MOULIN Mathieu
Laviolette Etienne

Le 24 octobre 2014

Compte rendu TP3

Exercice – 1 : Exercice Préliminaire :

a. Objectifs de l'exercice :

L'objectif de ce premier exercice est de comprendre le fonctionnement d'un diviseur de fréquence à deux entrées et une sortie dont le code VHDL commenté est présenté ci-dessous.

Nous utiliserons pour cela les éléments suivants :

- Clk100Mhz l'horloge,
- PB_0 le bouton de la carte
- LED_0 la première LED lumineuse de la carte

Le but est de faire clignoter la première LED de la carte toutes les secondes en divisant les 100MHz d'entrée en 1Hz

b. Le code VHDL

```
entity blinker is
       port(Clk100MHz, PB_0 : in bit;
       LED_0 : out bit);
end blinker;

architecture Behavioral of blinker is
     alias reset is PB_0; -- alias pour le signal de réinitialisation
     signal clk_out : bit := '0'; -- signal d'horloge après division

     -- Constante de division, ici pour une sortie à 1Hz.
     constant clock_divisor : integer := 100000000;

     begin
     -- Diviseur de fréquence : divise la fréquence du signal Clk100MHz
           par clock_div.
           clock_divider : process(Clk100Mhz, reset)

     variable c : integer range 0 to clock_divisor - 1 := 0;
     begin
           if reset = '1' then
                 c := 0;
                 clk_out <= '0';
           elsif Clk100MHz'event and Clk100MHz = '1' then
                 if c < (clock_divisor - 1) / 2 then
                       c := c + 1;
                       clk_out <= '0';
                 elsif c = (clock_divisor - 1) then
                       c := 0;
                       clk_out <= '0';
                 else
                       c := c + 1;
                       clk_out <= '1';
                 end if;
           end if;
     end process;
     -- Sortie sur la LED
     LED_0 <= clk_out;
end Behavioral;
```

c. Fonctionnement

Le bouton PB_0 est le bouton de reset de l'horloge.

Pour diviser la fréquence, on utilise un compteur et une constante clock_divisor qui correspond au nombre de fois qu'on veut diviser la fréquence de l'horloge. Celle-ci étant cadencé à 100Mz, il faut la divisé par `100 000 000 pour avoir un signal de 1Hz.`

A chaque front montant de l'horloge, on regarde :

Si le compteur est inférieur à la moitié de la constante de division, on l'incrémente et le signal de sorti est 0

Si le compteur est égal à la constante de division -1, on a effectué les 100 000 000 tour d'horloge, alors on met la variable de sortie à 0 et le compteur repasse à 0. En effet, on recommence les 100 000 000 tours d'horloge suivant.

Si non, si le compteur est entre la moitié de la constante de division et la constante de division ($c \in \left]\frac{cste}{2} ; cste\right[$) alors on incrémente le compteur et on passe le signal de sortie est à 1.

Ainsi, visuellement, on s'intéresse au signal d'horloge par groupe de 100 000 000. Pour les 49 999 999 premiers fronts montants de l'horloge, la LED_0 est éteinte, et pour les suivant la LED_0 est allumé. Ainsi, elle clignote toute les secondes.

Exercice – 2 Feux de circulation :

a. objectifs de l'exercice :

Afin de programmer la simulation de feux de cicrulation sur la carte nous allons utiliser les éléments suivants :

- clk_out : le diviseur de fréquence donné dans l'exercice numéro un,
- PB_0 le premier bouton de la carte,
- LED_7654, l'ensemble de quatre LEDs (premier groupe de feux)
- LED_3210 , l'ensemble des quatre autres LEDs, (second groupe de feux)

Le feu vert sera la diode la plus à droit de la carte, l'orange juste à sa gauche et le rouge encore à gauche.

Pour modéliser ce fonctionnement nous allons faire tourner un compteur (variable c).

De plus nous avons décidé d'inclure un décalage de deux secondes entre le moment où le feu passe au rouge et où celui opposé passe au vert.

Le tableau suivant représente la situation

Temps cumulé	Temps	Etat	Feux 1 (LED_3210)		Feux 2 (LED_7654)	
			Couleur	Valeur	Couleur	Valeur
8	8	0	R	4	V	1
10	2	1	R	4	O	2
12	2	2	R	4	R	4
20	8	3	V	1	R	4
22	2	4	O	2	R	4
24	2	5	R	4	R	4

b. Le code VHDL

```
entity exo2 is
        port(Clk100MHz, PB_0 : in bit;
             LED_3210, LED_7654 : out integer range 0 to 4);
end exo2;
architecture Behavioral of exo2 is
alias reset is PB_0; -- alias pour le signal de reinitialisation
signal clk_out : bit := '0'; -- signal d'horloge apres division
constant clock_divisor : integer := 100000000; -- Constante de division, ici pour une sortie a 1Hz

begin
       -- Diviseur de frequence : divise la frequence du signal Clk100MHz par clock_div
       clock_divider : process(Clk100Mhz, reset)
       -- c permet de parcourir tout le cycle
       variable c : integer range 0 to clock_divisor - 1 := 0;
       begin
       if reset = '1' then -- si on appui sur reset, on redemarre un cycle
                c := 0; clk_out <= '0';
        elsif Clk100MHz'event and Clk100MHz = '1' then
             -- on est sensible a chaque front montant a 100Mhz
                if c < (clock_divisor - 1) / 2 then
             -- premiere partie du cycle, on defini la sortie a 0
                        c := c + 1; clk_out <= '0'; -- sur la moitie du cycle (de 0 a 0,5 s)
                elsif c = (clock_divisor - 1) then
             -- fin de cycle, quand on arrive au bout d'un cycle
                        c := 0; clk_out <= '0'; -- on recommence (a 1 seconde)
                else
             -- deuxieme partie du cycle, on defini la sortie a 1
                  c := c + 1; clk_out <= '1'; -- (de 0,5 a 1 seconde)
             end if;
        end if;
end process;

process(clk_out)
        variable c : integer range 0 to 24;
        variable etat : integer range 0 to 5;
        begin
                if reset = '1' then
                        c := 0;
                        etat := 0;
                elsif clk_out'Event and clk_out = '1' then
                        if c < 8 then
                                c := c + 1; etat := 0;
                        elsif c < 10 then
                                c := c + 1; etat := 1;
                        elsif c < 12 then
                                c := c + 1; etat := 2;
                        elsif c < 20 then
                                c := c + 1; etat := 3;
                        elsif c < 22 then
                                c := c + 1; etat := 4;
                        elsif c < 24 then
                                c := c + 1; etat := 5;
                        end if;
                        if c = 24 then
                           c := 0;
                        end if;
                end if;

                case etat is
                        when 0 => LED_3210 <= 1; LED_7654 <= 4;
                        when 1 => LED_3210 <= 2; LED_7654 <= 4;
                        when 2 => LED_3210 <= 4; LED_7654 <= 4;
                        when 3 => LED_3210 <= 4; LED_7654 <= 1;
                        when 4 => LED_3210 <= 4; LED_7654 <= 2;
                        when 5 => LED_3210 <= 4; LED_7654 <= 4;
                        when others => LED_3210 <= 1; LED_7654 <= 4;
                end case;

        end process;
end Behavioral;
```

c. fonctionnement

Grâce au diviseur de fréquence de l'exercice numéro 1 nous nous basons sur un cycle total de 24 secondes (10s rouge, 2s orange, 8s vert et 2 fois 2 secondes de battement).

Une pression sur le premier bouton de la carte redémarre en début de cycle (à t=0)

Nous utilisons 2 process : le premier pour diviser la fréquence de l'horloge est avoir un signal de période d'une seconde que nous utilisons dans le second process afin de calculer les nouveaux états toutes les secondes.

La deuxième process réutilise le tableau précédent pour calculer les états et ainsi allumer les LED correspondantes à chaque changement de valeur de clk_out.

Exercice 3 – ajout d'un capteur en présence de voiture

a. Objectif

Nous allons reprendre le code VHDL utilisé à l'exercice numéro 2 sans inclure les battements de deux secondes (pour permettre aux piétons de traverser).

Afin de modéliser le capteur de présence de deux voitures nous ajoutons la gestion de PB_1 et PB_2, deux boutons (autre que le PB_0 servant au RESET) permettant de simuler la détection d'une voiture. Le PB_1 pour l'axe LED_3210 et le PB_2 pour l'axe LED_7654.

L'idée est que si une voiture est détectée sur un axe dont le feu est au rouge, le feu opposé passe à l'orange puis au rouge pour permettre de faire passer le feu ayant détecter la voiture de passer au vert et ainsi la laisser passer.

b. VHDL

```
entity exo3 is
port(Clk100MHz, PB_0, PB_1, PB_2 : in bit;
LED_3210, LED_7654 : out integer range 0 to 4);
end exo3;

architecture Behavioral of exo3 is
alias reset is PB_0; -- alias pour le signal de reinitialisation
signal clk_out : bit := '0'; -- signal d'horloge apres division
-- Constante de division, ici pour une sortie a 1Hz.
constant clock_divisor : integer := 100000000;
begin
       -- Diviseur de frequence : divise la frequence du signal Clk100MHz par clock_div
       clock_divider : process(Clk100Mhz, reset)
       -- c permet de parcourir tout le cycle
       variable c : integer range 0 to clock_divisor - 1 := 0;
       begin
       if reset = '1' then -- si on appui sur reset, on redemarre un cycle
                c := 0; clk_out <= '0';
        elsif Clk100MHz'event and Clk100MHz = '1' then
```

```
            -- on est sensible a chaque front montant a 100Mhz
               if c < (clock_divisor - 1) / 2 then
            -- premiere partie du cycle, on defini la sortie a 0
                       c := c + 1; clk_out <= '0'; -- sur la moitie du cycle (de 0 a 0,5 s)
               elsif c = (clock_divisor - 1) then
            -- fin de cycle, quand on arrive au bout d'un cycle
                       c := 0; clk_out <= '0'; -- on recommence (a 1 seconde)
               else
            -- deuxieme partie du cycle, on defini la sortie a 1
                 c := c + 1; clk_out <= '1'; -- (de 0,5 a 1 seconde)
            end if;
        end if;
end process;

process(clk_out)
variable c : integer range 0 to 20;
variable etat : integer range 0 to 3;
variable detecteur_voiture : integer range 0 to 1;
begin
if reset = '1' then
       c := 0;
       etat := 0;
elsif clk_out'Event and clk_out = '1' then
       if c < 8 then
              c := c + 1;
              etat := 0;
       elsif c = 8  AND detecteur_voiture = 0 then
              if PB_1 = '1' then
                     detecteur_voiture:= 1;
                     c := 18;
              end if;
       elsif c < 10 then
              c := c + 1;
              detecteur_voiture:= 0;
              etat := 1;
       elsif c < 18 then
              c := c + 1;
              etat := 2;
       elsif c = 18 AND detecteur_voiture = 0 then
              if PB_2 = '1' then
                     c := 8;
                     detecteur_voiture:= 1;
              end if;
       elsif c < 20 then
              c := c + 1;
              detecteur_voiture:= 0;
              etat := 3;
       end if;
       if c = 20 then
          c := 0;
       end if;
end if;

case etat is
       when 0 => LED_3210 <= 1; LED_7654 <= 4;
       when 1 => LED_3210 <= 2; LED_7654 <= 4;
       when 2 => LED_3210 <= 4; LED_7654 <= 1;
       when 3 => LED_3210 <= 4; LED_7654 <= 2;
when others => LED_3210 <= 1; LED_7654 <= 4;
end case;
end process;

end Behavioral;
```

MOULIN Mathieu
Laviolette Etienne

Le 24 octobre 2014

c. Fonctionnement

Temps cumulé	Temps	Etat	Feux 1 (LED_3210)		Feux 2 (LED_7654)	
			Couleur	Valeur	Couleur	Valeur
8	**8**	**0**	**V**	**1**	**R**	**4**
10	2	1	O	2	R	4
18	**8**	**2**	**R**	**4**	**V**	**1**
20	2	3	R	4	O	2

A 8 secondes et 18 secondes (les lignes en gras), avant de changer d'état (de passer de 0 à 1 ou de 2 à 3), il faut vérifier si une voiture a déjà été détectée. Si c'est le cas, on continue normalement, sinon il faut vérifier si le capteur capte une nouvelle voiture. Si tel est le cas, on change la valeur du compteur pour le mettre au temps correspondant au nouvel état souhaité, et on met la variable de détection de voiture à 1 pour mémorisé la détection.