSIGNED.ALL ;
```

Nous avons également utilisé le même fichier BASYS qui permet d'initialiser les composants à utiliser comme la clock, les LEDs, les 7 segments, les an et les boutons. Ainsi, pour chaque sujet nous mettions à jour ce sujet pour que notre code s'affiche ou nous le souhaitions.

2. PROJET 1 : CLIGNOTEMENT D'UNE LED

a. Présentation

L'objectif du sujet est de faire clignoter une led. Pour cela nous avons programmé la clock pendant une courte durée et après nous avons programmé la led pour qu'elle s'allume. Cette alternance entre allumé-éteinte a produit leqlignotement.



b. Explication du code

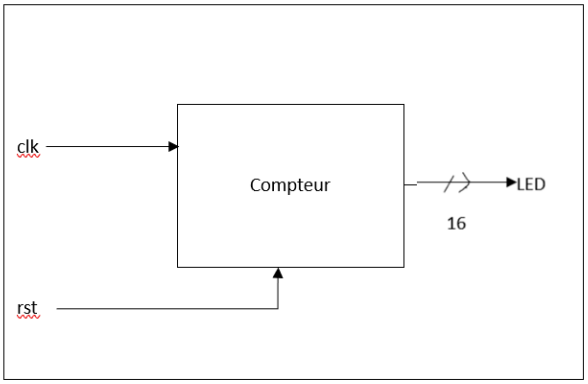
Voici le processus général et principal du programme pour allumer une led. Afin de créer le clignotement, on le répète à l'infini :

```
process (CLK,RST)
begin
    if (RST = '1') then
        cnt <= (others => '0');
    elsif (CLK'event and CLK = '1') then
        cnt <= cnt +1;
    end if;
end process;


process(cnt(23), rst)
begin
    if (RST = '1') then
        cnt_out <= (others => '0');
    elsif (cnt(23)'event and cnt(23) = '1') then
        cnt_out <= cnt_out +1;

    end if;
end process;
```

3. PROJET 2 : CLIGNOTEMENT D'UNE LED

a. Présentation

L'objectif est de coder un chenillard de LED sur la carte. Pour cela nous avons besoin de la clock en entrée, puis du reset qui nous permet de réinitialiser le chenillard au début. Puis en sortie nous avons utilisé les 16 leds de la carte. C'est à dire allumer alternativement les LEDS de la LD0 à LD15.



b. Explication du code

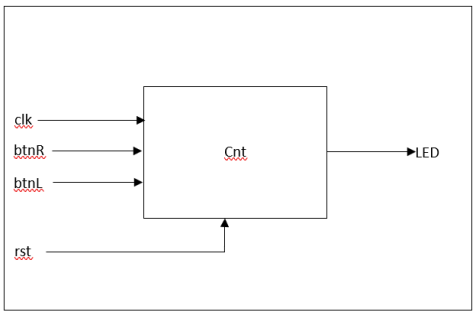
Pour ce code, on a implémenté la variable *cnt_out* qui correspond aux 32 états soit 2^5 états. L'horloge rythme le changement d'état, et c'est la variable *cnt_out* qui va changer d'état et va allumer et éteindre une led pour créer ce chenillard. Une fois les 16 leds allumées, les états suivants vont éteindre une à une les leds pour revenir au départ.

```
process (CLK,RST)
begin
    if (RST = '1') then
        cnt <= (others => '0');
    elsif (CLK'event and CLK = '1') then
        cnt <= cnt +1;
    end if;
end process;


process(cnt(23), rst)
begin
    if (RST = '1') then
        cnt_out <= (others => '0');
    elsif (cnt(23)'event and cnt(23) = '1') then
        cnt_out <= cnt_out +1;

    end if;
end process;
```

4. PROJET 3 : COMPTEUR ET DECOMPTEUR

a. Présentation

Le but du projet 3, est de gérer un allumage de led sur la carte avec des boutons poussoirs. Pour incrémenter les leds à gauche on utilise le *btnL* et pour incrémenter les leds à droite, on utilise le *btnR.* Il utilise deux processus, le premier qui met à jour la variable *cnt* et le deuxième qui est déclenché uniquement lorsque le bit le plus significatif de la variable *cnt* change. Pour résumer, Finalement, la sortie LED est mise à jour en fonction de la variable LED_int. Cela allume ou éteint les LED selon l'état de LED_int.



b. Explication du code

Il utilise deux processus, le premier qui met à jour la variable *cnt,* qui est utilisé pour le décalage de l'horloge et le deuxième qui est déclenché uniquement lorsque le bit le plus significatif de la variable *cnt* change. Pour résumer, Finalement, la sortie LED est mise à jour en fonction de la variable LED_int. Cela allume ou éteint les LED selon l'état de LED_int.

```
process (CLK,RST)
begin
    if (RST = '1') then
        cnt <= (others => '0');
    elsif (CLK'event and CLK = '1') then
        cnt <= cnt +1;
    end if;
end process;


process(cnt(23), RST, btnL, btnR)
begin
    if (RST = '1') then
        LED_int <= "0000000000000000";
    elsif (cnt(23)'event and cnt(23) = '1') then
        if(btnL = '1') then
            LED_int <= LED_int (14 downto 0) & '1';
        elsif(btnR = '1') then
            LED_int <= '0' & LED_int (15 downto 1);
        end if;
    end if;

LED <= LED_int;
end process;
end rtl;
```

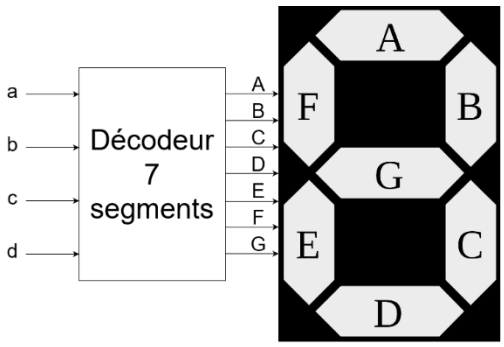
5. PROJET 4 : AFFICHAGE 4 SEGMENTS IDENTIQUES

a. Explication sur l'affichage

Dans ce projet et celui qui suit, on utilise un décodeur et un afficheur. Nous allons avoir besoin d'une clk et d'un rst. Pour créer une entité comportant ces deux éléments ainsi qu'en sorties les 4 signaux d'anode qui définissent la position des chiffres et les 7 cathodes à segments.

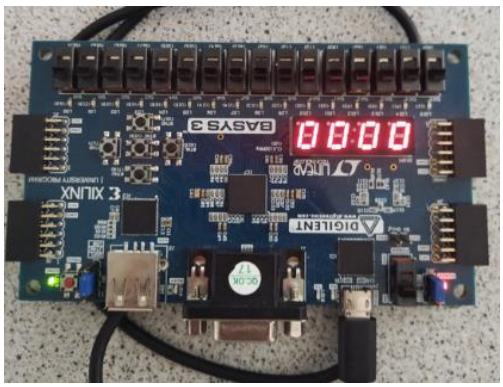
Le codeur BCD à 7 segments fonctionne de cette manière :

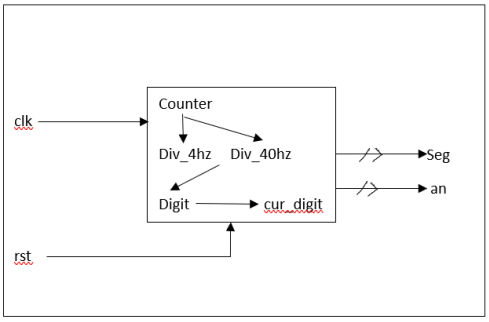
- Sur notre carte BASYS3 il a plusieurs afficheurs à 7 segments. Il y en a exactement 4. Chaque afficheur est contrôlé par 7 signaux qui proviennent du FPGA.
- Grâce à l'état du bit de chaque segment, on peut obtient l'extinction ou l'allumage du segment correspondant.
- Pour qu'un segment soit allumé il faut qu'il y ait un 0. Ainsi s'il y a un 1, le segment sera éteint.
- L'index 0 est égal au segment A, l'index 1 est B, et ainsi de suite, jusqu'à l'index 6, qui contrôle le segment G
- Avec le schéma de la Figure 2, si vous souhaitez afficher un 8, il faudra mettre tous les segments a 0. Si on veut afficher un zéro, on aura 1000000.



b. Présentation

Nous avons codé un afficheur 7 segments à 4 chiffres identique sur la carte mis à notre disposition :





c. Explication du code

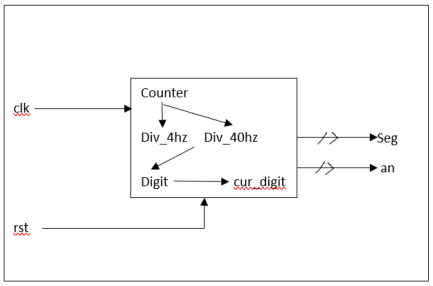
Les afficheurs se programment en utilisant une suite de sept 1 ou 0 bien défini. En se basant sur le schéma à gauche, représentant l'un des afficheurs, le segment G correspond au premier chiffre de la suite, le F au deuxième, jusqu'au A qui correspond au septième chiffre de la suite. Ainsi, pour coder le chiffre 7, il faudra implémenter la variable seg (servant donc à l'affichage du chiffre) de la façon suivante : « 1111000 ». À chaque front montant de la clock, une variable compteur est incrémenté, de telle sorte à ce que les chiffres affichés changent au fur et à mesure du temps. Le bouton RESET sert à revenir à quatre 1 affichés

```
process(CLK, RST)
begin
    if RST = '1' then
        counter <= 0;
        div_4hz <= 0;
        div_1hz <= 0;
        digit <= "0000";
        cur_digit <= 0;
    elsif rising_edge(CLK) then
        counter <= counter + 1;
        if counter = 1000 then -- 2500000
            counter <= 0;
            div_4hz <= div_4hz + 1;
            if div_4hz = 4 then
                div_4hz <= 0;
                div_1hz <= div_1hz + 1;
                if div_1hz = 10 then
                    div_1hz <= 0;
                    digit <= digit + 1;
                    if digit = "10000" then
                        digit <= "0000";
                    end if;
                end if;
            end if;
            cur_digit <= (cur_digit + 1) mod 4;
        end if;
    end if;
end process;
```

6. PROJET 5 : AFFICHAGE 4 SEGMENTS DIFFERENTS

a. Présentation

Ce code VHDL décrit un circuit électronique qui contrôle l'affichage de quatre chiffres sur un afficheur à sept segments. Le circuit utilise deux signaux d'horloge, CLK et RST, pour synchroniser ses opérations avec le reste du système. La fréquence de l'horloge CLK est utilisée pour mesurer le temps écoulé et pour diviser la fréquence de l'horloge en différentes fréquences. Le compteur "counter" est utilisé pour compter le nombre de fronts montants de l'horloge CLK.



b. Explication du code

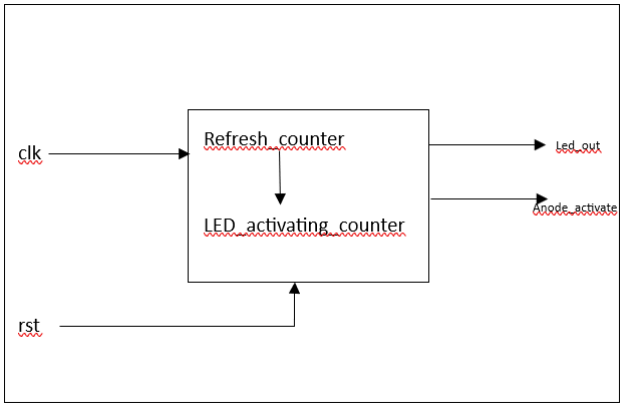
Le programme principal de ce projet est le même que celui précédent, Les variables horloge et RESET ne sont pas nécessaires. Le projet précédent a la même boîte noire, mais avec des seg différents dans le programme afin d'afficher des chiffres différents (la seule partie qui change est la partie segment) :

```
seg <= "0100100" when cur_digit = 0 else -- 2
       "0010010" when cur_digit = 1 else -- 5
       "0011001" when cur_digit = 2 else -- 4
       "1000000" when cur_digit = 3; -- 0
```

7. PROJET 6 : CHRONOMETRE

a. Présentation

L'objectif du projet est de réaliser un chronomètre sur les afficheurs 7 segments de la carte mis à notre disposition. Pour cela nous nous sommes servis des 2 sujets précédents. Afin de mieux appréhender la demande du sujet, on a pu s'aider des deux sujets à affichage de 4 chiffres.



b. Explication du code

Le premier processus permet d'associer des valeurs aux signaux « LED_BCD » et « LED_out », puis nous le compteur BCD nous permet de décoder et envoyer les informations à l'afficheur.

Le deuxième processus est un diviseur, qui nous permet de dénombrer les cycles d'horloge ainsi que les actualiser. Puis la sortie de ce diviseur entre dans le process compteur par 10. Le compteur par 10 stocke dans un signal sa valeur à chaque coup d'horloge, et si l'entrée RST est active le signal s'incrémente. Après 9 le compteur repasse a 0.

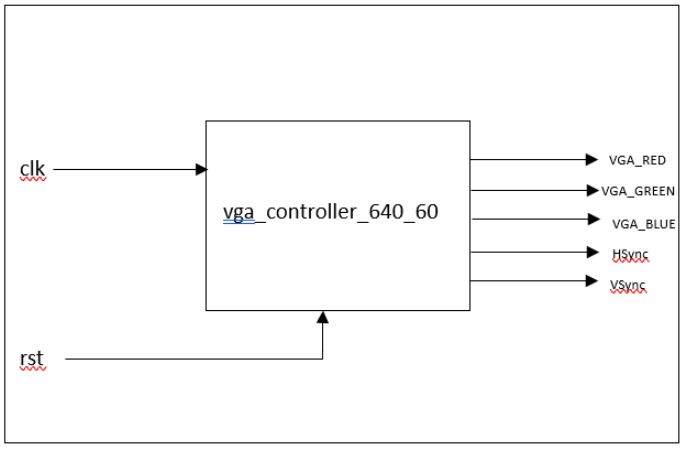
```
process(clock_100Mhz,reset)
begin
    if(reset='1') then
        refresh_counter <= (others => '0');
    elsif(rising_edge(clock_100Mhz)) then
        refresh_counter <= refresh_counter + 1;
    end if;
end process;
```

```
begin
    case LED_activating_counter is
    when "00" =>
        Anode_Activate <= "0111";
        LED_BCD <= displayed_number(15 downto 12);
    when "01" =>
        Anode_Activate <= "1011";
        LED_BCD <= displayed_number(11 downto 8);
    when "10" =>
        Anode_Activate <= "1101";
        LED_BCD <= displayed_number(7 downto 4);
    when "11" =>
        Anode_Activate <= "1110";
        LED_BCD <= displayed_number(3 downto 0);
    end case;
end process;
```

8. PROJET 7 : IMAGE FIXE MONOCHROME

a. Présentation

Ce projet nous a permis d'apprendre à utiliser le port VGA, afin d'afficher une simple image monochrome à l'aide de la carte. Afin de gérer le port VGA, on utilise contrairement aux autres projets, un fichier différent. On utilise un fichier vga_controller_640_60.



b. Explication du code

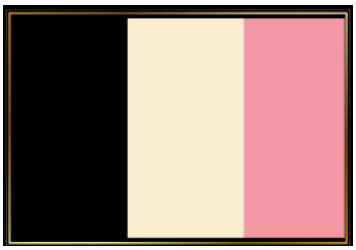
Pour commencer, il y a le besoin d'initialiser les variables contenues dans le fichier vga_controller_640_60, puis il faut charger le fichier image qui est contenu dedans. La couleur affichée est définie en RGB, chaque couleur étant une variable distincte des autres. Ici, il nous suffisait de faire passer l'une des trois variables, à 1, pour que la couleur voulue soit affichée.

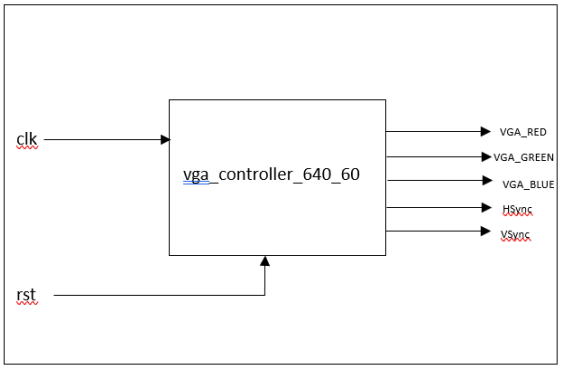
```
begin
    I0 : vga_controller_640_60 port map (RST, pixel_clk(1), Hsync, Vsync, hcount, vcount, blank);

    process (CLK,RST)
    begin
        if (RST = '1') then
            pixel_clk <= (others => '0');
        elsif (CLK'event and CLK = '1') then
            pixel_clk <= pixel_clk + '1';
        end if;
    end process;

    process (blank)
    begin
        if (blank = '0') then
            vgaRed <= (others => '1');
            vgaGreen <= (others => '1');
            vgaBlue <= (others => '0');
        else
            vgaRed <= (others => '0');
            vgaGreen <= (others => '0');
            vgaBlue <= (others => '0');
        end if;
    end process;
```

9. PROJET 8 : IMAGE FIXE 3 COULEURS

a. Présentation

Le but de ce projet est de générer un signal VGA, qui nous permet d'afficher une image à 3 couleurs sur un l'écran. Pour l'affichage du drapeau, nous avons choisi le drapeau de la Belgique





b. Explication du code

Pour coder ce projet, on utilise la même base de code que le projet précèdent. La différence est que nous avons dû séparer l'écran en trois parties pour pouvoir attribuer à chaque zone une partie. Pour chaque couleur, une nouvelle definition de l'état du VGA green/blue/red est mise en place :

- Pour le noir le vgaBlue, vgaGreen et vgaRed doivent être à 1
- Pour le jaune c'est vgaBlue et vgaGreen qui doivent être à 1
- Pour le rouge c'est vgaRed qui doit être à 1

```
process (blank1, blank2)
begin

    if (blank = '0') then if((hcount>=0 and hcount<=150) AND (vcount>=0 and vcount<=500))
    then
        vgaRed <= (others => '1');
        vgaGreen <= (others => '1');
        vgaBlue <= (others => '1');

    elseif (blank = '0') then if((hcount>=150 and hcount<=300) AND (vcount>=0 and vcount<=500))
    then
        vgaRed <= (others => '1');
        vgaGreen <= (others => '1');
        vgaBlue <= (others => '0');

    elseif (blank = '0') then if((hcount>=300 and hcount<=450) AND (vcount>=0 and vcount<=500))
    then
        vgaRed <= (others => '1');
        vgaGreen <= (others => '0');
        vgaBlue <= (others => '0');

    else
        vgaRed <= (others => '0');
        vgaGreen <= (others => '0');
        vgaBlue <= (others => '0');
    end if;
    end if;
```

10. PROJET 9 : PONG

a. Présentation

Dans ce sujet nous avons dû réaliser un Pong. À la vue de la complexité de ce sujet, nous nous sommes réunis pour le réaliser. Nous nous sommes principalement appuyés sur les sujets avec les ports VGA, afin de pouvoir afficher notre Pong sur un écran. Dans la limite de notre projet nous nous sommes arrêtes au fond du Pong ainsi que la ligne centrale et la position des raquettes sans mouvement.

11. CONCLUSION

Au début, nous avons dû parler d'ambition et de planification pour tenir des délais, mais nous nous sommes confrontés à problèmes internes, externes, mais aussi lié a la volonté de chacun.

Donc pour conclure ce projet, nous pouvons parler d'apprentissage sur deux plans.

Le premier est le plan personnel. Nous avons dû apprendre à travailler tous ensemble, avec des niveaux différents, des envies différentes. Savoir gérer son temps de planification, apprendre à écouter l'autre, et enfin savoir être autonome et travailler de son coté pour ne pas prendre de retard.

Ensuite, le deuxième plan est la partie technique. Travailler sur 9 projets différents, permet d'apprendre de larges compétences en VHDL. Que ce soit comme l'utilisation de la carte, un nouveau type de langage, Vivado, etc. Cela nous a permis d'accroitre nos connaissances et nos compétences dans ce domaine.