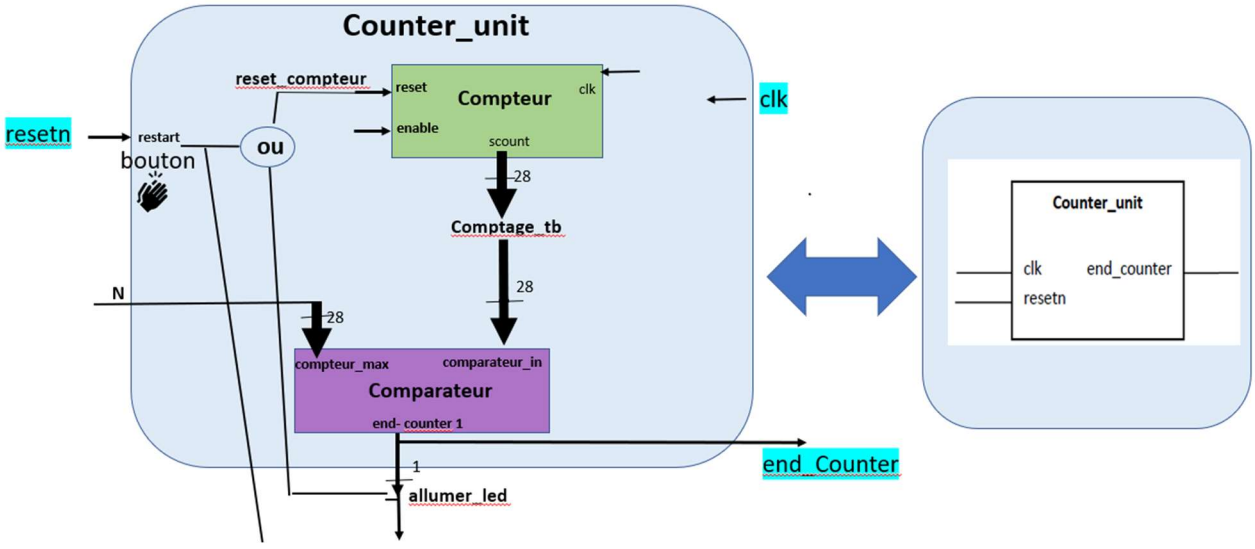
Compte rendu de – TP03 –Machine à états (FSM)

1.1 Objectif de ce TP

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) Question 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.

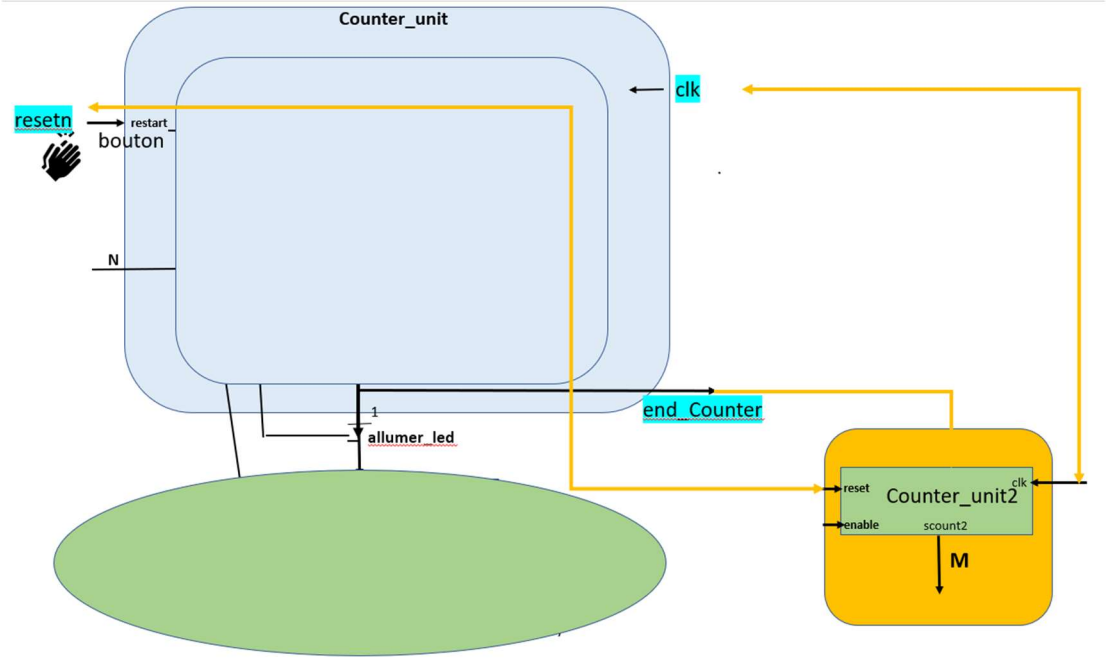


Le code de Counter_unit ne sera plus modifié ensuite

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.

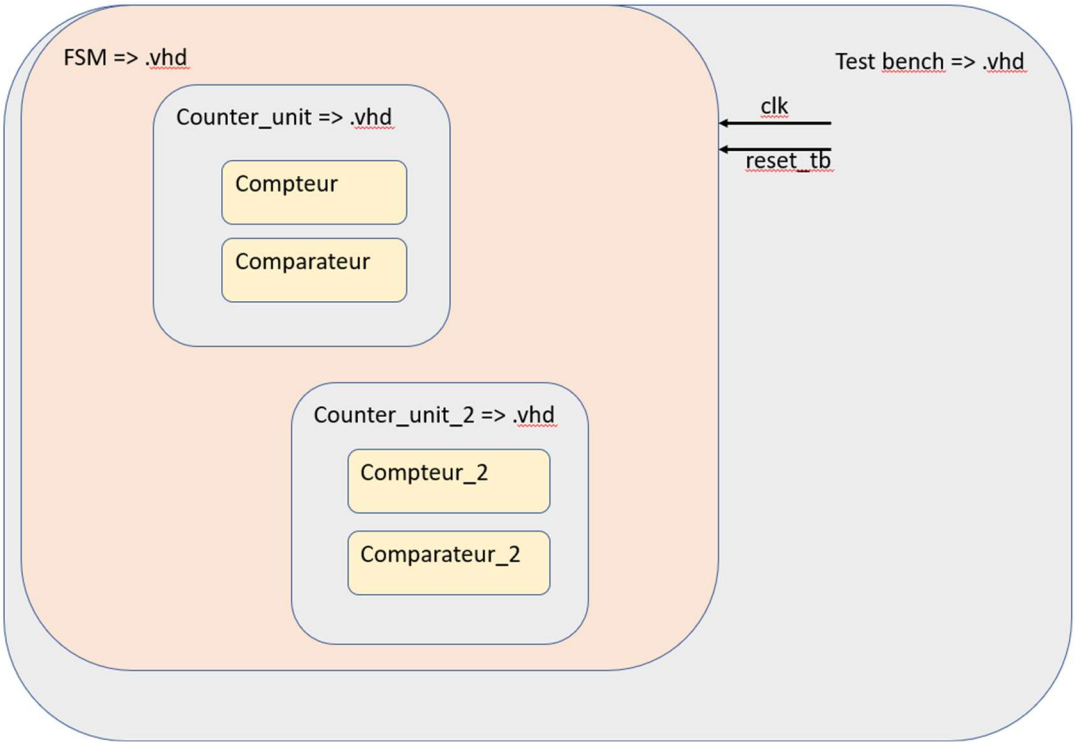
Je pars de notre ancien TP : je défini les limites de mon **COUNTER_UNIT** en bleu :

Compte rendu Eve CHAR du TP3 - 10/05/2023 – 12/05/2023

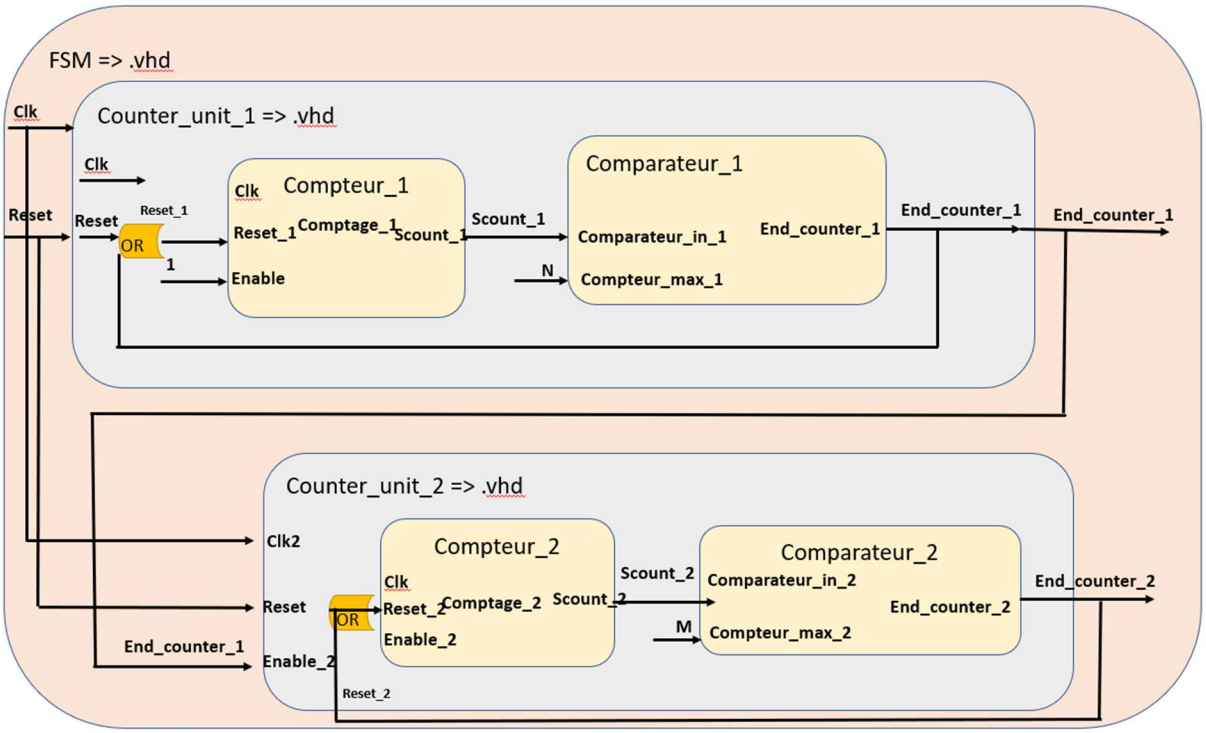


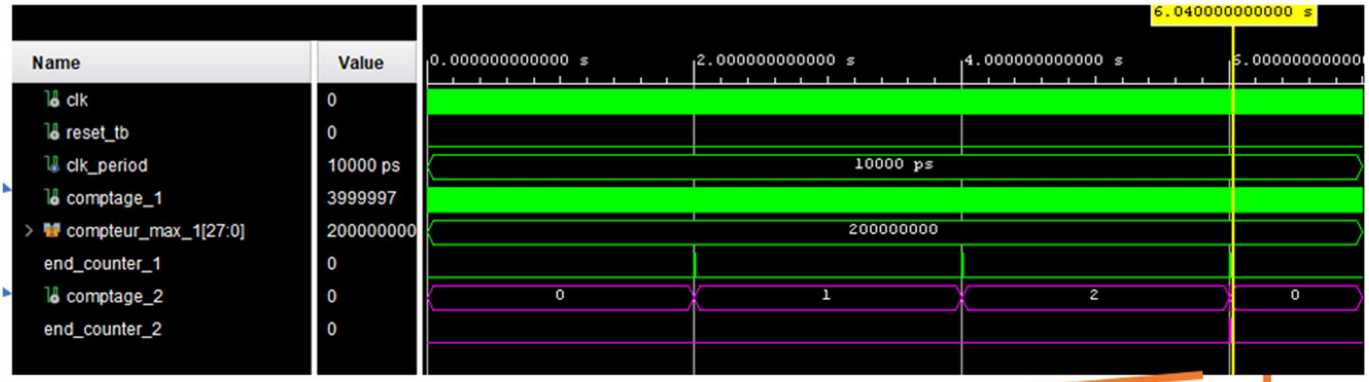
3. Ecrivez un code VHDL décrivant ce compteur de cycle, vous utiliserez le module Counter_unit.

4. Tester votre architecture avec un testbench.

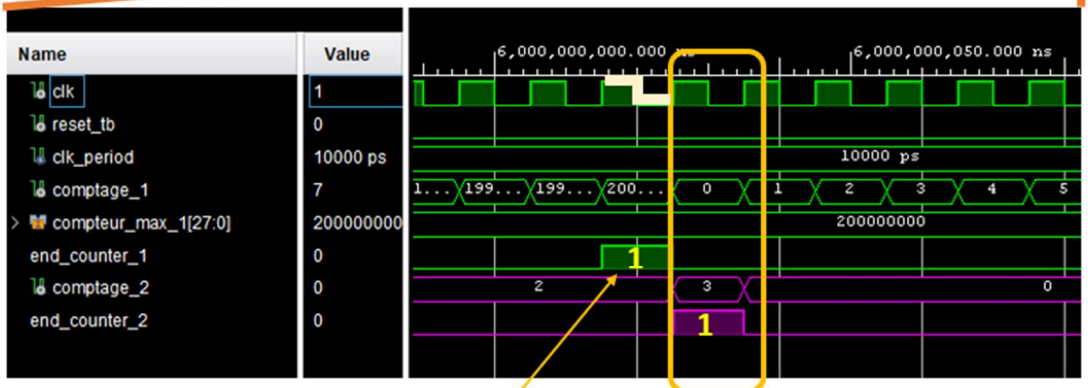


Compte rendu Eve CHAR du TP3 - 10/05/2023 – 12/05/2023





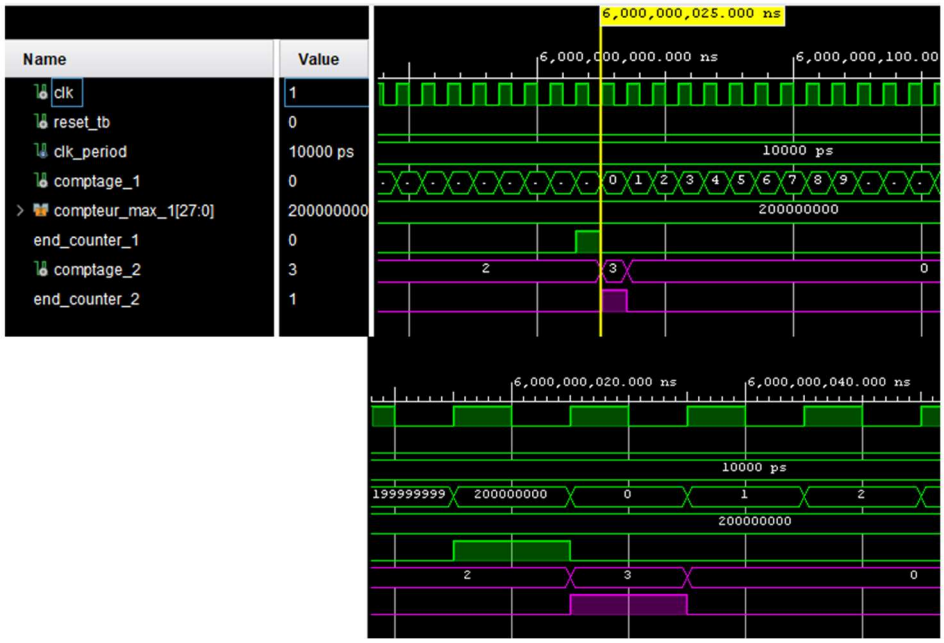
zoom



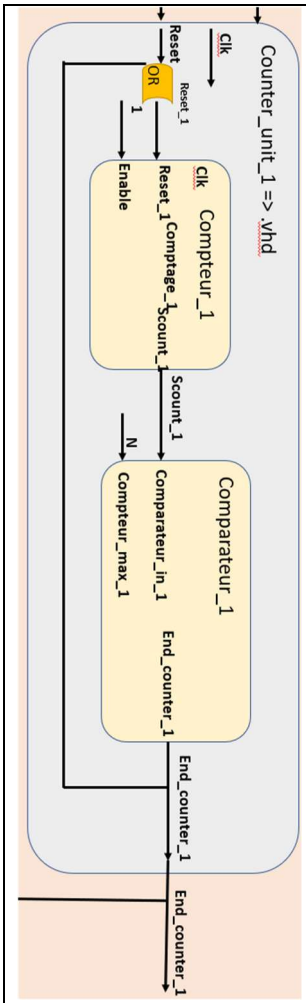
N= 200000000: mon comptage1 se reboucle

M= 3: mon comptage2 se reboucle

Compte rendu Eve CHAR du TP3 - 10/05/2023 – 12/05/2023

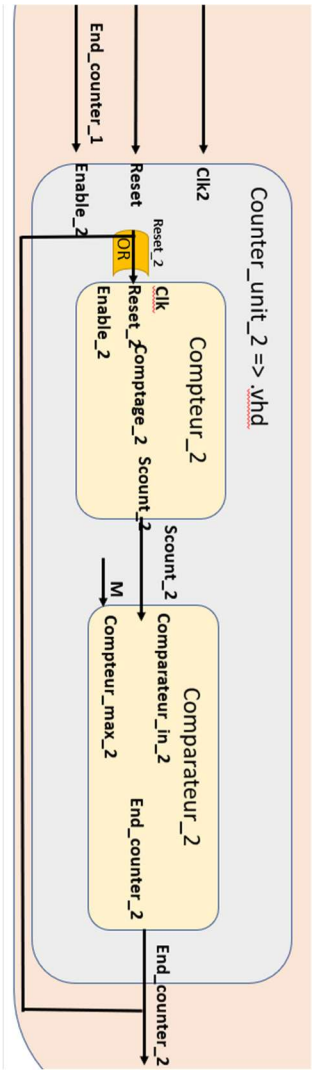


Définition de mon design source Counter_unit_1




```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
USE work.ALL;

entity Counter_unit_1 is
generic ( N : integer := 200000000); -- le nombre de coup d'horloge à compter
    Port ( clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           end_counter_1 : inout STD_LOGIC := '0');
end Counter_unit_1;
----Description comportementale de mon Counter_unit
architecture arch_Counter_unit_1 of Counter_unit_1 is

    signal scount_1 : std_logic_vector (27 downto 0);
    signal reset_1 : std_logic := '0';
begin
        reset_1 <=(end_counter_1 or reset);

    inst_compteur : entity compteur_1 port map(
        clk => clk,
        reset_1 => reset_1,
        enable_1 => '1',
        scount_1 => scount_1 );

    inst_comparateur : entity comparateur_1 port map(
        comparateur_in_1 => scount_1,
        compteur_max_1 => std_logic_vector(to_unsigned(N,28)),
        end_counter_1 => end_counter_1 );

end arch_Counter_unit_1;
```

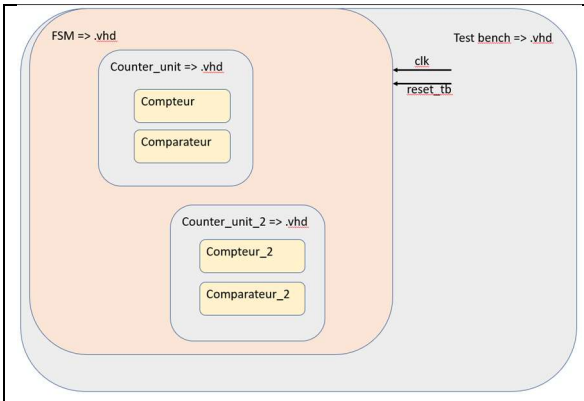
Définition de mon design source Counter_unit_2




```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
USE work.ALL;

entity Counter_unit_2 is
generic ( M : integer := 3);--255); -- le nombre de coup à compter
    Port ( clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           Enable_2 : in STD_LOGIC;
           end_counter_2 : inout STD_LOGIC := '0');
end Counter_unit_2;
----Description comportementale de mon Counter_unit
architecture arch_Counter_unit_2 of Counter_unit_2 is
  signal scount_2 : std_logic_vector (7 downto 0);
  signal reset_2 : std_logic := '0';
    --constant N : integer := 200000000;
begin
reset_2 <=(end_counter_2 or reset);
    inst_compteur_2 : entity compteur_2 port map(
           clk => clk,
           reset_2 => reset_2,
           enable_2 => Enable_2,
           scount_2 => scount_2 );

    inst_comparateur_2 : entity comparateur_2 port map(
           comparateur_in_2 => scount_2,
           compteur_max_2 => std_logic_vector(to_unsigned(M,8)),
           end_counter_2 => end_counter_2 );

end arch_Counter_unit_2;
```

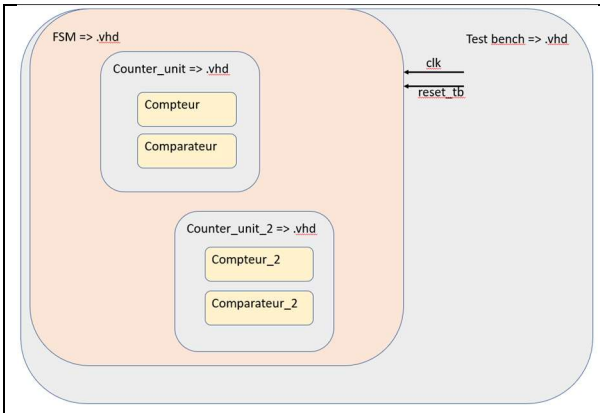
Compte rendu Eve CHAR du TP3 - 10/05/2023 – 12/05/2023

Définition de mon design source FSM_unit




```
-- Declaration de ma bibliothèque
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
USE work.ALL;
entity FSM_unit is
    Port ( clk : in STD_LOGIC;
           Reset : in STD_LOGIC);
          -- enable : in STD_LOGIC ;
           --scount2 : out integer);
end FSM_unit;
architecture arch_FSM_unit of FSM_unit is
--signal comptage2 : integer :=0;
signal end_counter_1 : STD_LOGIC := '0';
signal end_counter_2 : STD_LOGIC := '0';
--signal reset_compteur : std_logic := '0';
begin
    inst_Counter_unit_1 : entity Counter_unit_1 port map (
           clk => clk,
           Reset => Reset,
           end_counter_1 => end_counter_1);

    inst_Counter_unit_2 : entity Counter_unit_2 port map (
           clk => clk,
           Reset => Reset,
           enable_2 => end_counter_1,
           end_counter_2 => end_counter_2);

end arch_FSM_unit;
```

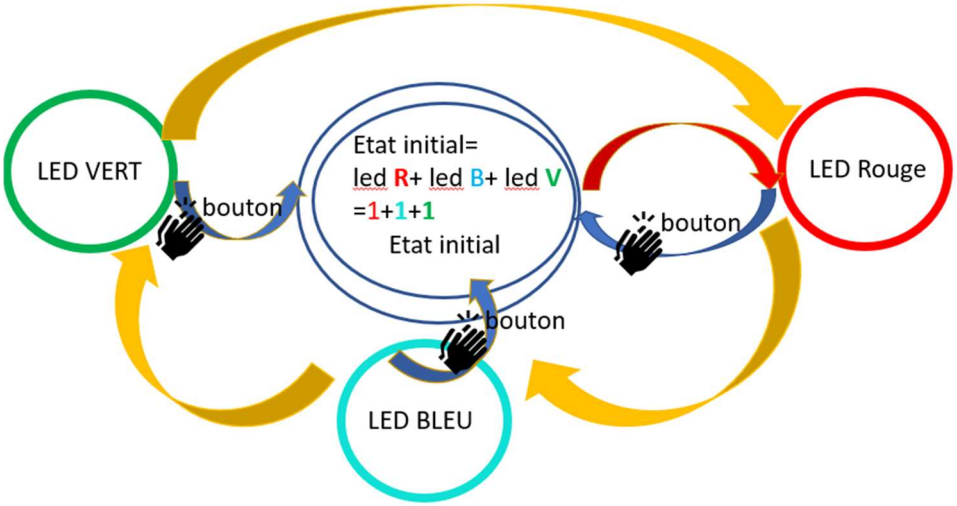
Définition de mon design source TEST-BENCH




```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
Use work .all;

entity FSM_tb is
end FSM_tb;

architecture arch_FSM_tb of FSM_tb is
   signal clk : std_logic ;
   --signal Resetn :std_logic ;
   signal reset_tb :std_logic :='0';
   --signal end_counter_unit_tb :  std_logic ;
    --signal end_counter : std_logic ;
    constant clk_period : time :=  10 ns;

begin
    uut_FSM_unit : entity FSM_unit Port map (
           clk => clk,
           Reset => reset_tb);

    ---Clock process definitions
    clk_process : process
    begin
    clk <= '0';
    wait for clk_period/2;
    clk <= '1';
    wait for clk_period/2;
    end process;

end arch_FSM_tb;
```

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). + Question6 +Question7

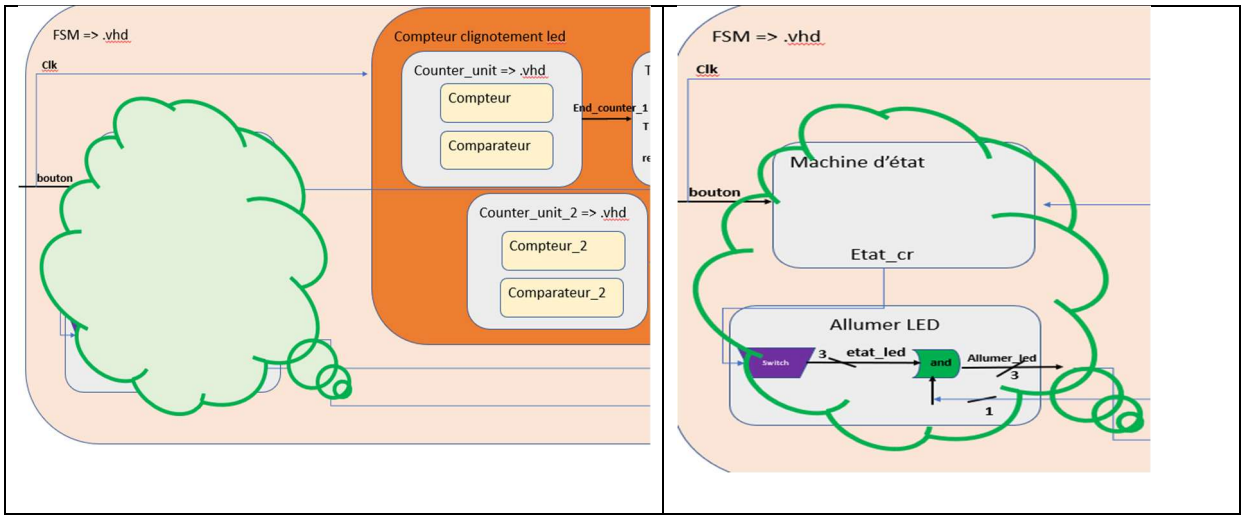
Ma machine à état est la suivante :



Pour coder ma machine à etat : je vais repartir le code sur 2 :

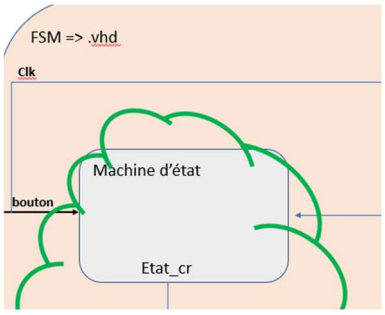
- une partie sur la description de la machine à etat en elle-même
- et une deuxième partie sur les actions appliquées à notre machine

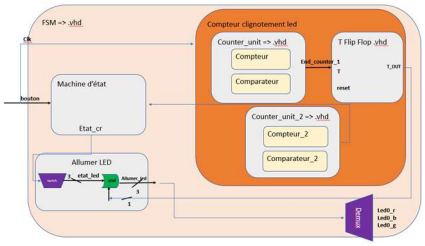
Avant tout je reviens à mon architecture et je dessine les entrées, les sorties, les signaux…



Compte rendu Eve CHAR du TP3 - 10/05/2023 – 12/05/2023

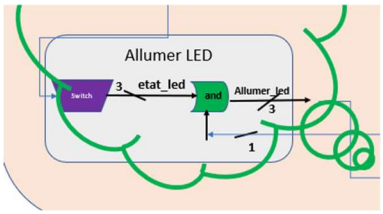
Définition de mon design source de la machine à état





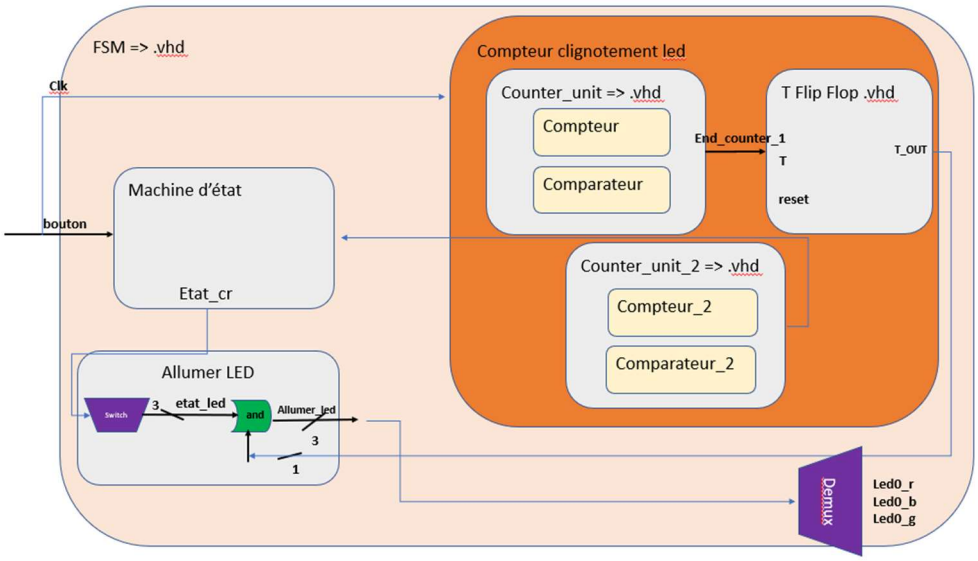

```
-- Initialisation de ma machine à etat--
process( clk , Bouton)
begin -- définition de l'initialisation de ma machine
      if Bouton='1' then etat_cr <= etat_initial;
      elsif rising_edge(clk) then etat_cr <= etat_sv;
      end if;
end process;

-- Définition de ma machine à etat--
process(etat_cr, end_counter_2)
begin
--initialisation des etats:
etat_sv <= etat_cr;
        case etat_cr is
            when etat_initial =>
                if end_counter_2 = '1' then
                    etat_sv <= led_rouge;
                end if;
            when led_rouge =>
                if end_counter_2 = '1' then
                    etat_sv <= led_bleu;
                end if;
            when led_bleu =>
                if end_counter_2 = '1' then
                    etat_sv <= led_vert;
                end if;

            when led_vert =>
                if end_counter_2 = '1' then
                    etat_sv <= led_rouge;
                end if;
        end case;
end process;
```

Définition de mon design source de la machine à état partie Actions appliquées




```
process(etat_cr, bouton, end_counter_1)--state_switch
begin
case etat_cr is
    when etat_initial =>
        etat_led <= v_led_blanc;        -- blanc => rouge + bleu + vert
    when led_rouge =>
        etat_led <= v_led_rouge; -- rouge
    when led_bleu =>
        etat_led <= v_led_bleu; -- bleu
    when led_vert =>
        etat_led <= v_led_vert; -- vert
    end case;
end process;

process (clk,etat_led)
begin
if rising_edge(clk) then
    allumer_led <= etat_led and (t_out, t_out, t_out);
    led0_r <= allumer_led(0);
    led0_b <= allumer_led(1);
    led0_g <= allumer_led(2);
end if;
end process;

end arch_FSM_unit;
```

Architecture finale (à la main) de mon FSM GLOBAL :



Pour ma deuxième partie de FSM j'ai besoin d' un autre élément à notre architecture (du tp precedent) →pour maintenir la led allumée 2 sec et eteinte 2 sec en mémorisant la dernière entrée, et ce pendant N valeur de comptage de front mentant ➔ ce composant est le T_FLIP-FLOP

Instanciation de mon FSM et les différents composants de mon architecture :	

```
entity FSM_unit is
    Port ( clk : in STD_LOGIC;
           Bouton : in STD_LOGIC;
           led0_r, led0_b, led0_g : out STD_LOGIC);
          -- enable : in STD_LOGIC ;
           --scount2 : out integer);
end FSM_unit;

architecture arch_FSM_unit of FSM_unit is

    -- Definition des signaux
    signal end_counter_1 : STD_LOGIC := '0';
    signal end_counter_2 : STD_LOGIC := '0';
    type state is (etat_initial, led_rouge, led_bleu,led_vert);
    signal etat_cr, etat_sv : state ; --etat dans lequel on se trouve
    signal etat_led : std_logic_vector(2 downto 0);
    signal allumer_led : std_logic_vector(2 downto 0);
    constant v_led_rouge :std_logic_vector := "001";
    constant v_led_bleu :std_logic_vector := "010";
    constant v_led_vert :std_logic_vector := "100";
    constant v_led_blanc :std_logic_vector := "111";
    signal t_out : std_logic;

begin

    inst_Counter_unit_1 : entity Counter_unit_1 port map (
           clk => clk,
           Reset => Bouton,
           end_counter_1 => end_counter_1);

    inst_Counter_unit_2 : entity Counter_unit_2 port map (
           clk => clk,
           Reset => Bouton,
           enable_2 => end_counter_1,
           end_counter_2 => end_counter_2);

    inst_Bascule_T_Flip_Flops : entity Bascule_T_Flip_Flops port map(
          clk => clk,
          reset => bouton,
          T => end_counter_1,
          t_out => t_out);
```

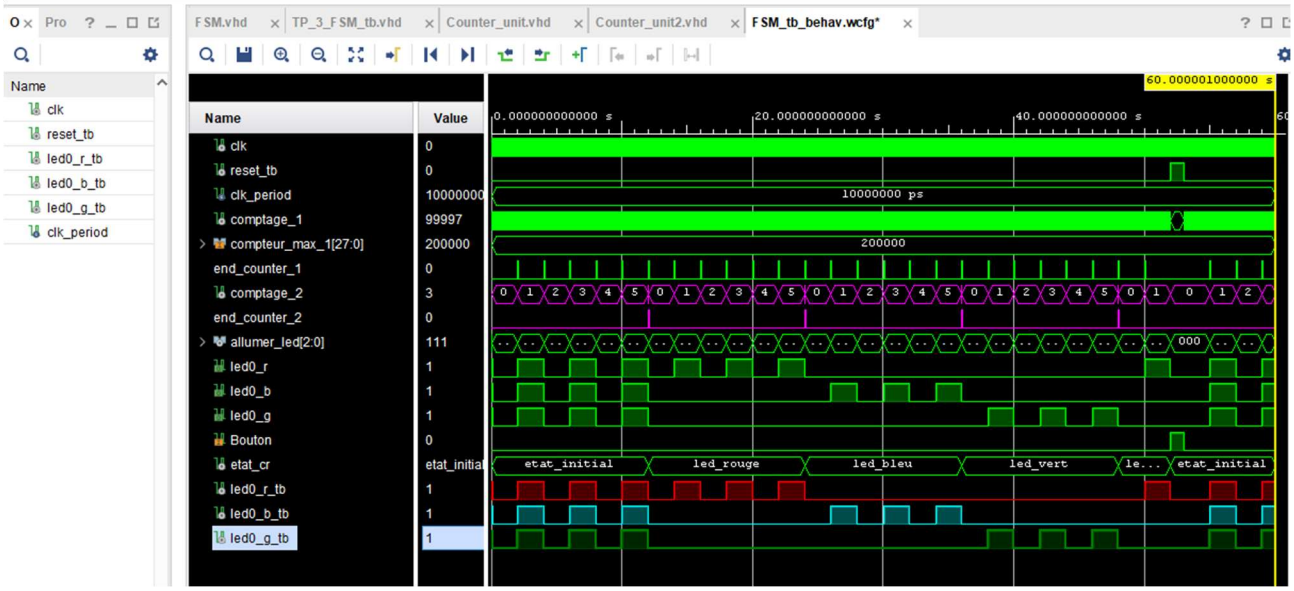
Compte rendu Eve CHAR du TP3 - 10/05/2023 – 12/05/2023

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

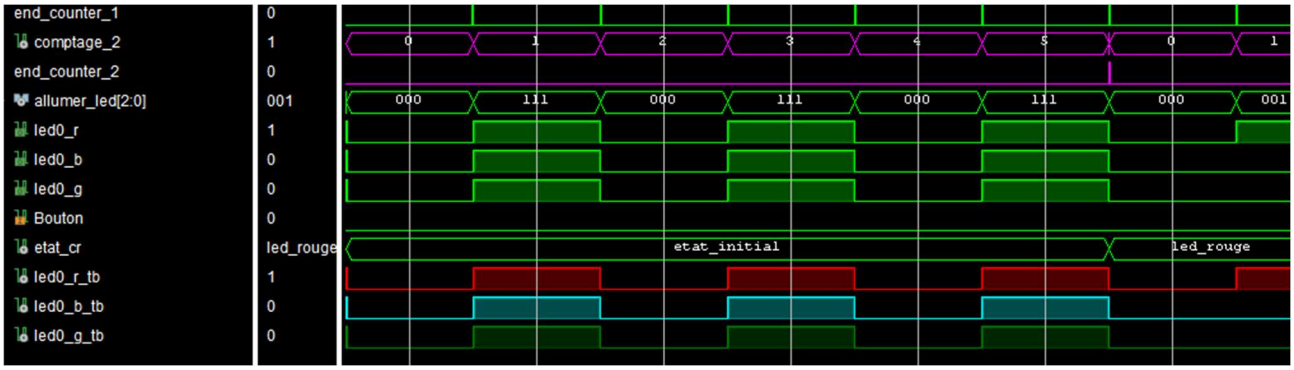
Test bench de mon FSM

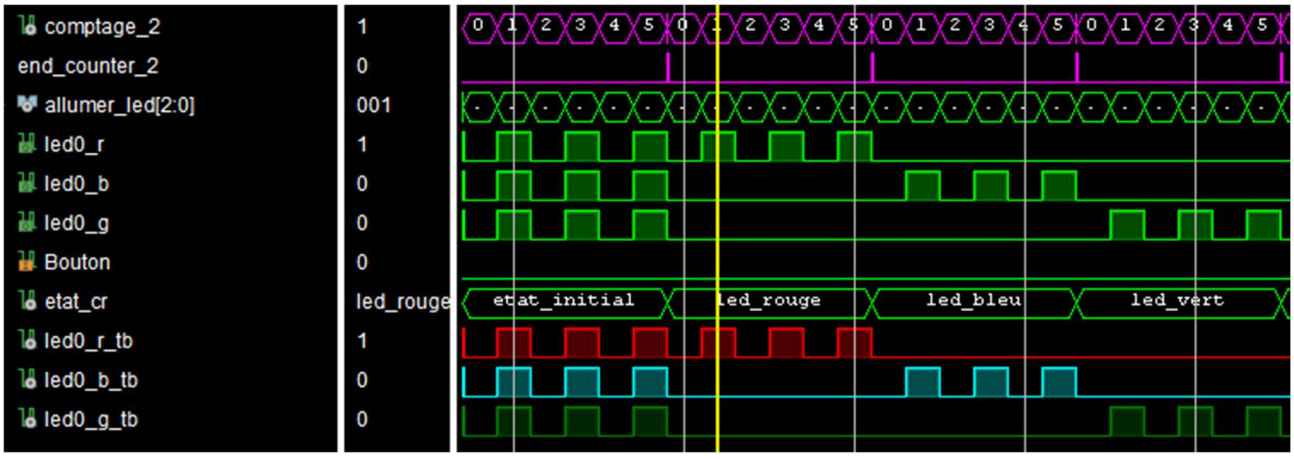
(pour des raison de temps de simulation j'ai écourté le nombre de N (200000 au lieu de 200 000 000) et la période T à 10ms : seulement pour obtenir cette photo ci_dessous

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
Use work .all;

entity FSM_tb is
end FSM_tb;

architecture arch_FSM_tb of FSM_tb is
   signal clk : std_logic ;
   --signal Resetn :std_logic ;
   signal reset_tb :std_logic :='0';
   signal led0_r_tb : std_logic :='0';
   signal led0_b_tb : std_logic :='0';
   signal led0_g_tb : std_logic :='0';
   --signal end_counter_unit_tb :  std_logic ;
    --signal end_counter : std_logic ;
    constant clk_period : time :=  10 ns;--10000 ns;--

begin
    uut_FSM_unit : entity FSM_unit Port map (
           clk => clk,
           Bouton => reset_tb,
           led0_r => led0_r_tb,
           led0_b => led0_b_tb,
           led0_g => led0_g_tb);

    ---Clock process definitions
    clk_process : process
    begin
    clk <= '0';
    wait for clk_period/2;
    clk <= '1';
    wait for clk_period/2;
    end process;
```

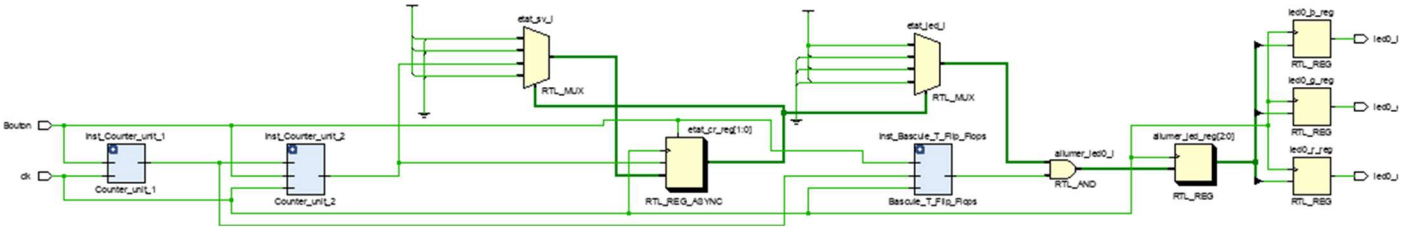


Compte rendu Eve CHAR du TP3 - 10/05/2023 – 12/05/2023



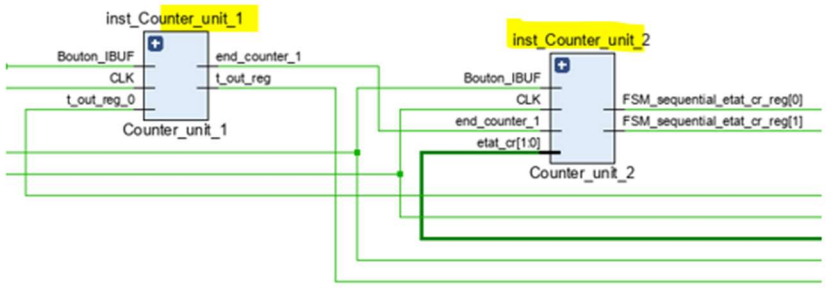


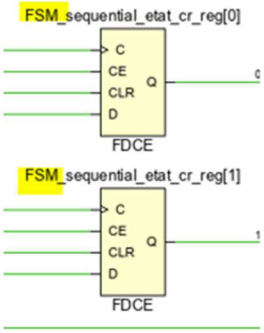
1.2 Schématique :



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

1.3 Rapport de synthèse






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

```
Report Cell Usage:
+------+-------+------+
|      |Cell   |Count |
+------+-------+------+
|1     |BUFG   |     1|
|2     |CARRY4 |     9|
|3     |LUT1   |     2|
|4     |LUT2   |     2|
|5     |LUT3   |     1|
|6     |LUT4   |     5|
|7     |LUT5   |     1|
|8     |LUT6   |     7|
|9     |FDCE   |     3|
|10    |FDRE   |    42|
|11    |IBUF   |     2|
|12    |OBUF   |     3|
+------+-------+------+
```

```
Report Instance Areas:
+------+-----------------------+---------------------+------+
|      |Instance               |Module               |Cells |
+------+-----------------------+---------------------+------+
|1     |top                    |                     |    78|
|2     |  inst_Bascule_T_Flip_Flops |Bascule_T_Flip_Flops |     4|
|3     |  inst_Counter_unit_1  |Counter_unit_1       |    44|
|4     |    inst_compteur      |compteur_1           |    44|
|5     |  inst_Counter_unit_2  |Counter_unit_2       |    16|
|6     |    inst_compteur_2    |compteur_2           |    16|
+------+-----------------------+---------------------+------+
```

Question10. 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.

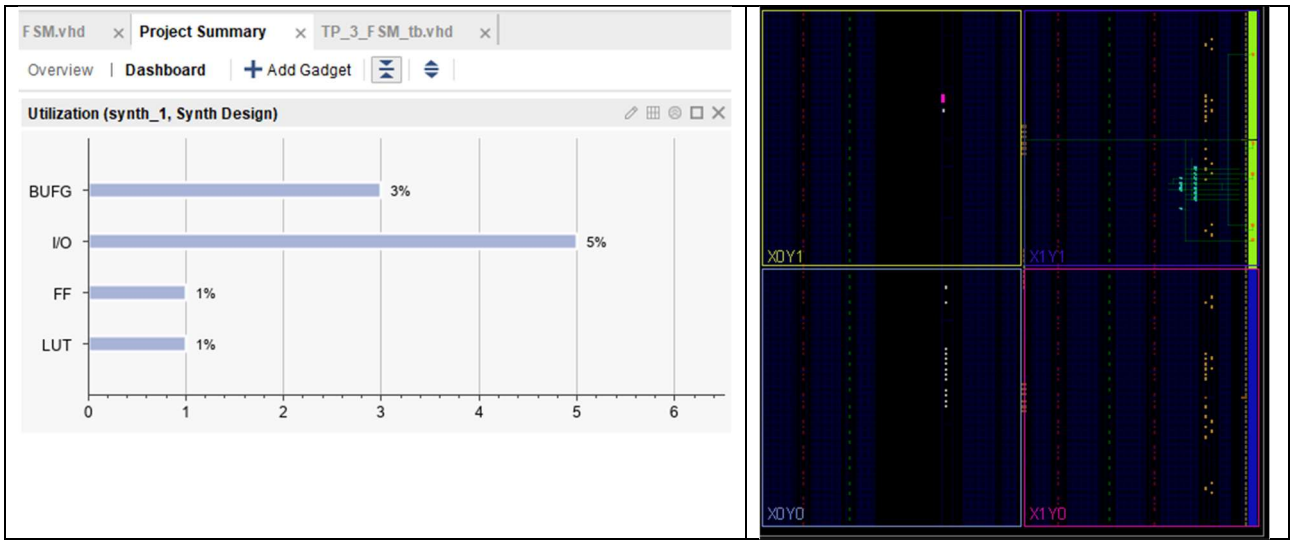
```
## This file is a general .xdc for the Cora Z7-07S Rev. B
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal

# 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 PACKAGE_PIN L15 [get_ports led0_b]
set_property IOSTANDARD LVCMOS33 [get_ports led0_b]
#set_property -dict {PACKAGE_PIN L15 IOSTANDARD LVCMOS33} [get_ports led0_b]
#set_property -dict {PACKAGE_PIN L15 IOSTANDARD LVCMOS33} [get_ports allumer_led]
set_property -dict {PACKAGE_PIN G17 IOSTANDARD LVCMOS33} [get_ports led0_g]
set_property -dict {PACKAGE_PIN N15 IOSTANDARD LVCMOS33} [get_ports led0_r]
#set_property -dict {PACKAGE_PIN G14 IOSTANDARD LVCMOS33} [get_ports led1_b]
#set_property -dict {PACKAGE_PIN L14 IOSTANDARD LVCMOS33} [get_ports led1_g]
#set_property -dict {PACKAGE_PIN M15 IOSTANDARD LVCMOS33} [get_ports led1_r]

# Buttons

set_property IOSTANDARD LVCMOS33 [get_ports Bouton]
set_property PACKAGE_PIN D20 [get_ports Bouton]
#set_property -dict {PACKAGE_PIN D20 IOSTANDARD LVCMOS33} [get_ports btn0]
#set_property -dict {PACKAGE_PIN D19 IOSTANDARD LVCMOS33} [get_ports {btn[1]}]
```

Question 11. Lancez limplémentation 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)
    -------      -------  ---------------------  -------------------      -------      -------
      5.684        0.000                      0                   89        0.122        0.000


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

Clock        Waveform(ns)       Period(ns)      Frequency(MHz)
-----        ------------       ----------      --------------
sys_clk_pin  {0.000 5.000}      10.000          100.000

------------------------------------------------------------------------------------------------
From Clock:  sys_clk_pin
  To Clock:  sys_clk_pin

Setup :            0  Failing Endpoints,  Worst Slack        5.684ns,  Total Violation        0.000ns
Hold  :            0  Failing Endpoints,  Worst Slack        0.122ns,  Total Violation        0.000ns
PW    :            0  Failing Endpoints,  Worst Slack        4.500ns,  Total Violation        0.000ns
------------------------------------------------------------------------------------------------
```

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

Annexe



Code :

```
-------------------------------------------------------------------------
----------------------------partie Actions appliquées de la machine à etat
-------------------------------------------------------------------------
-- on va rajouter un autre élément à notre architecture (du tp precedent)
---pour maintenir la led allumée 2 sec et eteinte 2 sec en mémorisant la dernière
--- entrée, et ce pendant N valeur de comptage de front mentant
    --**********************************************************
--##--declaration composant : Bascule_T_flip_flop
-- Declaration de ma bibliothèque
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Bascule_T_Flip_Flops is

    Port ( clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           --enable : in STD_LOGIC;
                                                                T : in STD_LOGIC;
           t_out : inout STD_LOGIC :='0');

end Bascule_T_flip_flops;

--Description comportementale
-----------------------------------------------
architecture arch_Bascule_T_Flip_Flops of Bascule_T_Flip_Flops is
```

```
begin

  process(Reset,clk)
      begin
      if reset='1' then t_out <= '0';
            elsif rising_edge(clk) then
         if T ='1' then
            if t_out ='1' then
                t_out <= '0';
                    else
               t_out <='1';

            end if;
                                                                        end if;
                                                                    end if;
 end process;
 end arch_Bascule_T_flip_flops;

-------------------------------------FSM--------------------------------
-------------------------------------------------------------------------------------------------------
-- Declaration de ma bibliothèque
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
USE work.ALL;
entity FSM_unit is
   Port ( clk : in STD_LOGIC;
          Bouton : in STD_LOGIC;
          led0_r, led0_b, led0_g : out STD_LOGIC);
         -- enable : in STD_LOGIC ;
          --scount2 : out integer);
end FSM_unit;

architecture arch_FSM_unit of FSM_unit is

   -- Definition des signaux
   signal end_counter_1 : STD_LOGIC := '0';
   signal end_counter_2 : STD_LOGIC := '0';
   type state is (etat_initial, led_rouge, led_bleu,led_vert);
   signal etat_cr, etat_sv : state ; --etat dans lequel on se trouve actuellement & etat dans lequel on
passera au prochain coup d'horloge
   signal etat_led : std_logic_vector(2 downto 0);
   signal allumer_led : std_logic_vector(2 downto 0);
   constant v_led_rouge :std_logic_vector := "001";
   constant v_led_bleu :std_logic_vector := "010";
   constant v_led_vert :std_logic_vector := "100";
   constant v_led_blanc :std_logic_vector := "111";
```

```
   signal t_out : std_logic;

begin

                                                                    inst_Counter_unit_1 : entity
Counter_unit_1 port map (
                                                                        clk => clk,
        Reset => Bouton,
        end_counter_1 => end_counter_1);

   inst_Counter_unit_2 : entity Counter_unit_2 port map (
                                                                        clk => clk,
        Reset => Bouton,
        enable_2 => end_counter_1,
        end_counter_2 => end_counter_2);

   inst_Bascule_T_Flip_Flops : entity Bascule_T_Flip_Flops port map(
       clk => clk,
       reset => bouton,
       T => end_counter_1,
                                                                      t_out => t_out);

-- Initialisation de ma machine à etat--
process( clk , Bouton)
begin -- définition de l'initialisation de ma machine à etat-- remise à zero
     if Bouton='1' then etat_cr <= etat_initial;
     elsif rising_edge(clk) then etat_cr <= etat_sv;
     end if;
end process;

-- Définition de ma machine à etat--
process(etat_cr, end_counter_2)
begin
--initialisation des etats:
etat_sv <= etat_cr;
       case etat_cr is
          when etat_initial =>
             if end_counter_2 = '1' then
                                                                            etat_sv
<= led_rouge;
                                                                        end if;
                                                                   when led_rouge =>
             if end_counter_2 = '1' then
                                                                            etat_sv
<= led_bleu;
                                                                        end if;
                                                                   when led_bleu =>
```

```
            if end_counter_2 = '1' then
                                                                    etat_sv
<= led_vert;
                                                                end if;

          when led_vert =>
            if end_counter_2 = '1' then
                                                                    etat_sv
<= led_rouge;
                                                                end if;
                                                            end case;
end process;

-------------------------------------------------------------------
-------------------------------------------------------------------
----------------------------partie Actions appliquées de la machine à etat
-------------------------------------------------------------------
--signal etat_led : std_logic_vector(2 down to 0);
--signal allumer_led : std_logic_vector(2 down to 0);
--constant v_led_rouge :std_logic_vector := "001";
--constant v_led_bleu :std_logic_vector := "010";
--constant v_led_vert :std_logic_vector := "100";
--constant v_led_blanc :std_logic_vector := "111";

process(etat_cr, bouton, end_counter_1)--state_switch
begin
case etat_cr is
                                                        when etat_initial =>
                                                            etat_led <= v_led_blanc;
                                                        -- blanc => rouge + bleu +
vert
                                                        when led_rouge =>
                                                            etat_led <= v_led_rouge;
-- rouge
                                                        when led_bleu =>
                                                            etat_led <= v_led_bleu; -
- bleu
                                                        when led_vert =>
                                                            etat_led <= v_led_vert; -
- vert
                                                        end case;
end process;

process (clk,etat_led)
begin
if rising_edge(clk) then
   allumer_led <= etat_led and (t_out, t_out, t_out);
```

```
    led0_r <= allumer_led(0);
    led0_b <= allumer_led(1);
    led0_g <= allumer_led(2);
end if;
end process;

end arch_FSM_unit;
```
