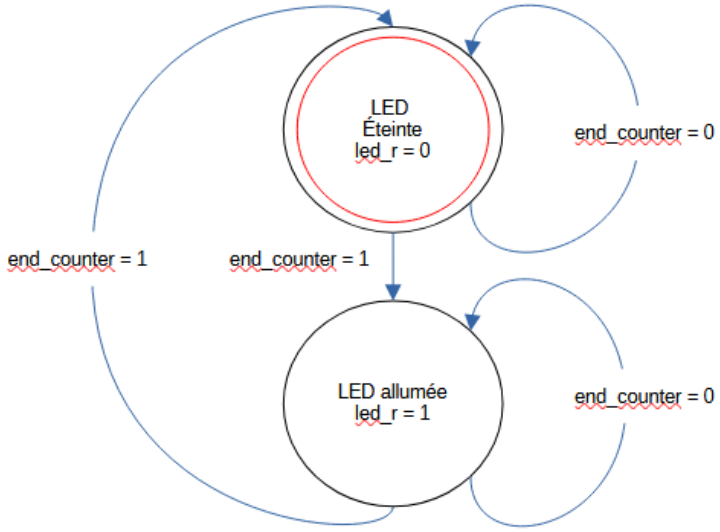
# Tp4 part 1 :

# led driver

1)

Soit la machine d'état suivante qui permet de gérer le clignotement des LED




Nous aurons en signaux d'entrée :

- *clk*
- *resetn*

En sortie :

- *led_r*

2)

Soit lors de l'appuie du *bouton_0*, nous aurons un signal green_cycle, qui indiquera dans le cas où nous sommes à 1, que nous devrons faire clignoter la LED verte, sinon la LED rouge. Soit sur la machine à état, *led_r* devient *led_on*.

Le schéma rtl permettant de faire commander quelle LED s'allume en appuyant sur le bouton_0.

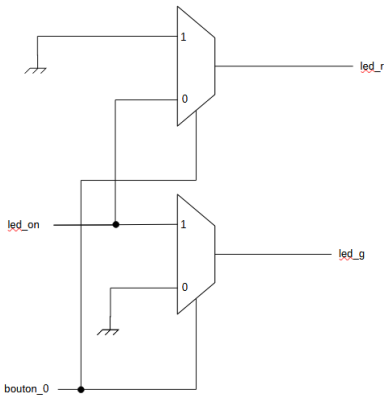
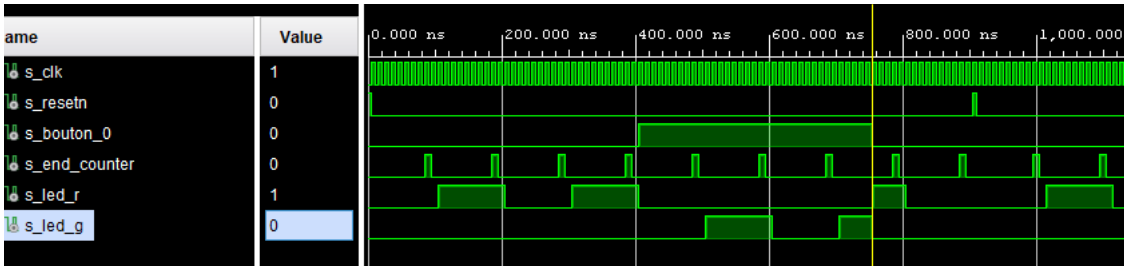



*Figure 1: schéma RTL pour commande de clignotement des LED*

4)




Nous observons bien le signal qui clignote en rouge (*s_led_r*) si le bouton_0 est à ‘0’, sinon c’est la led verte qui clignote, si le bouton est pressé sur plus d’un cycle d’horloge.

5)

Nous voulons la LED qui s’allume en verte juste sur un coup d’horloge, quand le bouton_0 est appuyé ; soit un signal *cmd* qui passe à l’état haut au moment où le bouton est appuyé, sur un coup d’horloge (voir chronogramme ci-dessous).

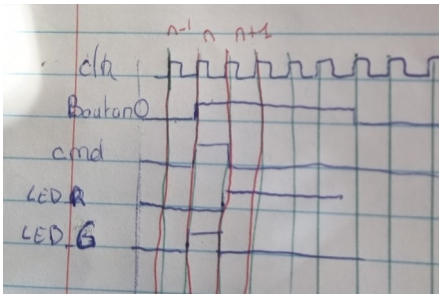



*Figure 2: chronogramme signal de commande (cmd)*

Soit le moment où le signal *cmd* est à l'état haut au temps n : le signal *bouton_0* est à l'état haut au temps n, et l'état bas au temps n-1. Un registre permet de récupérer ce temps n-1 et n à la fois, tel que montré sur le schéma rtl ci-dessous.

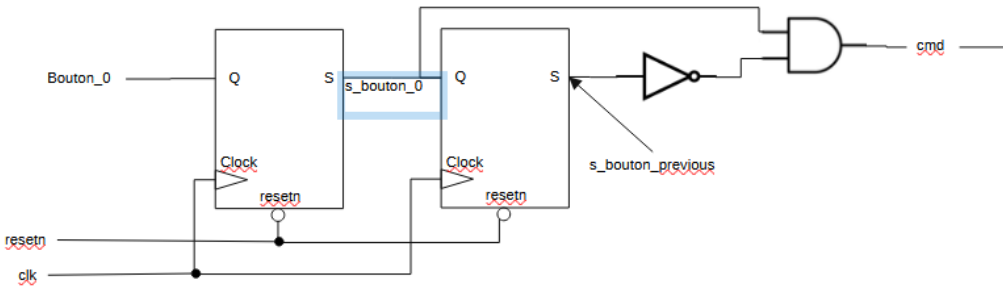



*Figure 3: schéma RTL pour détection de front montant*

*s_bouton_0* permet d'avoir le *bouton_0* synchronisé avec le signal d'horloge.

Pour résumer :

- signaux d'entrée :
  - *clk*
  - *resetn*
  - *bouton_0*
- signaux interne
  - *cmd* (il sera utilisé pour commander l'allumage des leds)
  - *s_bouton_previous*

6)

Registres dans le code VHDL, au sein d'un process et sur détection de front montant d'horloge :

```
s_bouton_0 <= bouton_0;
```

```
s_bouton_previous <= s_bouton_0;
```

logique combinatoire (en dehors du process)

```
cmd <= (not(s_bouton_previous) and s_bouton_0);
```

7)

Sur l'image suivante, nous voyons les signaux observés par simulation. Nous observons bien sur un cycle d'horloge le signal *cmd* qui passe à 1, et donc la LED verte associé également.

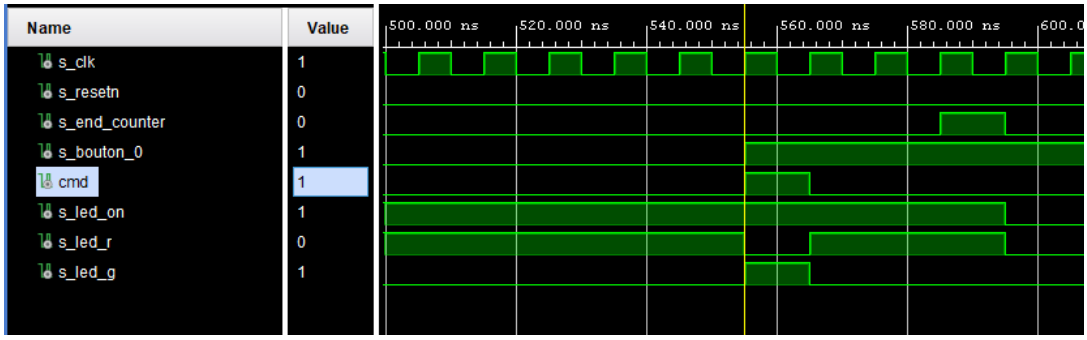



*Figure 4: allumage de led verte sur front montant*

8)

Pour pouvoir piloter les LED, nous aurons un multiplexeurs sur 4 entrées, en 2 bit, qui permet de sélectionner quelle LED doit clignoter, selon le signal *color_code_mem*, codé sur 2 bit.

*color_code_mem* est un signal interne qui met à jour *color_code_mem* selon le signal *update*. Si *update* vaut 1, alors on met à jour *color_code_mem* avec *color_code*, sinon nous gardons la même valeur de *color_code.*

Nous obtenons alors le module de pilotage des led RGB réalisé sur le schéma RTL suivant.

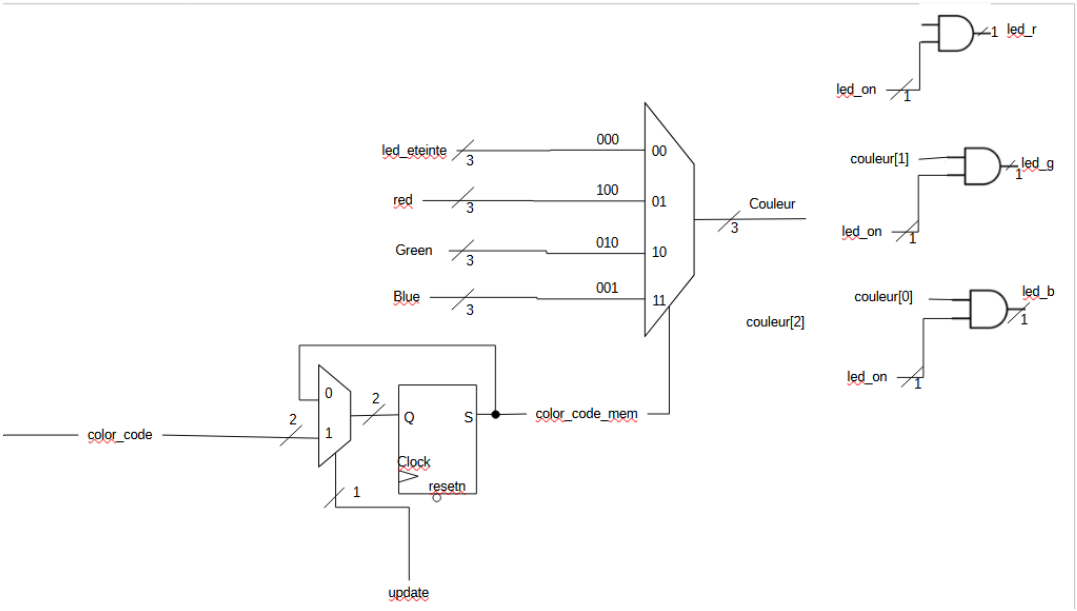



*Figure 5: RTL pour LED_driver*

9)

Pour le *bouton_0*, nous reprenons le schéma précédent pour la détection de front montant. Un multiplexeur sera utilisé pour le bouton_1.

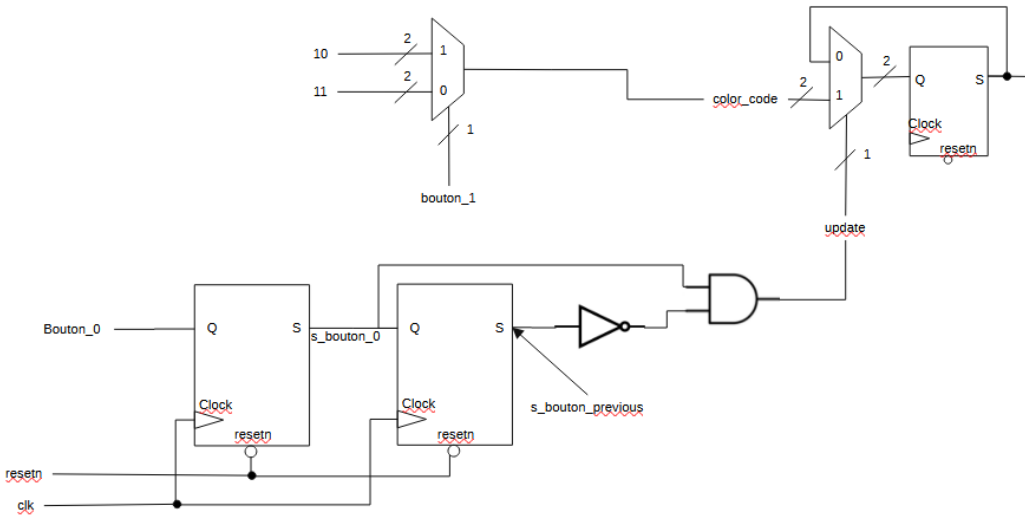



*Figure 6: schéma RTL pour pilotage entrée/sorties du module LED_Driver*

10)

Soit le code ci-dessous qui nous permettra de commander l'allumage des LED en fonction du code couleur indiqué

```
-------------------------------------------------------------------------
-- LED driver
  -------------------------------------------------------------------------
with color_code_mem select
   couleur <= "100" when "01",
              "010" when "10",
              "001" when "11",
              "000" when others;


led_r <= (couleur(2) and s_led_on);
led_g <= (couleur(1) and s_led_on);
led_b <= (couleur(0) and s_led_on);
```

le code ci-après nous permet de contrôler le code couleur en fonction du signal update (soit une mémorisation), sur front montant d'horloge pour être dans un registre.

```
if(update = '1') then
               color_code_mem <= color_code;
            else
               color_code_mem <= color_code_mem;
           end if;
```

2) 3) 4) 5)

Pour commander le code couleur en fonction de l'appuie sur le bouton_1 :

```
with bouton_1 select
        color_code <= "10" when '1',
                      "11" when '0';
```

11)

Après simulation via testbench, nous avons le résultat suivant pour le led_driver.

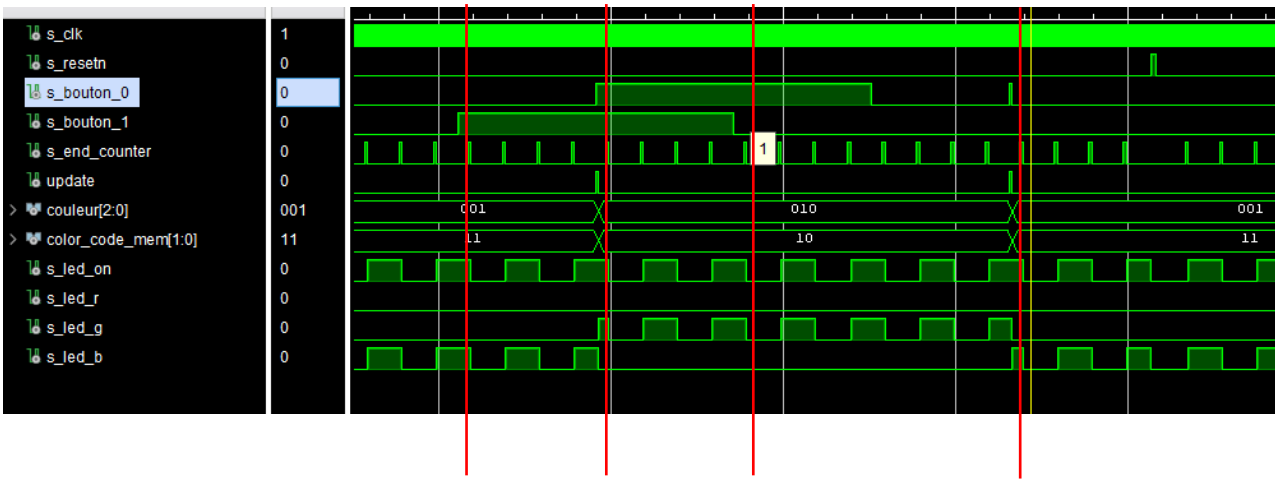



*Figure 7: test bench du LED_driver*

Nous pouvons  voir sur le graphique ci-dessus :

1) Initialement led bleue clignote (*s_led_bleu*)
2) si nous appuyons sur le *bouton_1* mais pas d'appui sur le *bouton_0*, la LED bleue clignote toujours
3) LED verte (*s_led_g*) clignote quand *s_bouton_0* passe à l'état haut.
4) Si *bouton_1* passe à 0 et *bouton_0* reste à 1, le clignotement reste à son état actuel : vert.
5) Si le bouton_0 passe à 1, le clignotement est bien mis à jour (*s_led_b* clignote)

13)

Soit le rapport suivant :

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

Nous retrouvons bien un registre sur 2 bit pour la mémorisation du code couleur.

3 registres sur 1 bit pour :

- 2 pour la détection sur front montant (s_bouton_0 et s_bouton_previous)
- 1 pour la machine d'état.

Les multiplexeurs :

1 à 4 entrée sur 3 bits : pour allumer les leds

1 à 2 entrée sur 2 bits : gestion du message code_color

1 à 2 entrée sur 1 bits : la machine à état.

```
Report Cell Usage:
+------+-------+------+
|      |Cell   |Count |
+------+-------+------+
|1     |BUFG   |     1|
|2     |CARRY4 |     7|
|3     |LUT2   |    32|
|4     |LUT4   |     5|
|5     |LUT6   |     3|
|6     |FDRE   |    32|
|7     |IBUF   |     3|
|8     |OBUF   |     3|
```

Nous nous intéressons sur les 3 dernière lignes :

- 32 registres à reset synchrone (FDRE) :
  - 28 pour le compteurs
  - 2 pour le *bouton_1* pour la détection de front montant
  - 1 pour la mémorisation de *code_color*
  - 1 pour la machine à état.
- 3 registres d'entrée (IBUF)
  - clk
  - bouton_0
  - $\text{bouton}_1$
  - (resetn a été mis sur un signal interne, car nous manquons de boutons sur la carte)
- 3 registres de sortie (OBUF)
  - led_b
  - led_g
  - led_r

Nous retrouvons dans le rapport de timing :

```
------------------------------------------------------------------------------------------------
| Design Timing Summary
| ---------------------
------------------------------------------------------------------------------------------------

    WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints      WHS(ns)      THS(ns)  THS F
    -------      -------  ---------------------  -------------------      -------      -------  -----
      3.636        0.000                      0                 4384        0.026        0.000
```

negative et hold slask qui ne sont pas négatif et à 0, Temps de chemin critique : 27.031ns et une période pour l'horloge de 10 ns .

15)

Après avoir générer le bitstream, nous programmons la carte. Avec ILA, nous pouvons alors observer le bon fonctionnement de la carte.

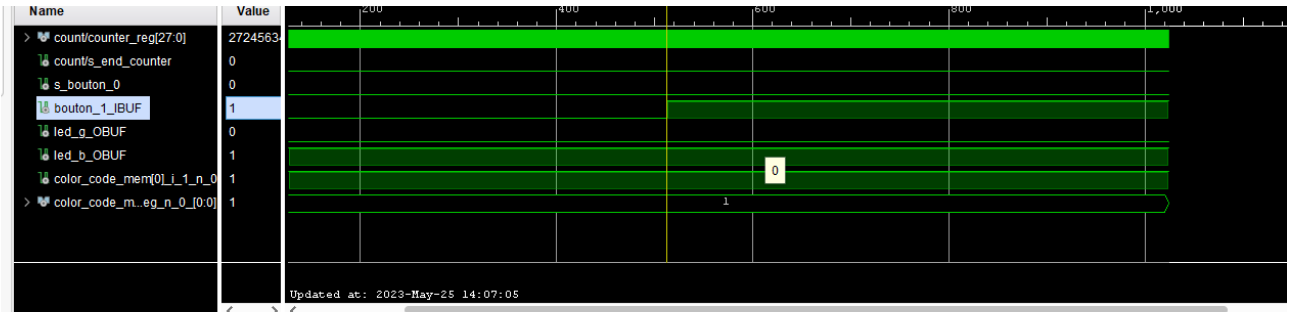
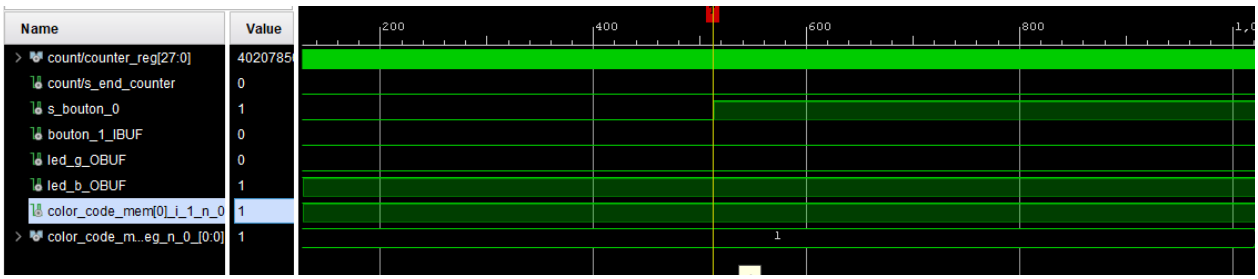
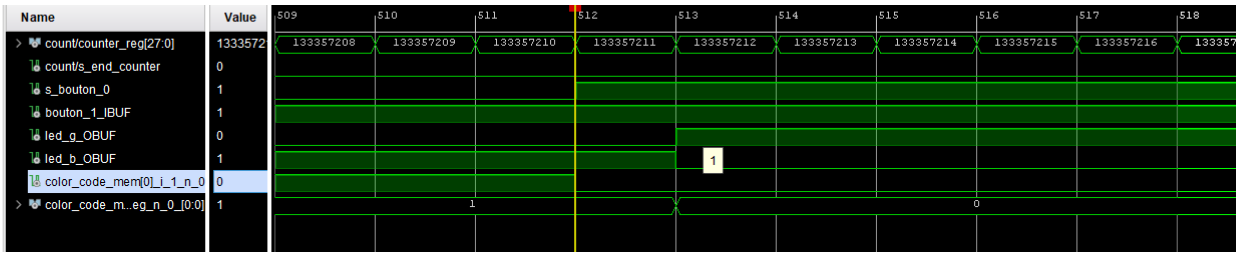



*Figure 8: appuie du bouton_1*

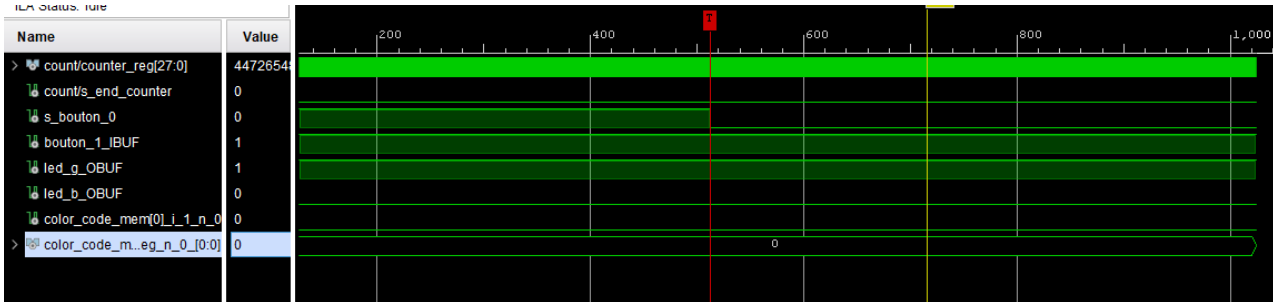
Sur la Figure 8, nous pouvons constater que lors de l'appuie sur le bouton_1, la led qui clignote en Bleue led_b reste bien allumée, et nous ne clignotons pas en vert.



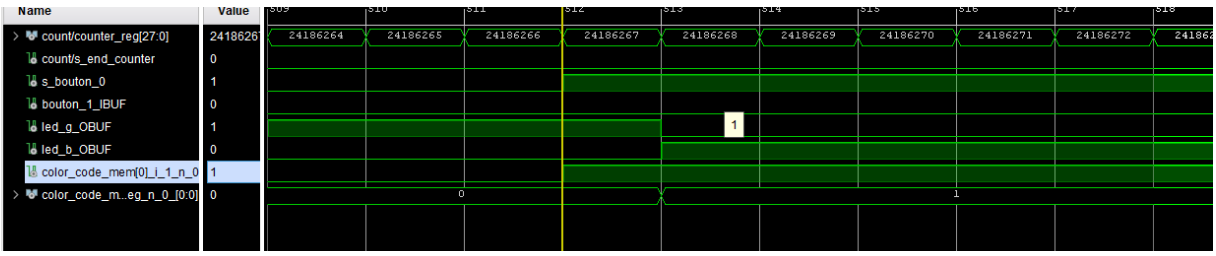

Lors de l'appuie du bouton_0, sans appuie sur bouton_1 en même temps, la led reste à bleu, conformément au cahier des charges.




Appuie du bouton_0 avec le bouton_1 appuyé : led_g passe à 1 et led_b passe à 0. (Remarquons au passage que le compteur n'est pas réinitialisé).




Puis après si nous relâchons le bouton_0 et maintenant bouton_1, nous voyons que le signal reste à vert (led_g reste allumé).




Appuie du bouton_0 avec bouton_1 à zéro : led_g s'éteint, et passe le relai à led_b.