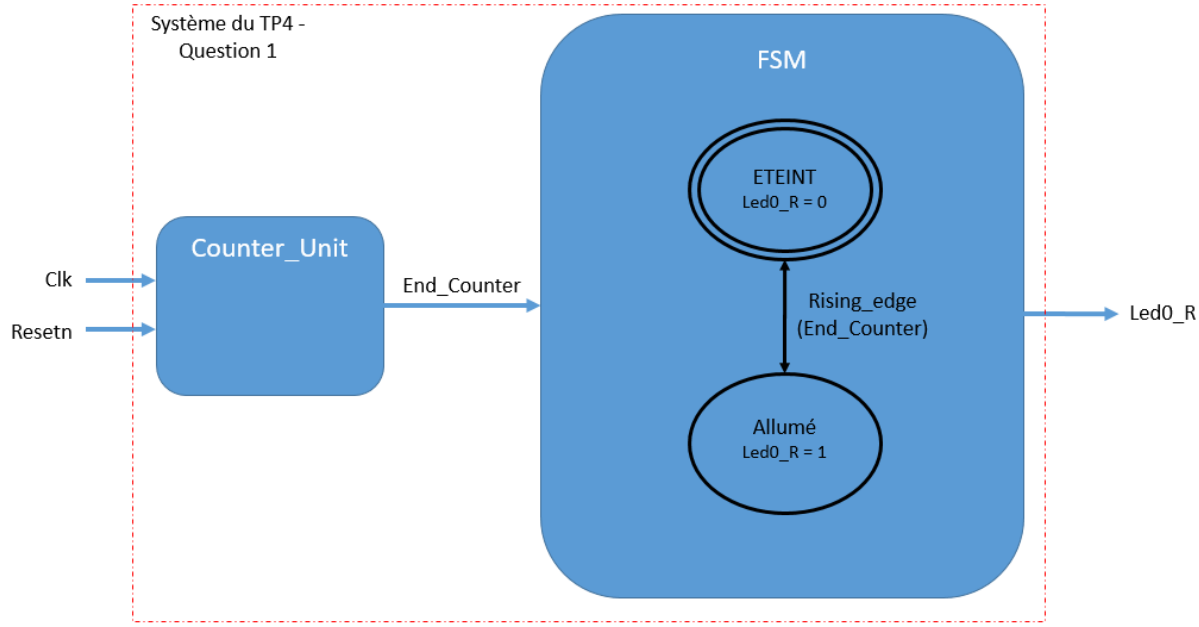
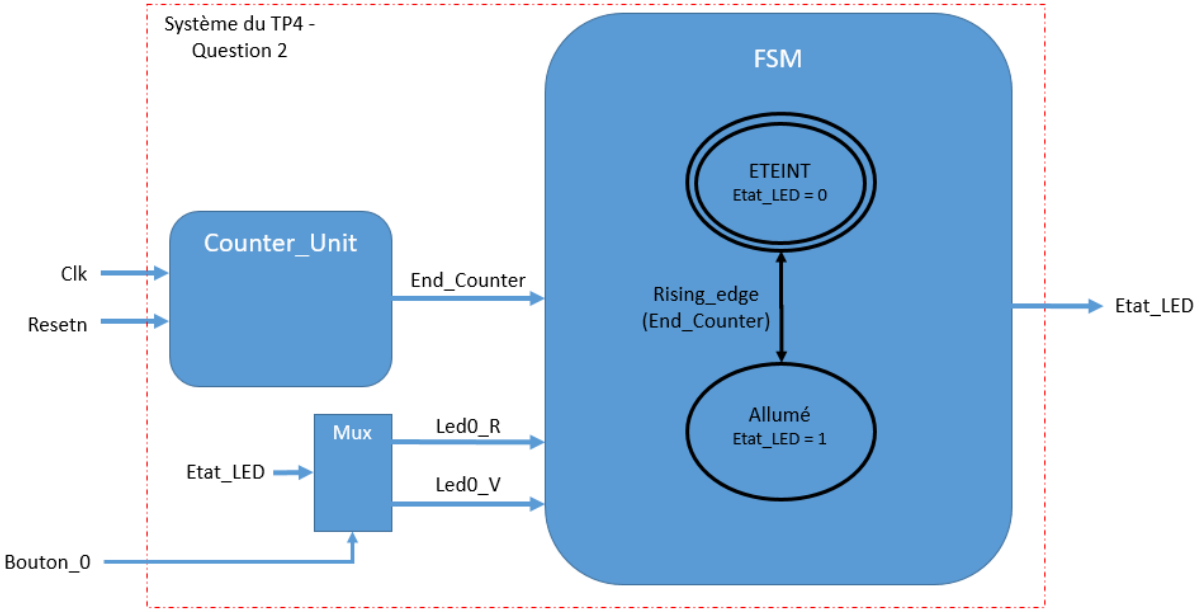
TP04 - Pilotage de LED et mémoire (Kamal KHERCHOUCH)

1. **Créez une architecture RTL permettant de faire clignoter une LED (par exemple la led0_r) en utilisant le module *Counter_unit* du TP2 et une machine à état.**



2. **Modifiez votre architecture pour piloter une LED rouge et une LED verte. Lorsque le bouton_0 est appuyé, la LED verte est allumée, sinon la LED rouge est allumée.**



3. **Rédigez le code VHDL de votre architecture.**

Voir code.

4. **Réalisez une simulation en rédigeant un testbench. Que se passe-t-il si le bouton est pressé pendant plus d'un cycle d'horloge ?**

- Lorsque le resten est activé, le système ne fonctionne pas.
- Lorsque le resetn est désactivé, le compteur fait son travail.
- Lorsque le bouton_0 est à 0 logique, seul le signal LED_0[0] clignote en fonction de End_Counter
- De même lorsque le bouton_0 est un 1 logique avec LED_0[1].

Si le bouton est pressé pendant plusieurs cycles d'horloge alors, le LED verte devrait se mettre à clignoter.

5. **Que faudrait-il faire pour que la LED ne clignote en vert qu'une seule fois même si le bouton est maintenu ?**

Il faut ajouter une condition que pour détecter le front montant sur le bouton 0. Cela permettra d'activer la LED verte une seule fois lors de la détection de l'appui, afin de revenir au clignotement de la LED rouge après coup.

La partie combinatoire a donc était remplacé par :

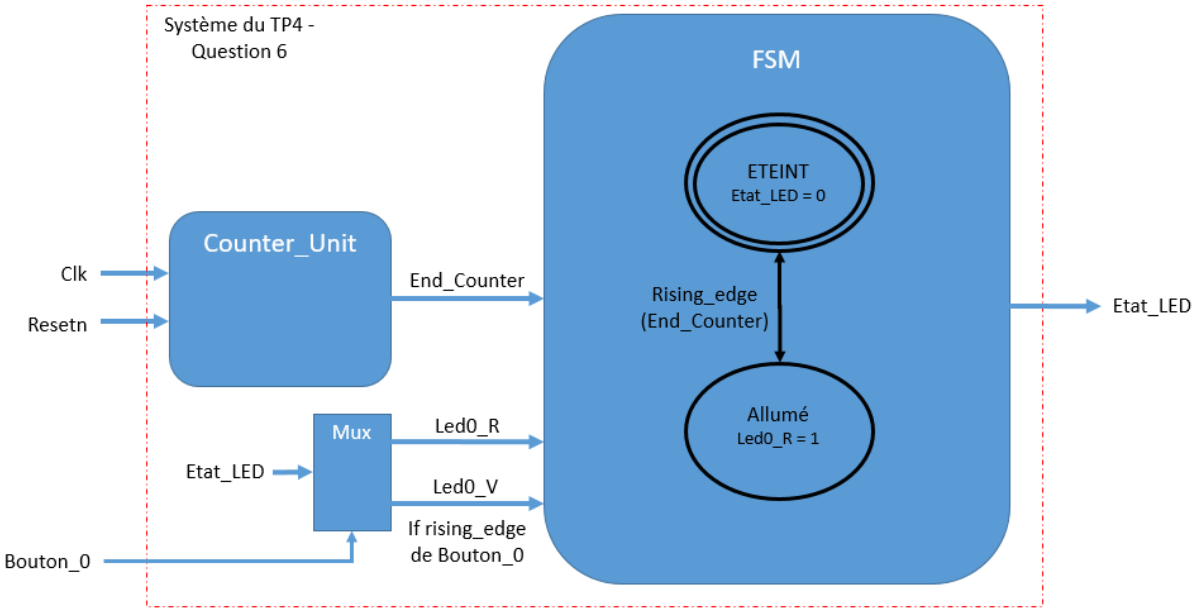
```
-- Gestion de la couleur en fonction de l'état du boutton.
S_LED_0 <= "01" when (LED_Commande = '1' and Bouton_0 <= '0')
  else "10" when (LED_Commande = '1' and rising_edge(Bouton_0))
      else "01" when (LED_Commande = '1' and Bouton_0 <= '1')
          else "00";

LED_0 <= S_LED_0;
```

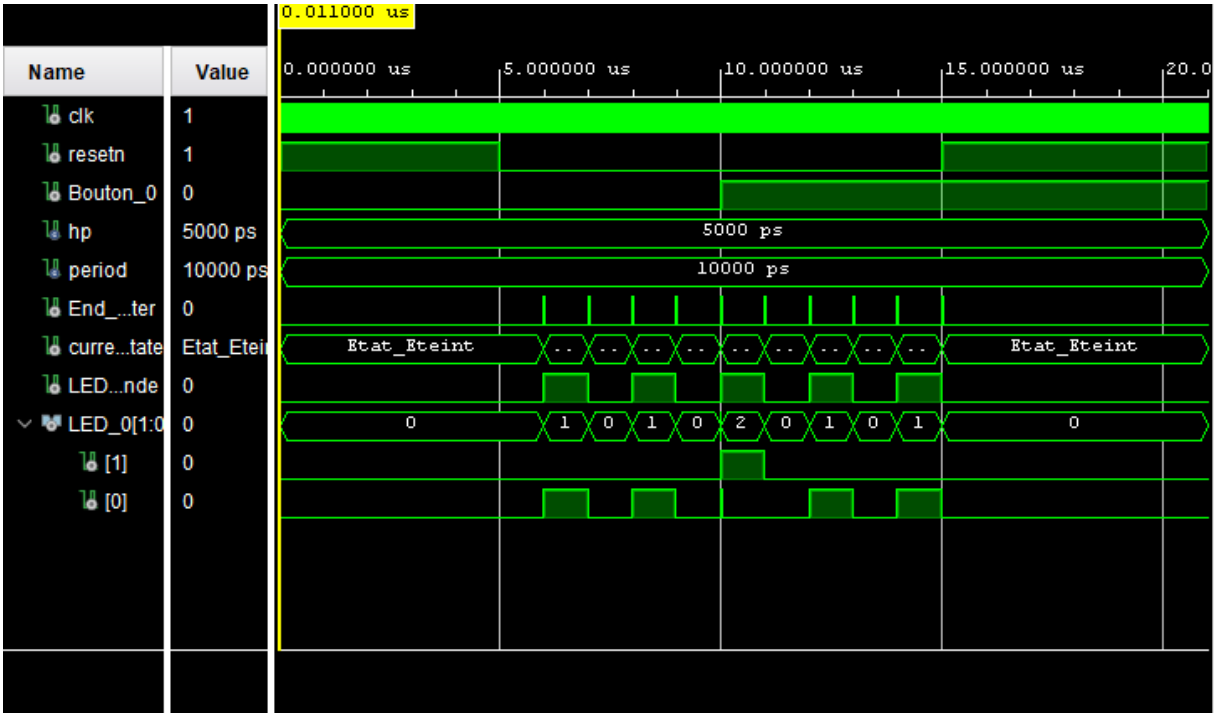
Au lieu de :

```
-- Gestion de la couleur en fonction de l'état du boutton.
S_LED_0 <= "01" when (LED_Commande = '1' and Bouton_0 <= '0')
  else "10" when (LED_Commande = '1' and Bouton_0 <= '1')
      else "00";
```

6. **Ajoutez cette solution à votre architecture RTL.**

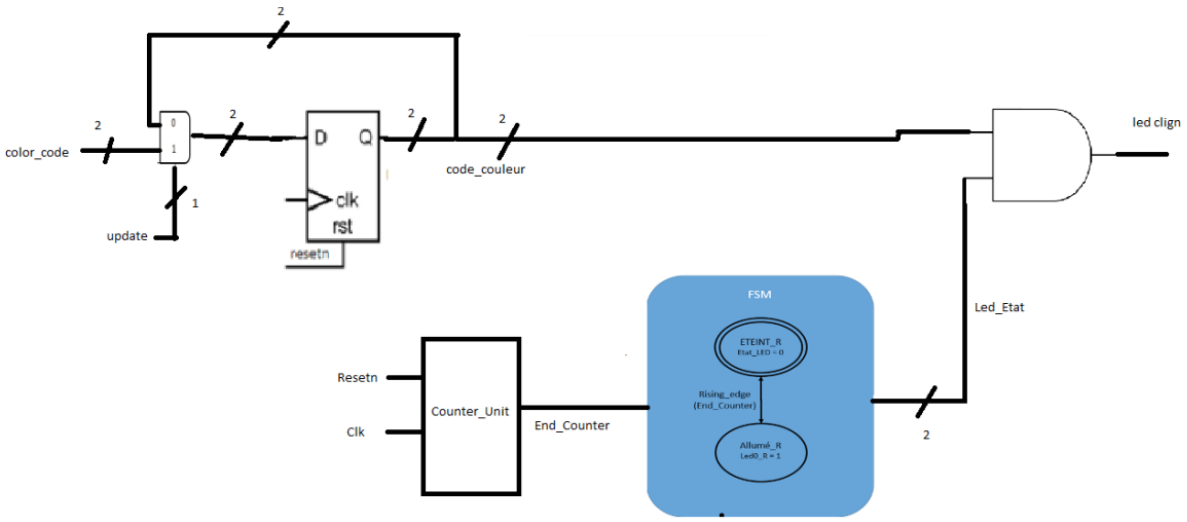


7. **Mettez à jour votre code VHDL et vérifiez votre résultat à la simulation.**

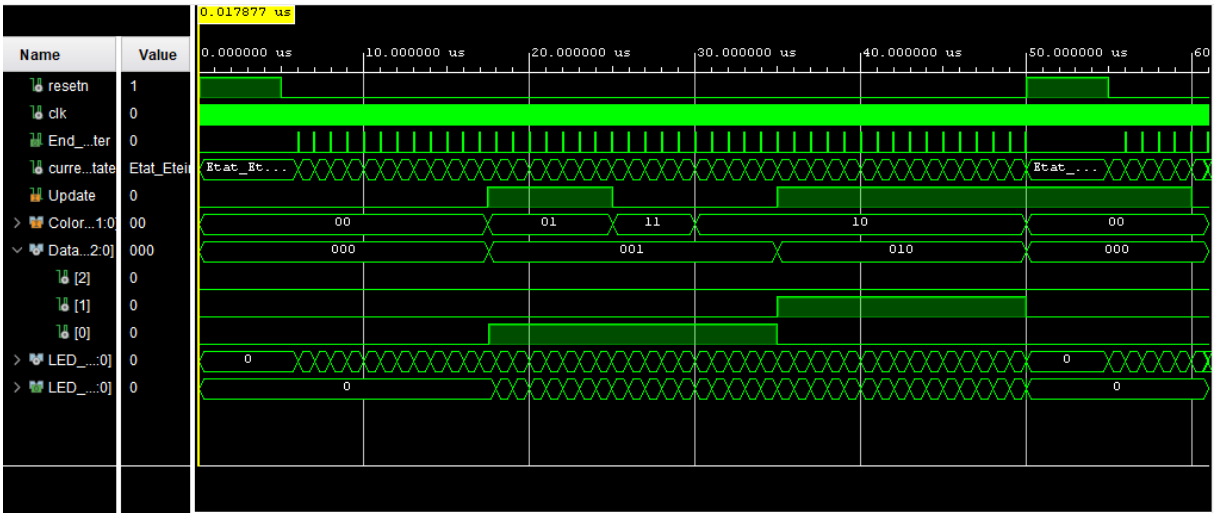


On voit bien que lors de la détection du front sur bouton 0, la LED verte clignote une fois, et la LED rouge reprend le relais.

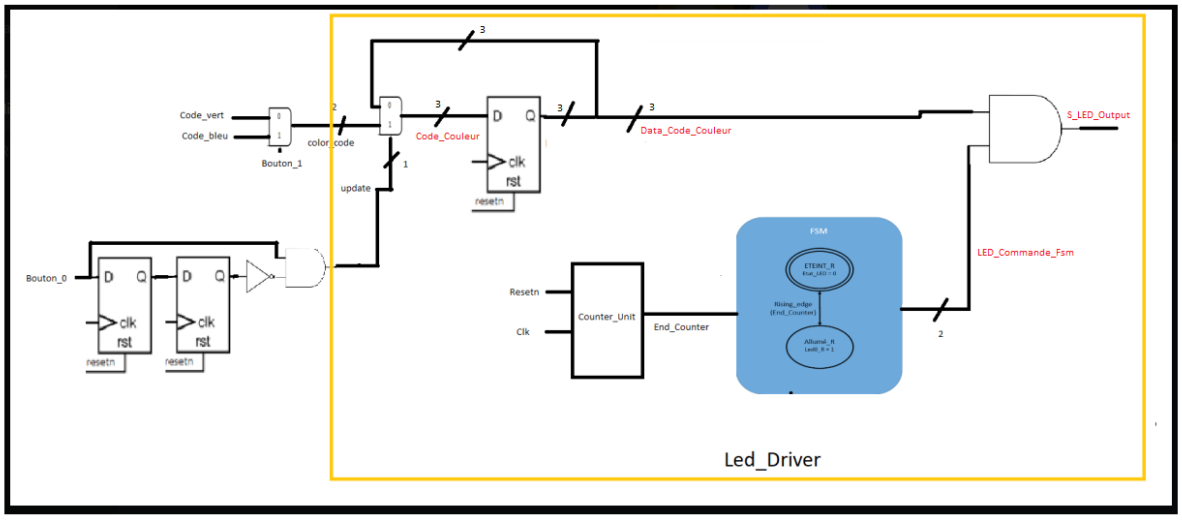
8. **Créez un module de pilotage d'une LED RGB en RTL. Ce dernier doit permettre de faire clignoter une LED RGB connectée en sortie d'une couleur définie par un code couleur donné en entrée. Le changement de couleur de la LED RGB n'a lieu que si un signal *update* est reçu.**



Le module LED Driver est créé (LedDriver.vhd) et testé via une simulation



9. Ajouter la logique nécessaire pour piloter les entrées/sorties de votre module.



Entrée :

- Clk
- Resetn
- Bouton_Color_Code (Bouton 1)
- Buton_Update (Bouton 0)

Sortie :

- LED_Output

10. Ecrivez le code VHDL correspondant à votre architecture.

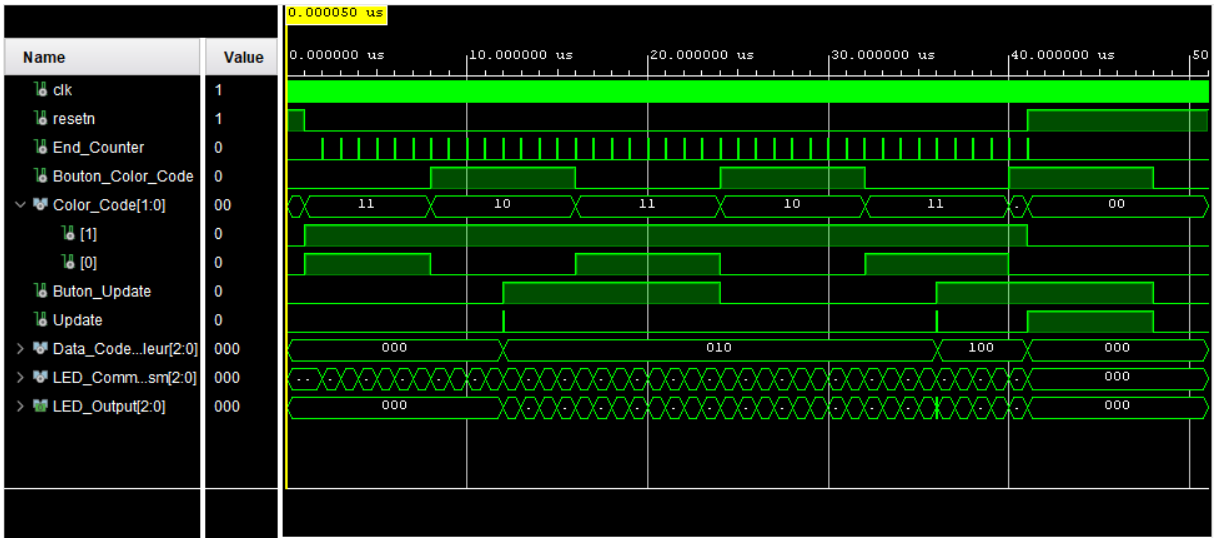
Un nouveau projet a été créé sous le nom PilotageLED_Avec_Module_LedDriver.vhd
Ce dernier utilise le module LedDriver qu'on vient de créer précédemment qui lui-même utilise le module Counter_Unit validé lors du TP2.

11. Ecrivez un testbench pour tester votre design.

Le testbench associé est à consulté via le fichier tb_PilotageLED_Avec_Module_LedDriver.vhd
Un prend en compte différents processus :

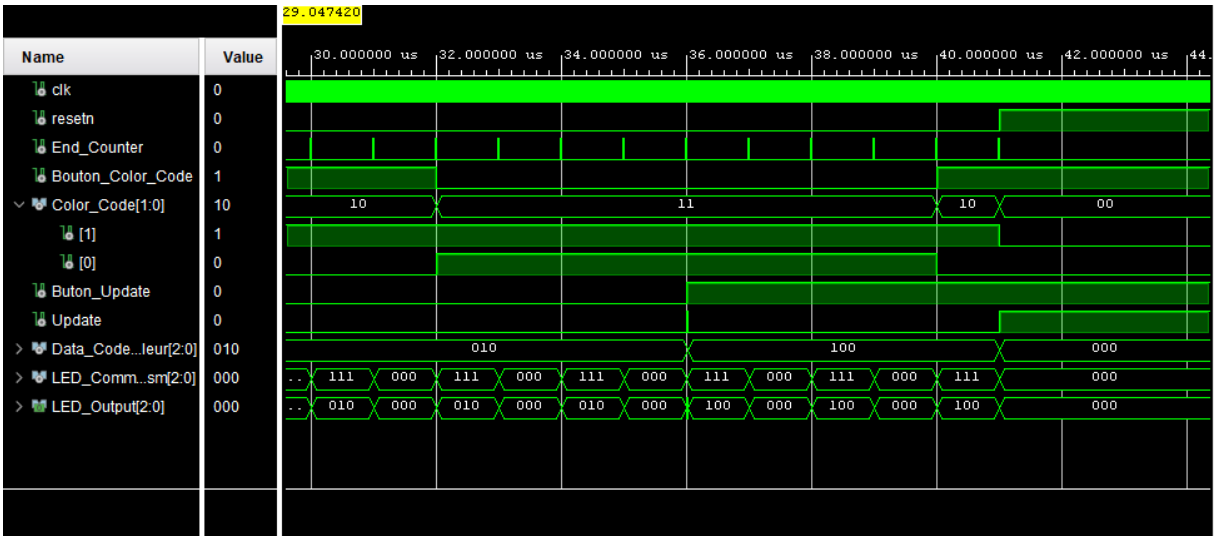
- Gestion de l'horloge : clk
- Gestion du reset : resetn
- Gestion du bouton 1 : Bouton_Color_Code
- Gestion du bouton 0 : Buton_Update

12. Vérifier votre résultat à la simulation.



Le système est validé :

- L'activation du reset désactive le comptage du signal End_Counter, la gestion du signal Color_Code et force notre sortie à 0 (LEDs OFF)
- Ici on voit bien que lorsque le Bouton_Color_Code est à 0 alors on selectionne Color_code = « 11 » ce qui correspond à la couleur bleu. Sinon on sélectionne la couleur vert avec Color_code = « 10 »
- On voit également que lorsqu'on appui sur le Buton_Update le front montant est détecté par le signal Update et que le signal Data_Code_Couleur prend la couleur sélectionné par Code_Couleur. Lorsque le Buton_Update est relâché, le signal Data_Code_Couleur conserve son ancienne valeur.
- Le clignotement des LEDs se fait correctement voir image ci-dessous.



13. Réalisez une synthèse et étudiez le rapport de synthèse, les ressources utilisées doivent correspondre à votre schéma RTL.

Une mise à jour du fichier de contrainte

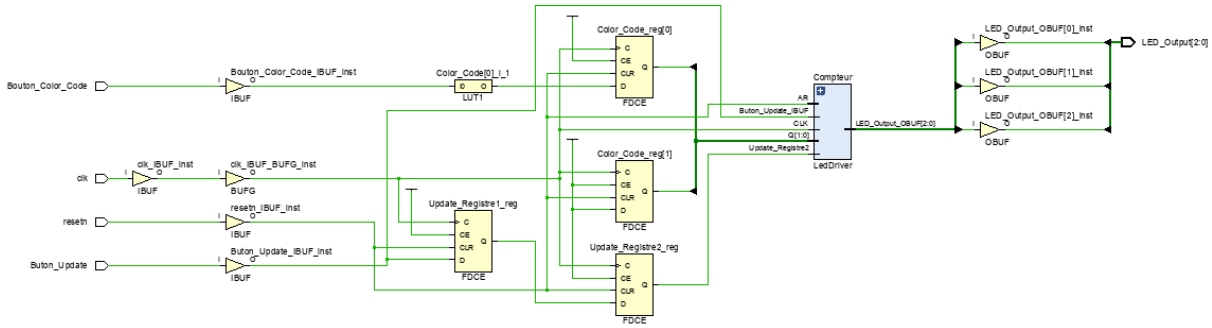
```
# PL System Clock
set_property -dict {PACKAGE_PIN H16 IOSTANDARD LVCMOS33} [get_ports clk]
create_clock -period 8.000 -name sys_clk_pin -waveform {0.000 4.000} -add [get_ports clk]
#set_property -dict { PACKAGE_PIN H16   IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L13P_T2_MRCC_35 Sch=sysclk
#create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4} [get_ports { clk }];#set

# RGB LEDs
set_property -dict { PACKAGE_PIN L15   IOSTANDARD LVCMOS33 } [get_ports { LED_OUT[2] }]; #IO_L22N_T3_AD7N_35 Sch=led0_b
set_property -dict { PACKAGE_PIN G17   IOSTANDARD LVCMOS33 } [get_ports { LED_OUT[1] }]; #IO_L16P_T2_35 Sch=led0_g
set_property -dict { PACKAGE_PIN N15   IOSTANDARD LVCMOS33 } [get_ports { LED_OUT[0] }]; #IO_L21P_T3_DQS_AD14P_35 Sch=led0_r
#set_property -dict { PACKAGE_PIN G14   IOSTANDARD LVCMOS33 } [get_ports { led1_b }]; #IO_0_35 Sch=led1_b
#set_property -dict { PACKAGE_PIN L14   IOSTANDARD LVCMOS33 } [get_ports { Output_On_Off }]; #IO_L22P_T3_AD7P_35 Sch=led1_g
#set_property -dict { PACKAGE_PIN M15   IOSTANDARD LVCMOS33 } [get_ports { led1_r }]; #IO_L23N_T3_35 Sch=led1_r

# Buttons
set_property -dict { PACKAGE_PIN D20   IOSTANDARD LVCMOS33 } [get_ports { Buton_Update }]; #IO_L4N_T0_35 Sch=btn[0]
set_property -dict { PACKAGE_PIN D19   IOSTANDARD LVCMOS33 } [get_ports { Bouton_Color_Code }]; #IO_L4P_T0_35 Sch=btn[1]
```

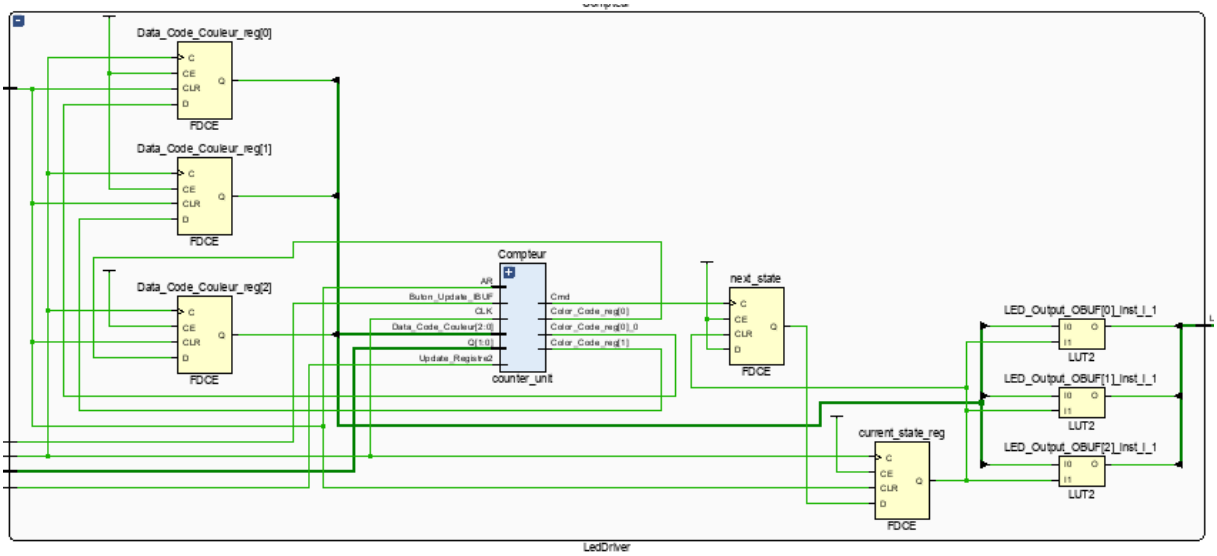
```
set_property -dict { PACKAGE_PIN C20   IOSTANDARD LVCMOS33 } [get_ports { resetn }]; #IO_L1P_T0_AD0P_35 Sch=ad_p[0]
```

On obtient le schéma RTL suivant :



Sur le Schéma RTL suivant on voit bien que l'on a :

- 4 registres (2 pour la gestion de l'Update et 2 pour la gestion du color_code)
- Le module LedDriver
- Nos entrées et sorties



Si on détaille notre module LedDriver on y retrouve bien Notre module Counter_Unit, nos registres pour le code Couleur sur 3 bits et notre machine d'état à 2 états.

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

Le rapport de synthèse nous confirme qu'on utilise bien 3 registres et 3 multiplexeurs, ce qui est conforme au schéma RTM proposé.

```
Report Cell Usage:
+------+-------+------+
|      |Cell   |Count |
+------+-------+------+
|1     |BUFG   |     1|
|2     |CARRY4 |     7|
|3     |LUT1   |     1|
|4     |LUT2   |     3|
|5     |LUT4   |     1|
|6     |LUT5   |     4|
|7     |LUT6   |    35|
|8     |FDCE   |    36|
|9     |IBUF   |     4|
|10    |OBUF   |     3|
+------+-------+------+
```

Ici, on voit bien qu'on a :

- 4 Entrées : IBUF
- 3 Sorties : OBUF
- 36 registres : FDCE (26 registres pour le module Counter_Unit, 6 pour le module LedDriver (avec la prise en compte de la Fsm) et 4 pour le système)

14. Effectuez le placement routage et étudiez les rapports.

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	3,190 ns	Worst Hold Slack (WHS):	0,201 ns	Worst Pulse Width Slack (WPWS):	3,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	31	Total Number of Endpoints:	31	Total Number of Endpoints:	36

All user specified timing constraints are met.

Notre système ne devrait pas rencontrer de métastabilité car on ne retrouve pas de stack (TNS et THS = 0ns)

```
Max Delay Paths
--------------------------------------------------------------------------------------
Slack (MET) :             3.190ns  (required time - arrival time)
  Source:                 Compteur/Compteur/D_out_reg[17]/C
                            (rising edge-triggered cell FDCE clocked by sys_clk_pin  {rise@0.000ns fall@4.000ns per
  Destination:            Compteur/Compteur/D_out_reg[25]/D
                            (rising edge-triggered cell FDCE clocked by sys_clk_pin  {rise@0.000ns fall@4.000ns per
  Path Group:             sys_clk_pin
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            8.000ns  (sys_clk_pin rise@8.000ns - sys_clk_pin rise@0.000ns)
  Data Path Delay:        4.810ns  (logic 2.282ns (47.438%)  route 2.528ns (52.562%))
  Logic Levels:           10  (CARRY4=7 LUT6=3)
```

On observe que chemin le plus long est celui du module compteur, ce qui ne me semble pas aberrent vu qu'il utilise beaucoup de registre.

15. Générez le bitstream et vérifiez que vous avez le comportement attendu sur carte.

Voir la Video