TP VHDL séquentiel, comptage du temps

Dans le précédent TP, nous avons étudié l'utilité d'une machine à état pour « contrôler » une séquence d'événements. L'objectif de ce TP est de compléter cette étude par l'application d'une machine à états au comptage du temps, avec ou sans prise en compte d'entrées extérieures (hormis l'horloge).

Exercice 1. Exercice Préliminaire

La plateforme de TP met à votre disposition d'une horloge cadencée à 100MHz (`Clk100MHz` dans le fichier `carte_tp.ucf`). Les temps considérés dans les exercices suivants étant de l'ordre de la seconde, il est utile de générer un signal interne d'horloge à une fréquence de 1 Hz. On peut pour cela utiliser un diviseur de fréquence donc le code vous est donné ci-dessous.

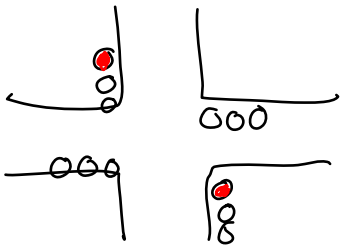
```
entity blinker is
  port(Clk100MHz, PB_0 : in bit;
       LED_0 : out bit);
end blinker;

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;
```

- Quel est le principe de fonctionnement de ce diviseur de fréquence ?

Dans la suite, vous incorporerez à votre code VHDL le process `clock_divider` afin de produire le signal d'horloge que vous utiliserez pour faire évoluer vos machines à états.

Exercice 2. Feu de circulation

Le contrôleur d'un feu de circulation simple mettant en œuvre un carrefour à 2 axes principaux. Il vous est demandé de développer ce contrôleur sachant que :



Les deux axes sont symétriques, la durée de la phase rouge est de 10 s, orange 2 s.

- Définir les durées de toutes les phases, et le déroulement de la séquence.
- Réfléchir aux types de modélisation possibles en VHDL et définir celle qui est plus appropriée.
- Coder, simuler et intégrer dans la FPGA.
- Tester la fonction
- Comment peut-on intégrer une mise à zéro fiable du système (reset) ?
- Pour chacune des solutions, coder, simuler, programmer le FPGA et tester. Quelles sont vos conclusions ?

Vous utiliserez les signaux suivants :

- Feux de l'axe 1 : LED_3210
- Feux de l'axe 2 : LED_7654
- Reset : PB_0 (bouton poussoir de droite).

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

Dans cette deuxième étape, on prendra en considération une entrée venant d'un capteur de véhicules sur les deux axes (boucles d'induction, radar, etc.). Le principe de commande cette fois-ci est le suivant :

- En cas d'absence de véhicules sur l'axe 1, laisser le feu sur 'vert' sur l'axe 2.
- Si un véhicule se présente au feu sur l'axe 1, passer à l'orange sur l'axe 2, une fois que la durée du vert est respectée.
- Et vice-versa

Apportez les modifications à votre conception pour intégrer ces nouvelles consignes :

- définir les phases, et le déroulement de la séquence.
- Modifier la modélisation de la séquence, sans prise en compte du reset.
- Coder (sans prise en compte du reset), simuler, programmer le FPGA et tester.
- Recommencer la dernière étape en prenant en compte le reset.
- Quel type de reset doit-on choisir (synchrone ou asynchrone) et à quel moment peut-il intervenir ?

Vous utiliserez les mêmes signaux que pour l'exercice 2. Les capteurs de présence de véhicules seront simulés par les signaux PB_2 et PB_3 (deux boutons poussoirs du centre).