Exercice 1: Exercice Préliminaire

Le but de ce premier exercice est d'étudier le fonctionnement de l'horloge que l'on nous fournit pour le TP.

```
entity blinker is
    port(Clk100MHz, PB 0 : in bit;
                                                              Déclaration des bits
    LED 0 : out bit);
                                                             d'entrées/sorties
end blinker;
architecture Behavioral of blinker is
                                                             Architecture
   alias reset is PB_0;
                                                             alias : signal de reset
   signal clk out : bit := '0';
                                                             clk out : signal après division
   constant clock divisor : integer := 100000000;
                                                             clock divisor : Cste de division
begin
clock divider : process(Clk100Mhz, reset)
                                                             Process diviseur de f :
  variable c: integer range 0 to clock divisor - 1 :=0;
                                                             Clk100MHz / clock div
begin
   if reset = '1' then
      c := 0:
      clk out <= '0';
   elsif Clk100MHz'event and Clk100MHz = '1' then
      if c < (clock_divisor - 1) / 2 then</pre>
         c := c + 1;
        clk_out <= '0';
      elsif c = (clock divisor - 1) then
        c := 0;
         clk_out <= '0';
        c := c + 1;
         clk out <= '1';
      end if:
   end if;
end process;
-- LED 0 <= clk out;
                                                             Sortie sur la LED
end Behavioral;
```

Le principe de fonctionnement de ce diviseur de fréquence est d'utiliser un signal pour permettre de partager la sortie clk_out avec les autres process que nous utiliserons dans les exercices 2 et 3. Il contient un reset qui suit le même résonnement à savoir un signal partageable pour pouvoir remettre l'horloge à zéro à l'aide du bouton poussoir PB 0.

L'horloge qui nous ai donné est cadencée à 100MHz or nous souhaitons une période de une seconde, on doit donc utiliser une constante clock_divisor initialisé à 1e8 pour obtenir une fréquence de 1Hz.

Le principe est simple, le fonctionnement du reset est normal, il remet le compteur à zero, sinon on incrémente une variable tant que, elle est nulle initialement. Tant qu'elle est strictement inférieure à la moitié du la constante de division la sortie et nulle, lorsque est supérieure la sortie passe à un jusqu'a ce qu'elle soit égale à la constante de division et la on reprend à zéro.

On utilise une LED pour visionner la sortie : LED 0.

Exercice 2: Feu de circulation

Cet exercice se propose de simuler le fonctionnement d'un feu rouge sur une intersection. Les contraintes sont : 10 secondes de rouge, 2 secondes d'orange. Nous avons fait le choix de considérer que les feux d'un axe passent au vert aussitôt que ceux de l'autre axe sont rouges. Il y a donc 8 secondes de vert.

Choix fait sur l'utilisation des 2 groupes de 4 LED de la carte FPGA :

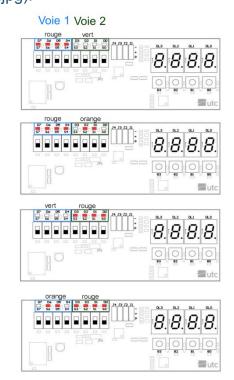
- 4 LED allumées : feux rouges sur l'axe correspondant.
- 2 LED centrales allumées : feux oranges sur l'axe correspondant.
- Toutes les LED sont éteintes : feux verts sur l'axe correspondant.
- LED 3210 et LED 7654 représentent chacun les feux d'un axe.

Voici le code correspondant :

```
entity exo2 is
                                                                       Entité
   PORT( PB 0 : IN BIT ;
                                                                       On définit les ports qu'on compte
          Clk100MHz: in bit;
                                                                       utiliser.
          LED_3210, LED_7654 : OUT BIT_VECTOR (3 downto 0));
end exo2;
architecture Behavioral of exo2 is
                                                                       Architecture
   signal clk_out : bit := '0';
                                                                       Définition de signaux et variables
   constant clock divisor : integer := 100000000;
                                                                       utiles au code VHDL par la suite.
   alias reset is PB 0;
begin
                                                                       Premier process : diviseur de fréquence fourni dans l'exercice.
PROCESS(Clk100Mhz, reset)
                                                                       Ce process est sensible à Clk100Mhz que nous avons bien défini en amont, ainsi qu'au reset
   variable c : integer range 0 to clock divisor - 1 := 0;
   if reset = '1' then
                                                                       qui est en fait PB_0, également
      c := 0;
                                                                       défini, dans l'entité.
      clk_out <= '0';
   elsif \overline{Clk100MHz}'event and Clk100MHz = '1' then
      if c < (clock_divisor - 1) / 2 then</pre>
          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)
                                                                       Deuxième process : il gère les
   VARIABLE etat : INTEGER RANGE 0 TO 19 :=0 ;
                                                                       feux et est sensible à clk_out, la
BEGIN
                                                                       sortie du process précédent.
   IF (clk_out 'event AND clk_out='1') THEN
       CASE etat IS
          WHEN 0 \Rightarrow LED_{7654} (3 DOWNTO 0) <= "1111";
                     LED 3210 (3 DOWNTO 0) <= "0000";
                     etat := etat+1 :
                                                                       Etat = 0 (Début du cycle) : un axe
                                                                       aux feux verts, l'autre aux feux
          WHEN 8 => LED_7654 (3 DOWNTO 0) <= "1111";
                                                                       rouges.
```

```
LED_3210 (3 DOWNTO 0) <= "0110";
                   etat := etat+1 ;
                                                                  Au bout de 8 secondes passées (etat devient 8) : les feux rouges
         WHEN 10 => LED 7654 (3 DOWNTO 0) <= "0000";
                    LED_3210 (3 DOWNTO 0) <= "1111";
                    etat := etat+1 ;
                                                                  passent à l'orange.
         WHEN 18 => LED_7654 (3 DOWNTO 0) <= "0110"; LED_3210 (3 DOWNTO 0) <= "1111";
                                                                  2 secondes plus tard : les feux
                                                                  oranges deviennent rouges,
                                                                  feux rouges deviennent verts.
                    etat := etat+1;
          WHEN 19 => etat := 0 ;
                                                                  8 secondes plus tard : les feux
                                                                  rouges passent à l'orange.
          WHEN OTHERS => etat := etat+1 ;
      END CASE ;
                                                                  Quand on arrive à 19, plutôt que
                                                                  d'incrément etat, on revient à 0
   END IF ;
                                                                  pour reprendre le cycle depuis le
                                                                  début.
   IF(reset='1') THEN etat:=0;
                                                                  On incrémente etat chaque seconde
   END IF ;
                                                                  grâce à l'horloge.
END PROCESS ;
END Behavioral;
                                                                  Reset asynchrone : plus performant
                                                                  dans les situations d'urgence. Il
                                                                  permet au système de revenir à un
                                                                  état où un feu est vert et l'autre
                                                                  rouge, l'état 0. Un
                                                                  synchrone devrait attendre un
                                                                  front montant d'horloge, et cette
                                                                  horloge à une fréquence faible
                                                                  (1Hz).
```

<u>Fonctionnement après programmation de la carte FGPA :</u> (jointe en annexe tableau.jpg).



On constate à l'utilisation que les l'affichage le temps entre chaque phase est correcte.

Exercice 3 : Prise en compte d'un capteur de voiture

On peut considérer cet exercice comme un prolongement du précédent : il faut considérer la présence de détecteurs sur chaque axe. Si aucune voiture de se présente sur une voie, le feux reste au vert sur l'autre, sinon on suit le fonctionnement normal.

On utilise en entrée comme détecteur, les deux boutons poussoirs PB_1 et PB_3.

Voici le code VHDL correspondant :

```
entity exo3 is
                                                                         Entité
   PORT( PB_0, PB_1, PB_3 : IN BIT ;
                                                                         PB_0 est notre reset
         Clk100MHz: in bit;
                                                                         PB_1 et 2 nos detecteurs
         LED 3210, LED 7654 : OUT BIT VECTOR (3 downto 0));
                                                                         Représentation feu
end exo3:
                                                                         Architecture
architecture Behavioral of exo3 is
   signal clk out : bit := '0';
                                                                        Package de l'horloge
   constant clock_divisor : integer := 100000000;
   alias reset is PB 0;
begin
                                                                         Process de l'horloge 1s
PROCESS(Clk100Mhz, reset)
   variable c : integer range 0 to clock_divisor - 1 := 0;
BEGIN
   if reset = '1' then
      c := 0;
      clk out <= '0';
   elsif \overline{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';</pre>
      end if;
   end if;
END PROCESS;
                                                                        Nouveau process pour feu
PROCESS(clk_out)
   VARIABLE etat : INTEGER RANGE 0 TO 19 :=0;
BEGIN
   IF (clk_out 'event AND clk_out='1') THEN
      CASE etat IS
         WHEN 0 => LED_7654 (3 DOWNTO 0) <= "1111";
LED_3210 (3 DOWNTO 0) <= "0000";
                    etat := etat+1 ;
                                                                        On rajoute la condition PB_3 = '1' pour indiquer
         WHEN 8 \Rightarrow IF PB 3='1' THEN
                        si des voitures sont sur
                        LED_3210 (3 DOWNTO 0) <= "0110";
                                                                         l'axe 1.
                        etat := etat+1 ;
                    END IF ;
```

```
WHEN 10 => LED_7654 (3 DOWNTO 0) <= "0000";
LED_3210 (3 DOWNTO 0) <= "1111";
                         etat := etat+1 ;
                                                                                     De même condition PB_1 = '1' pour indiquer si des
           WHEN 18 => IF PB_1='1' THEN
                         LED_7654 (3 DOWNTO 0) <= "0110";
LED_3210 (3 DOWNTO 0) <= "1111";
                                                                                      voitures sont sur l'axe 2.
                         etat := etat+1 ;
                         END IF ;
            WHEN 19 => etat := 0 ;
            WHEN OTHERS => etat := etat+1 ;
       END CASE ;
   END IF ;
   IF (reset='1') THEN etat :=0;
   END IF ;
END PROCESS ;
END Behavioral;
```

Comme précédemment on choisi un reset asynchrone pour avoir une plus grande réactivité. Malgré ce que dans notre cas l'horloge soit d'une seconde et donc le temps d'attente et court, si nous avions une horloge de 5 secondes il faudrait attendre au moins 4 seconde dans le pire des cas pour remettre la circulation dans le bon état ce qui est problématique si des voitures se présentent dans sur les 2 axes.

Conclusion:

Ce TP nous a permis de manipuler un process représentant une horloge par rapport au précédent TP où l'horloge était simulée par l'action de l'opérateur sur un bouton poussoir. Nous avons ainsi pu mettre en évidence les soucis liés à la synchronisation des signaux et continuer d'étudier les machines à états au travers d'un exercice simulant un carrefour routier.