Segment 2

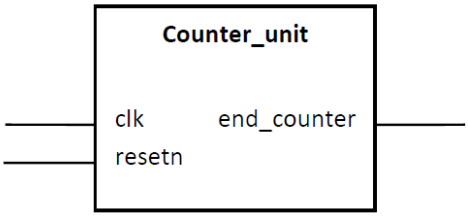
TP3 : FSM

Objectif

L'objectif de cet TP est faire clignoter une LED en utilisant un compteur de temporisation. Un compteur de temporisation permet de compter le nombre de coup d'horloge nécessaire pour attendre un temps voulu. En connaissant la fréquence de l'horloge il est possible de déterminer combien de périodes d'horloge il faut compter pour attendre 3 secondes par exemple.

Questions

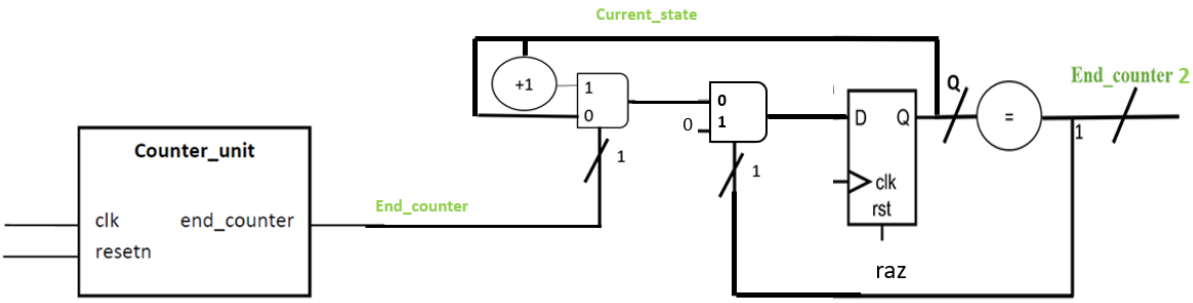
1. Dans un fichier *.vhd*, créez un module *Counter_unit* à partir du compteur du TP1. Le module prendra en entrée un signal d'horloge et de resetn, et donnera en sortie le signal *end_counter*. Utilisez un paramètre *generic()* pour définir le nombre de coup d'horloge à compter. Le code de Counter_unit 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 (
         max_count : integer :=10;
         nb_bit : integer := 4
         );
    port (
        clk         : in std_logic;
        resetn      : in std_logic;

        end_counter  : out std_logic
     );
end counter_unit;

architecture behavioral of counter_unit is

     signal Q : std_logic_vector((nb_bit-1) downto 0);
     signal end_count : std_logic;

begin
        --Partie sequentielle
        process(clk,resetn)
        begin
            if(resetn = '1') then
            Q   <=(others => '0');

            elsif(rising_edge(clk)) then

              Q <= Q+ 1;
              if(end_count= '1') then
             Q  <=(others => '0');

            end if;
            end if;

        end process;

        --Partie combinatoire
      end_count <= '1' when (Q =(max_count)-1)
                else '0';

    end_counter <= end_count;

end behavioral;
```

Dans ce module Counter_unit, nous avons utilisé un paramètre générique max_count pour définir le nombre à compter. Le compteur counter_unit est une valeur non signée de 4 bits qui compte jusqu'à max_count - 1. Lorsque le compteur atteint max_count - 1, le signal end_counter est mis à '1' pour indiquer que le comptage est terminé.

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.



Le passage à 1 d'end_counter 2 avec le paramètre générique permet de définir le temps des états des leds(allumé/éteint). Le nombre de cycles allumé/éteint dans notre tp est de 6. On a trois étapes ou cycle d'allumer et trois d'états d'éteints.

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

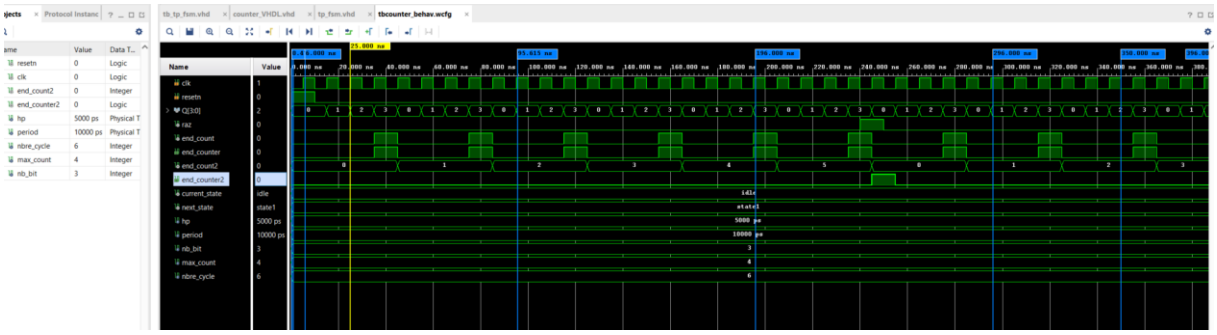
```
 library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith;

entity tp_fsm is
    generic (
       max_count : integer :=4;
       nb_bit : integer := 3;
       nbre_cycle : positive :=6  --cycle de clignotement allumé et éteint
    );
    port (
        clk         : in std_logic;
        resetn      : in std_logic;
                --a completer
        --restart     : in std_logic;
        end_counter2 : out std_logic;
        led_out_R  : out std_logic;
        led_out_B  : out std_logic;
        led_out_V  : out std_logic
     );
end tp_fsm;

architecture behavioral of tp_fsm is


    signal raz : std_logic;
    signal end_count : std_logic;
    signal end_count2 : positive :=0;

 --Declaration de l'entite a tester
   component counter_unit
   generic (
         max_count : integer :=4;
         nb_bit : integer := 3
         );
        port (
            clk           : in std_logic;
            resetn        : in std_logic;
```

```
            end_counter    : out std_logic
        );
 end component;

    begin

compteur : counter_unit
generic map (
        max_count =>4,
        nb_bit => 3
        )
port map (
            clk => clk,
            resetn=>resetn,
            end_counter => end_count
        );

    -- Process séquentielle

        process(clk,resetn)
        begin

       if(resetn='1') then
            end_count2<= 0;

            -- current_state <= idle;

            elsif(rising_edge(clk)) then
                  if(raz='0') then
                        if (end_count='1') then
                     end_count2<=end_count2 +1;
                         elsif(end_count='0') then
                    end_count2<=end_count2;
                        end if;
                 else end_count2<=0;

                --current_state <= next_state;
                  end if;
        end if;
                --a completer avec votre compteur de cycles

        raz <= '1'  when (end_count2 = ((nbre_cycle)-1) and end_count = '1')
                else '0';
         end_counter2 <= raz;

       end process;

end behavioral;
```

4. Tester votre architecture avec un testbench.

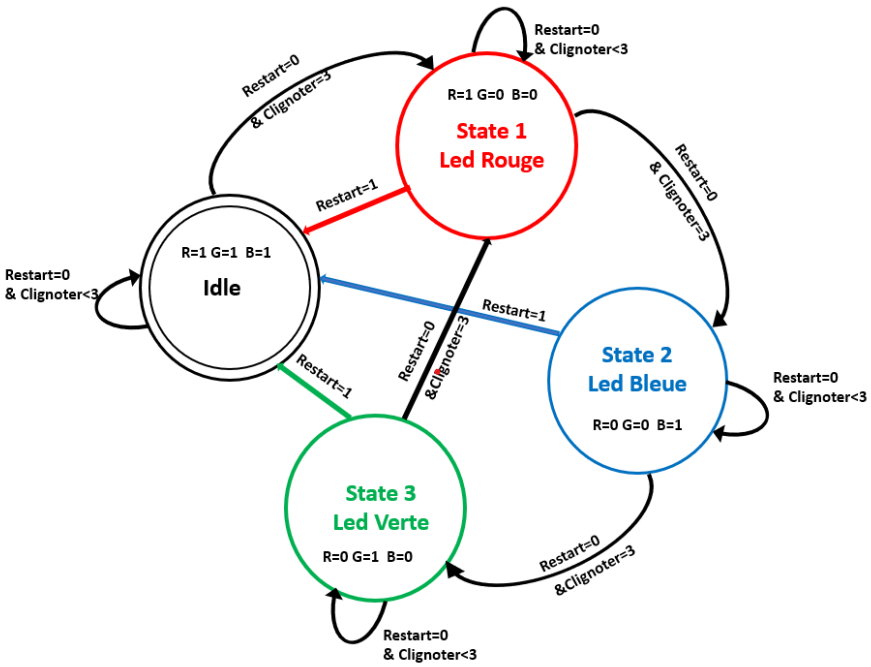
```
 library ieee;
use ieee.std_logic_1164.all;


entity tb_tp_fsm is
end tb_tp_fsm;

architecture behavioral of tb_tp_fsm is

    signal resetn      : std_logic := '0';
    signal clk         : std_logic := '0';
    --a completer
    signal end_count2 : positive :=0;
    signal end_counter2 : std_logic:='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 nbre_cycle : positive := 6 ;
     constant  max_count : integer :=4;
      constant   nb_bit : integer := 3;

    component tp_fsm
        port (
            clk         : in std_logic;
            resetn      : in std_logic;
            --a completer
             end_counter2  : out std_logic
         );
    end component;



    begin
    dut: tp_fsm
        port map (
            clk => clk,
            resetn => resetn,
            --a completer
            end_counter2 => end_counter2
        );

    --Simulation du signal d'horloge en continue
    process
```

```
    --Simulation du signal d'horloge en continue
    process
    begin

        wait for hp;
        clk <= not clk;


    end process;


    process
    begin

        resetn <= '1';
        wait for period*1;
        resetn <= '0';
         assert end_counter2='0'
             report "end_counter2 : test failed";
         wait for period*2;
            --a completer
        wait for period*nbre_cycle*max_count;
        assert end_counter2='1'
        report "end_counter2 : test failed";
        wait;
    end process;


end behavioral;
```



On remarque que le comportement du compteur permet de déterminer le nombre de cycles allumé/éteint qui ont été effectués par la LED. Le compteur est ensuite remis à 0, pour maintenir sa valeur actuelle ou s'incrémenter. On compte 6 fronts montant d'end_counter et end_count2. End_couter2 est alors à 1, puis remise à 0 pour un nouveau cycle.

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.

Les signaux d'entrées

-Clk : l'horloge

-Resetn : le reset

-Restart : la remise à l'état initial de notre FSM

Les signaux de sorties

-end_counter2 : Sortie du compteur

-Led_out : Leds RGB

Les signaux internes

current_state : état dans lequel se trouve la led actuellement

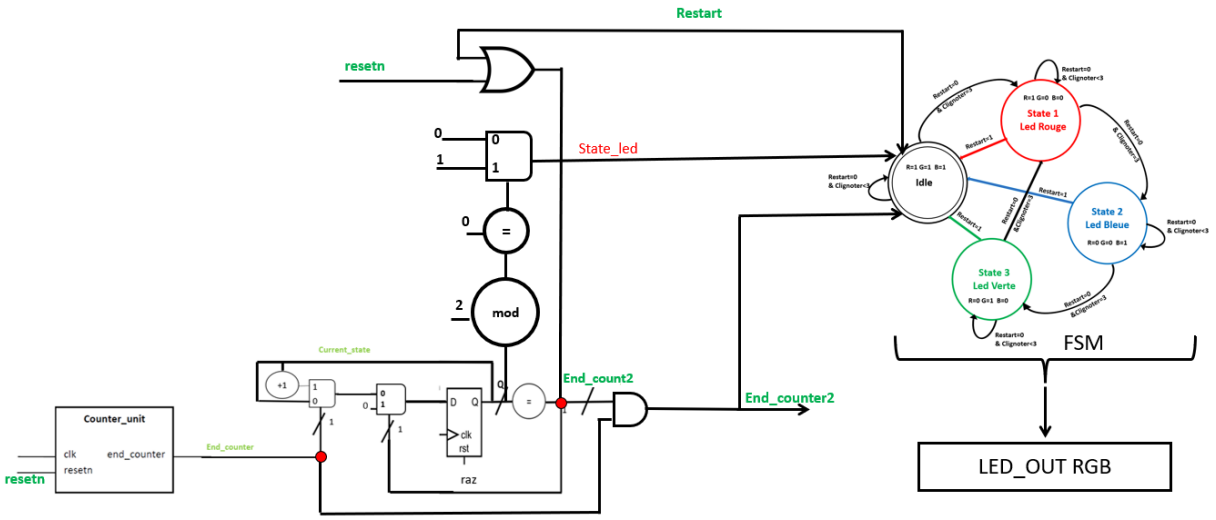
-next_state : état dans lequel va se trouver la led au prochain coup d'horloge

-s_led_out : signal interne pour led_out RGB

-state_led : état de la led.

On garde tous les signaux internes déclarer pour le compteur unit et la remise à zéro.

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



Pour les sortie de leds en RGB, un mux est utilisé dans la fsm(current_state) avec le state_led pour chaque couleur

Le schéma de l'ensemble, les signaux clock et resten sont reliés entre eux. Les points rouges sont les points de connexions du même signal. Sur ce schéma RTL, nous rajoutons à la FSM qui gère les états (state1, stade2, state3, et idle) de la led, le module du compteur, des opérateurs logiques et le restart pour un bon fonctionnement. Pour gérer le clignotement de la led(éteint/allumé), un opérateur modulo et un diviseur 2 est utilisé. En sortie d'end_count2, nous avons 0 pour le pair ou 1 pour les impairs. Les leds sont allumées sur les impairs (state_led). La FSM vient ensuite gérer les états : initial, led rouge, led bleue et led verte. Le compteur changerait d'état après 6 cycles au front montant.

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

entity tp_fsm is
    generic (
--        max_count : integer :=200000000;
--        nb_bit : integer := 28;
       max_count : integer :=4;
       nb_bit : integer := 3;
       nbre_cycle : positive :=6  --cycle de clignotement allumé et éteint
    );
    port (
        clk         : in std_logic;
        resetn      : in std_logic;
                --a completer
        restart     : in std_logic;
        end_counter2 : out std_logic;
        led_out_R  : out std_logic;
        led_out_B  : out std_logic;
        led_out_V  : out std_logic
     );
end tp_fsm;


architecture behavioral of tp_fsm is

    type state is (idle, state1, state2, state3); --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  s_led_out_B  : std_logic;  -- signal de sortie de LED RGB
    signal  s_led_out_V  : std_logic;
    signal  s_led_out_R  : std_logic;
    signal raz : std_logic;                  -- signal de remise à zéro
    signal end_count : std_logic;             -- signal de sortie de counter_unit
    signal end_count2 : integer range 0 to 5:=0;                -- signal de sortie de counter2
    signal state_led: std_logic;                        -- signal de'état de led

 --Declaration de l'entite a tester counter_unit
   component counter_unit
   generic (
--         max_count : integer :=200000000;
--         nb_bit : integer := 28
         max_count : integer :=4;
       nb_bit : integer := 3
        );
        port (
            clk           : in std_logic;
            resetn        : in std_logic;

            end_counter   : out std_logic
        );
 end component;

    begin


compteur : counter_unit
generic map (
--         max_count =>200000000,
--         nb_bit => 28
         max_count =>4,
       nb_bit => 3
        )
port map (
            clk => clk,
            resetn=>resetn,
            end_counter => end_count
       );
```

```vhdl
    -- Process séquentielle du clk, resetn, restart

        process(clk,resetn,restart)
        begin

    if(resetn='1' or restart='1') then
            end_count2<= 0;
            current_state <= idle;

    elsif(rising_edge(clk)) then
            current_state <= next_state;

             if(raz='0') then
```

```
            if(raz='0') then
                  if (end_count='1') then
                    end_count2<=end_count2 +1;
                   elsif(end_count='0') then
                       end_count2<=end_count2;
                   end if;
               else
                    end_count2<=0;
                end if;

        end if;

end process;

                --Partie combinatoire a completer avec votre compteur de cycles

                --remise à zero à la fin du compteur de cycles
        raz <= '1'  when (end_count2 = ((nbre_cycle)-1) and end_count = '1')
                else '0';
         end_counter2 <= raz;


       -- Vérification de l'état de end_counter2 pour determine si la led doit etre 'on' or 'off'
         state_led <= '0' when( end_count2 mod 2 = 0 )else '1';


         -- FSM
process(current_state, restart,state_led,end_counter2) --a completer avec les signaux
current_state, restart,state_led,end_counter2

begin
             --signaux pilotes par la fsm les signaux sont affectés dans les différentes cas

           case current_state is
              when idle =>
              s_led_out_R<=state_led;
              s_led_out_B<=state_led;
              s_led_out_V<=state_led;
             if restart='0' then

                  if(end_counter2='1') then
                    next_state <= state1;

                    else

                    next_state <= idle;              --prochain etat
                    end if;
                else
                next_state<=idle;

                   end if;

              when state1 =>
                s_led_out_R<=state_led;
                s_led_out_B<='0';
                s_led_out_V<='0';
             if restart='0' then

                    if(end_counter2='1') then
                    next_state <= state2;              --prochain etat

                    else

                    next_state <= state1;

                    end if;
                else
                next_state <= idle;

                end if;

              when state2 =>
                s_led_out_R<='0';
                s_led_out_B<=state_led;
                s_led_out_V<='0';
             if restart='0' then
                  if(end_counter2='1') then

                   next_state <= state3;                    --prochain etat

                    else

                    next_state <= current_state;
                    end if;
                  else
                next_state <= idle;
                end if;
```

```
171
172             when state3 =>
173                s_led_out_R<='0';
174                s_led_out_B<='0';
175                s_led_out_V<=state_led;
176             if restart='0' then
177
178                     if(end_counter2='1') then
179                    next_state <= state1;                    --prochain etat
180
181                    else
182
183                    next_state <= current_state;
184                    end if;
185                    else
186                 next_state <= idle;
187                 end if;
188
189                --signaux pilotes par la fsm
190                when others =>
191
192              next_state <= idle;
193
194             end case;
195
196 end process;
197
198 --Partie combinatoire a completer avec les leds affectations de signaux
199  led_out_R<=s_led_out_R;
200  led_out_B<=s_led_out_B;
201  led_out_V<=s_led_out_V;
202
203 end behavioral;
```

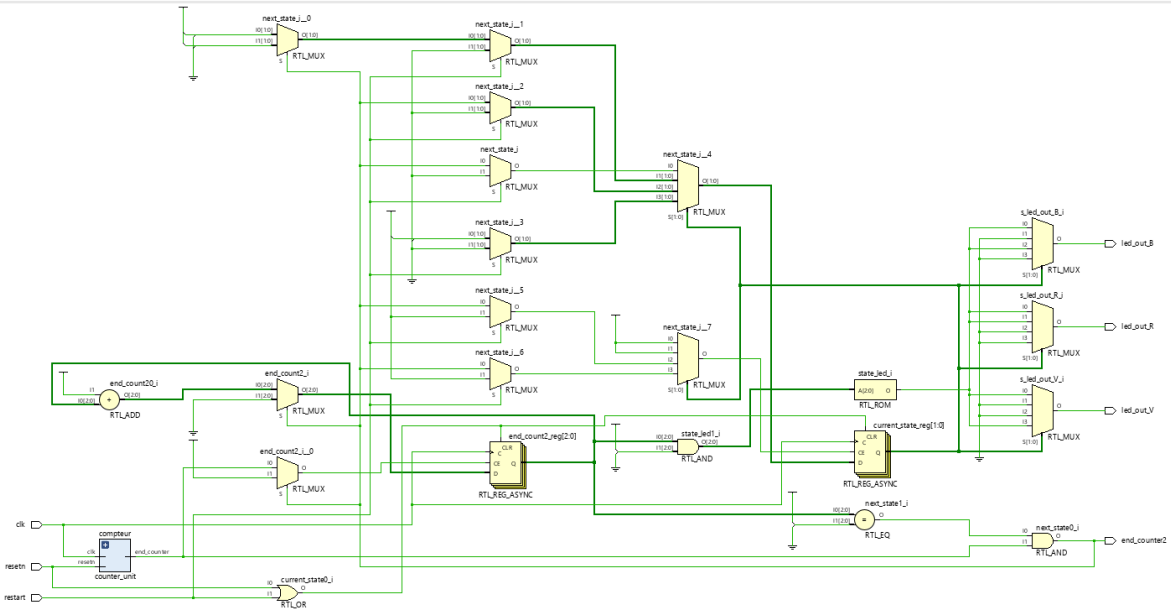
8. Ecrivez un testbench pour tester votre architecture. Vérifiez à la simulation que vous obtenez le résultat attendu.

```
207
208  library ieee;
209  use ieee.std_logic_1164.all;
210  use ieee.std_logic_unsigned.all;
211  use ieee.std_logic_arith;
212
213
214 entity tb_tp_fsm is
215  end tb_tp_fsm;
216
217 architecture behavioral of tb_tp_fsm is
218
219      --signaux de clk, resetn et restart
220     signal resetn      : std_logic := '0';
221     signal clk         : std_logic := '0';
222     signal restart          : std_logic := '0';
223
224     --Signaux de sorties
225     signal end_count2 : integer range 0 to 5:=0;
226     signal end_counter2 : std_logic:='0';
227      signal  s_led_out_B  : std_logic;
228     signal  s_led_out_V  : std_logic;
229     signal  s_led_out_R  : std_logic;
230
231     -- Les constantes suivantes permette de definir la frequence de l'horloge
232     constant hp : time := 5 ns;      --demi periode de 5ns
233     constant period : time := 2*hp;  --periode de 10ns, soit une frequence de 100Hz
234     constant nbre_cycle : positive := 6 ;
235 --  constant  max_count : integer :=200000000;
236 --    constant   nb_bit : integer := 28;
237      constant  max_count : integer :=4;
238       constant   nb_bit : integer := 3;
239
240      --Déclaration du composant fsm
241     component tp_fsm
242         port (
243             clk         : in std_logic;
244             resetn      : in std_logic;
245             restart     : in std_logic;
246             --a completer
247              end_counter2  : out std_logic;
248              led_out_V:  out std_logic;
249              led_out_R:  out std_logic;
250              led_out_B:  out std_logic
251          );
252     end component;
```

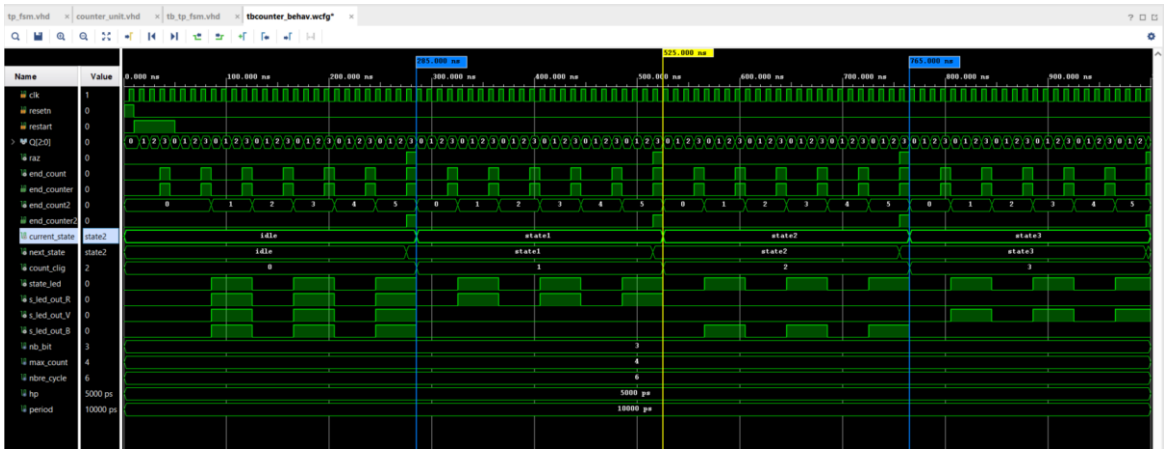
```
    begin
    dut: tp_fsm
        port map (
            clk => clk,
            resetn => resetn,
            restart => restart,
            --a completer
            end_counter2 => end_counter2,
            led_out_B => s_led_out_B ,
            led_out_V => s_led_out_V ,
            led_out_R  => s_led_out_R
        );

    --Simulation du signal d'horloge en continue
    process
    begin

        wait for hp;
        clk <= not clk;

    end process;

     --Simulation du signal restn en continue avec affichage de end_counter2 qui fonctionne après
     le nombre de cycle
    process
    begin

        resetn <= '1';
```

```
        --wait for hp;
        wait for period*1;
        resetn <= '0';
         assert end_counter2='0'
           report "end_counter2 : test failed";
         wait for period*2;
            --a completer
        wait for period*nbre_cycle*max_count;
        assert end_counter2='1'
          report "end_counter2 : test failed";
        wait;
    end process;
--simulation de leds et clignotement
process
    begin
   wait for period*1;
-- Restart
    restart <= '1';
    wait for 40 ns;
    restart <= '0';
    wait for period*nbre_cycle*max_count;

    -- Verifions l'état initial idle (LED blanche)
    assert (s_led_out_R = '1' and s_led_out_V = '1' and s_led_out_B = '1')
      report "Initial state mismatch" severity error;

    -- Wait for 3 clignotements en blanc (INITIAL state : idle)
    wait for period*nbre_cycle*max_count;

    -- Verifions RED state1
    assert (s_led_out_R = '1' and s_led_out_V = '0' and s_led_out_B = '0')
      report "RED state mismatch" severity error;

    -- Wait for 3 clignotements en rouge (RED state)
    wait for period*nbre_cycle*max_count;

    -- Verifions BLUE state2
    assert (s_led_out_R = '0' and s_led_out_V = '0' and s_led_out_B = '1')
      report "BLUE state mismatch" severity error;

    -- Wait for 3 clignotements en bleu (BLUE state)
    wait for period*nbre_cycle*max_count;

    -- Verifions GREEN state3
    assert (s_led_out_R = '0' and s_led_out_V = '1' and s_led_out_B = '0')
      report "GREEN state mismatch" severity error;

    -- Wait for 3 clignotements en vert (GREEN state)
    wait for period*nbre_cycle*max_count;

--Pour finir la simulation
wait;
end process;

end behavioral;
```

Le RTL de l'ensemble



Les résultats de simulations sur le chronogramme montrent que cela fonctionne bien.



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

Nous retrouvons dans la synthèse les états de nos leds. Ils sont stockés dans un registre de 2 bits.

```
INFO: [Synth 8-802] inferred FSM for state register 'current_state_reg' in module 'tp_fsm'
---------------------------------------------------------------------------------------------
                   State |                     New Encoding |               Previous Encoding
---------------------------------------------------------------------------------------------
                    idle |                               00 |                               00
                  state1 |                               01 |                               01
                  state2 |                               10 |                               10
                  state3 |                               11 |                               11
---------------------------------------------------------------------------------------------
INFO: [Synth 8-3354] encoded FSM with state register 'current_state_reg' using encoding 'sequential' in module 'tp_fsm'
```

La description du RTL dans la synthèse.

```
Start RTL Component Statistics
---------------------------------------------------------------------------------
Detailed RTL Component Info :
+---Adders :
      2 Input     3 Bit       Adders := 2
+---Registers :
                  3 Bit    Registers := 2
+---Muxes :
      2 Input     3 Bit        Muxes := 2
      4 Input     2 Bit        Muxes := 1
      2 Input     2 Bit        Muxes := 6
      2 Input     1 Bit        Muxes := 2
      4 Input     1 Bit        Muxes := 4
---------------------------------------------------------------------------------
Finished RTL Component Statistics
---------------------------------------------------------------------------------
```

```
Start Writing Synthesis Report
------------------------------

Report BlackBoxes:
+-+--------------+----------+
| |BlackBox name |Instances |
+-+--------------+----------+
+-+--------------+----------+

Report Cell Usage:
+------+-----+------+
|      |Cell |Count |
+------+-----+------+
|1     |BUFG |     1|
|2     |LUT1 |     1|
|3     |LUT2 |     4|
|4     |LUT3 |     2|
|5     |LUT4 |     2|
|6     |LUT5 |     3|
|7     |FDCE |     7|
|8     |IBUF |     3|
|9     |OBUF |     4|
```

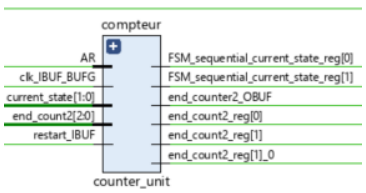
Tous les ressources sont répertoriées ci-dessus. Pour un générique de 4, on a 7 registres (deux pour la machine d'état, 3 pour le counter2 et 2 pour le counter_unit).

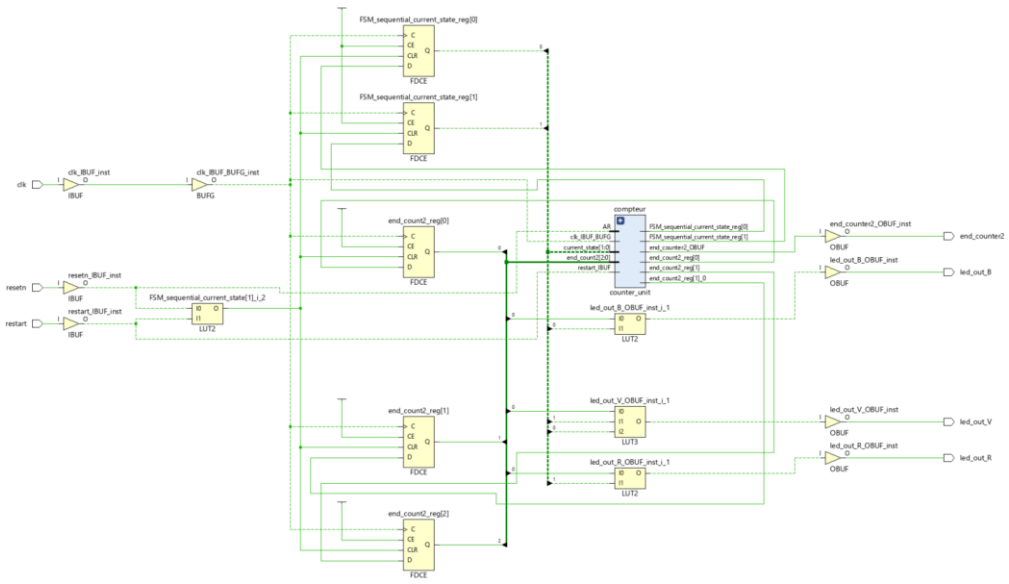
Lorsque le générique est de 200000000, on adonc 33 registres (deux pour la machine d'état, 3 pour le counter2 et 28 pour le counter_unit).

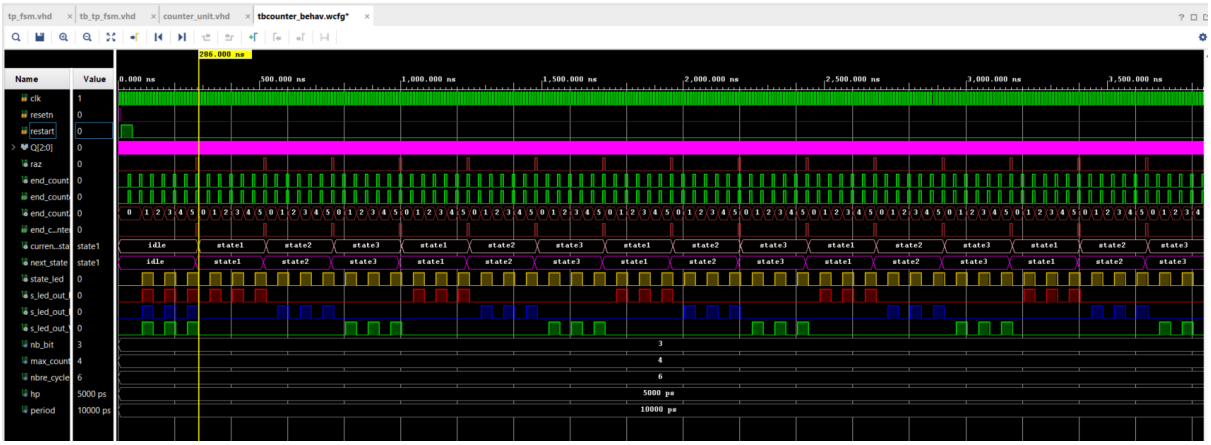
```
---------------------------------------------------------------------
Start RTL Component Statistics
---------------------------------------------------------------------
Detailed RTL Component Info :
+---Adders :
      2 Input     3 Bit       Adders := 1
+---Registers :
                  3 Bit    Registers := 1
+---Muxes :
      2 Input     3 Bit        Muxes := 1
      4 Input     2 Bit        Muxes := 1
      2 Input     2 Bit        Muxes := 6
      2 Input     1 Bit        Muxes := 2
      4 Input     1 Bit        Muxes := 4
---------------------------------------------------------------------
```

```
Report Cell Usage:
+------+-------+------+
|      |Cell   |Count |
+------+-------+------+
|1     |BUFG   |     1|
|2     |CARRY4 |     7|
|3     |LUT2   |    33|
|4     |LUT3   |     2|
|5     |LUT4   |     7|
|6     |LUT5   |     2|
|7     |LUT6   |     3|
|8     |FDCE   |    33|
|9     |IBUF   |     3|
|10    |OBUF   |     4|
+------+-------+------+
```

Dans la synthèse : voici le compteur de cycle. On retrouve autour de lui, les registres qui gère le clignotement et la FSM.







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.

```
## This file is a general .xdc for the Cora Z7-07S Rev. B
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project

# 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]

# RGB LEDs
set_property -dict {PACKAGE_PIN L15 IOSTANDARD LVCMOS33} [get_ports {led_out_R}]
set_property -dict {PACKAGE_PIN G17 IOSTANDARD LVCMOS33} [get_ports {led_out_B}]
set_property -dict {PACKAGE_PIN N15 IOSTANDARD LVCMOS33} [get_ports {led_out_V}]
#set_property -diot { 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 { end_counter2 }]; #IO_L22P_T3_AD7P_35 Sch=led1_g
#set_property -diot { 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 { resetn }]; #IO_L4N_T0_35 Sch=btn[0]
set_property -dict {PACKAGE_PIN D19 IOSTANDARD LVCMOS33} [get_ports {restart}]
```

La fréquence est réglée à 100MHZ. La schématique est représentée ci-dessous.

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

Le Clock

```
| Clock Summary
| -------------

Clock                                                                                  Waveform(ns)       Period(ns)      Frequency(MHz)
-----                                                                                  ------------       ----------      --------------
dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_inst/SERIES7_BSCAN.bscan_inst/TCK  {0.000 16.500}     33.000          30.303
sys_clk_pin                                                                            {0.000 5.000}      10.000          100.000
```

On vérifie bien que la période est à 10ns et la fréquence est de 100MHz

Les valeurs dans le THS et TNS sont à 0, il n'y a pas de violation du set up et du hold. Pas de métastabilité.

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

    WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints      WHS(ns)      THS(ns)  THS Failing Endpoints  THS Total Endpoints     WPWS(ns)     TPWS(ns)  TPWS Failing Endpoints  TPWS Total Endpoints
    -------      -------  ---------------------  -------------------      -------      -------  ---------------------  -------------------     --------     --------  ---------------------  --------------------
      4.026        0.000                      0                 3929        0.023        0.000                      0                 3913        3.750        0.000                      0                 2200


All user specified timing constraints are met.
```

Le chemin critique est :

Max Delay Paths

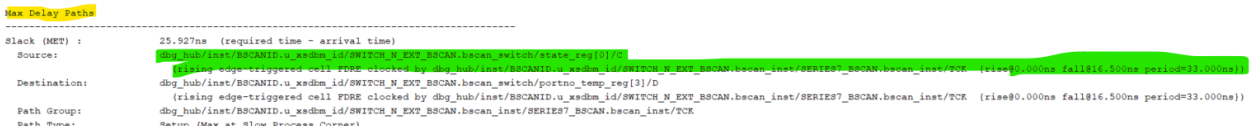
Slack (MET) : 25.927ns (required time - arrival time)

Source: dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_switch/state_reg[0]/C

(rising edge-triggered cell FDRE clocked by dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_inst/SERIES7_BSCAN.bscan_inst/TCK {rise@0.000ns fall@16.500ns period=33.000ns})

Destination: dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_switch/portno_temp_reg[3]/D

(rising edge-triggered cell FDRE clocked by dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_inst/SERIES7_BSCAN.bscan_inst/TCK {rise@0.000ns fall@16.500ns period=33.000ns})

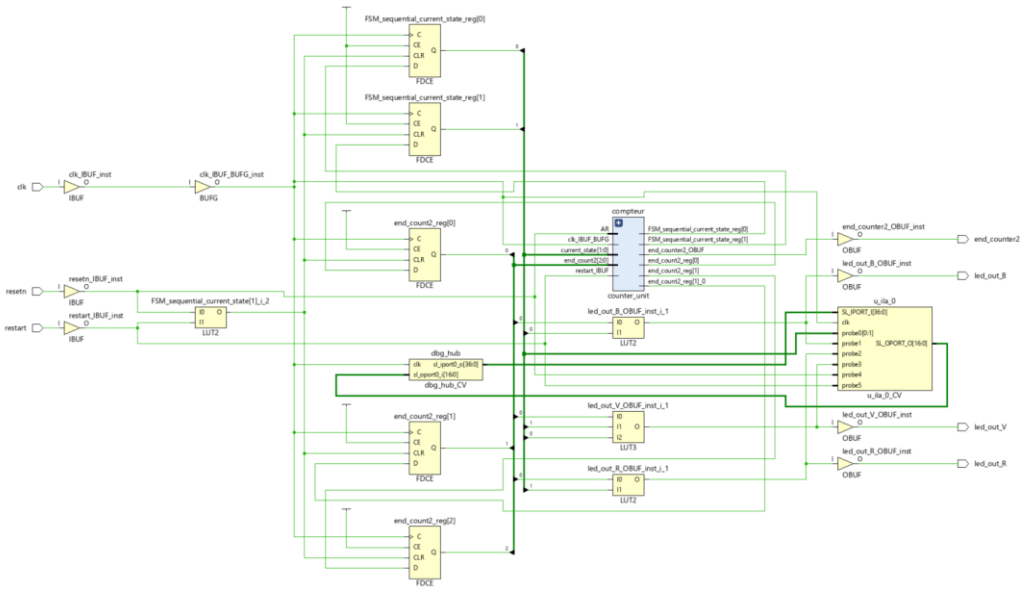


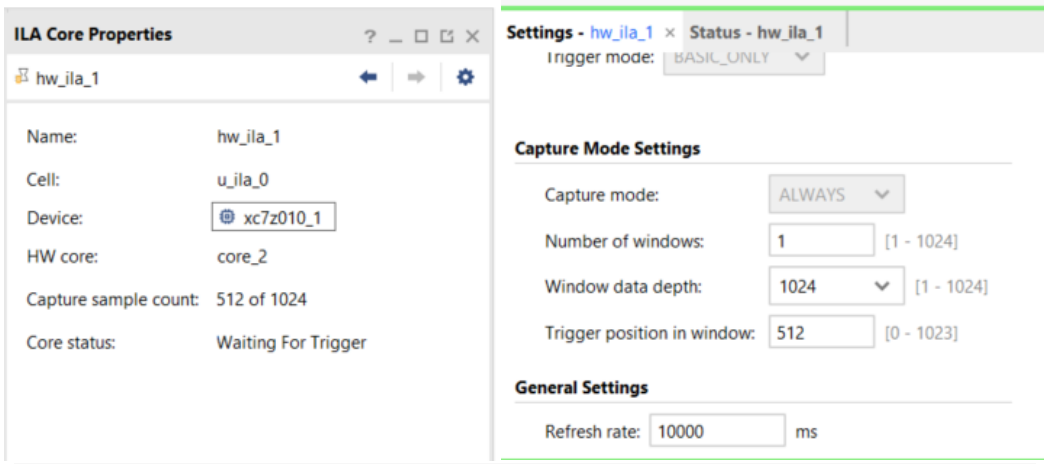
12. Générez le bitstream pour vérifier le système sur carte.

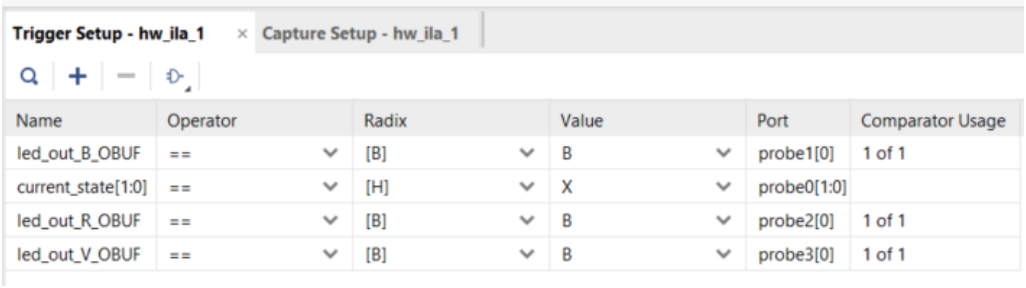
Les signaux leds_out, et current_state sont observés dans le trigger. Hw_ila_1

Pour rappel, la principale fonction d'une ILA est de surveiller et d'analyser les signaux numériques à l'intérieur d'un circuit intégré ou d'un FPGA (Field-Programmable Gate Array). Elle permet de détecter les erreurs, de valider le fonctionnement des circuits et de comprendre le comportement des signaux lors de l'exécution d'un programme ou d'une séquence d'opérations.

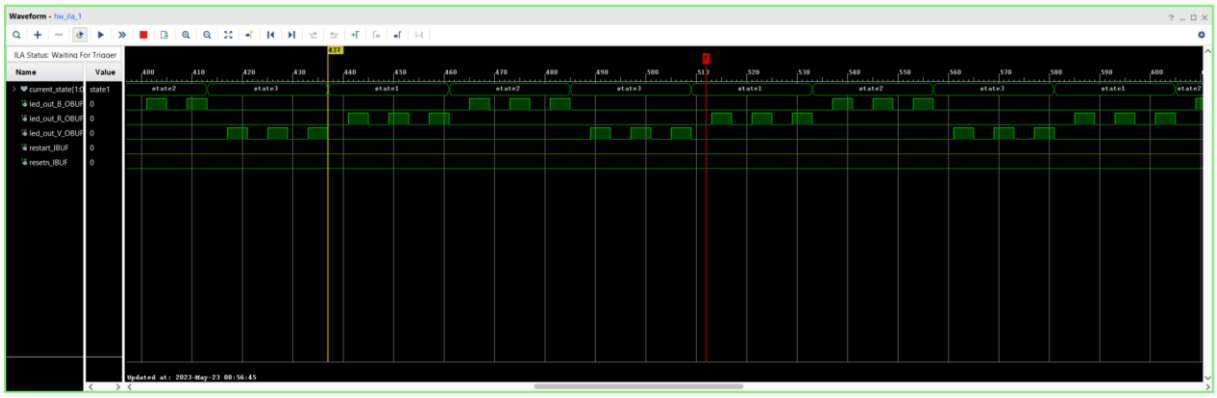
Nous démarrons l'enregistrement de l'ILA sur le front montant et descendant du port de sortie du current_state. On procède à un changement de valeur sur le signal de déclenchement entraînera l'ILA pour commencer à enregistrer les signaux sondés (Led_out RGB). Ceci est fait dans le déclencheur (trigger setup).

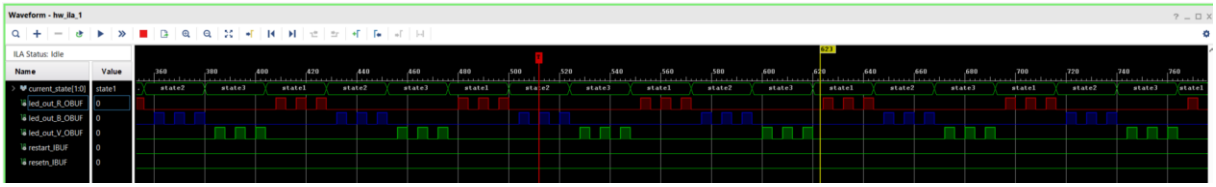






Nous voyons la ligne verticale rouge (marqueur) sur le front montant de notre signal de déclenchement (port trigger de led), et il est en position 512. Nous pouvons également vérifier que le signal compte se comporte correctement et change de couleur suivant les changements d'état.





Lorsque que le bouton restart est appuyé, on retourne dans l'état initial quel que soit l'état dans lequel on se situe.