TP02 - VHDL séquentiel

Geoffrey PERRIN
Océane DUBOIS

0.1 Exercice préliminaire

Voici le code d'un diviseur de fréquence :

```
entity blinker is
   port(Clk100MHz, PB_0 : in bit; --on utilise l'horloge 100MHz en
       signal d'entree
      LED_0 : out bit);
end blinker;

architecture Behavioral of blinker is
   alias reset is PB_0; --PB\_0 est defini comme le signal de reset
   signal clk_out : bit := '0'; -- signal d'horloge apres division
   constant clock_divisor : integer := 100000000; --constante de
       division pour un signal a 1Hz
   begin
      clock_divider : process(Clk100Mhz, reset)
      variable c : integer range 0 to clock_divisor - 1 := 0;
      begin
      if reset = '1' then -- si on declanche le reset on recommence
          le cycle de l'horloge a 0
         c := 0;
         clk_out <= '0';
      elsif Clk100MHz'event and Clk100MHz = '1' then --si le signal
          Clk100MHz est sur un front montant
         if c < (clock_divisor - 1) / 2 then --entre 0 et
             0,5secondes on incremente c et on laisse la sortie a 0
            c := c + 1;
            clk_out <= '0';
         elsif c = (clock_divisor - 1) then -- lorsqu'on arrive a
             une seconde, on recommence le cycle a 0
            c := 0;
            clk_out <= '0';
         else -- sur la deuxieme partie du cycle (de 0,5 a
             1seconde), on continue d'incrementer c et on passe la
             sortie a 1.
               c := c + 1;
         clk_out <= '1';
         end if;
      end if;
   end process;
-- Sortie sur la LED
LED_0 <= clk_out;
end Behavioral;
```

Dans ce code, on définit d'abord une entité "blinker", consituté de 2 ports d'entrée : Clk100MHz, PB_0 et une sortie LED_0.

Puis dans l'architecture, on définit un alias pour PB_0 qui sera désomais appelé reset. On définit un nouveau signal de type bit, appelé clk_out qui est initialisé à 0 qui sera le signal de sortie. Et une constante de type entier nommé clock_divisor initialisé à 100000000.

Puis on déclare un process donc la liste de signaux de sensibilité est composée des signaux Clk100Mhz et reset.C'est donc un reset asynchrone.

Dans le process si le reset est activé, on met la variable c à 0 et le signal clock_out à 0. Si le signal Clk100MHz est sur un front montant et que la variable c est inferieur à (100000000-1)/2 =49999999,5 alors on incrémente la variable c et le signal clock_out est mis à 0. Si c = 99999999 on met c à 0 et clv_out à 0. Entre 49999999,5 et 99999999 on met la sortie à 1. A la fin du process on égalise clk_out sur LED_0.

La LED_0 est donc activée (à 1) à chaque cycle (de 1 entre 0,5 secondes et 1 seconde le reste du temps elle est à 0.

0.2 Feu de circulation

Dans cet exercices on cherchera à implémenter un contrôleur de circulation, permettant de réguler la circulation d'un croisement à 2 axes principaux.

Le feu rouge dure 10 secondes et le feu orange 2 secondes. Le feu vert doit donc durer 8 secondes. Puisqu'il faut que Trouge = Tvert + Torange.

La plus petite unité de temps utilisée est donc 2 secondes (Tsync), on peut donc utiliser une fréquence d'horloge de f=1/Tsync=0,5Hz

On nomme les 2 axes A et B qui sont perpendiculaires. Lorsque le feu de l'axe A est au rouge, le feu de l'axe B doit être vert puis passer au orange. Puis le feu de l'axe B passe enfin au rouge, et le feu de l'axe A passe au vert puis 8 secondes plus tard au orange.

On réalise un modélisation à partir d'un machine à états.

On utilisera la valeur 1 pour signaler un feu rouge, 2 pour le feu orange et 3 pour le feu vert.

0.2.1 Code VHDL

```
ENTITY feu IS
   PORT(Clk100MHz, PB_0 : INBIT;
      LED_3210, LED_7654 : OUT INTEGER RANGE 0 TO 4);
END feu
```

```
ARCHITECTURE Behavioural OF feu IS
alias reset is PB_0;
signal clk_out : bit := '0';
constant clock_divisor : integer := 100000000;
BEGIN
   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;

   process(clk_out)

   variable c : integer range 0 to 20; --temps total d'un cycle
   variable etat : integer range 0 to 3;
   variable feuA, feuB : integer range 0 to 4;

   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 < 18 then
               c := c + 1;
               etat := 2;
            elsif c < 20 then
               c := c + 1;
               etat := 3;
            end if;
            if c = 20 then
               c := 0;
            end if;
         end if;

         case etat is
            when 0 => feuA <= 1; feuB <= 3; --le feu de l'axe A est
                vert et le feu de l'axe B est rouge
            when 1 => feu1 <= 2; feuB <= 3; --le feu de l'axe A est
                orange, le feu de l'axe B est rouge
            when 2 => feu1 <= 3; feuB <= 1; --le feu de l'axe A est
                rouge, le feu de l'axe B est vert
            when 3 => feu1 <= 3; feuB <= 2; --le feu de l'axe A est
                rouge et le feu de l'axe B est orange
            when others => feu1 <= 1; feuB <= 4;
         end case;

      end process;

      LED_3210 <= feuA;
      LED_7654 <= feuB;

      END Behavioural;
```

Sachant que le signal d'entrée du deuxième process est clk_out et qu'il change d'état toutes les seconde, nous avons décidé de créer un compteur de temps, modélisé par la variable c, qui permet d'incrémenter la variable d'état toutes les 8 puis 2 secondes, plutôt que de faire une machine à 20 états différents.

0.3 Prise en compte d'un capteur de voiture

Dans cet exercice on doit toujours réaliser un feu de voiture, mais cette fois-ci en considérant la présence d'un capteur de véhicule qui permet de savoir si il y a un véhicule sur l'axe ou non, si il n'y en a pas, on peut laisser le feu du deuxième axe vert. On ne prendra pas le reset en compte pour le

moment.

Les signaux utilisés pour symboliser les capteurs de véhicules sont PB_2 et PB_3.

On définit donc les séquences comme suivant :

- si aucun capteur de véhicule n'est activé, on définit que le comportement à suivre sera de mettre au vert le feu de l'axe A et au rouge le deu de l'axe B.
- Si un véhicule est capté sur l'axe A et aucun sur l'axe B, on passe le feu A au vert et le feu B au rouge. C'est le même comportement que lorsqu'aucun véhicule n'est détecté.
- à l'inverse si un véhicule est capté sur l'axe B et aucun sur l'axe A, on passe le feu B au vert et le feu A au rouge.
- si on capte des véhicules sur l'axe A et B en même temps, on alterne le feu rouge et vert grâce à la séquence déjà définie dans l'exercice précédent.

0.3.1 Code VHDL

```
process(clk_out)

variable c : integer range 0 to 20; --temps total d'un cycle
variable etat : integer range 0 to 3;
variable feuA, feuB : integer range 0 to 4;

begin
   if reset = '1' then
      c := 0;
      etat := 0;
   else if clk_out'Event and clk_out = '1' then
      if (PB_3 = 1 and etat = 0) or (PB_2 = 1 and etat = 2) or
          (PB_2 = 1 and PB_3 = 1) then
         if c < 8 then
            c := c + 1;
            etat := 0;
         elsif c < 10 then
            c := c + 1;
            etat := 1;
         elsif c < 18 then
```

```
            c := c + 1;
            etat := 2;
         elsif c < 20 then
            c := c + 1;
            etat := 3;
         end if;
         if c = 20 then
            c := 0;
         end if;
      end if;
   end if;

   case etat is
      when 0 => feuA <= 1; feuB <= 3;-- le feu de l'axe A est
          vert et le feu de l'axe B est rouge
      when 1 => feu1 <= 2; feuB <= 3;--le feu de l'axe A est
          orange, le feu de l'axe B est rouge
      when 2 => feu1 <= 3; feuB <= 1;--le feu de l'axe A est
          rouge, le feu de l'axe B est vert
      when 3 => feu1 <= 3; feuB <= 2; --le feu de l'axe A est
          rouge et le feu de l'axe B est orange
      when others => feu1 <= 1; feuB <= 4;
   end case;

end process;

LED_3210 <= feuA;
LED_7654 <= feuB;

END Behavioural;
```