1) Soit le schéma RTL suivant. Nous avons un compteur avec le signal interne *s_end_count_10* qui indique quand une LED aura clignotée 10 fois (10 fois s_end_cycle_1). Une la machine à état qui permet d'indiquer le code couleur de la led à clignoter en fonction du *s_end_count_10*, et un signal update qui dépend du signal de fin de compteur, pour arriver en même temps que le *s_color_code* fourni par la machine à état.

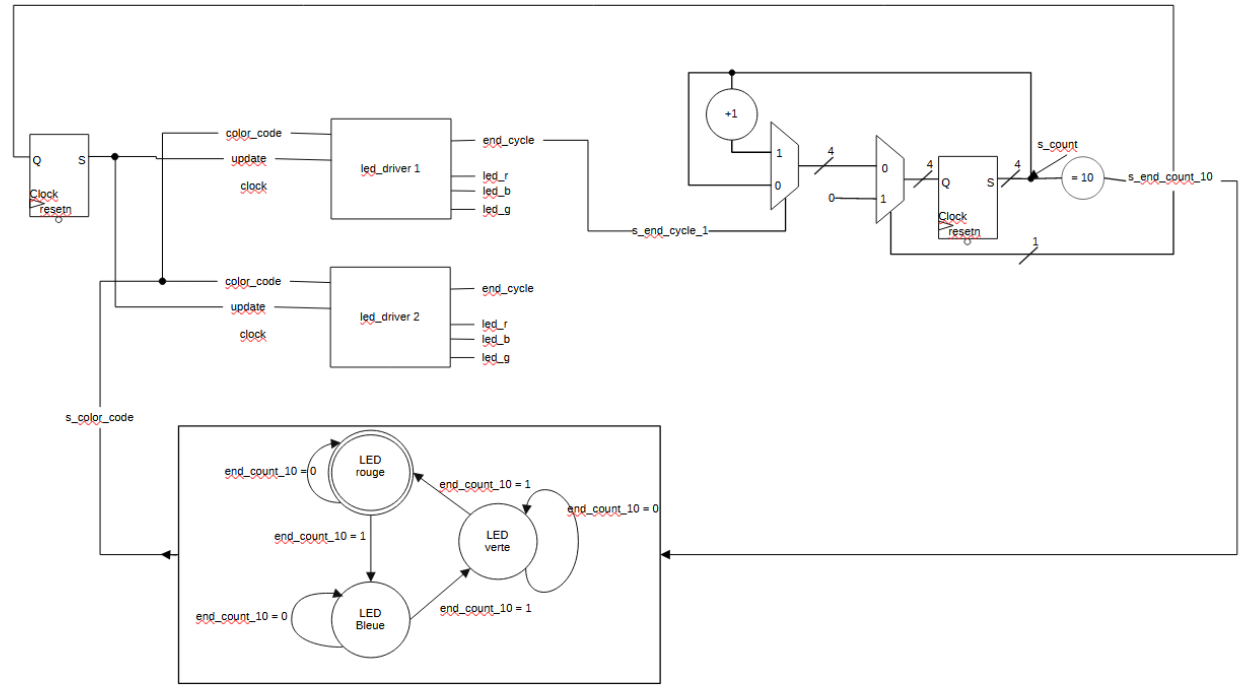


*Figure 1: RTL pour faire clignoter led0 et led1*

2) machine à état :

- déclaration :

```
type state_color is (red, blue, green);
   signal current_state_color  : state_color;
   signal next_state_color     : state_color;
```

process :

```
   fsm_color : process(current_state_color, s_update)
    begin
        -- update <= '0';
        case current_state_color is
            when red =>
                s_color_code <= "01";
                if (s_update = '1') then
                    next_state_color <= blue;
                else
                    next_state_color <= red;
                end if;
            when blue =>
                s_color_code <= "11";
```

```
            if(s_update = '1') then
                next_state_color <= green;
            else
                next_state_color <= blue;
            end if;
        when green =>
            s_color_code <= "10";
            if(s_update = '1') then
                next_state_color <= red;
            else
                next_state_color <= green;
            end if;
    end case;
end process;
```

synchronisation :

```
current_state_color <= next_state_color;
```

- compteur

  registre

```
 if(s_end_count_10 = '1') then       -- remise à zéro
            s_count <= (others => '0');
        elsif(s_end_cycle_1 = '1') then     -- incrémentation du compteur
            s_count <= s_count + 1;
        else            -- on maintient le compteur à sa valeur actuelle
            s_count <= s_count;
        end if;
```

  hors registre

```
 with s_count select
     s_end_count_10 <= '1' when std_logic_vector(to_unsigned(cycle_led_max-1,4)),
     '0' when others;
```

- signal update (dans registre)

```
s_update <= s_end_count_10;
```

3) Nous réalisons un testbench sur ce signal et obtenons les courbes suivantes :

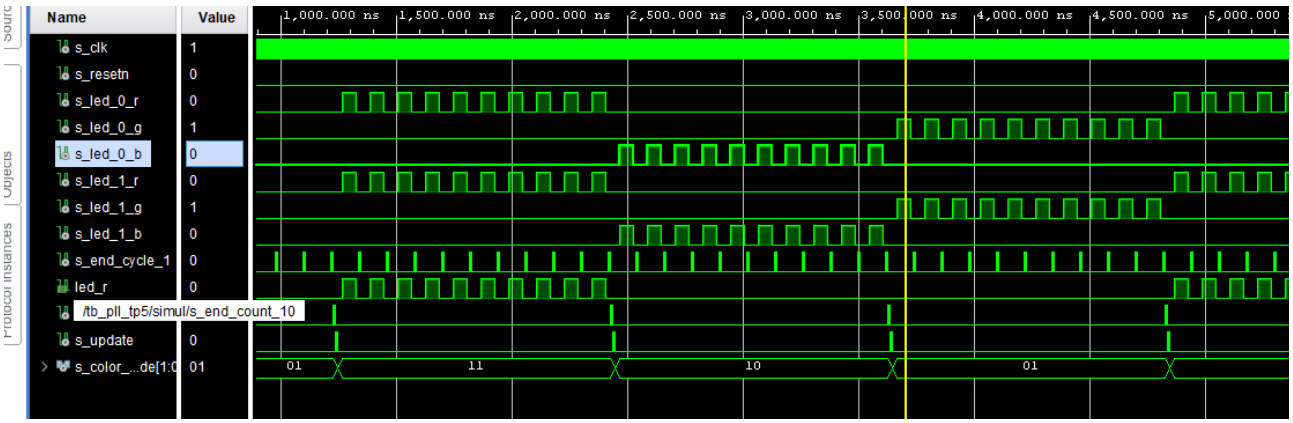


*Figure 2: led clignotante avec une même horloge*

Nous observons bien le code couleur qui change tous les 10 clignotement de LED, et ça commence par le rouge, le bleue, puis le vert avant de recommencer.

6) Après avoir modifier le design et le testbench pour clkA à 250 Mhz et clkB à 50 Mhz, nous observons alors que le driver de la led_0 fonctionne bien, alors que celui du driver de la LED_1 ne fait pas clignoter les LED. Ceci s'explique par le fait que pour la 2me LED, le update n'arrive pas synchronisé au front montant de clkB.

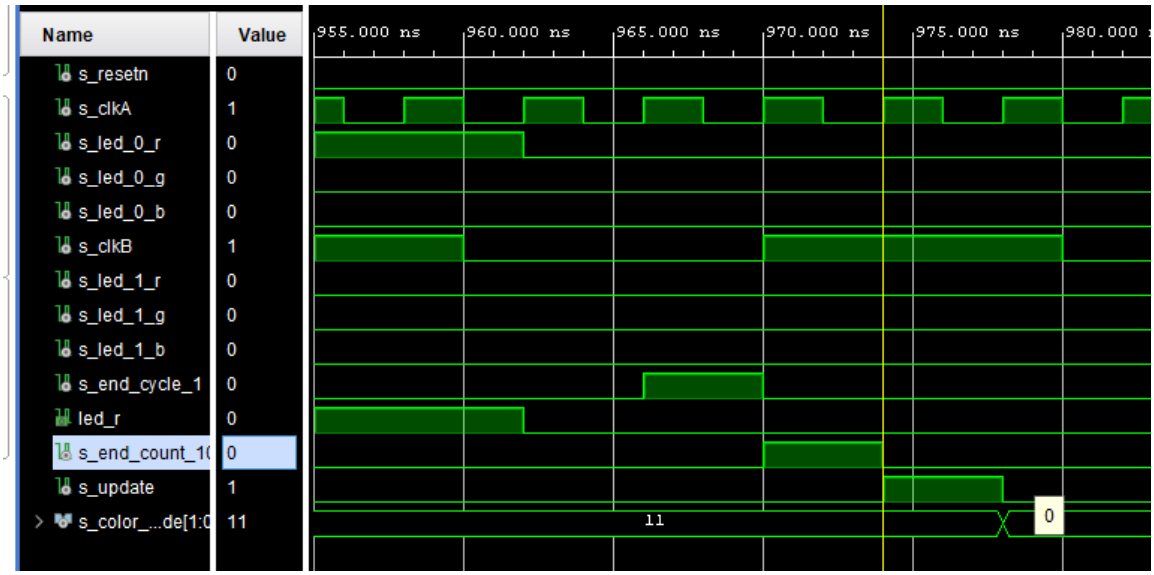


*Figure 3: décalage de update par rapport à clkB*

7) D'après https://nandland.com/lesson-14-crossing-clock-domains/ , nous allons étirer s_update pour avoir s_update_stretch qui soit à un quand clkB a un front montant, soit 4 signaux d'horloge de clkA.

```
Process[…]
        if(rising_edge(clkA))
                if s_update = '1' then
                     s_count_stretch <= 4;
                 elsif s_count_stretch > 0 then
                     s_count_stretch <= s_count_stretch -1;
                end if;
[…] end process ;

 s_update_stretch <= '1' when s_count_stretch > 0 else '0';
```

8)

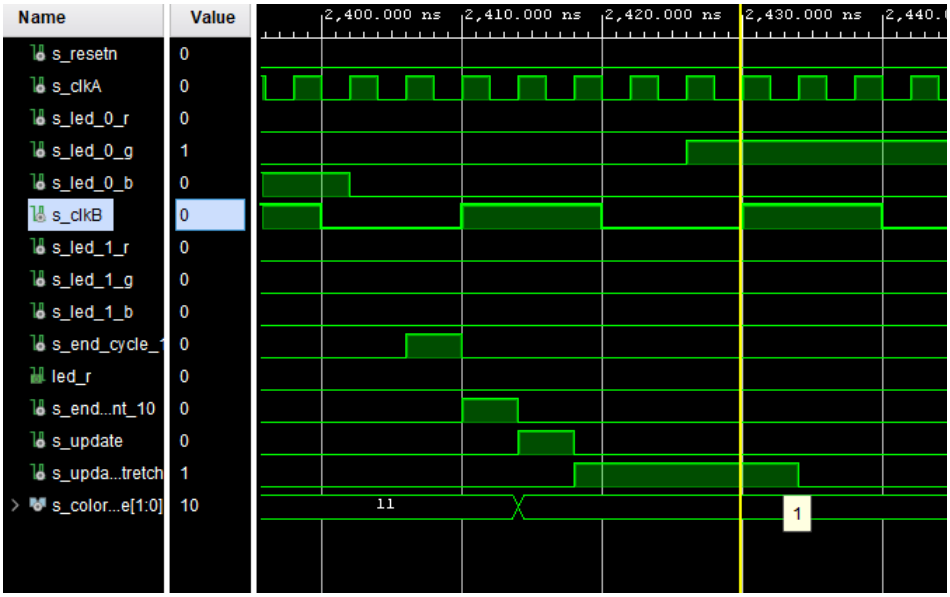


*Figure 4: signal update étiré pour être à 1 au front montant de clkB*

9) l'ip pour pouvoir réaliser la PLL sera « clocking wizard ».

Nous la relierons aux signaux tel que montrés sur la figure ci-dessous.

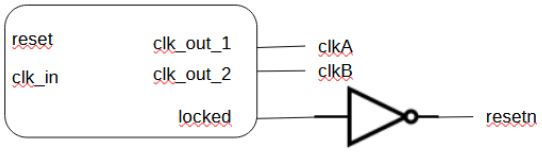


*Figure 5: schéma RTL et signaux*

Locked est un signal qui passera à 1 si les horloges de sortie sont stables. Donc tant que ce signal est à 0, les courbes ne sont pas relancée.

```
Detailed RTL Component Info :
+---Adders :
          2 Input    4 Bit       Adders := 1
          2 Input    3 Bit       Adders := 1
+---Registers :
                       4 Bit    Registers := 1
                       3 Bit    Registers := 1
                       2 Bit    Registers := 2
                       1 Bit    Registers := 7
+---Muxes :
          2 Input    4 Bit        Muxes := 1
          4 Input    3 Bit        Muxes := 2
          3 Input    2 Bit        Muxes := 2
          2 Input    2 Bit        Muxes := 3
          2 Input    1 Bit        Muxes := 4
--------------------------------------------------------------------------------
```

Dans le rapport de synthèse ci-dessus, nous pouvons vérifier :

- 2 adders :
  - 1 adder sur 4 bit pour le compteur sur 10 bits (
  - 1 adder sur 3 bit, pour le décompte pour s_update_stretch (étale sur 4 signal d'horloge
- 11 registres
  - 1 registre sur 4 bit pour le compteur de 10 bit
  - 1 registre sur 3 bit pour le décompte qui permet d'étaler update
  - 3 registre sur 2 bit :
    - machine a état pour le code la commande code couleur envoyé au led_drivers
    - 2 pour les led_drivers, la gestion du code couleur
  - 7 sur 1 bit
    - 4 registre pour les led_driver pour le signal led_on
    - 2 registre pour la machine à état, le lien entre next_stage et current_stage dans les led_drivers
    - 1 pour la machine à état du code couleur, dans le projet pll
- 11 multiplexeurs
  - 1 à 2 entrées sur 4 bit pour le compteur de 10
  - 2 à 4 entrées sur 3 bit, dans les led_driver pour le message qui commande les leds qui clignotent
  - 2 à 3 entrées sur 2 bits
    - pour la machine à état qui commande le code_couleur envoyé au led_driver

        - pour le code_couleur de next_state_color
    - 3 à 2 entrées sur 2 bits
        - 2 pour le signal de code_couleur dans la machine à état
    - 4 à 2 entrées sur 1 bits qui sont pour led_driver, utilisés pour la machine à état.

```
|9     |FDCE           |    71|
|10    |FDRE           |     3|
|11    |IBUF           |     1|
|12    |OBUF           |     6|
```

- 71 registres à reset asynchrones
    - 56 pour les compteurs dans les led_drivers (2 x 28 bits)
    - 4 registres (2 dans chaque led_driver) pour le code_color_mem sur 2 bits
    - 2 registres (1 dans chaque led_driver) pour la machine à état qui commande led_on
    - 2 registres dans led_driver_1, pour la détection de front descendant de led_on.
    - 1 registre pour décaler le s_count_10 en s_update
    - 4 registres pour le compteur de 10 (4 bits)
    - 2 registre pour la machine à état qui commande le code couleur
- 3 registres à resets synchrone pour les 3 bits à étirer pour le update
- un buffer d’entrée : reset
- 6 buffers de sorties : toutes les led

Dans le « timing summary », nous retrouvons bien nos trois signaux d’horloges à 100, 250 et 50 Mhz. Mais suite à un problème de routage pour le timing, nous avons passé le compteur de led_driver en reset synchrone.

Nous pouvons vérifier le bon fonctionnement sur la carte, sur les courbes de debug généré par ILA qui suivent.

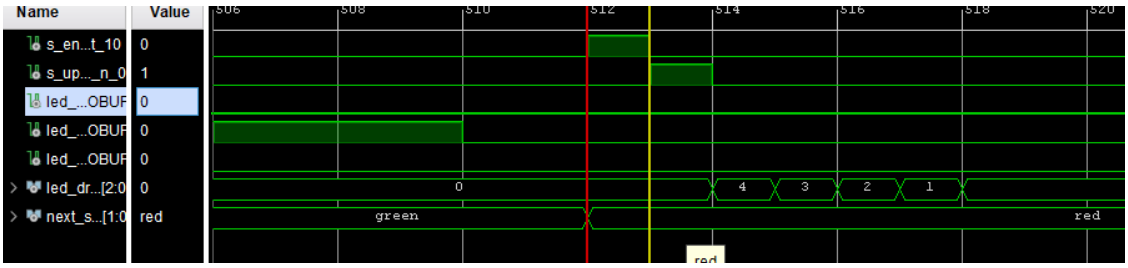


*Figure 6: fin de décompte de 10 clignotement*

Sur la figure ci-dessus, nous pouvons constater que lorsque le clignotement de la led à atteint 10, le signal l'indiquant passe à 1, et met à jour le code couleur des leds, faisant par la suite clignoter de vert à rouge.

Nous observons aussi le décompte utilisé pour avoir un update qui soit à 1 pour le led_driver_2.

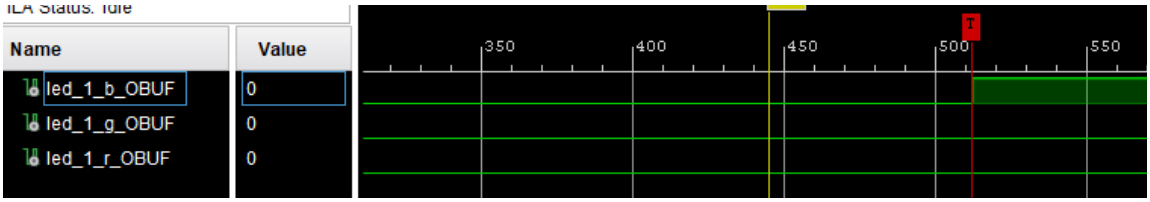


*Figure 7: clignotement led 1 bleue*

Nous pouvons constater sur la figure ci-dessus la led 1 b s'allumer

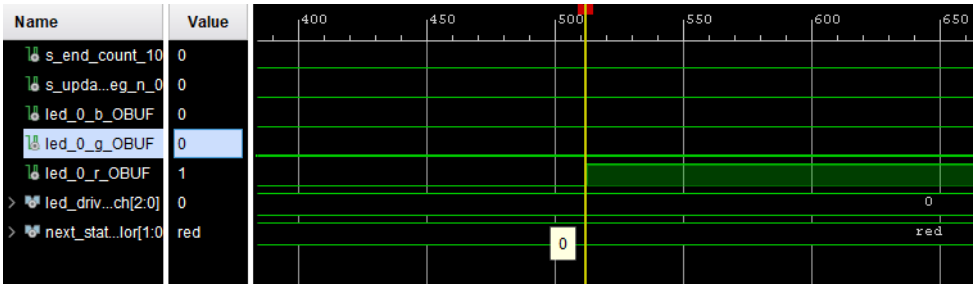


*Figure 8: led_0 rouge allumée*

Sur la figure ci-dessus:led_rouge qui s'allume, lors d'un clignotement.