TP3 compteurs
Jean Baptiste NARI

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.

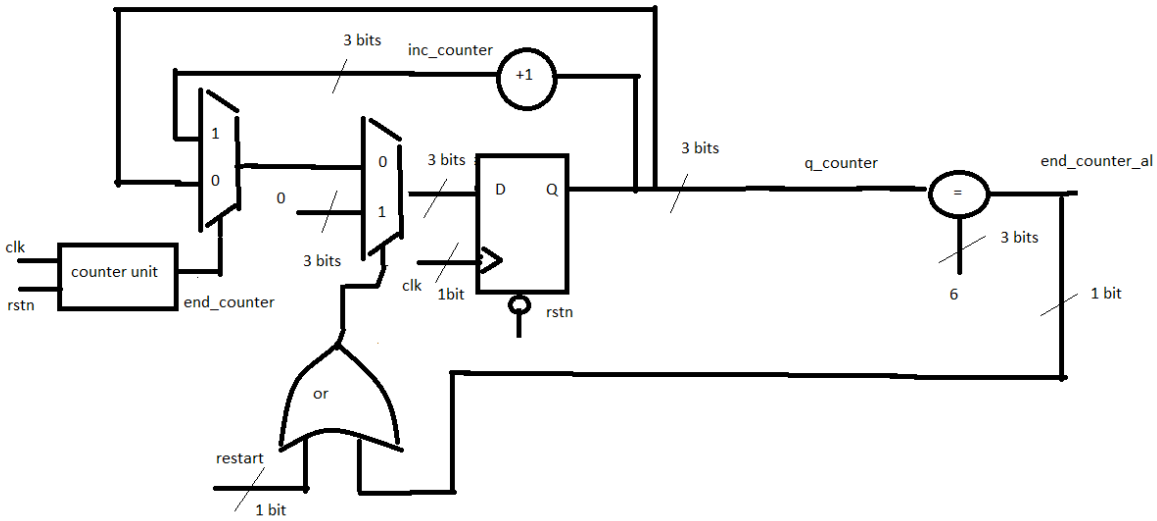
Question 1

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

entity counter_unit is
        generic (
        cte : positive := 20 ;
        );
   port (
                clk : in std_logic;
                rstn : in std_logic;
                end_counter : out std_logic
    );
end counter_unit;

architecture behavioral of counter_unit is

        --Declaration des signaux internes

   --constant cte : positive := 20;

        signal Q : std_logic_vector(27 downto 0);
        signal end_counter_int : std_logic;



begin
process(clk, rstn)
begin
        if (rstn = '1') then
                Q <= (others => '0');

        elsif (rising_edge(clk)) then

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

                else

                    Q <= Q + 1;
```

```
            end if;

        end if;
end process;



        end_counter_int <= '1' when Q =  cte - 1
                           else '0';

        end_counter <= end_counter_int;

end behavioral;
```

Question 2

Schéma RTL permettant de compter le nombre de cycles allumés/eteints.



Choix de comptage du nombre de changements d'état, il faudra comparer la valeur à 6 pour detecter 3 cycles allumés/éteints.

Question 3

Description vhdl du compteur de cycles :

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


entity counter_al is
   generic (
         cte : positive := 6
         );
   port (
                  clk : in std_logic;
                  rstn : in std_logic;
                  restart : in std_logic;
                  end_counter_al : out std_logic;
                  end_counter_2 : out std_logic
    );
end counter_al;

architecture behavioral of counter_al is

         --Declaration des signaux internes
         signal Q : std_logic_vector(2 downto 0);
         signal end_counter : std_logic;
         signal end_counter_int_al : std_logic;

         component counter_unit
                  port (
                           clk                  : in std_logic;
                           rstn          : in std_logic;
                           end_counter : out std_logic
                    );
         end component;


begin

counter_unit_comp: counter_unit
       port map (
          clk => clk,
          rstn=>rstn,
          end_counter => end_counter
       );

process(clk, rstn)
begin
         if (rstn = '1') then
```

```
                Q <= (others => '0');


        elsif (rising_edge(clk)) then

                if (restart='1' or end_counter_int_al='1') then
                        Q <= (others => '0');

                else
                        if (end_counter ='1') then
                            Q <= Q + 1;
                        else
                            Q <= Q;
                        end if;
                end if;



        end if;
end process;

        end_counter_int_al <= '1' when (Q =  cte - 1) and (end_counter ='1')
                                else '0';

        end_counter_al <= end_counter_int_al;

   end_counter_2 <= end_counter;
end behavioral;
```

Description vhdl du counter unit :

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

entity counter_unit is
        generic (
        cte : positive := 20
        );
   port (
                clk : in std_logic;
                rstn : in std_logic;
                end_counter : out std_logic
    );
end counter_unit;

architecture behavioral of counter_unit is
```

```
        --Declaration des signaux internes

    --constant cte : positive := 20;

        signal Q : std_logic_vector(27 downto 0);
        signal end_counter_int : std_logic;



begin
process(clk, rstn)
begin
        if (rstn = '1') then
                Q <= (others => '0');

        elsif (rising_edge(clk)) then

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

                else

                     Q <= Q + 1;
                end if;

        end if;
end process;




        end_counter_int <= '1' when Q =  cte - 1
                                else '0';

        end_counter <= end_counter_int;

end behavioral;
```

Question 4

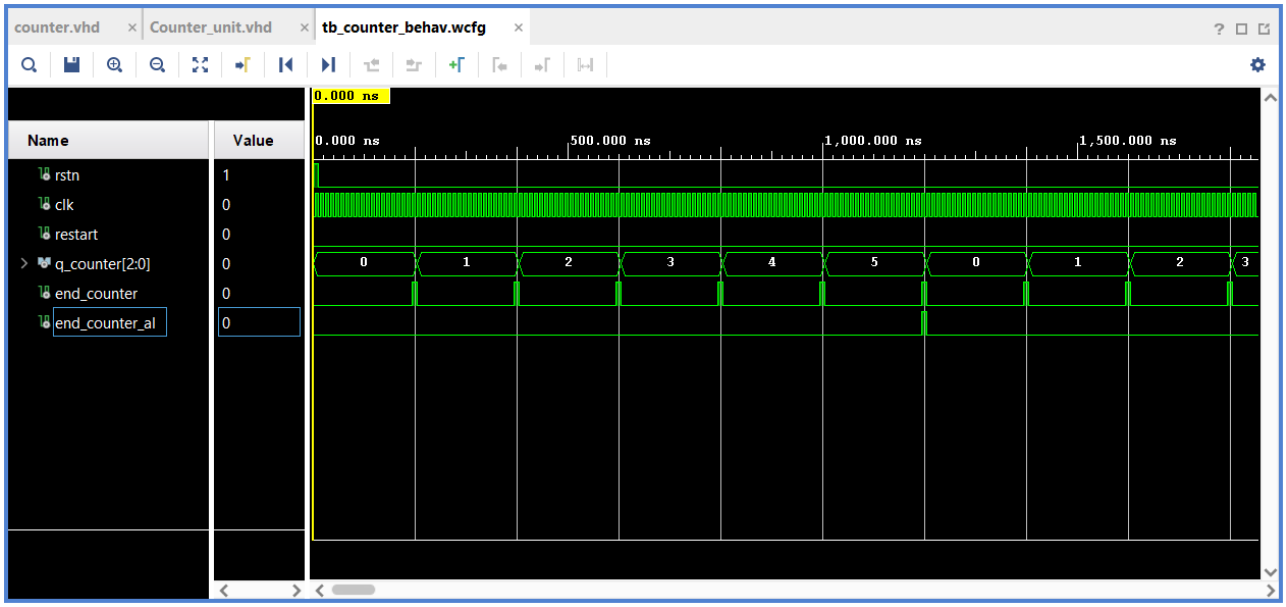
Description vhdl du test bench :

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

entity tb_counter_v1 is
end tb_counter_v1;
```

```
architecture behavioral of tb_counter_v1 is

        signal rstn      : std_logic := '0';
        signal clk         : std_logic := '0';
        signal restart         : std_logic := '0';
        signal end_counter_al : std_logic;


        -- 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 long_time : time := 2000 ms;

        --Declaration de l'entite a tester
        component counter_al
                port (
                        clk : in std_logic;
                   rstn : in std_logic;
                   restart : in std_logic;
                   end_counter_al : out std_logic
                 );
        end component;



        begin

        --Affectation des signaux du testbench avec ceux de l'entite a tester
        u_counter_al: counter_al
      port map (
        clk => clk,
        rstn=>rstn,
        restart=>restart,
        end_counter_al => end_counter_al
      );

        --Simulation du signal d'horloge en continue
        process
    begin
                wait for hp;
                clk <= not clk;
        end process;


        process
        begin

          -- TESTS A EFFECTUER
          rstn <= '1';
          wait for 10ns;
          rstn <= '0';
```

```
        wait for 10000ns;

    end process;

end behavioral;
```

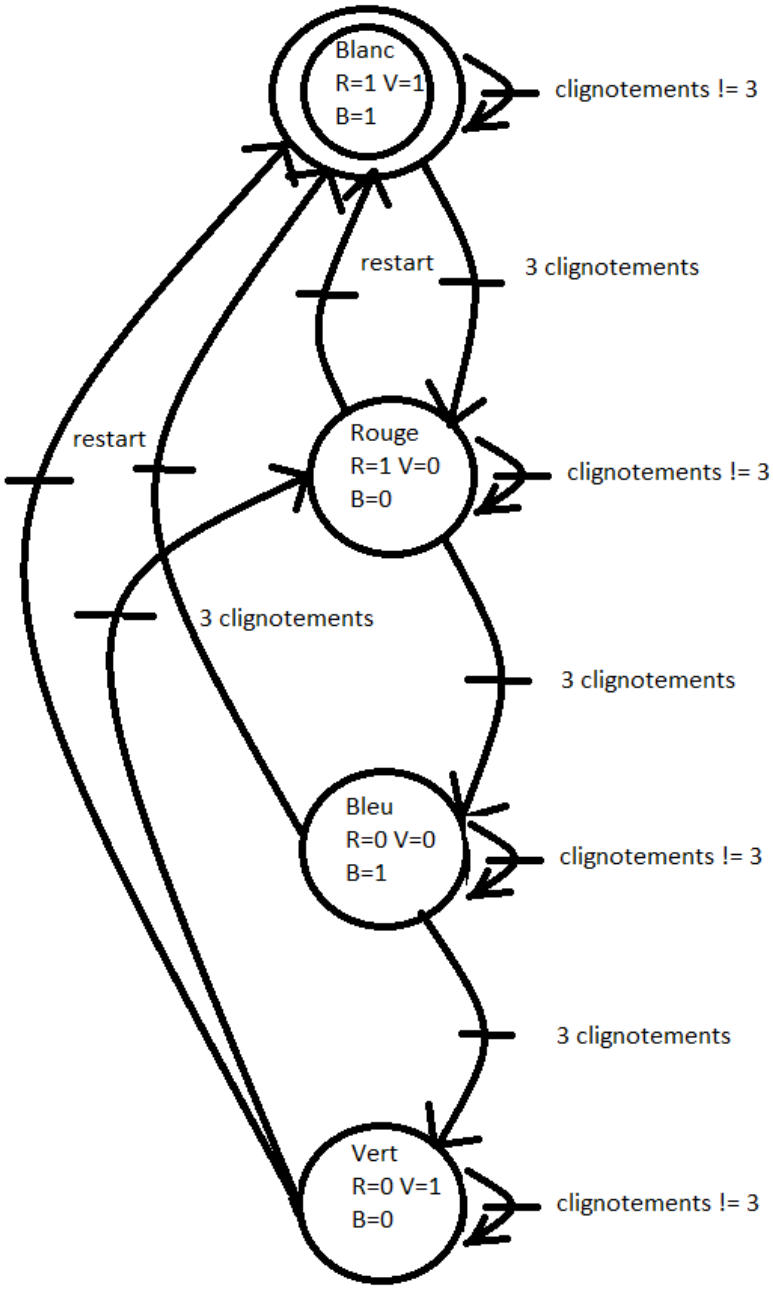
Résultat de simulation du compteur réalisé avec un compteur unit.



Le compteur compte bien jusqu'a 6 (de 0 à 5) puis le signal end_counter_al passe à l'état 1 et le comptage repard de 0 pour un nouveau cycle.

Question 5

Représentation de la machine à états



Question 6

Signaux d'entrées : rstn , restart, clk
Signaux internes : end_counter_al,end_counter_2 ,r_int,v_int, b_int, Q, led_on
Signaux sorties : R , V , B

Question 7

```
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
--    );
    port (
                clk                         : in std_logic;
        resetn              : in std_logic;
        restart             : in std_logic;
        r           : out std_logic;
        v           : out std_logic;
        b           : out std_logic
                    --a completer
    );
end tp_fsm;

architecture behavioral of tp_fsm is

    type state is (idle_blanc, rouge, bleu, 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_al : std_logic;
     signal end_counter_2 : std_logic;

     signal r_int: std_logic;
     signal v_int: std_logic;
     signal b_int: std_logic;

     signal Q: std_logic;
     signal led_on:std_logic;


component counter_al
                    port (
                    clk : in std_logic;
```

```
                rstn : in std_logic;
                restart : in std_logic;
                end_counter_al : out std_logic;
                end_counter_2 : out std_logic
                );
        end component;


        begin
counter_al_comp: counter_al
      port map (
        clk => clk,
        rstn=>resetn,
        restart => restart,
                  end_counter_al => end_counter_al,
                  end_counter_2 => end_counter_2
      );




process(clk, resetn)
begin
        if (resetn = '1') then

                Q <= '1';


        elsif (rising_edge(clk)) then


                 if(end_counter_2='1') then
        Q <= not Q;

                  else
                   Q <= Q;

                  end if;


        end if;
end process;

      led_on<=Q;

                r <= r_int when led_on='1'
                else '0';

                v <= v_int when led_on='1'
                else '0';

                b <= b_int when led_on='1'
```

```
            else '0';

--v <= not (v_int) when end_counter_2 = '1';



process(clk,resetn)
            begin
   if(resetn='1') then

      current_state <= idle_blanc;

                  elsif(rising_edge(clk)) then

                        current_state <= next_state;

                        --a completer avec votre compteur de cycles


   end if;
            end process;



            -- FSM
            process(current_state,end_counter_al) --a completer avec vos signaux
            begin
  case current_state is
    when idle_blanc  =>
       r_int <= '1';
       v_int <= '1';
       b_int <= '1';

       if (end_counter_al = '1') then
            next_state <= rouge; --prochain etat

       else
            next_state<=idle_blanc;

       end if;


       --signaux pilotes par la fsm

    when rouge =>
                        r_int <= '1';
       v_int <= '0';
       b_int <= '0';

                        if (end_counter_al = '1') then
```

```
            next_state <= bleu; --prochain etat

        else
            next_state<=rouge;

        end if;
       --signaux pilotes par la fsm

    when bleu =>
                        r_int <= '0';
        v_int <= '0';
        b_int <= '1';

                            if (end_counter_al = '1') then
            next_state <= vert; --prochain etat

        else
            next_state<=bleu;

        end if;

     when vert =>
                        r_int <= '0';
        v_int <= '1';
        b_int <= '0';

                            if (end_counter_al = '1') then
            next_state <= rouge; --prochain etat

        else
            next_state<=vert;

        end if;

    end case;


            end process;




end behavioral;
```

Detail sur la structure des composants

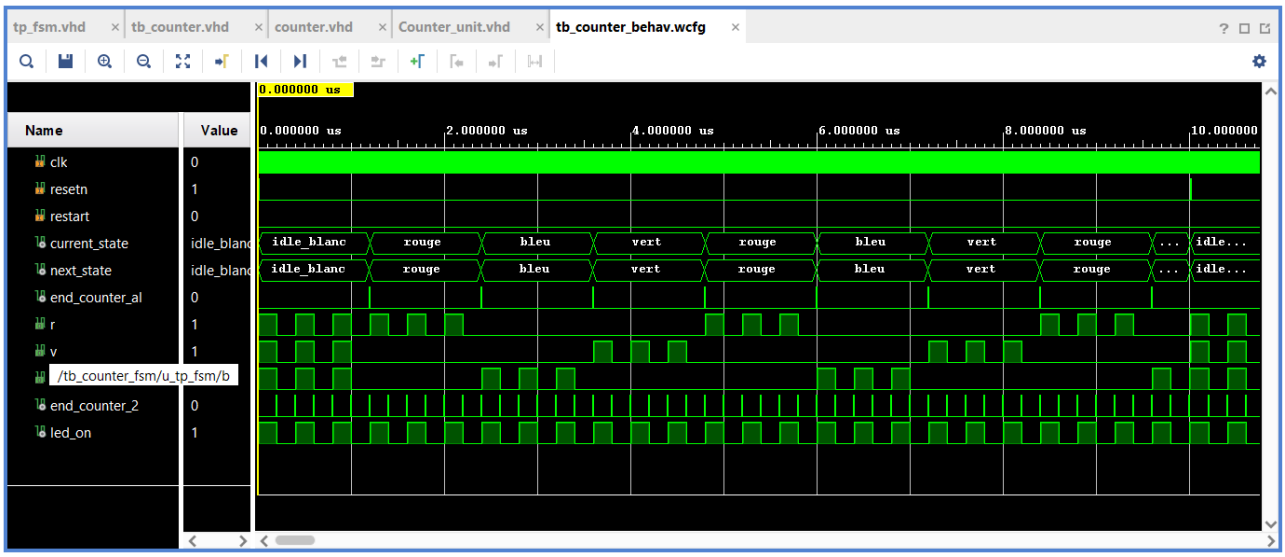
La sortie end_counter du module counter_unit passe à travers mon module counter_al (qui utilise le module counter_unit) et est présente sur sa sortie (nomé end_counter_2).
Le module counter_al est utilisé par le composant tp_fsm. C'est dans ce dernier qu'il est nécéssaire d'avoir le signal end_counter_2, afin de créer le signal led_on. Le signal led_on indique si la led est allumée (état haut) ou éteinte (état bas) cette information permet de faire clignoter la led.

Question 8

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



entity tb_counter_fsm is
end tb_counter_fsm;


architecture behavioral of tb_counter_fsm is

        signal resetn      : std_logic := '0';
        signal clk         : std_logic := '0';
        signal restart         : std_logic := '0';
        signal r           :  std_logic;
    signal v           :  std_logic;
    signal b           :  std_logic;

        -- 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 u_time : time :=1.25 us;
        --constant long_time : time := 2000 ms;

        --Declaration de l'entite a tester
        component  tp_fsm
                port (
                        clk : in std_logic;
                   resetn : in std_logic;
                   restart : in std_logic;
                   r          : out std_logic;
        v          : out std_logic;
        b          : out std_logic
                   );
        end component;



        begin

        --Affectation des signaux du testbench avec ceux de l'entite a tester
        u_tp_fsm: tp_fsm
      port map (
        clk => clk,
        resetn=>resetn,
        restart=>restart,
        r => r,
        v => v,
        b => b
      );
```

```
        --Simulation du signal d'horloge en continue
        process
    begin
                wait for hp;
                clk <= not clk;
        end process;


        process
        begin

          -- TESTS A EFFECTUER
          resetn <= '1';
          wait for 10ns;
          resetn <= '0';

          wait for period;
          assert r='1' report "test fail" severity failure;
          assert v='1' report "test fail" severity failure;
          assert b='1' report "test fail" severity failure;

          wait for u_time;
          assert r='1' report "test failr1" severity failure;
          assert v='0' report "test fail" severity failure;
          assert b='0' report "test fail" severity failure;

          wait for u_time;
          assert r='0' report "test fail" severity failure;
          assert v='0' report "test fail" severity failure;
          assert b='1' report "test fail" severity failure;

          wait for u_time;
          assert r='0' report "test fail" severity failure;
          assert v='1' report "test fail" severity failure;
          assert b='0' report "test fail" severity failure;




          wait for 10000ns;


        end process;


end behavioral;
```

Resultat de simulation:



La machine à états passe de l'état idle_blanc à rouge puis bleu puis vert.
Les sorties rvb correspondent bien aux états:

idle_blanc r=1 v=1 b=1
rouge r =1 v,b =0
vert v=1 r,b=0
bleu b=1 r,v=0

Question 9

Rapport de synthèse :

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

Dans cette sous partie du rapport un registre de 28 bits n'apparait pas.
Alors qu'il apparait dans la partie suivante.
Peut être que l'outil réalise un traitement particulier pour les registres de grande taille (28bits).

Report Cell Usage:

	Cell	Count
1	BUFG	1
2	CARRY4	7
3	LUT1	2
4	LUT2	30
5	LUT3	1
6	LUT4	4
7	LUT5	2
8	LUT6	5
9	FDCE	33
10	IBUF	3
11	OBUF	3

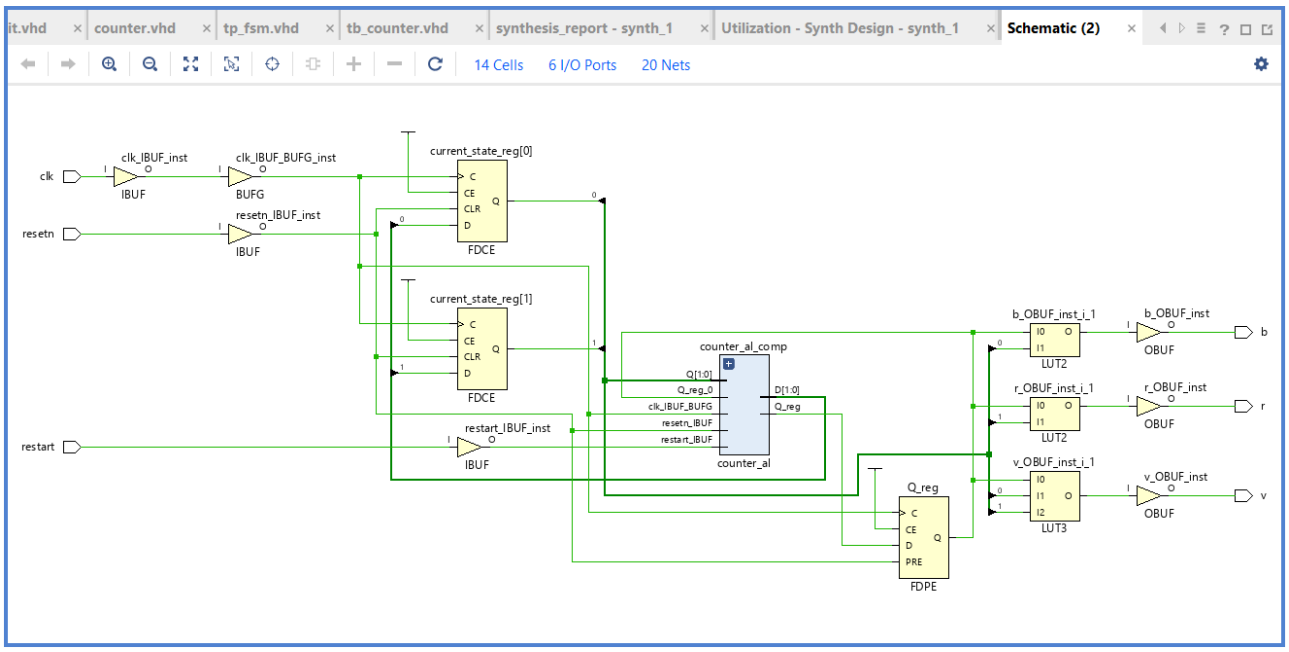
Report Cell Usage:

	Cell	Count
1	BUFG	1
2	CARRY4	7
3	LUT2	32
4	LUT3	2
5	LUT4	4
6	LUT5	2
7	LUT6	5
8	FDCE	33
9	FDPE	1
10	IBUF	3
11	OBUF	3

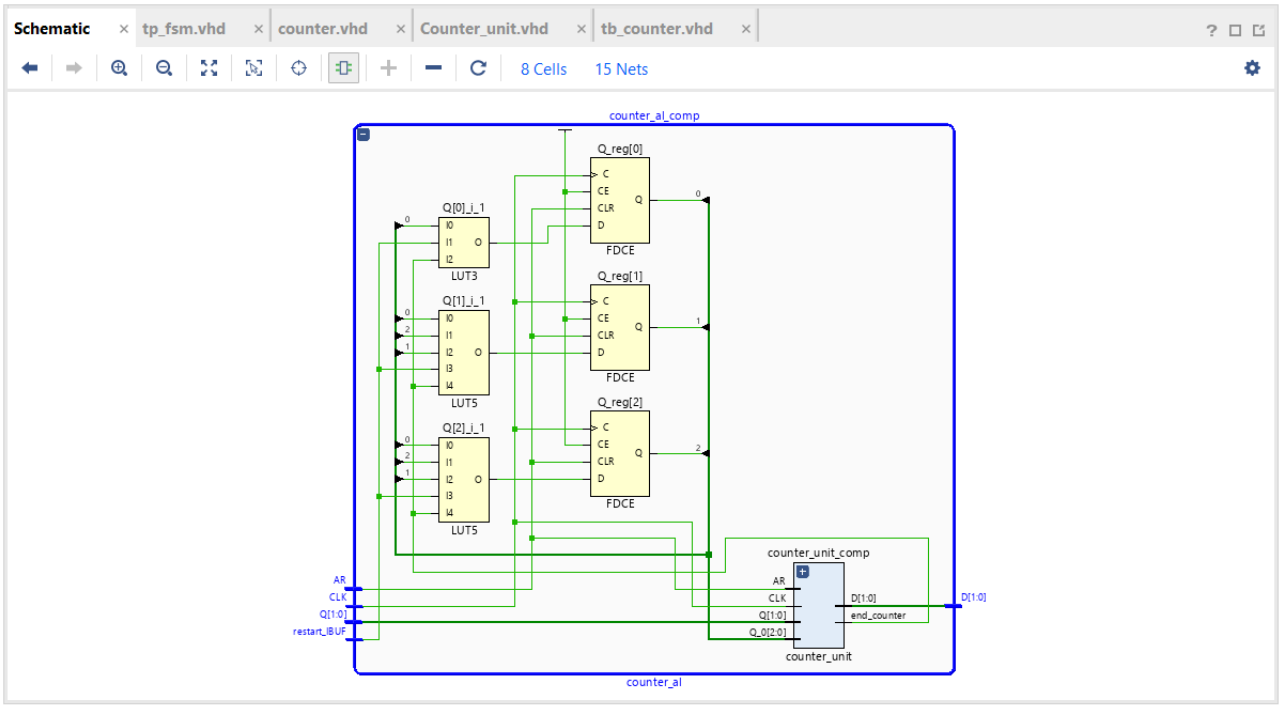
Il y a 33 FDCE (Registre à reset asynchrone) : 28 pour le counter unit, 3 pour le compteur d'impulsion, 2 pour la FSM.

Il y a 1 FDPE (registre à set asynchrone) pour la creation du signal led_on. Le signal led_on indique quand la led est allumée ou éteinte.

Schematic :



Le compteur de cycle est le bloc en bleu noté counter_al_comp.
Il apparait en un seul block car j'ai fait le choix de créer un module pour le compteur de cycles.



A l'intérieur de notre compteur de cycles on retrouve notre module counter_unit_comp.

Question 10

Modification du fichier de contraintes:

```
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 10.00 -waveform {0 5} [get_ports { clk }];#set

# RGB LEDs
set_property -dict { PACKAGE_PIN L15   IOSTANDARD LVCMOS33 } [get_ports { b }];
#IO_L22N_T3_AD7N_35 Sch=led0_b
set_property -dict { PACKAGE_PIN G17   IOSTANDARD LVCMOS33 } [get_ports { v }];
#IO_L16P_T2_35 Sch=led0_g
set_property -dict { PACKAGE_PIN N15   IOSTANDARD LVCMOS33 } [get_ports { r }];
#IO_L21P_T3_DQS_AD14P_35 Sch=led0_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 }];
#IO_L4P_T0_35 Sch=btn[1]
```

Question 11

Rapport de timing :

WNS(ns)	TNS(ns)	TNS Failing Endpoints	TNS Total Endpoints
4.953	0.000	0	34

WHS(ns)	THS(ns)	THS Failing Endpoints	THS Total Endpoints
0.248	0.000	0	34

TNS et THS sont à 0 il n'y a pas de violation de setup ou de hold.

Chemin critique :

Max Delay Paths

Slack (MET) : 4.953ns (required time - arrival time)
Source: counter_al_comp/counter_unit_comp/Q_reg[3]/C
(rising edge-triggered cell FDCE clocked by sys_clk_pin {rise@0.000ns fall@5.000ns period=10.000ns})
Destination: counter_al_comp/counter_unit_comp/Q_reg[25]/D
(rising edge-triggered cell FDCE clocked by sys_clk_pin {rise@0.000ns fall@5.000ns period=10.000ns})

Question 12

Génération du bitstream et progammation de la carte.

La led clignote blanche au premier démarrage de la carte ou si l'on appui sur le bouton reset.

La led prend les couleurs rouge clignotant, puis bleu puis vert et recommence un cycle avec la couleur rouge.

Le fonctionnement sur carte est conforme à la description.