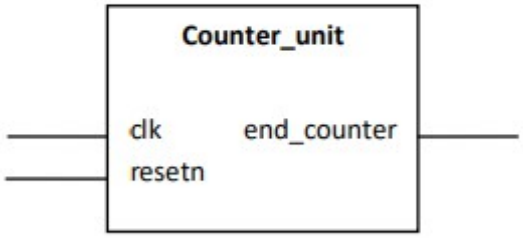
Atmani Hicham TP3-Machine à états

Objectif

L'objectif de ce TP est de réaliser une architecture permettant de faire clignoter deux LEDs RGB en rouge, vert et bleu. Le pilotage des LEDs se fera à l'aide de machines à états. Lors de ce TP vous apprendrez à utiliser des paramètres génériques ainsi que des modules.

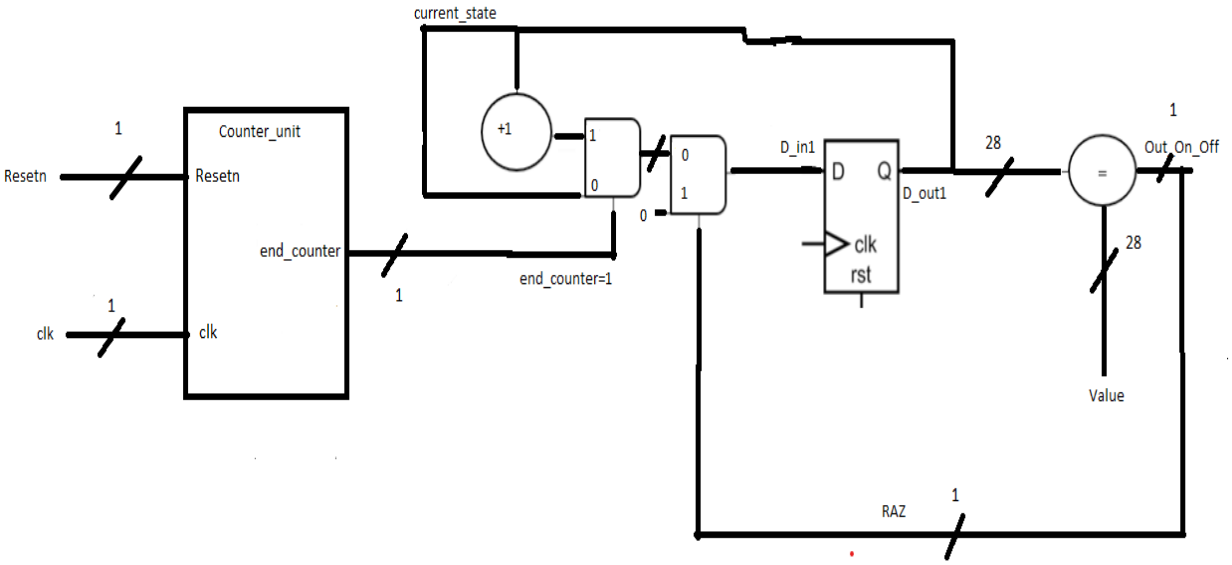
1. Dans un fichier *.vhd*, créez un module *Counter_unit* à partir du compteur du TP1. Le module prendra en entrée un signal d'horloge et de resetn, et donnera en sortie le signal *end_counter*. Utilisez un paramètre *generic()* pour définir le nombre de coup d'horloge à compter.



Le code de *Counter_unit* ne sera plus modifié ensuite.

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

entity counter_unit is

    generic ( constant Cst : positive := 200000000);
    port (
        clk         : in std_logic;
        resetn      : in std_logic;
        End_Counter : out std_logic;
     );
end counter_unit;

architecture behavioral of counter_unit is

    --Declaration des signaux internes
    signal D_out : positive range 0 to Cst ;
    Signal Cmd   : std_logic := '0';

    begin

        --Partie sequentielle
        process(clk,resetn)
        begin
            if(resetn = '1') then
                D_out <= 0;
                elsif(rising_edge(clk)) then
                    D_out <= D_out + 1 ;
                    if (Cmd = '1') then
                        D_out <= 0;
                    end if;
                end if;
            end if;
    end process;

    --Partie combinatoire
    Cmd <= '1'  when (D_out = Cst-1 )
           else '0';
    End_Counter <= Cmd;

end behavioral;
```

2. **En schéma RTL, créez un compteur du signal end_counter. Ce compteur doit permettre de déterminer le nombre de cycles allumé/éteint qui ont été effectués par la LED. Le compteur doit pouvoir être remis à 0, maintenir sa valeur actuelle ou s'incrémenter.**



3. **Ecrivez un code VHDL décrivant ce compteur de cycle, vous utiliserez le module Counter_unit.**

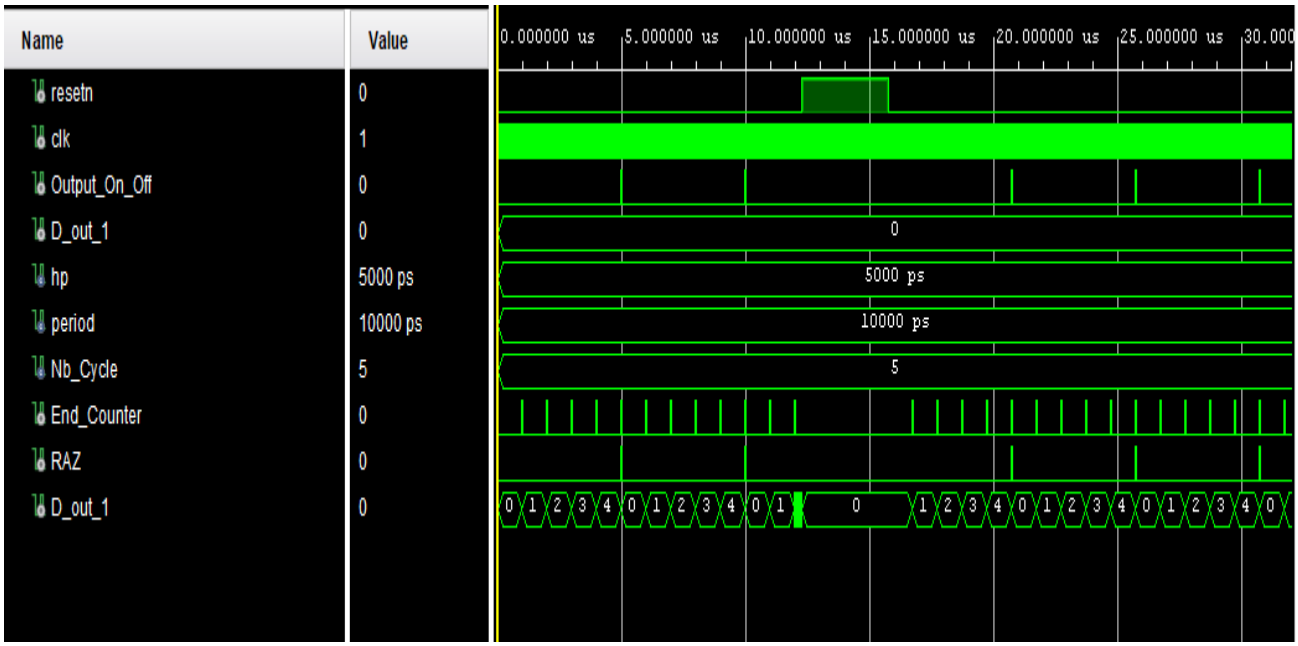
```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;


entity tp_fsm is
    generic (
        constant Nb_Cycle : positive := 5
    );
    port (
        clk         : in std_logic;
        resetn      : in std_logic;
        --a completer
        Output_On_Off : out std_logic
    );
end tp_fsm;

architecture behavioral of tp_fsm is

    --type state is (idle, state1, state2); --a modifier avec vos etats

    --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 D_out_1 : positive range 0 to Nb_Cycle ; -- Pour la synthèse
    signal D_out_1 : positive := 0;
    signal RAZ        : std_logic;


    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
        );


        process(clk,resetn)
        begin
            if(resetn = '1') then
                D_out_1 <= 0;
                --current_state <= idle;
            elsif(rising_edge(clk)) then

                --current_state <= next_state;

                --a completer avec votre compteur de cycles
                if(RAZ = '0') then
                    if(End_counter = '1') then
                        D_out_1 <= D_out_1 + 1;
                    elsif(End_counter = '0') then
                        D_out_1 <= D_out_1;
                    end if;
                else
                    D_out_1 <= 0;
                end if;
            end if;
        end process;

        --Partie combinatoire
    RAZ <= '1'  when (D_out_1 = Nb_Cycle-1 and End_Counter ='1')
            else '0';
    Output_On_Off <= RAZ;

end behavioral;
```

4. Tester votre architecture avec un testbench.

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

entity tb_tp_fsm is
end tb_tp_fsm;

architecture behavioral of tb_tp_fsm is

    signal resetn      : std_logic := '1';
    signal clk         : std_logic := '0';
    --a completer
    signal Output_On_Off : std_logic := '0';
    --constant nb_loop   : positive := 100;
    signal D_out_1 : positive := 0;


    -- 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
    constant Nb_Cycle : positive := 5;
    constant Nb_loop : positive := 350;

    component tp_fsm
        port (
            clk            : in std_logic;
            resetn         : in std_logic;
            --a completer
            Output_On_Off : out std_logic
        );
    end component;
begin
    dut: tp_fsm
        port map (
            clk => clk,
            resetn => resetn,
            --a completer
            Output_On_Off => Output_On_Off
        );
```

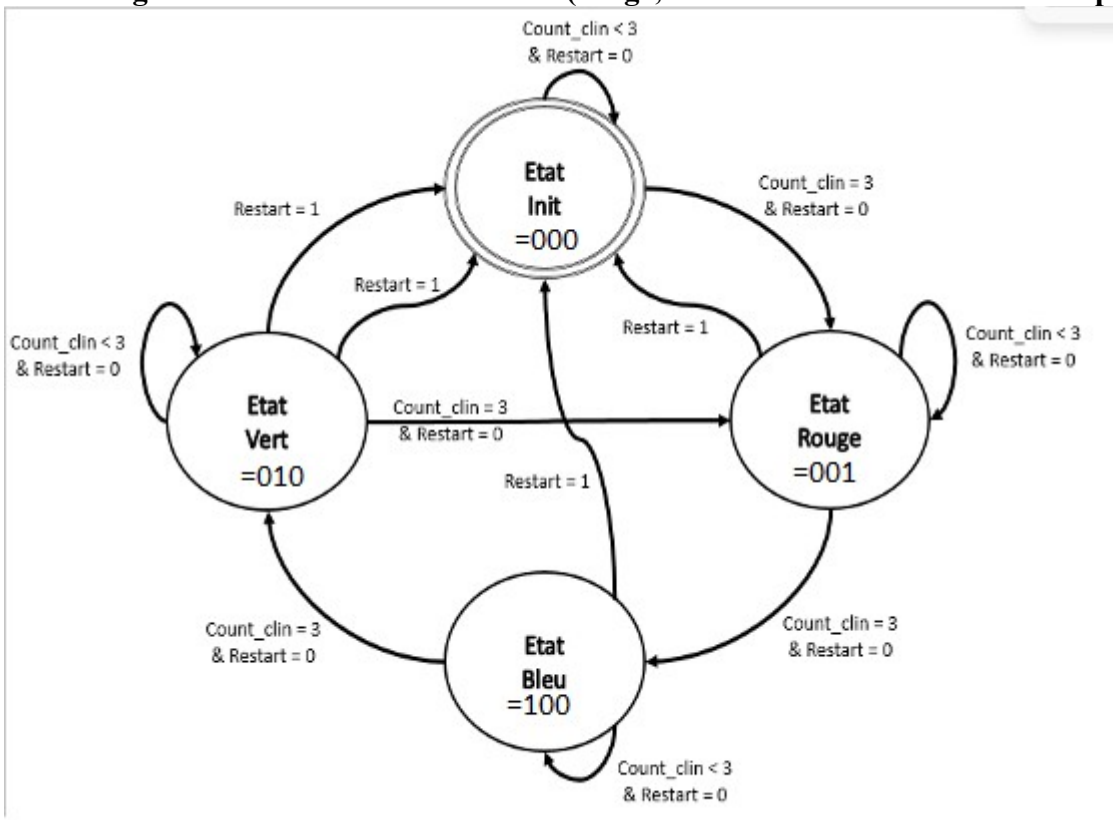
```
    process
    begin

      -- Test avec Reset non active
       resetn <= '0';
       for i in 1 to 2 loop
           for i in 1 to nb_loop loop
             clk <= not clk;
             wait for hp;
             if (D_out_1 = Nb_Cycle-1) then
                 assert Output_On_Off = '0'
                 report "Output_On_Off : test failed, il devrait etre a  0" severity failure;
              end if;
           end loop;
        end loop;
       -- Test avec Reset active
       resetn <= '0';
       for i in 1 to 10 loop
           if (i > 5 and i < 8) then
               resetn <= '1';
           else
               resetn <= '0';
           end if;
           for i in 1 to nb_loop loop
              clk <= not clk;
              wait for hp;
              if (D_out_1 = Nb_Cycle-1) then
                 assert Output_On_Off = '1'
                 report "Output_On_Off : test failed, il devrait etre a  1" severity failure;
              end if;
            end loop;
        end loop;
    end process;
end behavioral;
```



On peut voir que le comportement est respecté. On compte bien les fronts montants de End_counter et à 5 (Valeur pour la simulation), on a bien une remise à zéro et notre sortie Output_On_Off qui se met à un. De plus quand le reset est activé on ne compte plus donc le reset fonctionne.

5. **Créez en RTL une machine à états (FSM) permettant de faire clignoter une LED RGB en rouge puis bleu et enfin en vert avant de recommencer le cycle (rouge, bleu, vert, …). Dans chaque état la LED devra clignoter 3 fois. De plus, si le bouton restart est appuyé, on retourne dans l'état initial quel que soit l'état dans lequel on se situe. L'état initial est l'état dans lequel on se situe au démarrage, on passe à l'état rouge après 3**

clignotements de la LED en blanc (rouge, vert et bleu actifs en même temps).



6. **Listez les signaux d'entrée, de sortie et les signaux internes de votre architecture**
7.

Les signaux d'entrée sont :

- Clk : in std_logic ; → Horloge de notre système
- Resetn : in std_logic ; → Reset de notre système
- Restart : in std_logic ; → RAZ de la machine d'état

Les signaux de sortie sont :

- Output_On_Off : out std_logic ; → Sortie du compteur de notre Syst.
- LED_OUT : out std_logic_vector (2 downto 0) → LEDs RGB pour la carte CORA

Les signaux internes sont :

- S_LED_OUT : std_logic_vector (2 downto 0) ; → Signal interne pour LED_OUT
- Compteur_clignotement : positive :=0 ; → Comptage du Nb de clignotement
- Validation_clignotement : std_logic ; → Permet la validation du chgt d'état

8. **Ajoutez à votre code VHDL les éléments que vous venez de créer**

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


entity tp_fsm is
    generic (
        --vous pouvez ajouter des parameres generics ici
        constant Nb_Cycle : positive := 3
    );
    port (
        clk         : in std_logic;
        resetn      : in std_logic;
        --a completer
        Restart      : in std_logic;
        Output_On_Off : out std_logic;
        LED_OUT : out std_logic_vector( 2 downto 0)
     );
end tp_fsm;

architecture behavioral of tp_fsm is

    type state is (Etat_Init, Etat_Rouge, Etat_Bleu, Etat_Vert); --a modifier avec vos etats

    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 D_out_1 : positive := 0; -- Pour la simulation !!!!
    signal RAZ     : std_logic;
    signal S_LED_OUT : std_logic_vector(2 downto 0);
    signal Count_clign : positive := 0;
    signal Validation_Clignotement : std_logic;
    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
        );
```

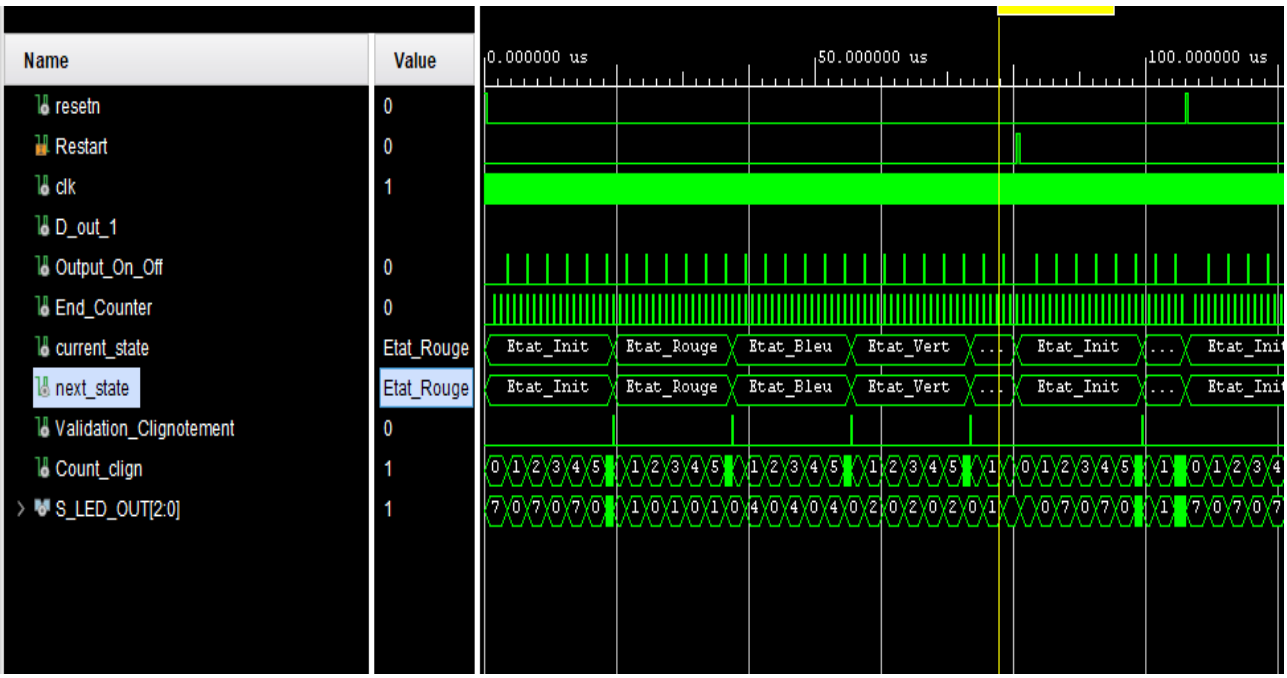
```
process(clk,resetn,Restart)
begin
    if(resetn = '1' or Restart ='1') then
        Count_clign <= 0;
        current_State <= Etat_Init;
        D_out_1 <= 0;
    elsif(rising_edge(clk)) then

         current_state <= next_state;
        --a completer avec votre compteur de cycles
        if(RAZ = '0') then
            if(End_counter = '1') then
                D_out_1 <= D_out_1 + 1;
            elsif(End_counter = '0') then
                D_out_1 <= D_out_1;
            end if;
         else
            D_out_1 <= 0;
         end if;
         if(D_out_1 = Nb_Cycle-1 and End_Counter = '1') then
            Count_clign <= Count_clign+1;
           -- Validation_Clignotement <= '0';
            else if (Validation_Clignotement ='1') then
                Count_clign <= 0;
              --  Validation_Clignotement <= '1';
            else
                Count_clign <= Count_clign;
               -- Validation_Clignotement <= '0';
            end if;
         end if;
    end if;
end process;

--Partie combinatoire
RAZ <= '1' when (D_out_1 = Nb_Cycle-1 and End_Counter = '1')
        else '0';
Output_On_Off <= RAZ;
LED_OUT<=S_LED_OUT;
Validation_Clignotement <= '1' when (Count_clign = Nb_Cycle*2 and End_Counter = '1')
        else '0';



-- FSM
process(current_state, Restart, Validation_Clignotement, count_clign) --a completer avec vos signaux
begin
   case current_state is
      when Etat_Init =>
        if(Restart = '0') then
               if (Validation_Clignotement = '0') then
                    if(Count_clign mod 2 = 0 ) then
                        S_LED_OUT <= "000";
                    else
                        S_LED_OUT <= "111";
                    end if;
                    next_state <= Etat_Init;
                else
                    next_state <= Etat_Rouge;
                end if;
        else
            next_state <= Etat_Init;

        end if;
```

```
                when Etat_Rouge =>
                  if(Restart = '0') then
                       if (Validation_Clignotement = '0') then
                            if(Count_clign mod 2 = 0 ) then
                                S_LED_OUT <= "000";
                            else
                                S_LED_OUT <= "001";
                            end if;
                           next_state <= Etat_Rouge;
                           else
                               next_state <= Etat_Bleu;
                           end if;
                  else
                      next_state <= Etat_Init;

                  end if;
                when Etat_Bleu =>
                    if(Restart = '0') then
                          if (Validation_Clignotement = '0') then
                             if(Count_clign mod 2 = 0 ) then
                                 S_LED_OUT <= "000";
                             else
                                 S_LED_OUT <= "100";
                             end if;
                             next_state <= Etat_Bleu;
                         else
                             next_state <= Etat_Vert;
                         end if;
                     else
                          next_state <= Etat_Init;

                     end if;

               when Etat_Vert =>
                  if(Restart = '0') then
                       if (Validation_Clignotement = '0') then
                            if(Count_clign mod 2 = 0 ) then
                                S_LED_OUT <= "000";
                            else
                                S_LED_OUT <= "010";
                            end if;
                            next_state <= Etat_Vert;
                           else
                               next_state <= Etat_Rouge;
                           end if;
                  else
                     next_state <= Etat_Init;

                  end if;

                  When OTHERS =>
                    next_state <= Etat_Init;
                    S_LED_OUT <= "000";
              end case;
        end process;

end behavioral;
```

8. **Ecrivez un testbench pour tester votre architecture. Vérifiez à la simulation que vous obtenez le résultat attendu.**

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

entity tb_tp_fsm is
end tb_tp_fsm;

architecture behavioral of tb_tp_fsm is

signal resetn      : std_logic := '1';
    signal clk         : std_logic := '0';
    --a completer
    signal Restart      : std_logic := '0';
    signal Output_On_Off : std_logic := '0';
    signal LED_OUT : std_logic_vector( 2 downto 0) := "000";

    --constant nb_loop   : positive := 100;
    signal D_out_1 : positive := 0;


    -- 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
    constant Nb_Cycle : positive := 3;
    constant Nb_loop : positive := 250;
    signal Indice : std_logic := '0';

    component tp_fsm
        port (
            clk            : in std_logic;
            resetn        : in std_logic;
            Restart     : in std_logic;
            --a completer
            Output_On_Off : out std_logic;
            LED_OUT : out std_logic_vector( 2 downto 0)
        );
    end component;

    begin
    dut: tp_fsm
        port map (
            clk => clk,
            resetn => resetn,
            Restart => Restart,
            --a completer
            Output_On_Off => Output_On_Off,
            LED_OUT => LED_OUT
        );
```

```
process
    begin

        for i in 1 to 100 loop
            resetn <= '1';
            clk <= not clk;
            wait for hp;
        end loop;
        for i in 1 to 16000 loop
            resetn <='0';
            restart <= '0';
            clk <= not clk;
            wait for hp;
        end loop;
        for i in 1 to 100 loop
            restart <= '1';
            clk <= not clk;
            wait for hp;
        end loop;
        for i in 1 to 5000 loop
            restart <= '0';
            clk <= not clk;
            wait for hp;
        end loop;
        for i in 1 to 100 loop
            resetn <= '1';
            clk <= not clk;
            wait for hp;
        end loop;
        for i in 1 to 5000 loop
            resetn <= '0';
            clk <= not clk;
            wait for hp;
        end loop;

    end process;

end behavioral;
```



On peut voir que l'on a bien le fonctionnement attendu, tout d'abords on passe bien de l'état blanc, à l'état rouge, à l'état bleu à l'état vert puis à l'état rouge. Quand on a un reset ou un restart notre compteur se réinitialise et on repasse à l'état initiale. De plus pour chaque état notre compteur compte jusqu'à 6, notre led s'éteint et s'allume (3 fois allumer et 3 fois état) de la couleur souhaitée. On change d'état car notre validation_clignotement passe à 1.

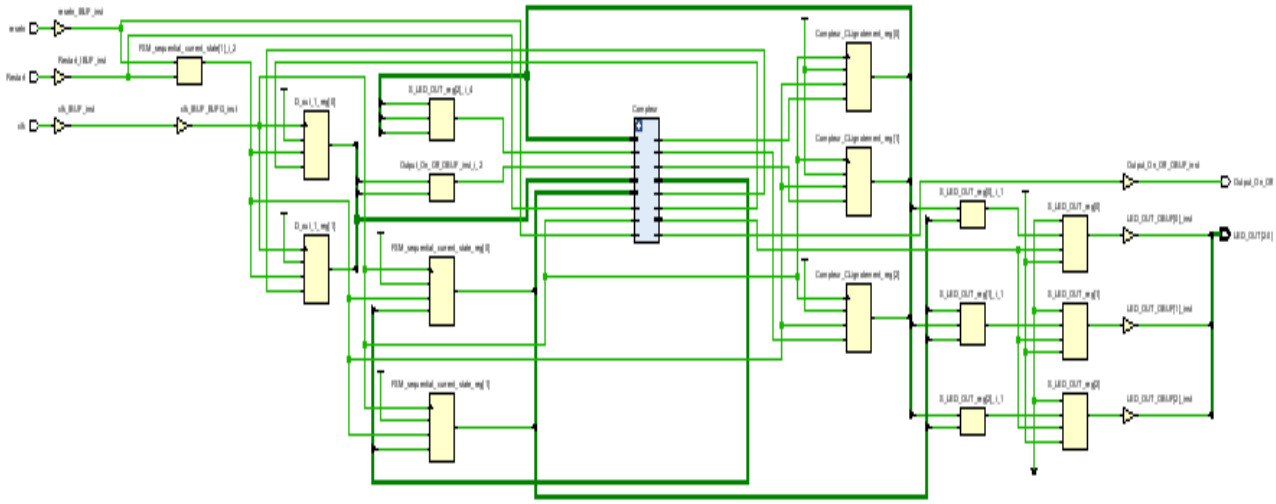
9. **Exécutez la synthèse et relevez les ressources utilisées (y compris la FSM). Sur la schématique, identifiez où se situe votre compteur de cycle.**

On peut voir ci-dessous dans le rapport de synthèse notre machine d'états avec nos 4 états :

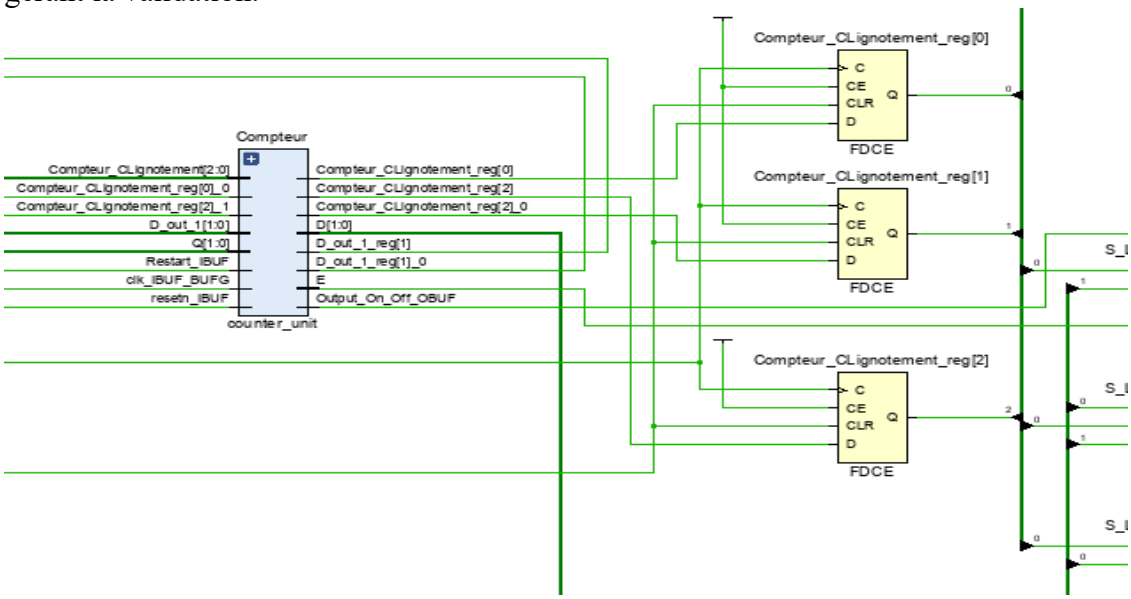
State	New Encoding	Previous Encoding
etat_init	00	00
etat_rouge	01	01
etat_bleu	10	10
etat_vert	11	11

On peut voir ci-dessous les composants de notre schéma RTL :

```
---------------------------------------------------------------------------------
Start RTL Component Statistics
---------------------------------------------------------------------------------
Detailed RTL Component Info :
+---Adders :
       2 Input     3 Bit       Adders := 1
       2 Input     2 Bit       Adders := 1
+---Registers :
                   3 Bit    Registers := 1
                   2 Bit    Registers := 1
+---Muxes :
       2 Input     3 Bit        Muxes := 1
       4 Input     3 Bit        Muxes := 1
       2 Input     2 Bit        Muxes := 8
       4 Input     2 Bit        Muxes := 1
       2 Input     1 Bit        Muxes := 2
---------------------------------------------------------------------------------
Finished RTL Component Statistics
---------------------------------------------------------------------------------
```



On peut voir notre module counter_unit au centre et sa droite notre compteur de clignotement en gérant la validation.



10. Modifiez le fichier de contraintes pour connecter vos entrées / sorties du système avec les broches de la carte. Réglez l'horloge pour que sa fréquence soit à 100MHz

```
# 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 { led1_g }]; #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
Sset_property -dict { PACKAGE_PIN D20   IOSTANDARD LVCMOS33 } [resetn]]; #IO_L4N_T0_35 Sch=btn[0]
set_property -dict { PACKAGE_PIN D19   IOSTANDARD LVCMOS33 } [Restart]; #IO_L4P_T0_35 Sch=btn[1]

# PL System Clock
set_property -dict {PACKAGE_PIN H16 IOSTANDARD LVCMOS33} [get_ports clk]
create clock -period 10.000 -name sys clk pin -waveform {0.000 5.000} -add [get ports clk]
```

11. Lancez l'implémentation puis étudiez le rapport de timing (vérifiez les violations de set up et de hold et identifiez le chemin critique).

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

WNS(ns)	TNS(ns)	TNS Failing Endpoints	TNS Total Endpoints	WHS(ns)	THS(ns)
2.746	0.000	0	34	0.267	0.000

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	2,746 ns	Worst Hold Slack (WHS):	0,267 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:	34	Total Number of Endpoints:	34	Total Number of Endpoints:	35

On peut voir que l'on n'a aucune violation (pas de slack) donc il n'y aura pas de métastabilité car on peut voir qu'aucune valeur n'est dans le négatif pour le THS et dans le TNS.

Chemin critique :

```
Slack (MET) :           2.746ns  (required time - arrival time)
  Source:               Compteur/D_out_reg[3]/C
                          (rising edge-triggered cell FDCE clocked by sys_clk_pin  {rise@0.000ns fall@4.000ns period=8.000ns})
  Destination:          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})
```

12. Générez le bitstream pour vérifier le système sur carte.

Voir vidéo dans le dossier