

TP02 - VHDL séquentiel temporisé

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", constitué 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 désormais 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 inférieure 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 clk_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.

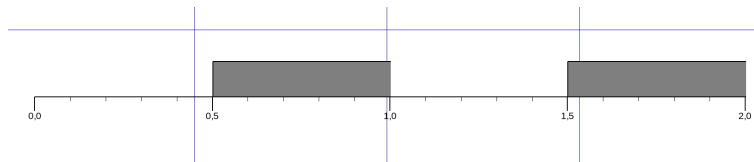


Figure 1: Chronogramme du fonctionnement du blinker

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 $T_{rouge} = T_{vert} + T_{orange}$.

La plus petite unité de temps utilisée est donc 2 secondes (T_{sync}), on peut donc utiliser une fréquence d'horloge de $f=1/T_{sync}=0,5\text{Hz}$

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 peut réaliser cette modélisation en comportemental ou en structurel.

On choisi ici de modéliser les feu de circulation à l'aide d'un code en VHDL comportemental car cela nous semble plus simple d'inclure le blinker dans le code VHDL. En structurel nous aurions du créer une entité blinker qui aurait ensuite été utilisée dans le code VHDL des feux de circulation.

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

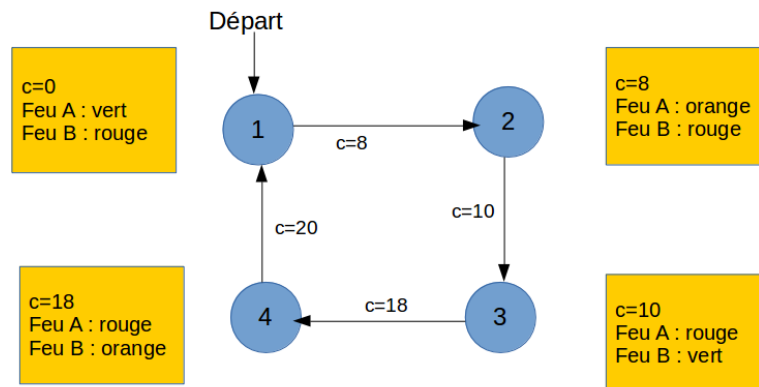


Figure 2: Machine à état des feux

Dans notre schéma, les états sont les cercles bleus, les flèches représentent les transitions accompagnées de la variable qui déclanche la transition. Les rectangles oranges déterminent ce qui est fait à chaque fois qu'on entre dans l'état.

La machine reste dans l'état 1 pendant 8 secondes, puis dans l'état 2 pendant 2 secondes puis dans l'état 3 pendant 8 secondes et enfin dans l'état 4 pendant 2 secondes.

Ainsi de 0 à 8 secondes, le feu A est vert et le feu B est au rouge, de 8 à 10 secondes, le feu A est orange et le feu B est rouge, de 10 à 18 secondes, le feu A est rouge et le feu B est vert et de 18 à 30 secondes, le feu A est rouge et le feu B est orange. A la fin, on recommence le cycle depuis le début.

On utilisera dans le code VHDL des entiers pour différencier les couleurs de feu, soit la valeur 1 pour signaler un feu rouge, 2 pour le feu orange et 4 pour le feu vert.

0.2.1 Code VHDL sans prendre en compte le reset

```
ENTITY feu IS
  PORT (
    Clk100MHz, PB_0 : IN BIT;
    LED_3210, LED_7654 : OUT INTEGER RANGE 0 TO 15
  );
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 (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 := 4; --le feu de l'axe A est
        vert et le feu de l'axe B est rouge
    WHEN 1 => feuA := 2; feuB := 4; --le feu de l'axe A est
        orange, le feu de l'axe B est rouge
    WHEN 2 => feuA := 4; feuB := 1; --le feu de l'axe A est
        rouge, le feu de l'axe B est vert
    WHEN 3 => feuA := 4; feuB := 2; --le feu de l'axe A est
        rouge et le feu de l'axe B est orange
    WHEN OTHERS => feuA := 1; feuB := 4;
END CASE;

LED_3210 <= feuA;
LED_7654 <= feuB;
END PROCESS;
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.

0.2.2 Simulation sans le reset

Ici on voit bien que le feu vert de l'axe A dure 8 seconde, le feu orange de l'axe A 2 secondes et le feu rouge de l'axe B 10 secondes. (on commence au premier front montant de l'horloge d'où le décalage de 0,5s sur la simulation) La deuxième partie du cycle de la 10ème à la 20ème seconde est l'inverse de

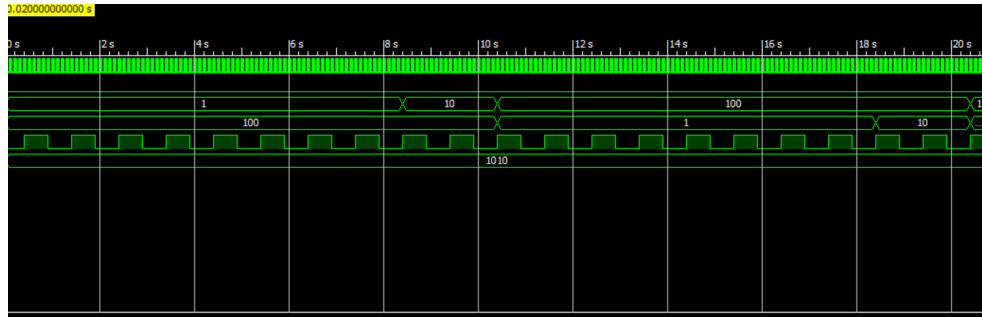


Figure 3: Simulation sans le reset

la première partie du cycle.

0.2.3 Code VHDL avec le reset

Nous cherchons maintenant à intégrer un reset au code précédent.

Pour le reset nous avons défini le comportement du reset pour chacun des 4 états de la machine, afin d'essayer d'avoir le reset le plus sécurisé possible pour les automobilistes.

Ainsi l'état de reset est défini ainsi : feu A au vert et feu B au rouge.

Ainsi dans l'état 0 (feu A au vert et feu B au rouge), si on appuie sur le reset, on remet le compteur à 0 et on reste dans cet état.

Dans l'état 1 (le feu A est orange et le feu B est au rouge), si on appuie sur le reset on peut directement repasser dans l'état 0 aussi, car passer du orange vers le vert n'est, pour nous, pas un danger pour les automobilistes.

Dans l'état 2 (le feu A est rouge et le feu B est vert), si on appuie sur reset, il faut passer le feu B au orange pendant 2 secondes puis ensuite passer au rouge et passer le feu A au vert. Cette séquence d'instruction correspond à celle qui se passe lorsqu'on est dans l'état 3 et qu'on repasse à l'état 0.

Dans l'état 3 (le feu A est au rouge et le feu B est au orange), si on appuie sur le reset il ne se passe rien, car on souhaite finir normalement le cycle et repasser ensuite à l'état 0, comme prévu normalement dans le code VHDL.

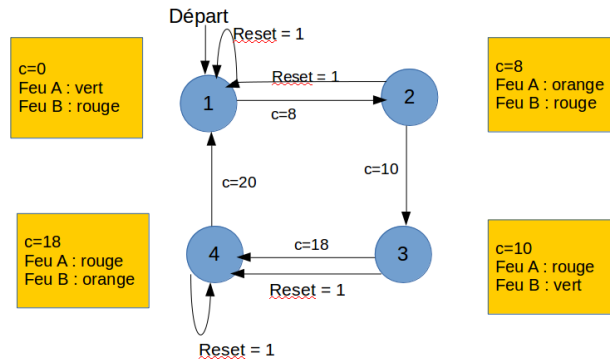


Figure 4: Machine à état des feu avec reset

Voici le code VHDL du PROCESS avec le reset :

```
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
IF etat = 2 THEN
etat := 3;
c := 18;
ELSIF (etat = 0 OR etat = 1) THEN
c := 0;
etat := 0;
ELSE
c := c - 1;
etat := 3;
END IF;

IF (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 := 4; --le feu de l'axe A est vert
              et le feu de l'axe B est rouge
    WHEN 1 => feuA := 2; feuB := 4; --le feu de l'axe A est
              orange, le feu de l'axe B est rouge
    WHEN 2 => feuA := 4; feuB := 1; --le feu de l'axe A est rouge,
              le feu de l'axe B est vert
    WHEN 3 => feuA := 4; feuB := 2; --le feu de l'axe A est rouge
              et le feu de l'axe B est orange
    WHEN OTHERS => feuA := 1; feuB := 4;
END CASE;

LED_3210 <= feuA;
LED_7654 <= feuB;
END PROCESS;
END Behavioural;
```

0.2.4 Simulation avec reset

Le reset est activé pendant l'état 1 (feu vert sur l'axe A). Le compteur est donc remis à 0 et le cycle de 8 secondes recommence au front montant de l'horloge qui suit le reset.

Le reset est activé pendant l'état 2 (feu orange sur l'Axe A). On retourne donc à l'état 0 (feu vert sur l'axe A) et on recommence un cycle de 8 secondes.

Le reset est activé pendant l'état 3 (feu vert sur l'axe B) On passe à l'état 4 (on passe le feu orange sur l'axe B) puis on passe à l'état 1.

Enfin si le reset est activé à l'état 4 on ne fait rien et on attend de repasser à l'état 0.

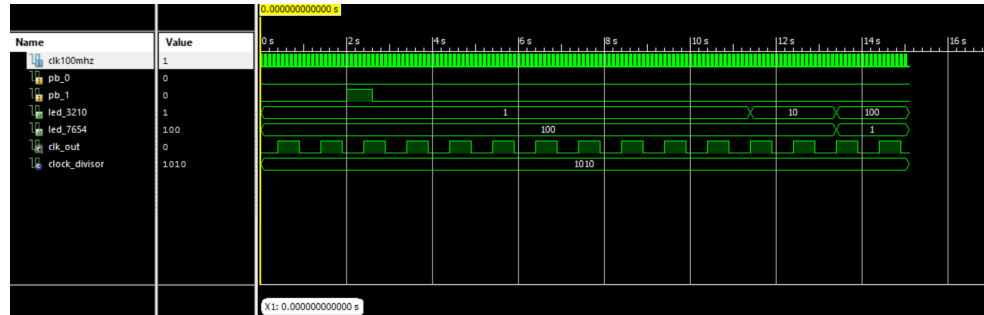


Figure 5: Simulation avec le reset : Cas 1

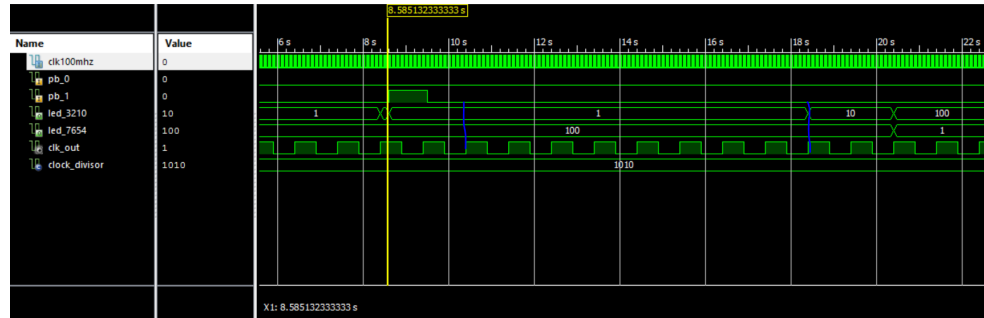


Figure 6: Simulation avec le reset : Cas 2

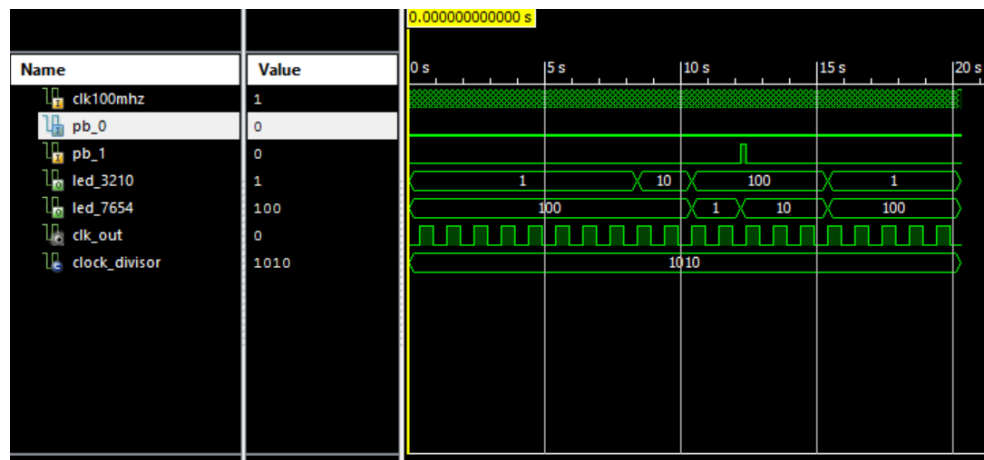


Figure 7: Simulation avec le reset : Cas 3

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 :

- S'il n'y a pas de véhicules sur l'axe A, laisser le feu au vert sur l'axe B.
- Si un véhicule arrive sur l'axe A, le feu de l'axe B doit passer au orange, tout en ayant respecté les 8 secondes de feu vert sur l'axe A
- Ces comportements sont valable pour le feu de l'axe B.

On décide donc, de définir les 4 états de notre machine à état ainsi :

- Dans l'état 1, on attend les 8 secondes puis on passe a l'état 2 si on detecte une voiture avec le capteur B.
- Dans l'état 2, on ne s'occupe pas des capteurs de voitures, dans tous les cas on passe à l'état 3.
- Dans l'état 3, on attend les 8 secondes puis on passe a l'état 4 si on detecte une voiture avec le capteur A.
- Dans l'état 4, quelque soit la configuration des capteurs on passe à l'état 1 lorsque c=20.

On va donc bloquer le compteur qui permet de passer de l'état feu verre a l'état feu orange avec une condition sur le capteur au moment de passer au orange.

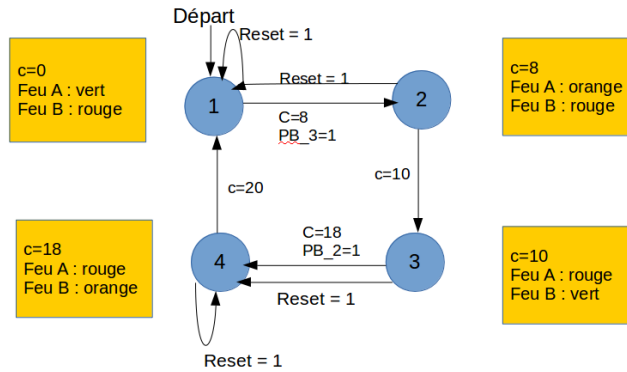


Figure 8: Machine à états des feu avec le reset et les capteurs

0.3.1 Code VHDL

```

PROCESS (clk_out, reset)
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
    IF etat = 2 THEN
      etat := 3;
      c := 18;
    ELSIF (etat = 0 OR etat = 1) THEN
      c := 0;
      etat := 0;
    ELSE
      c := c - 1;
      etat := 3;
    END IF;
  END IF;

  ELSIF (clk_out'EVENT AND clk_out = '1') THEN
    IF c < 8 THEN
      c := c + 1;
      etat := 0;
    ELSIF c < 10 AND PB_3 = '1' AND c > 7 THEN

```

```
        c := c + 1;
        etat := 1;
    ELSIF c < 18 AND c > 9 THEN
        c := c + 1;
        etat := 2;
    ELSIF c < 20 AND PB_2 = '1' AND c > 17 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 := 4; --le feu de l'axe A est vert et le feu de l'axe B
            est rouge
    WHEN 1 => feuA := 2;
        feuB := 4; --le feu de l'axe A est orange, le feu de l'axe B
            est rouge
    WHEN 2 => feuA := 4;
        feuB := 1; --le feu de l'axe A est rouge, le feu de l'axe B
            est vert
    WHEN 3 => feuA := 4;
        feuB := 2; --le feu de l'axe A est rouge et le feu de l'axe B
            est orange
    WHEN OTHERS => feuA := 1;
        feuB := 4;
END CASE;

LED_3210 <= feuA;
LED_7654 <= feuB;
END PROCESS;
```

Pour le reset nous avons décidé de garder le comportement que nous avions défini précédemment. En effet, il ne nous a pas semblé utile de changer son comportement et son mode de fonctionnement puisque ce dernier répond déjà aux exigences que nous nous étions fixé.

0.3.2 Simulation

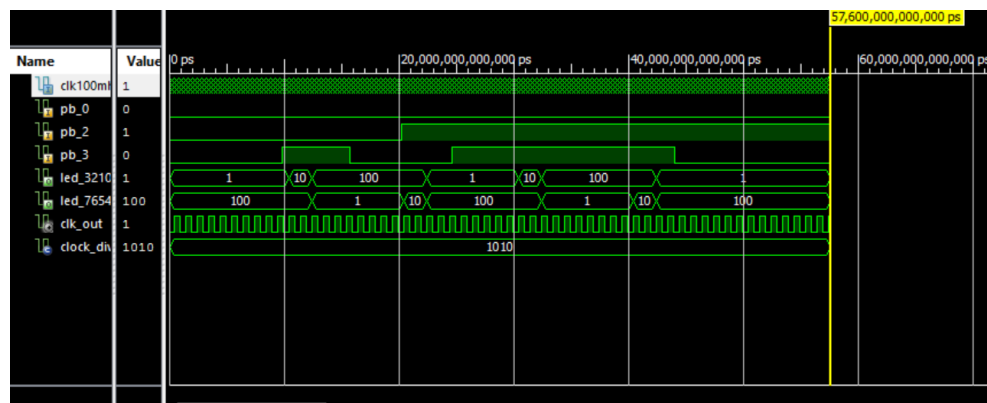


Figure 9: Simulation des feux avec reset et capteurs