Rapport de TP FPGA

Sous Titre?

Ayrault Maxime **3203694** - *Nguyen Gabriel* **32?????**

Intro

Ce qu'on compte faire dans cette matière, les différents objectif et tout

TP 1

Titre du TP

I. prise en main du modèle

1) Création d'un module VHDL

Dans cette première partie nous allons creer un module permetant d'allumer des leds selon la valeur d'interrupteurs sur la carte.

```
entity TEST is
   Port (  -- input signal, switch 0
           SW0 : in STD_LOGIC;
           -- input signal, switch 1
           SW1 : in STD_LOGIC;
           -- input signal, switch 2
           SW2 : in STD_LOGIC;
           -- output signal the first 3 leds of the board
           LED : out STD_LOGIC_VECTOR (2 downto 0)
         );
end TEST;

architecture Behavioral of TEST is
begin
  -- when switch 0 is on - led 0 is on
  LED(0) <= SW0;
  -- when switch 1 is on - led 1 is on
  LED(1) <= SW1;
  -- when the three switchs are all on - all the led are on too
  LED(2) <= SW0 and SW1 and SW2;
end Behavioral;
```

II. Cas d'études-Synthèse VHDL

On doit traiter 3-4 parties séparées, chaque partie est composé d'un petit système permettant de réaliser une fonction simple.
Dans la dernière partie de cette partie nous allons les regrouper en un seul bloc qui effectura une action plus grande.

1) Compteurs imbriqués

Le fichier Test_CPT permet d'instancier une petite structure comprenant deux compteurs et une gestion de l'affichage des 4 premières leds.

Un Premier compteur *Cpt* inverse la valeur du signal `start` une fois tout les *20000000 cycles*, initialisé à `'0'`. (bof comme phrase..)

A chaque fois que le signal `start` vaut `'1'` le compteur *Cpt2* (sur 28 bits) est incrémenté.

Il y a deux mode pour l'affichage sur les LEDS

- Le bouton gauche est appuyé -> Les 4 LEDS de gauches de la cartes sont allumées.
- Le bouton gauche est relaché -> Affichage des 4 MSB de CPT2 sur les 4 premières LEDS.

Il y avait plusieurs erreurs dans le code qui l'empechait de fonctionner normalement :

- Le compteur *CPT* n'était pas assez grand pour atteindre 20000000, le signal start n'était jamais actioné.
- ? Je sais plus

Code corrigé:

```
entity Test_CPT is
   Port ( -- Clock
          Clk : in  STD_LOGIC;
          -- Asynchronous Reset
          Reset : in  STD_LOGIC;
          -- Left Button
          Button_L : in  STD_LOGIC;
          -- The 4 output for the LED
          LED : out  STD_LOGIC_VECTOR (3 downto 0));   -- LED de sortie
end Test_CPT;

architecture Behavioral of Test_CPT is

-- modulo N counter
signal Cpt: integer range 0 to 20000000;
-- 28 bits counter
signal Cpt2: std_logic_vector(27 downto 0);
-- start signal
signal start: std_logic;                                                    -

begin
  ------------------------------
  -- Gestion Cpt et Start --
  ------------------------------
  process(Clk,Reset)
  begin


    if Reset = '1' then
      -- Asynchronous Reset
      Cpt <= 0;
      start <='0';

    if rising_edge(Clk) then

      -- increment Cpt
      Cpt <= Cpt + 1;
```

```
      -- if the bound limit is reach
      if Cpt = 20000000 then -- *** Correction de 20000000 au lieu de 70000000 ***
        -- invert start level
        start <= not start;
        -- Reset Cpt
        Cpt <= 0;
      end if;

    end if;
  end process;
```

```
----------------------
-- Gestion CPT2
----------------------
process(Clk,Reset)
begin

 -- Asynchronous Reset
 if Reset = '1' then
       Cpt2 <= (others => '0');
 end if;

 if rising_edge(Clk) then

   if start = '1' then
     -- increment Cpt2 when start signal equal '1'
     Cpt2 <= Cpt2 + 1;
   end if;

 end if;
end process;
```

```
  ------------------
  -- Gestion LED
  ------------------
    -- Bouton Relâché --> Affichage des 4 MSB de CPT2
    -- Bouton Appuyé  --> Les 4 LED sont Allumées

    LED <= Cpt2(27 downto 24) when Button_L='0' else "1111";

end Behavioral;
```

2) Compteur d'impulsions

Le fichier Test_Impulse permet d'instancier une petite structure permettant d'utiliser deux boutons. le *bouton de gauche* sert à incrementer la valeur de notre compteur, celui du *centre* pour le décrementer.
Le compteur est un compteur sur 4 bits dont la valeur est affichée en binaire sur les 4 premières leds de la carte. Il y a aussi la led 15 qui s'allume une fois une valeur seuil dépasée.

Lors de l'implémentation, on remarque que la fonction écrite dans le fichier VHDL ne fonctione pas car on constate que la synchronisation est faite par 2 signaux (*Button_L* et *Button_C*) ce qui n'est pas possible.

Nous avons aussi rencontré un problème de fréquence. En effet la carte tournant à 100MHz nous ne pouvons pas gerer notre compteur seulement par l'appuis que l'on fait sur celui ci.
Nous avons eu besoin d'introduitre des `stamps` pour gerer le temps entre deux appuis. Chaque stamp sert a limiter le temps entre deux appuis consécutifs reconnu dans l'implementation. La vitesse d'incrémentation du compteur est donc bloquée à 1 appuis toutes les 2? secondes.

Code corrigé:

```
entity IMPULSE_COUNT is
  Port ( -- clock
         Clk      : in STD_LOGIC;
         -- Reset Asynchrone
         Reset    : in  STD_LOGIC;
         -- LEDS Values
         Count    : out  STD_LOGIC_VECTOR (3 downto 0);
         -- Indicateur Valeur Seuil
         Sup      : out  STD_LOGIC;
         -- Center Button
         Button_C : in  STD_LOGIC;
         -- Left   Button
         Button_L : in  STD_LOGIC);
end IMPULSE_COUNT;

architecture Behavioral of IMPULSE_COUNT is

  -- Impulse counter
  signal cpt : std_logic_vector(3 downto 0);
  -- Signal to avoid bounce for left button
  signal stamp1 : integer range 0 to 100000000;
  -- Signal to avoid bounce for center button
  signal stamp2 : integer range 0 to 100000000;

 begin

   -- put the cpt value as led value
  count <= cpt;


  process(reset, clk)
  begin
```

```
    -- Asynchrnous Reset
    if reset='1' then
        cpt<="0000";
    end if;


    if rising_edge (clk) then
        -- Increment stamp1
        stamp1 <= stamp1 + 1;
        -- Increment stamp2
        stamp2 <= stamp2 + 1;

        -- if left button is hit and at the right timming
        if Button_L = '1' and stamp1 > 20000000 then
           -- Reset stamp1
           stamp1 <= 0;
           -- increment cpt value
           cpt<=cpt+1;
        end if;

        -- if center button is hit and at the right timming
        if Button_C = '1' and stamp2 > 20000000 then
           -- Reset stamp1
           stamp2 <= 0;
           -- decrement cpt value
           cpt<=cpt-1;
        end if;

   end if;
 end process;

 process(Cpt)
 begin

   -- if cpt is greater than 9 then sup output is equal to 1
   if (cpt > 9) then
      Sup<='1';
   else
     Sup<='0';
   end if;
 end process;

end Behavioral;
```

3) Décodeur

Le fichier Selector permet d'instancier un decodeur qui prends en entré les signaux de sortie (*sup* et *cout*) du fichier impulse_Count, et initialise le signal *Limit* qui affichera differents motifs sur les 16 leds en fonction de ces deux signaux.

Il y a eu des erreur de compilation car dans le fichier originel toutes les conditions du `case` n'étaient pas déclarées. Il a fallut retirer le commentaire indiquant `when others => NULL` pour résoudre ce problème. il a aussi fallut rajouter au niveau de l'initialisation pour les valeurs du signal Decode, la ligne suivante `else "00"`.

Code corrigé:

```
entity Selector is
Port (-- Clock
       Clk : in  STD_LOGIC;
       -- Asynchronous Reset
       Reset : in  STD_LOGIC;
       -- Right Button
       Button_R: in STD_LOGIC;
       -- Compteur d'entré
       Count : in  STD_LOGIC_VECTOR (3 downto 0);
       -- Valeur Seuil
       Sup : in  STD_LOGIC;
       -- Bound Value
       Limit : out  STD_LOGIC_VECTOR (27 downto 0));
end Selector;


architecture Behavioral of Selector is

  -- Commande du Decodeur
  signal Decode: std_logic_vector(1 downto 0);

begin

---------------------------
-- Gestion du Décodeur
---------------------------
   process(Clk,Reset)
   begin

       -- Reset Asynchrone
       if Reset = '1' then
          Limit <= (others =>'0');


       -- Si On A un Front d'Horloge
       elsif rising_edge (Clk) then

         -- Si On Appuie sur le Bouton Right
         if Button_R = '1' then

       -- Signification de Limit (Pour la Machine à États de la Suite du TP)
```

```
        -- Les 2 MSB définissent le Mode de Clignotement
        -- 00 -->     LEDs Toujours Éteintes
        -- 10 -->     Clignotement des LEDs
        --                La Fréquence de Clignotement
        --             Dépend des LSB de Limit
        --             24 Millions --> 1 fois par Seconde
        --                 8 Millions --> 3 fois par Seconde
        -- 11 -->     LEDs Toujours Allumées

             case (Decode) is

               when "00" => Limit <= (others => '0');
               when "01" => Limit <= X"96E3600";-- 24 000 000 en Décimal
               when "10" => Limit <= X"87A1200";-- 8 000 000 en Décimal
               when "11" => Limit <= (others => '1');
               when others => NULL;

            end case;
          end if;
        end if;

    end process;




        -- Si Count > 9       --> Decode = 11
    Decode <= "11" when Sup = '1'
        -- Si Count = 6,7,8,9 --> Decode =10
        else "10" when Count > 5
        -- Si Count = 3,4,5   --> Decode = 01
        else "01" when Count > 2
        -- Si Count = 0,1,2   --> Decode = 00
        else "00";

end Behavioral;
```

4) FSM

Dans cette étape le but est d'implémenter une machine à état qui va permetre de réaliser le fonctionnement des
leds (éteintes, allumés ou clignement).

Code corrigé:

```
entity FSM is
Port ( --Horloge
       Clk : in  STD_LOGIC;
       --Reset Asynchrone
       Reset : in  STD_LOGIC;
```

```
      --Mode d'Affichage des LEDs
      Mode : in  STD_LOGIC_VECTOR (1 downto 0);
      --Seuil du Compteur pour Vitesse
      Seuil : in  STD_LOGIC_VECTOR (25 downto 0);
      --Commande des LEDs
      LED : out  STD_LOGIC_VECTOR (3 downto 0));
end FSM;

architecture Behavioral of FSM is

-- Compteur de Temporisation
signal cpt: integer range 0 to 24000000;

-- FSM States
type etat is (LED_OFF, CLIGN_OFF, LED_ON, CLIGN_ON);
-- État Présent, État Futur
signal EP,EF: etat;

begin

-----------------------------------------------
-- Gestion du Compteur de Temporisation
-----------------------------------------------

process(Clk,Reset)
begin

   -- Reset Asynchrone
   if Reset='1' then
      Cpt <= 0;

   -- Si on A un Front d'Horloge...
   elsif rising_edge(Clk) then

      -- Si On Est en Mode Clignotement, le Compteur s'incrémente
      if (EP = CLIGN_OFF) or (EP = CLIGN_ON) then
         Cpt <= Cpt + 1;
      -- Sinon, on Remet le Compteur à 0
      else
         Cpt <= 0;
         end if;
         end if;
         end process;

---------------------------
-- MAE - Registre d'État
---------------------------

process(Clk,Reset)
begin

       -- Reset Asynchrone
       if Reset = '1' then
          EP <= LED_OFF;

       -- Si on a un Front d'Horloge
```

```
       elsif rising_edge (Clk) then
          -- Mise à Jour du Registre d'Etat
          EP <= EF;
       end if;
end process;

-------------------------------------------------
-- MAE - Évolution des États et des Sorties
-------------------------------------------------
process(Cpt,EP,Mode,Seuil)
begin

  -- Par Défaut les LEDs sont Éteintes
  LED <= "0000";

  ---------------------------------------------------------------------
  --         Modes de Fonctionnement
  --         Mode = 00 --> LEDs Éteintes
  --         Mode = 10 --> LEDs Clignotent
  --         Mode = 11 --> LEDs Allumées
  ---------------------------------------------------------------------

  case (EP) is

     -- LEDs Éteintes
     -- On Reste dans cet État Tant que Mode est à 00
     -- Si Mode Passe à 10, On Passe en LEDs Clignotement
     -- Si Mode Passe à 11, On Passe en LEDs Allumées

       when LED_OFF  => LED <= "0000";
       --Rajout de la valeur en sortie
                        if Mode = "10" then
                           EF <= CLIGN_OFF;
                        elsif Mode = "11" then
                           EF <= LED_ON;
                        end if;

       -- LEDs Clignotement - (Eteint)
       -- Le Compteur Compte Jusqu'au Seuil puis on Passe à l'Etat Suivant
       when CLIGN_OFF        =>  LED <= "0000";
       --Rajout de la valeur de sortie
                        if Mode = "00" then
                           EF <= LED_OFF;
                        elsif Mode = "11" then
                           EF <= LED_ON;
                        end if;
                        if Cpt = Seuil then
                           EF <= LED_ON;
                        end if;

       -- LEDs Allumées
       -- On Reste dans cet état tant que Mode est à 11
       -- Si Mode Passe à 10, On Passe en LEDs Clignotement
       -- Si Mode Passe à 00, On Passe en LEDs Éteintes
       when LED_ON   => LED <= "1111";
                        if Mode = "10" then
```

```
                           EF <= CLIGN_ON;
                        elsif Mode = "00" then
                           EF <= LED_OFF;
                        end if;

        -- LEDs Clignotement - (Allumé)
        -- Le Compteur Compte Jusqu'au Seuil puis on Passe à l'état Suivant
        when CLIGN_ON => LED <= "1111";
                        if Mode = "00" then
                           EF <= LED_OFF;
                        elsif Mode = "11" then
                           EF <= LED_ON;
                        end if;
                        if Cpt = Seuil then
                           EF <= LED_OFF;
                        end if;
     end case;
   end process;

 end Behavioral;
```

TP 2

Titre du TP

I. Développement de l'application logicielle

Dans cette exercice nous devons écrire un programme C permettant qui sera executé sur le microcontroleur `Microblaze` afin d'allumer des leds en actionnanr des interrupteurs.

Code écrit:

```
#include "xgpio.h"
#include "xparameters.h"

int main (int argc, char **argv ) {

       /* déclarer GPIO */
       XGpio led, button;
       /* registre lecture écriture sur 32 bits */
       u32 lecture = 0;
       /* initialiser la struct XGPIO */
       XGpio_Initialize (&led, 1);
       /* fixer la direction des switch */
       XGpio_SetDataDirection (&led, 1, 1);
       /* fixer la direction des led */
```

```
        XGpio_SetDataDirection (&led, 2, 0);


        while (1) {
            /* lire la valeur des 4 switchs */
            lecture = XGpio_DiscreteRead (&led, 1);
            /* écrire la valeur lu */
            XGpio_DiscreteWrite (&led, 2, lecture);
        }
  return 0;
}
```