

## Exercice 1: Exercice Préliminaire

<pre> 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;     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 &lt;= '0';         elsif Clk100MHz'event and Clk100MHz = '1' then             if c &lt; (clock_divisor - 1) / 2 then                 c := c + 1;                 clk_out &lt;= '0';             elsif c = (clock_divisor - 1) then                 c := 0;                 clk_out &lt;= '0';             else                 c := c + 1;                 clk_out &lt;= '1';             end if;         end if;     end process;      -- LED_0 &lt;= clk_out;  end Behavioral; </pre>	<p><b>Entité</b> Déclaration des bits d'entrées/sorties</p> <p><b>Architecture</b> alias : signal de reset clk_out : signal après division clock_divisor : Cste de division</p> <p>Process diviseur de f : Clk100MHz / clock_div</p> <p>Sortie sur la LED</p>
---	---

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

Compte-rendu TP03 MI01

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 :

<pre> entity exo2 is     PORT( PB_0 : IN BIT ;           Clk100MHz: in bit;           LED_3210, LED_7654 : OUT BIT_VECTOR (3 downto 0)); end exo2;  architecture Behavioral of exo2 is     signal clk_out : bit := '0';     constant clock_divisor : integer := 100000000;     alias reset is PB_0;  begin  PROCESS(Clk100Mhz, reset)     variable c : integer range 0 to clock_divisor - 1 := 0; BEGIN     if reset = '1' then         c := 0;         clk_out &lt;= '0';     elsif Clk100MHz'event and Clk100MHz = '1' then         if c &lt; (clock_divisor - 1) / 2 then             c := c + 1;             clk_out &lt;= '0';         elsif c = (clock_divisor - 1) then             c := 0;             clk_out &lt;= '0';         else             c := c + 1;             clk_out &lt;= '1';         end if;     end if; END PROCESS;  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 =&gt; LED_7654 (3 DOWNT0 0) &lt;= "1111" ;                      LED_3210 (3 DOWNT0 0) &lt;= "0000" ;                      etat := etat+1 ;              WHEN 8 =&gt; LED_7654 (3 DOWNT0 0) &lt;= "1111" ; </pre>	<p><b><u>Entité</u></b></p> <p>On définit les ports qu'on compte utiliser.</p> <p><b><u>Architecture</u></b></p> <p>Définition de signaux et variables utiles au code VHDL par la suite.</p> <p>Premier process : diviseur de fréquence fourni dans l'exercice. Ce process est sensible à Clk100Mhz que nous avons bien défini en amont, ainsi qu'au reset qui est en fait PB_0, également défini, dans l'entité.</p> <p>Deuxième process : il gère les feux et est sensible à clk_out, la sortie du process précédent.</p> <p>Etat = 0 (Début du cycle) : un axe aux feux verts, l'autre aux feux rouges.</p>
--	--

### Compte-rendu TP03 MI01

<pre> LED_3210 (3 DOWNT0 0) &lt;= "0110" ; etat := etat+1 ;  WHEN 10 =&gt; LED_7654 (3 DOWNT0 0) &lt;= "0000" ; LED_3210 (3 DOWNT0 0) &lt;= "1111" ; etat := etat+1 ;  WHEN 18 =&gt; LED_7654 (3 DOWNT0 0) &lt;= "0110" ; LED_3210 (3 DOWNT0 0) &lt;= "1111" ; etat := etat+1 ;  WHEN 19 =&gt; etat := 0 ;  WHEN OTHERS =&gt; etat := etat+1 ;  END CASE ;  END IF ;  IF(reset='1') THEN etat:=0; END IF ;  END PROCESS ;  END Behavioral;</pre>	<p>Au bout de 8 secondes passées (etat devient 8) : les feux rouges passent à l'orange.</p> <p>2 secondes plus tard : les feux oranges deviennent rouges, les feux rouges deviennent verts.</p> <p>8 secondes plus tard : les feux rouges passent à l'orange.</p> <p>Quand on arrive à 19, plutôt que d'incrément etat, on revient à 0 pour reprendre le cycle depuis le début.</p> <p>On incrémente etat chaque seconde grâce à l'horloge.</p> <p>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 reset synchrone devrait attendre un front montant d'horloge, et cette horloge à une fréquence faible (1Hz).</p>
--	--

Fonctionnement après programmation de la carte FPGA :  
(jointe en annexe tableau.jpg).



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

Compte-rendu TP03 MI01

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 feu 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 :

<pre> entity exo3 is     PORT( PB_0, PB_1, PB_3 : IN BIT ;           Clk100MHz: in bit;           LED_3210, LED_7654 : OUT BIT_VECTOR (3 downto 0)); end exo3;  architecture Behavioral of exo3 is      signal clk_out : bit := '0';     constant clock_divisor : integer := 100000000;     alias reset is PB_0;  begin  PROCESS(Clk100Mhz, reset)     variable c : integer range 0 to clock_divisor - 1 := 0; BEGIN      if reset = '1' then         c := 0;         clk_out &lt;= '0';     elsif Clk100MHz'event and Clk100MHz = '1' then         if c &lt; (clock_divisor - 1) / 2 then             c := c + 1;             clk_out &lt;= '0';         elsif c = (clock_divisor - 1) then             c := 0;             clk_out &lt;= '0';         else             c := c + 1;             clk_out &lt;= '1';         end if;     end if; END PROCESS;  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 =&gt; LED_7654 (3 DOWNT0 0) &lt;= "1111" ;                       LED_3210 (3 DOWNT0 0) &lt;= "0000" ;                       etat := etat+1 ;              WHEN 8 =&gt; IF PB_3='1' THEN                       LED_7654 (3 DOWNT0 0) &lt;= "1111" ;                       LED_3210 (3 DOWNT0 0) &lt;= "0110" ;                       etat := etat+1 ;                     END IF ;         END CASE;     END IF; END PROCESS; </pre>	<p><b>Entité</b></p> <p>PB_0 est notre reset PB_1 et 2 nos detecteurs Représentation feu</p> <p><b>Architecture</b></p> <p>Package de l'horloge</p> <p>Process de l'horloge 1s</p> <p>Nouveau process pour feu</p> <p>On rajoute la condition PB_3 = '1' pour indiquer si des voitures sont sur l'axe 1.</p>
---	--

Compte-rendu TP03 MI01

<pre>WHEN 10 =&gt; LED_7654 (3 DOWNT0 0) &lt;= "0000" ;             LED_3210 (3 DOWNT0 0) &lt;= "1111" ;             etat := etat+1 ;  WHEN 18 =&gt; IF PB_1='1' THEN             LED_7654 (3 DOWNT0 0) &lt;= "0110" ;             LED_3210 (3 DOWNT0 0) &lt;= "1111" ;             etat := etat+1 ;             END IF ;  WHEN 19 =&gt; etat := 0 ;  WHEN OTHERS =&gt; etat := etat+1 ;  END CASE ;  END IF ;  IF (reset='1') THEN etat :=0; END IF ;  END PROCESS ;  END Behavioral;</pre>	<p>De même condition PB_1 = '1' pour indiquer si des voitures sont sur l'axe 2.</p>
--	---

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 avons 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.

**Compte-rendu TP03 MI01**

**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.