**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**;  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 <= '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**;    *--* LED\_0 <= clk\_out;  **end** Behavioral; | **Entité**  Déclaration des bits d'entrées/sorties  **Architecture**  alias : signal de reset  clk\_out : signal après division  clock\_divisor : *Cste de division*  *Process diviseur de f :*  *Clk100MHz / clock\_div*  *Sortie sur la LED* |

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 :

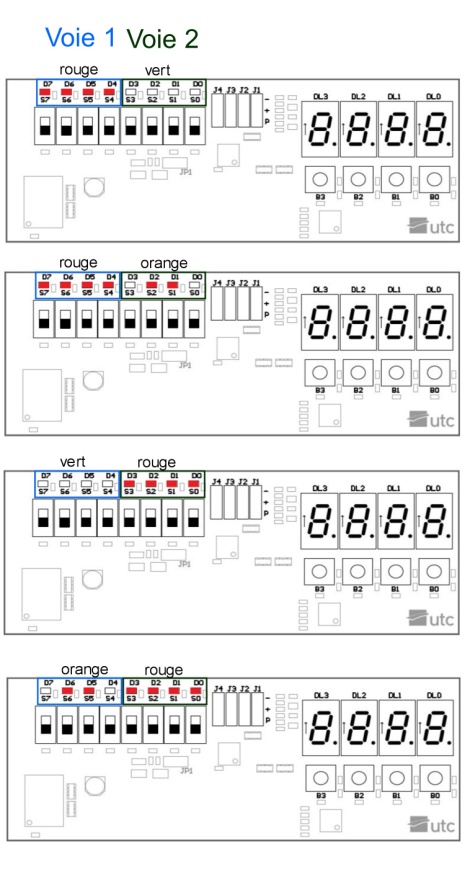
* 4 LED allumées : feux rouges sur l’axe correspondant.
* 2 LED centrales allumées : feux oranges sur l’axe correspondant.
* Toutes les LED sont éteintes : feux verts sur l’axe correspondant.
* LED\_3210 et LED\_7654 représentent chacun les feux d’un axe.

Voici le code correspondant :

|  |  |
| --- | --- |
| entity exo2 is  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 <= '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 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 ;  WHEN 8 => LED\_7654 (3 DOWNTO 0) <= "1111" ;  LED\_3210 (3 DOWNTO 0) <= "0110" ;  etat := etat+1 ;  WHEN 10 => LED\_7654 (3 DOWNTO 0) <= "0000" ;  LED\_3210 (3 DOWNTO 0) <= "1111" ;  etat := etat+1 ;  WHEN 18 => LED\_7654 (3 DOWNTO 0) <= "0110" ;  LED\_3210 (3 DOWNTO 0) <= "1111" ;  etat := etat+1 ;  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; | **Entité**  On définit les ports qu’on compte utiliser.  **Architecture**  Définition de signaux et variables utiles au code VHDL par la suite.  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.  Deuxième process : il gère les feux et est sensible à clk\_out, la sortie du process précédent.  Etat = 0 (Début du cycle) : un axe aux feux verts, l’autre aux feux rouges.  Au bout de 8 secondes passées (etat devient 8) : les feux rouges passent à l’orange.  2 secondes plus tard : les feux oranges deviennent rouges, les feux rouges deviennent verts.  8 secondes plus tard : les feux rouges passent à l’orange.  Quand on arrive à 19, plutôt que d’incrément etat, on revient à 0 pour reprendre le cycle depuis le début.  On incrémente etat chaque seconde grâce à l’horloge.  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). |

Fonctionnement  après programmation de la carte FGPA :

(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  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 <= '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 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 ;  WHEN 8 => IF PB\_3='1' THEN  LED\_7654 (3 DOWNTO 0) <= "1111" ;  LED\_3210 (3 DOWNTO 0) <= "0110" ;  etat := etat+1 ;  END IF ;  WHEN 10 => LED\_7654 (3 DOWNTO 0) <= "0000" ;  LED\_3210 (3 DOWNTO 0) <= "1111" ;  etat := etat+1 ;  WHEN 18 => IF PB\_1='1' THEN  LED\_7654 (3 DOWNTO 0) <= "0110" ;  LED\_3210 (3 DOWNTO 0) <= "1111" ;  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; | **Entité**  PB\_0 est notre reset  PB\_1 et 2 nos detecteurs  Représentation feu  **Architecture**  Package de l'horloge  Process de l'horloge 1s  Nouveau process pour feu  On rajoute la condition PB\_3 = '1' pour indiquer si des voitures sont sur l'axe 1.  De même condition PB\_1 = '1' pour indiquer si des voitures sont sur l'axe 2. |

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.