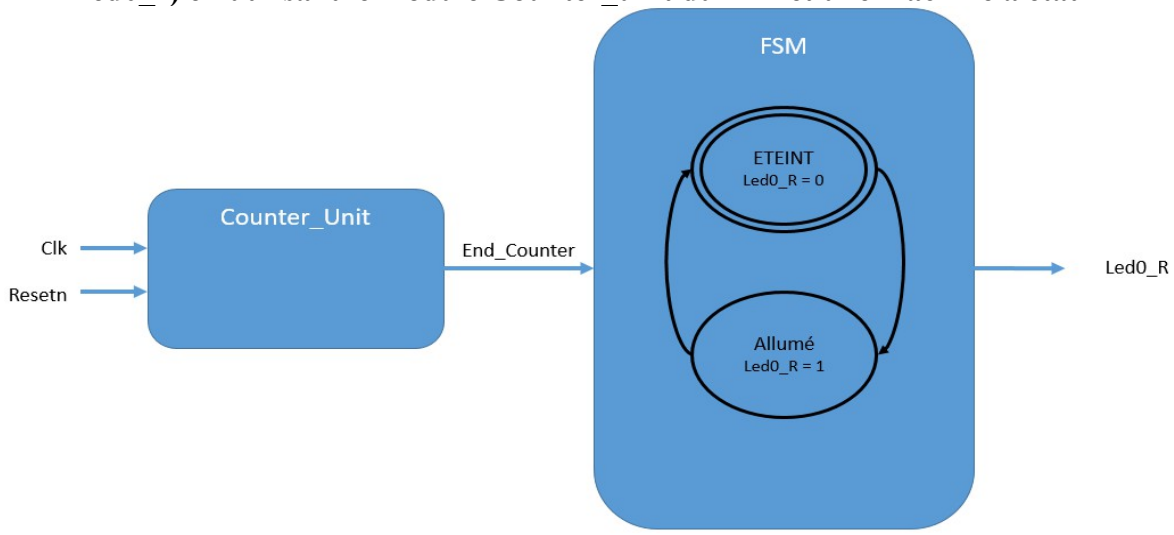
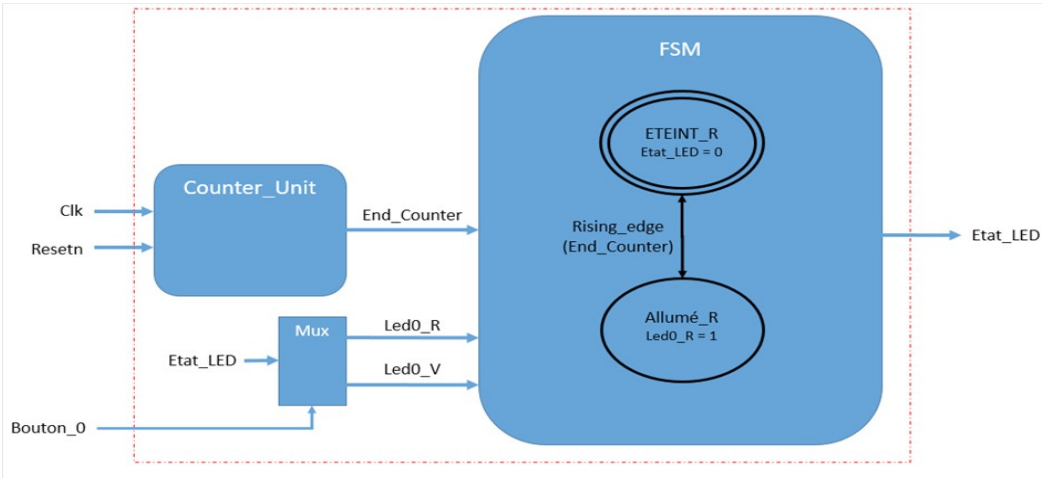
1. **Créez une architecture RTL permettant de faire clignoter une LED (par exemple la led0_r) en utilisant le module Counter_unit du TP2 et une machine à état**



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.**

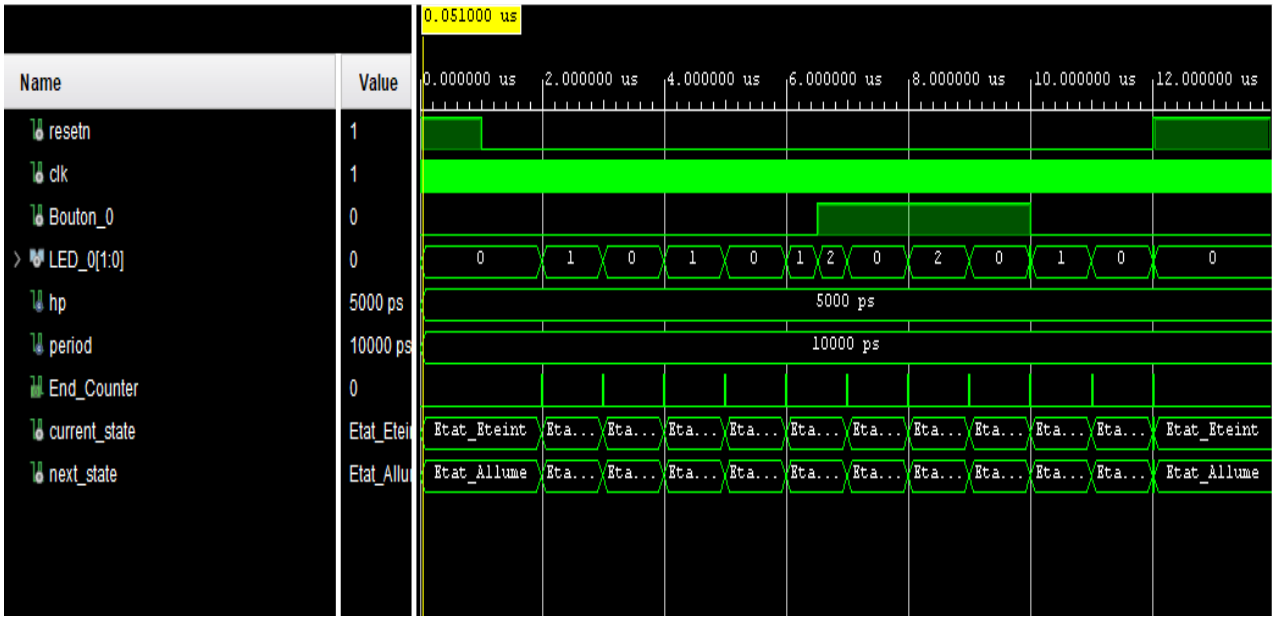


3. **Rédigez le code VHDL de votre architecture.**

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;


entity Tp4_FSM is
    port (
        clk         : in std_logic;
        resetn      : in std_logic;
        Bouton_0    : in std_logic;
        LED_0 : out std_logic_vector (1 downto 0)
    );
end Tp4_FSM;

architecture behavioral of Tp4_FSM is

    type state is (Etat_Eteint, Etat_Allume);

    signal current_state : state;  --etat dans lequel on se trouve actuellement
    signal next_state : state;     --etat dans lequel on passera au prochain coup d'horloge

    signal End_Counter  : std_logic;
    signal LED_Commande : std_logic := '1';
    signal S_LED_0 : std_logic_vector (1 downto 0) := "00";

    -- Composant Counter_Unit
    component Counter_unit
        port (
            clk         : in std_logic;
            resetn      : in std_logic;
            End_Counter : out std_logic
        );
    end component;


    begin
    --Affectation des signaux du testbench avec ceux de l'entite a tester
    Compteur : Counter_unit
        port map (
            clk => clk,
            resetn => resetn,
            End_Counter => End_Counter
        );

        -- Gestion du reset
        process(clk,resetn, END_Counter)
        begin
            if(resetn = '1') then
                current_state <= Etat_Eteint;
            elsif(rising_edge(END_Counter)) then
                current_state <= next_state;
            end if;
        end process;

        -- processe de la machine d'états
        process(current_state, next_state) -- Liste des signaux de sensibilité du processe
        begin
          case current_state is
                -- à remplir
                when Etat_Eteint =>
                    LED_Commande <= '0';
                    next_state <= Etat_Allume;

                -- à remplir
                when Etat_Allume =>
                    LED_Commande <= '1';
                    next_state <= Etat_Eteint;

           end case;
        end process;

        -- 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";

        LED_0 <= S_LED_0;

end behavioral;
```

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 ?**

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity tb_TP4_Fsm is
end tb_TP4_Fsm;

architecture behavioral of tb_TP4_Fsm is

    signal resetn      : std_logic := '1';
    signal clk         : std_logic := '0';
    signal Bouton_0    : std_logic := '0';
    signal LED_0       : std_logic_vector (1 downto 0) := "00";


    -- Les constantes suivantes permette de definir la frequence de l'horloge
    constant hp : time := 5 ns;      --demi periode de 5ns
    constant period : time := 2*hp;  --periode de 10ns, soit une frequence de 100Hz

    component Tp4_Fsm
        port (
            clk         : in std_logic;
            resetn      : in std_logic;
            Bouton_0     : in std_logic;
            LED_0 : out std_logic_vector (1 downto 0)
        );
    end component;

    begin
    dut: Tp4_Fsm
        port map (
            clk => clk,
            resetn => resetn,
            Bouton_0 => Bouton_0,
            LED_0 => LED_0
        );

    process
    begin


    for i in 1 to 5000 loop
       clk <= not clk;
       wait for hp;
       if(i>0 and i<200) then
          resetn <= '1';
          Bouton_0 <= '0';
       elsif (i>=200 and i<1300) then
            resetn <= '0';
            Bouton_0 <= '0';
       elsif (i>=1300 and i<2000) then
            resetn <= '0';
            Bouton_0 <= '1';
        elsif (i>=2000 and i<2400) then
            resetn <= '0';
            Bouton_0 <= '0';
       else
            resetn <= '1';
       end if;
    end loop;

    end process;

end behavioral;
```



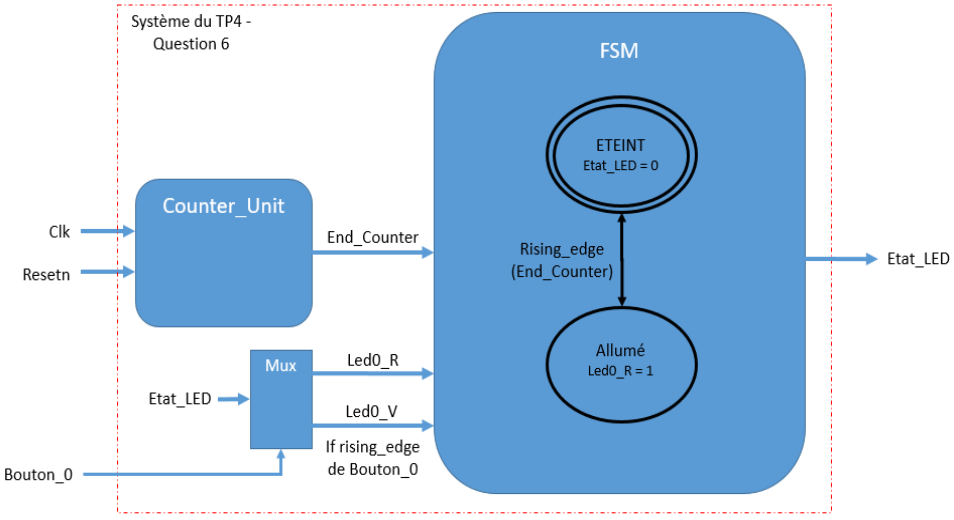
On peut voir que lorsque le reset est actif, On reste à l'état éteint. Ensuite tous les front montants de End_counter on passe bien de l'état allumé à l'état éteint. Lorsqu'on appuie sur le bouton la led, change de couleur on peut voir qu'on passe du rouge au vert. On peut voir tous les changements grâce à Led_0. Lorsque LED_0 est à 0, led éteinte. Lorsque LED_0 est à 1, led rouge. Lorsque LED_0 est à 2, led verte. Lorsque l'on reste appuyer sur le bouton pendant plusieurs cycles la led clignote plusieurs fois en vert.

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

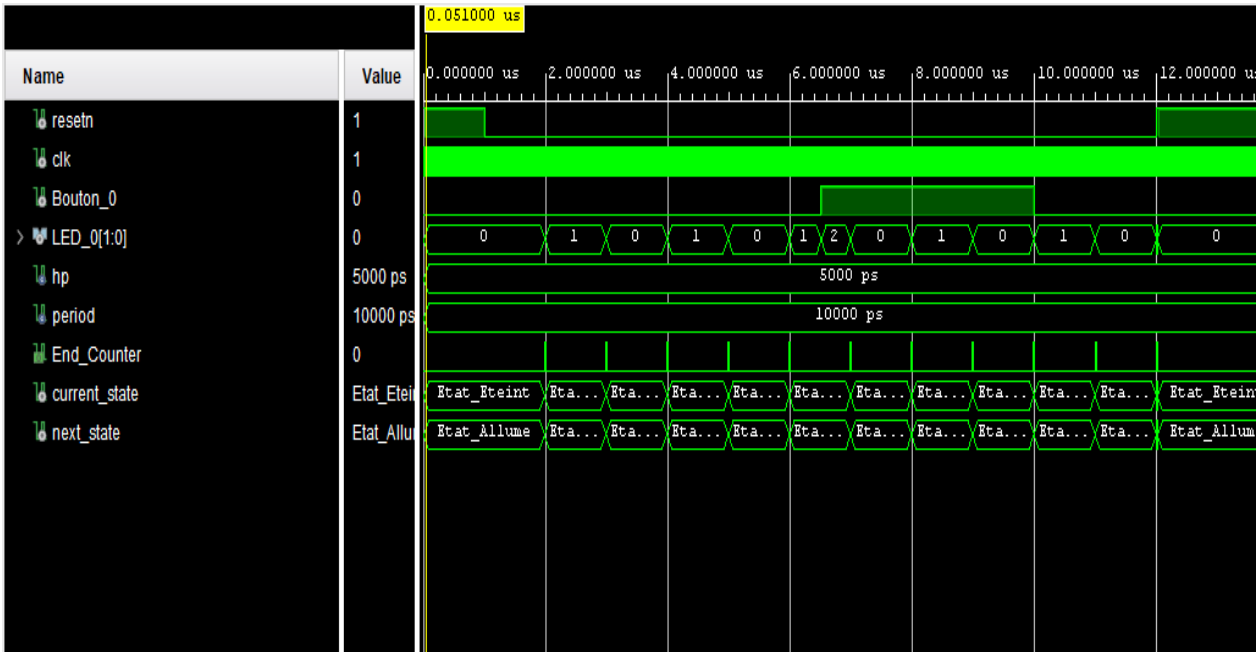
On change notre condition, au lieu de mettre lorsque bouton est à 1 on est vert. Il faut mettre lorsque l'on a front montant de bouton qu'on est vert sinon rouge. On a donc :

```
-- 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;
```

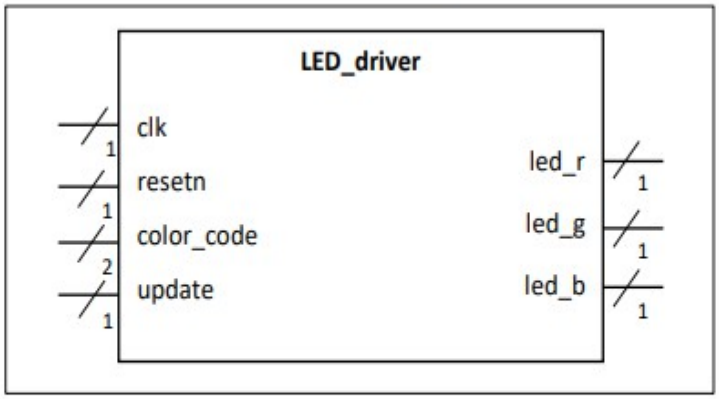
6. **Ajoutez cette solution à votre architecture RTL**



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

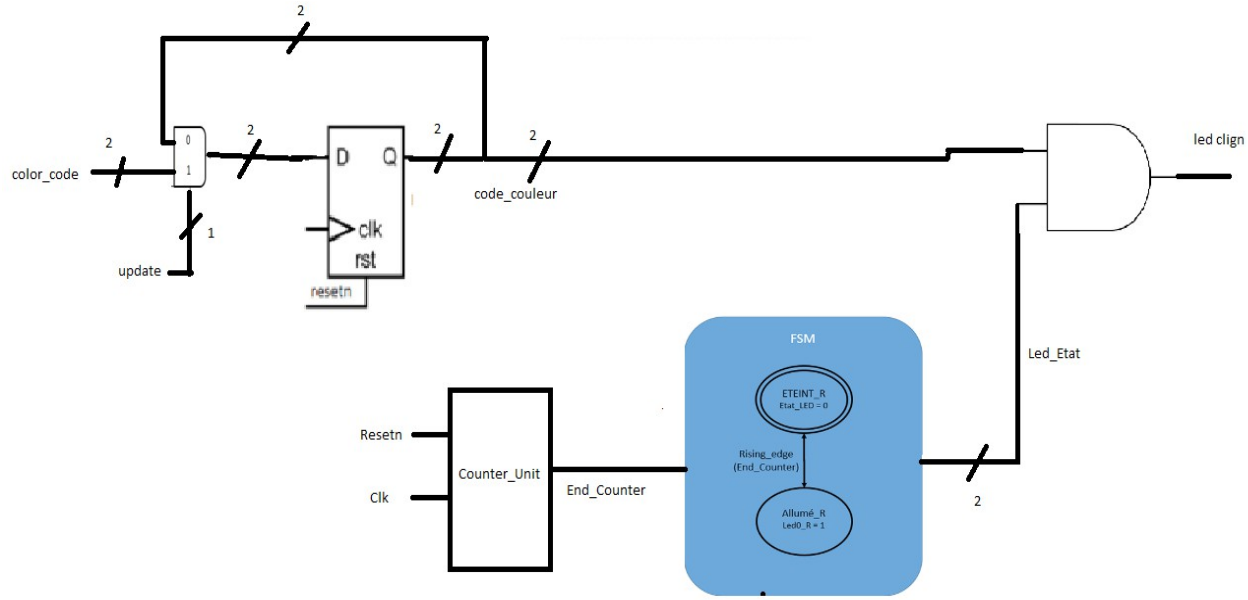


8. Créez un module de pilotage d'une LED RGB en RTL. Ce dernier doit permettre de faire clignoter une LED RGB connectée en sortie d'une couleur définie par un code couleur donné en entrée. Le changement de couleur de la LED RGB n'a lieu que si un signal *update* est reçu.



A titre d'exemple voici une table mettant en relation un code couleur avec une couleur de la LED RGB :

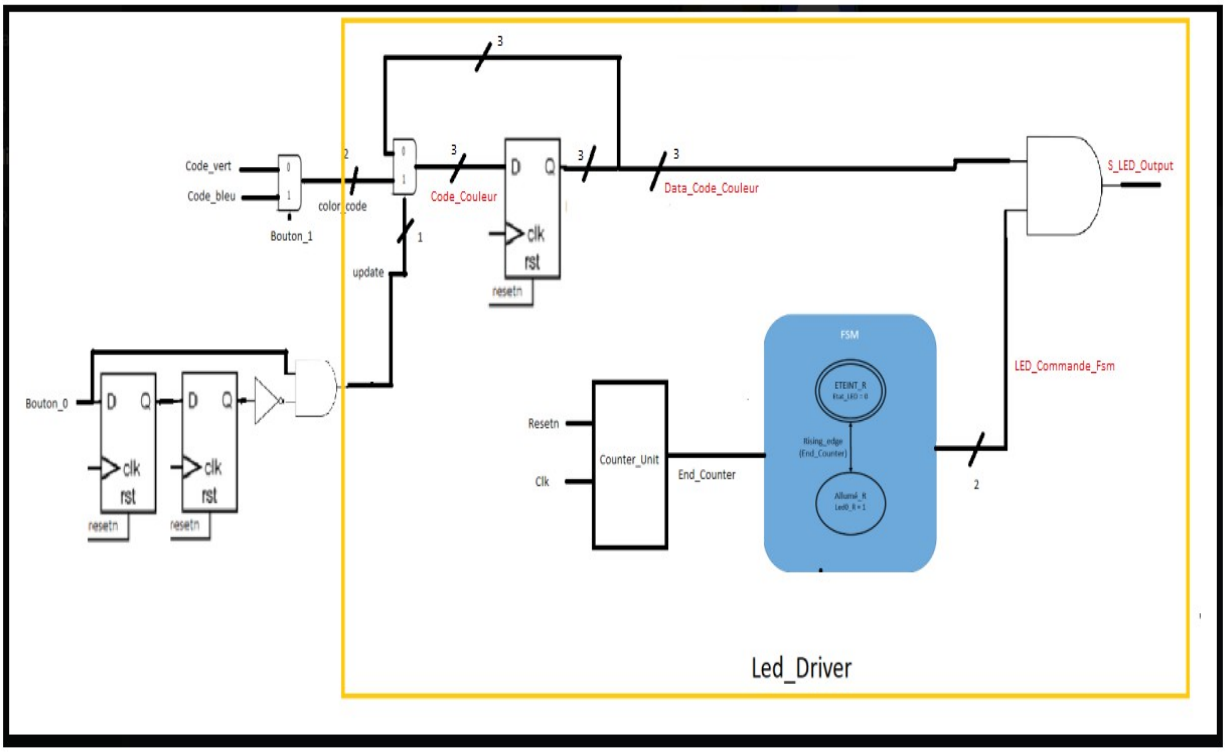
Code couleur	Couleur de la LED RGB
01	Rouge
10	Verte
11	Bleue
00	Eteinte




```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;


entity LedDriver is
    port (
        clk         : in std_logic;
        resetn      : in std_logic;
        Color_Code  : in std_logic_Vector (1 downto 0);
        Update      : in std_logic;
        LED_Output  : out std_logic_vector (2 downto 0)
    );
end LedDriver;

architecture behavioral of LedDriver is

    -- Lié à la machine d'état.
    type state is (Etat_Eteint, Etat_Allume);
    signal current_state : state;  -- Etat dans lequel on se trouve actuellement
    signal next_state : state;     -- Etat dans lequel on passera au prochain coup d'horloge
    signal LED_Commande_Fsm : std_logic_vector (2 downto 0); -- Signal indiquant le status de notre FSM (Leds ON ou Leds OFF)

    -- Lié au Compteur.
    signal End_Counter  : std_logic;

    -- Signaux interne
    signal S_LED_Output      : std_logic_vector (2 downto 0);
    signal Code_Couleur      : std_logic_vector (2 downto 0); -- Signal de sortie du multiplexeur vers la bascule
    signal Data_Code_Couleur : std_logic_vector (2 downto 0); -- Signal de sortie de la bascule

    -- Composant Counter_Unit
    component Counter_unit
        port (
            clk         : in std_logic;
            resetn      : in std_logic;
            End_Counter : out std_logic
        );
    end component;
```

```vhdl
    begin
    --Affectation des signaux du testbench avec ceux de l'entite a tester
    Compteur : Counter_unit
        port map (
            clk => clk,
            resetn => resetn,
            End_Counter => End_Counter
        );

        -- ************** Gestion du reset ************** --
        process(resetn, clk)
        begin
            if(resetn = '1') then
                -- Lors du reset on réinitialise la machine d'état.
                current_state <= Etat_Eteint;
            elsif(rising_edge(clk)) then
                -- On change d'état à chaque front montant de End_Counter.
                current_state <= next_state;
            end if;
        end process;
        -- ***************-- Fin du reset ***************** --

        -- ************** Début de processe de la Machine d'états ************** --
        process(current_state, next_state, END_Counter) -- Liste des signaux de sensibilité du processe
        begin
           case current_state is
                -- L'indicateur Led_Commande passe à 0.
                when Etat_Eteint =>
                    LED_Commande_Fsm <= "000";
                    if(rising_edge(END_Counter)) then
                        next_state <= Etat_Allume;
                    end if;

                -- L'indicateur Led_Commande passe à 1.
                when Etat_Allume =>
                    LED_Commande_Fsm <= "111";
                if(rising_edge(END_Counter)) then
                        next_state <= Etat_Eteint;
                    end if;

            end case;
        end process;
        -- ***************** FIN de processe de la Machine d'états ***************** --

        -- ************** Début de processe de la gestion du Code Couleur ************** --
        process(resetn, clk)
        begin
            if(resetn = '1') then
                -- Lors du reset on réinitialise les signaux lié au Code Couleur.
                Data_Code_Couleur    <= "000";
            elsif(rising_edge(clk)) then
                -- On change d'état à chaque front montant de End_Counter.
                if(update = '1') then
                    Data_Code_Couleur <= Code_Couleur;
                else
                    Data_Code_Couleur <= Data_Code_Couleur;
                end if;
            end if;
        end process;
        -- ************** Fin de processe de la gestion du Code Couleur ************** --

        -- ************** Partie combinatoire de la gestion du Code Couleur ************** --
        Code_Couleur <= "000" when (Color_Code = "00")
            else "001" when (Color_Code = "01")
              else "010" when (Color_Code = "10")
                else "100" when (Color_Code = "11")
                  else "000";
        -- ************** Fin combinatoire de la gestion du Code Couleur ************** --

        -- ************** Partie Combinatoire entre Fsm et Code Couleur ************** --
        LED_Output <= Data_Code_Couleur and LED_Commande_Fsm;
        -- Signal de sortie prends le signal interne.
        --LED_Output <= S_LED_Output;
        -- ************** Fin Combinatoire entre Fsm et Code Couleur ************** --

end behavioral;
```

9. **Ajouter la logique nécessaire pour piloter les entrées/sorties de votre module. - Le signal update doit recevoir 1 uniquement lorsque le bouton _0 vient d'être appuyé, maintenir le bouton enfoncé ne doit pas maintenir le signal update à 1 - Le signal color_code doit recevoir soit le code couleur « vert » si le bouton_1 est pressé, il doit recevoir le code couleur « bleu » sinon - Les LED R, G et B sont connectées avec les couleurs respectives de la LED_0**



10. **Ecrivez le code VHDL correspondant à votre architecture.**

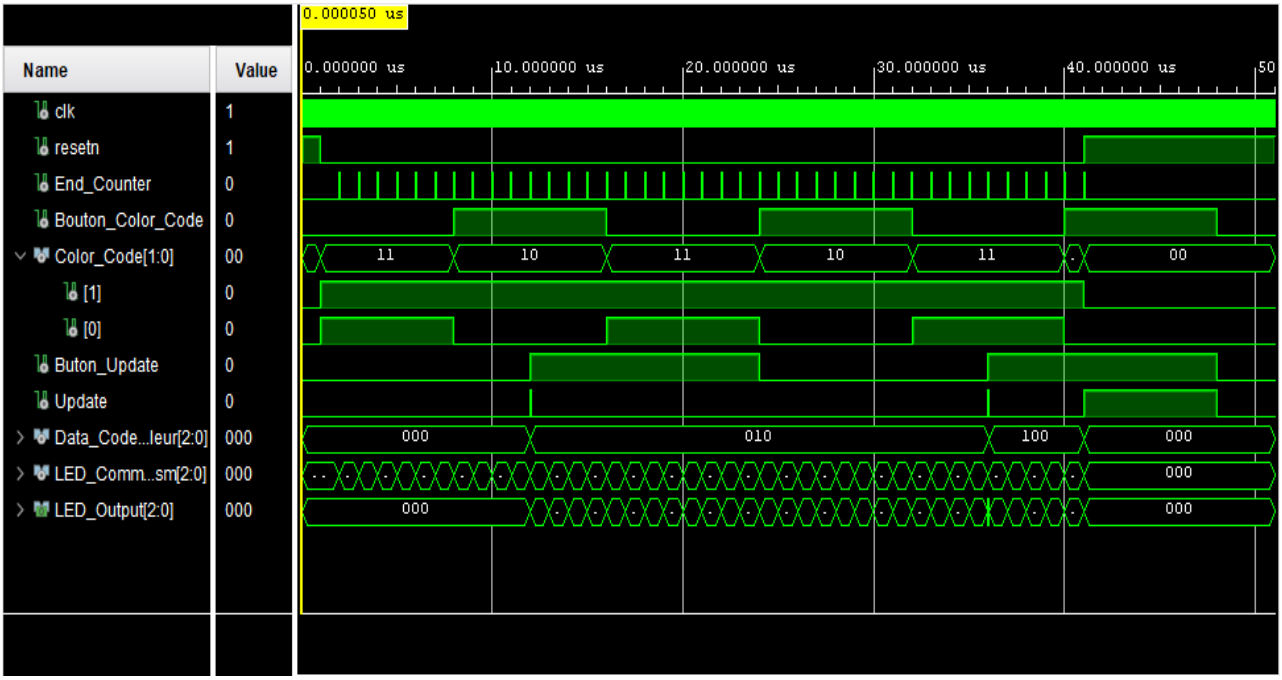
```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;


entity PilotageLED_Avec_Module_LedDriver is
    port (
        clk          : in std_logic;
        resetn       : in std_logic;
        Bouton_Color_Code    : in std_logic;
        Buton_Update         : in std_logic;
        LED_Output           : out std_logic_vector (2 downto 0)
    );
end PilotageLED_Avec_Module_LedDriver;

architecture behavioral of PilotageLED_Avec_Module_LedDriver is

    signal Color_Code   : std_logic_Vector (1 downto 0);
    signal Update       : std_logic;

    -- Signaux pour la gestion du Bouton_0
    signal Update_Registre1 : std_logic;
    signal Update_Registre2 : std_logic;

    -- Signaux la gestion des Codes Couleurs
    Signal Code_Rouge    : std_logic_Vector (1 downto 0) := "01";
    Signal Code_Vert     : std_logic_Vector (1 downto 0) := "10";
    Signal Code_Bleu     : std_logic_Vector (1 downto 0) := "11";
    Signal Code_Eteint   : std_logic_Vector (1 downto 0) := "00";

    -- Composant LedDriver
    component LedDriver
        port (
        clk          : in std_logic;
        resetn       : in std_logic;
        Color_Code   : in std_logic_Vector (1 downto 0);
        Update       : in std_logic;
        LED_Output   : out std_logic_vector (2 downto 0)
    );
    end component;

    begin
    --Affectation des signaux du testbench avec ceux de l'entite a tester
    Compteur : LedDriver
        port map (
            clk => clk,
            resetn => resetn,
            Color_Code => Color_Code,
            Update => Update,
            LED_Output => LED_Output
        );

        -- ** Début de processe Gestion du signal Update ** --
        process(resetn, clk)
        begin
            if(resetn = '1') then
                -- On réinitialise les registres
                Update_Registre1 <= '0';
                Update_Registre2 <= '0';
                Color_Code <= Code_Eteint;
            elsif(rising_edge(clk)) then
                Update_Registre1 <= Buton_Update;
                Update_Registre2 <= Update_Registre1;

                if(Bouton_Color_Code = '0') then
                    Color_Code <= Code_Bleu;
                else
                    Color_Code <= Code_Vert;
                end if;

            end if;
        end process;
        -- ** Fin de processe Gestion du signal Update ** --
        -- ** Partie combinatoire du processe Gestion du signal Update ** --
        Update <= Buton_Update and not(Update_Registre2);
        -- ** Fin Partie combinatoire du processe Gestion du signal Update ** --

end behavioral;
```

11. **Ecrivez un testbench pour tester votre design.**

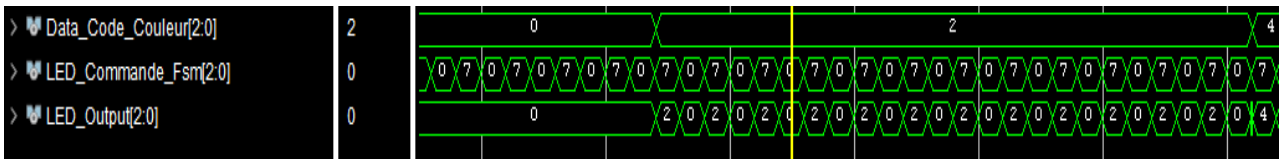
```
library ieee;
use ieee.std_logic_1164.all;

entity tb_PilotageLED_Avec_Module_LedDriver is
end tb_PilotageLED_Avec_Module_LedDriver;

architecture behavioral of tb_PilotageLED_Avec_Module_LedDriver is

    signal clk                : std_logic := '0';
    signal resetn             : std_logic := '1';
    signal Bouton_Color_Code  : std_logic := '0';
    signal Buton_Update       : std_logic := '0';
    signal LED_Output         : std_logic_vector (2 downto 0) := "000";


    -- Les constantes suivantes permette de definir la frequence de l'horloge
    constant hp : time := 5 ns;      --demi periode de 5ns
    constant period : time := 2*hp;  --periode de 10ns, soit une frequence de 100Hz

    component PilotageLED_Avec_Module_LedDriver
        port (
            clk                : in std_logic;
            resetn             : in std_logic;
            Bouton_Color_Code  : in std_logic;
            Buton_Update       : in std_logic;
            LED_Output         : out std_logic_vector (2 downto 0)
         );
    end component;

    begin
    dut: PilotageLED_Avec_Module_LedDriver
        port map (
            clk              => clk,
            resetn           => resetn,
            Bouton_Color_Code => Bouton_Color_Code,
            Buton_Update     => Buton_Update,
            LED_Output       => LED_Output
        );

    -- Process Gestion de la Clock
    process
       begin
          clk <= not clk;
          wait for hp;
    end process;
```

```
    -- Process Gestion du resetn
    process
        begin
            wait for 100*period;
            resetn <= '0';
    end process;

    -- Process Gestion du Bouton_1 : Code Couleur
    process
        begin
            for i in 1 to 4 loop
                wait for 800*period;
                Bouton_Color_Code <= not Bouton_Color_Code;
            end loop;
    end process;

    -- Process Gestion du Bouton_0 : Update
    process
        begin
            for i in 1 to 3 loop
                wait for 1200*period;
                Buton_Update <= not Buton_Update;
            end loop;
    end process;

end behavioral;
```

12. **Vérifier votre résultat à la simulation.**



On peut voir que l'on a bien le comportement recherché. Lorsque l'on a un reset actif les leds sont bien éteintes. Lorsque l'on désactive le reset et que le bouton color_code est actif, on est à vert (« 10 ») et qu'on il est inactif on est à bleu (« 11 ») pour la valeur de color_code. Ensuite, si le bouton update est actif data_code_couleur prend la valeur de code_color sinon il reste dans son état. Le changement d'état ne se fait que pour un front montant de Update.



Ensuite on peut voir ici que notre AND entre Data_Code_Couleur et la FSM se fait bien, on bien le comportement de clignotement recherché.

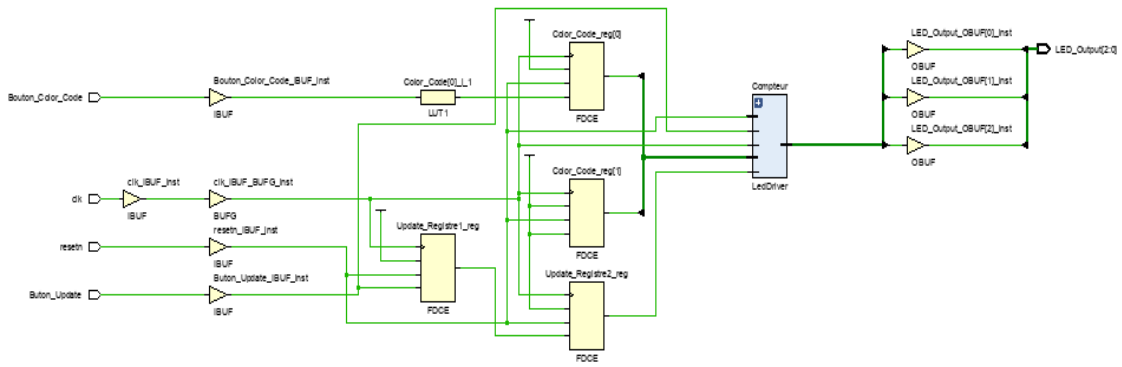
13. **Réalisez une synthèse et étudiez le rapport de synthèse, les ressources utilisées doivent correspondre à votre schéma RTL.**

J'ai modifié mon fichier de contrainte :

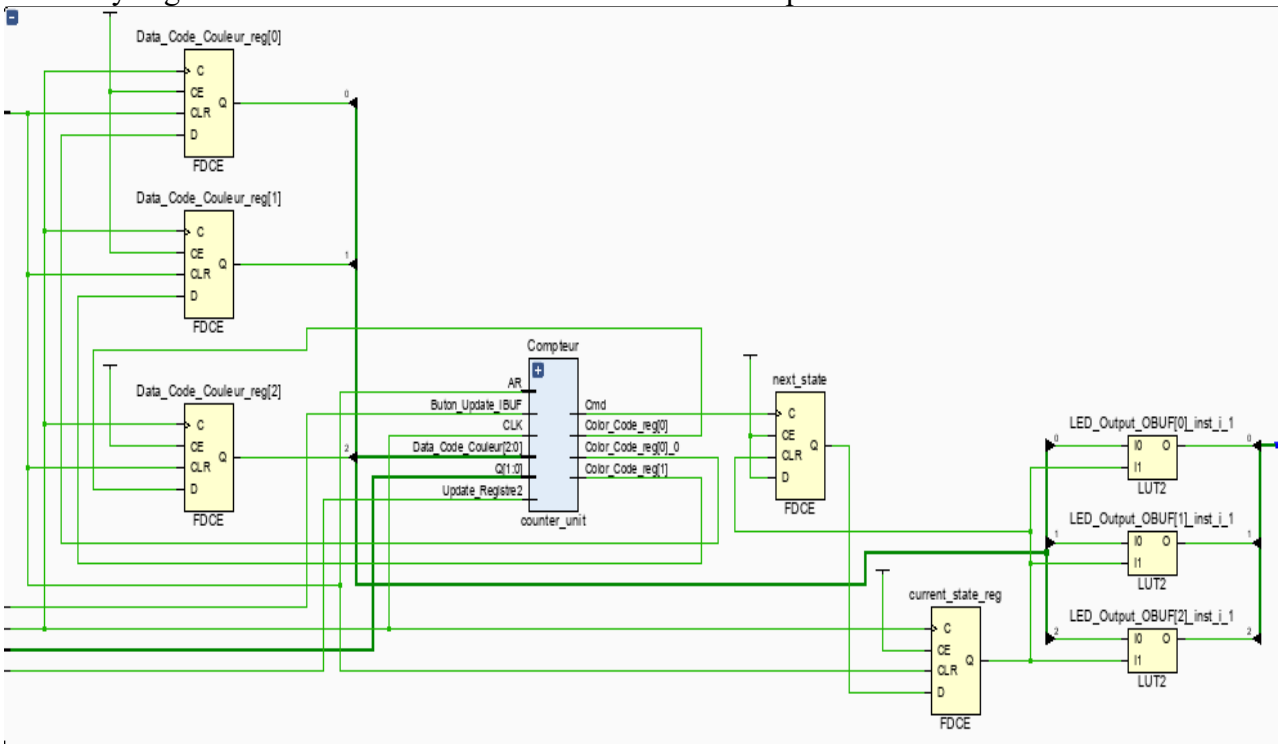
```
# PL System Clock
set_property -dict {PACKAGE_PIN H16 IOSTANDARD LVCMOS33} [get_ports clk]
create_clock -period 8.000 -name sys_clk_pin -waveform {0.000 4.000} -add [get_ports clk]
#set_property -dict { PACKAGE_PIN H16   IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L13P_T2_MRCC_35 Sch=sysclk
#create_clock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

J'obtiens ce schéma RTL :



On a bien 4 entrées, les deux boutons notre clk et le reset. On peut voir que l'on a bien 4 registres sur le schéma, deux pour la gestion du front montant de update et deux autres pour la gestion du color. Il ya également notre module ledDriver et en sortie on peut voir notre led.



On peut voir ici l'intérieur de notre module LedDriver, on voit bien nos 3 registres pour la gestion de code couleur, notre compteur_unit et la FSM qui permettent le clignotement des Leds.

```
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
```

Ici on peut voir nos 3 registres et nos 3 Muxes comme sur notre schéma RTL.

```
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|
+------+-------+------+
```

On peut voir que l'on a 4 entrées et 3 sorties. Il y a également nos 36 registres : 26 registres pour le module counter_unit, 6 pour le module led_driver avec la fsm et 4 pour notre dernier projet.

14. **Effectuez le placement routage et étudiez les rapports**

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

    WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints      WHS(ns)      THS(ns)
    -------      -------  ---------------------  -------------------      -------      -------
      3.264        0.000                      0                   31        0.156        0.000
```

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	3,264 ns	Worst Hold Slack (WHS):	0,156 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

On peut voir que l'on a pas de slack car TNS et THS sont à zéro. Nous n'avons pas de valeur négative donc pas de métastibilité.

Chemin critique :

```
Max Delay Paths
--------------------------------------------------------------------------------
Slack (MET) :             3.264ns  (required time - arrival time)
  Source:                 Compteur/Compteur/D_out_reg[15]/C
                            (rising edge-triggered cell FDCE clocked by sys_clk_pin  {rise@0.000ns fall@4.000ns period=8.000ns})
  Destination:            Compteur/Compteur/D_out_reg[25]/D
                            (rising edge-triggered cell FDCE clocked by sys_clk_pin  {rise@0.000ns fall@4.000ns period=8.000ns})
```

On observe que le chemin le plus long est celui du compteur car on passe beaucoup plus de registres.

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