

TP3 - MACHINE A ETATS (FSM)

Rendu

Votre rapport devra contenir :

- Vos schéma RTL
- Vos résultats de simulation avec vos chronogrammes commentés
- Vos résultats de synthèse (analyse de vos ressources utilisées)
- Vos résultats de STA (analyse du rapport de timing)
- Une démonstration de votre design

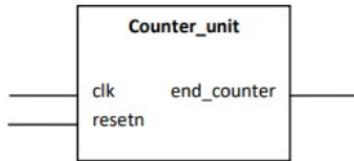
Vous fournirez également vos codes source commentés.

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.

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.

Reprise du code du TP2 en supprimant la partie LED et en modifiant la déclaration du paramètre count_max en generic.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity counter_unit is
    generic( count_max : natural := 2000 ); -- Valeur du nombre de périodes à compter
    port (
        clk           : in std_logic;          -- Horloge
        resetn       : in std_logic;          -- Entrée pour RESET des registres
        end_counter  : out std_logic         -- Sortie indiquant la fin du comptage
    );
end counter_unit;
architecture behavioral of counter_unit is
    --Déclaration des signaux internes
    signal Q_out_cnt : natural range 0 to count_max; -- Signal en sortie du registre du compteur
    signal cmd_cnt   : std_logic := '0';              -- Recopie du signal end_counter pour utilisation dans partie séquentielle
```

Reprise du code du TP2 en supprimant la partie LED et en modifiant la déclaration du paramètre count_max en generic.

```
23 begin
24
25     --Partie sequentielle
26     process(clk,resetn)
27     begin
28         if(resetn = '1') then
29             Q_out_cnt <= 0;
30         elsif(rising_edge(clk)) then
31             if (cmd_cnt = '1') then
32                 Q_out_cnt <= 0;
33             else
34                 Q_out_cnt <= Q_out_cnt + 1;
35             end if;
36         end if;
37     end process;
38
39     --Partie combinatoire
40     cmd_cnt <= '1' when (Q_out_cnt = (count_max-1))
41         else '0';
42     end_counter <= cmd_cnt;           -- Copie du signal end_counter
43
44 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.

SOLUTION 1

Un cycle allumé/éteint correspond au comptage de 2 signaux *end_counter* à 1.

end counter	Qout cycle	end counter cycle	count cycle
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
1	0	0	0
0	1	0	0
0	1	0	0
0	1	0	0
1	1	0	0
0	0	1	0
0	0	1	1
0	0	1	1
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	1	0	1
0	0	1	1
0	0	0	2

Initialisation

Registre

Logique combinatoire

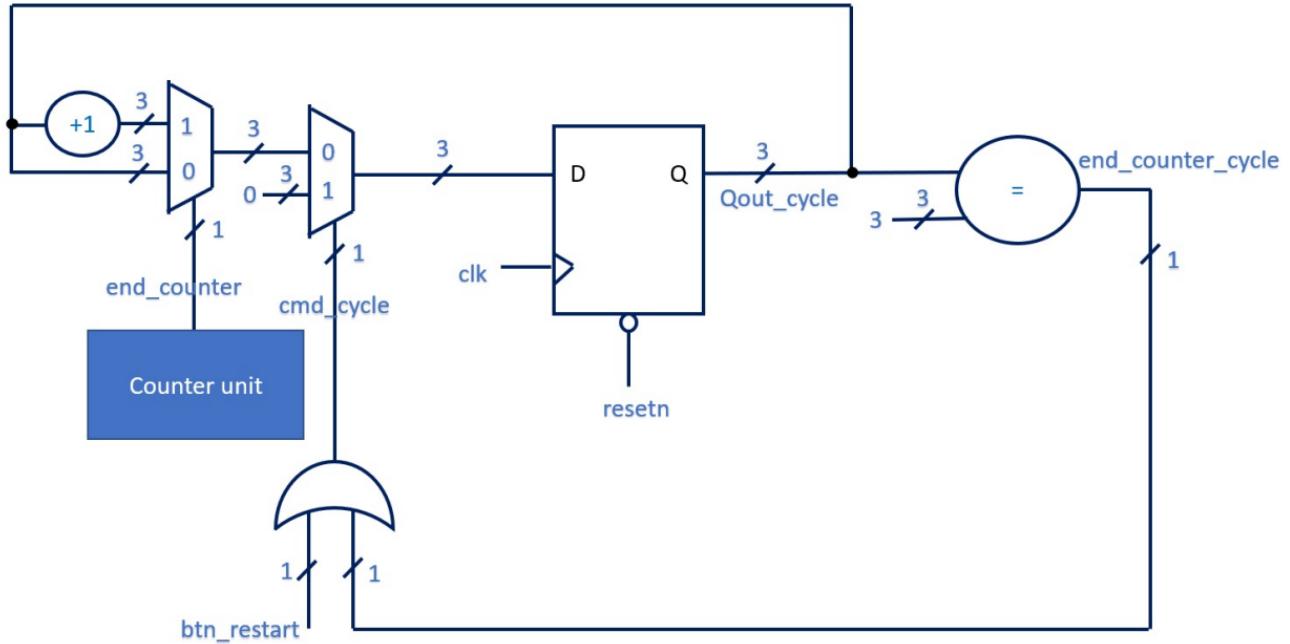


Schéma RTL permettant de générer un signal `end_counter_cycle` pour chaque cycle

A chaque cycle "éteint:allumé", le signal `end_counter_cycle` passe à 1.
Le bouton `btn_restart` permet de redémarrer le compteur à 0.

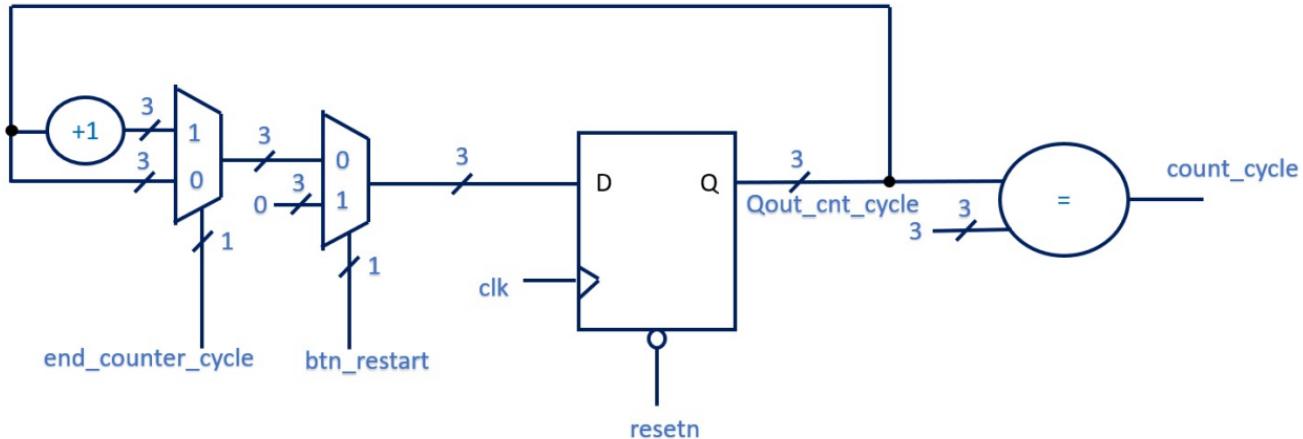


Schéma RTL permettant de compter les cycles

A chaque détection de fin de cycle "éteint/allumé", on incrémente le compteur count_cycle.
Si on appuie sur le bouton btn_restart, le compteur est remis à 0.

SOLUTION 2

AVANTAGE: un seul registre utilisé !!!

end_counter (cmd_cnt)	nb_cycle	Qout_cycle	Qout_cycle_dec	count_cycle	end_counter_cycle (cmd_restart)
0	0	0	0 = "0000"	0	0
0	0	0	1 = "0001"	0	0
0	0	0	1 = "0001"	0	0
0	0	0	1 = "0001"	0	0
1	1	0	1 = "0001"	0	0
0	1	1	2 = "0010"	1	0
0	1	1	2 = "0010"	1	0
0	1	1	2 = "0010"	1	0
1	2	1	2 = "0010"	1	0
0	2	2	3 = "0011"	1	0
0	2	2	3 = "0011"	1	0
0	2	2	3 = "0011"	1	0
1	3	2	3 = "0011"	1	0
0	3	3	4 = "0100"	2	0
0	3	3	4 = "0100"	2	0
0	3	3	4 = "0100"	2	0
1	4	3	4 = "0100"	2	0
0	4	4	5 = "0101"	2	0
0	4	4	5 = "0101"	2	0
0	4	4	5 = "0101"	2	0
0	4	4	5 = "0101"	2	0
1	5	4	5 = "0101"	2	0
0	5	5	6 = "0110"	3	0
0	5	5	6 = "0110"	3	0
0	5	5	6 = "0110"	3	0
0	5	5	6 = "0110"	3	0
1	6	5	6 = "0110"	3	1
0	0	0	1 = "0001"	0	0



Suppression du LSB

Registre



Logique combinatoire



Si les 2 conditions sont remplies
=> end_counter_cycle passe à 1

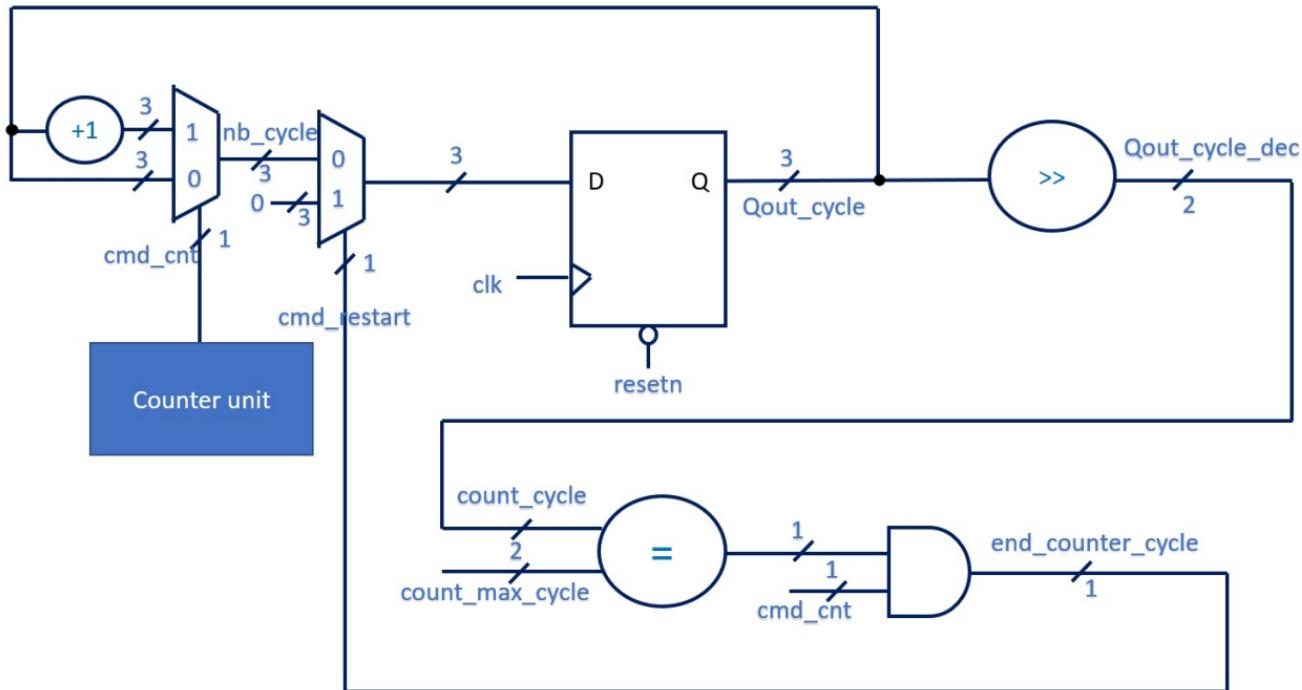


Schéma RTL permettant de générer un signal count_cycle

A chaque cycle "éteint:allumé", le signal `count_cycle` s'incrémente.

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

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_arith.all;
4  use ieee.std_logic_unsigned.all;
5
6
7  entity tp_fsm is
8    generic( count_max_cycle : natural := 4 ); -- Valeur du nombre de cycle à compter
9    port (
10      clk          : in std_logic;
11      resetn      : in std_logic;
12      end_counter_cycle : out std_logic
13
14    );
15  end tp_fsm;
16
17
18  architecture behavioral of tp_fsm is
19
20  --type state is (idle, state1, state2); --a modifier avec vos etats
21
22  --signal current_state : state; --etat dans lequel on se trouve actuellement
23  --signal next_state : state; --etat dans lequel on passera au prochain coup d'horloge
24
25  --constant count_max_cycle : natural := 4; -- nombre de cycle max correspondant au compteur de cycle
26  signal Qout_cycle     : std_logic_vector (4 downto 0) := (others => '0');           -- Signal en sortie du registre du compteur de cycle
27  signal Qout_cycle_dec : std_logic_vector (4 downto 0) := (others => '0');           -- Signal en sortie du registre du compteur de cycle + 1
28  signal count_cycle   : std_logic_vector (4 downto 0) := (others => '0');           -- Compteur de cycle
29  signal cmd_cnt       : std_logic := '0';           -- Commande du multiplexeur de comptage
30  signal cmd_restart   : std_logic := '0';           -- Commande du multiplexeur de restart
31
32
33  --Declaration de l'entite counter_unit (compteur générant le signal end_counter)
34  component counter_unit
35    generic(
36      count_max      : natural
37    );
38    port (
39      clk          : in std_logic;
40      resetn      : in std_logic;
41      end_counter : out std_logic
42    );
43  end component;
```

```
46 begin
47
48     --Affectation des signaux du compteur de cycle avec ceux de counter_unit
49     uut: counter_unit
50         generic map(
51             count_max => 2000
52         )
53         port map (
54             clk => clk,
55             resetn => resetn,
56             end_counter => cmd_cnt
57         );
58
59
60         --Partie sequentielle
61         process(clk, resetn)
62         begin
63             if(resetn='1') then
64                 --current_state <= idle;
65                 Qout_cycle <= (others => '0');
66
67             elsif(rising_edge(clk)) then
68                 --current_state <= next_state;
69                 if (cmd_restart = '1') then
70                     Qout_cycle <= (others => '0');
71                 elsif (cmd_cnt = '1') then
72                     Qout_cycle <= Qout_cycle + 1;
73                 end if;
74
75             end if;
76         end process;
77
78         --Partie combinatoire
79         Qout_cycle_dec <= Qout_cycle + 1;
80         count_cycle <= '0' & Qout_cycle_dec(4 downto 1);
81         cmd_restart <= '1' when ((count_cycle = (count_max_cycle)) AND (cmd_cnt = '1'))
82             else '0';
83         end_counter_cycle <= cmd_restart;
--
```

4. Tester votre architecture avec un testbench.

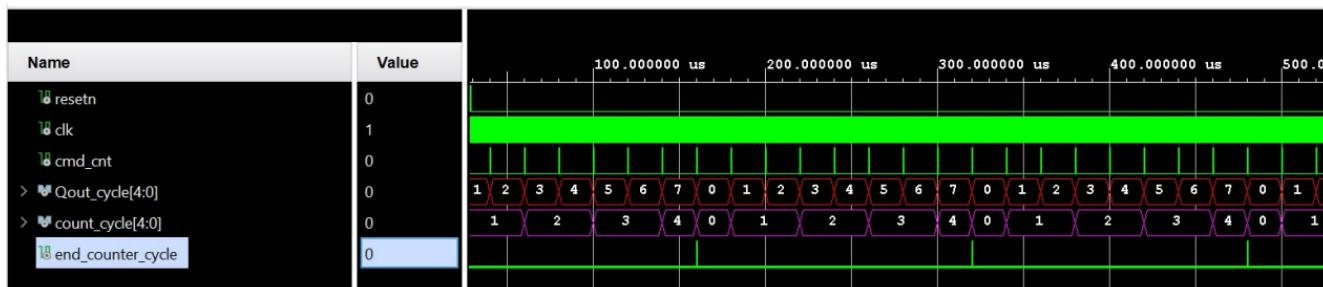
Test avec un comptage de 4 cycles (un cycle étant $2 * 2000$ périodes)

Le signal cmd_cnt est bien généré de façon périodique.

A chaque front montant du signal cmd_cnt, la valeur du signal Qout_cycle s'incrémente.

Tous les deux fronts montants du signal cmd_cnt, la valeur du signal count_cycle s'incrémente.

Lorsque count_cycle atteint 4 et on a un front montant de cmd_cnt, le signal end_counter_cycle passe à 1.

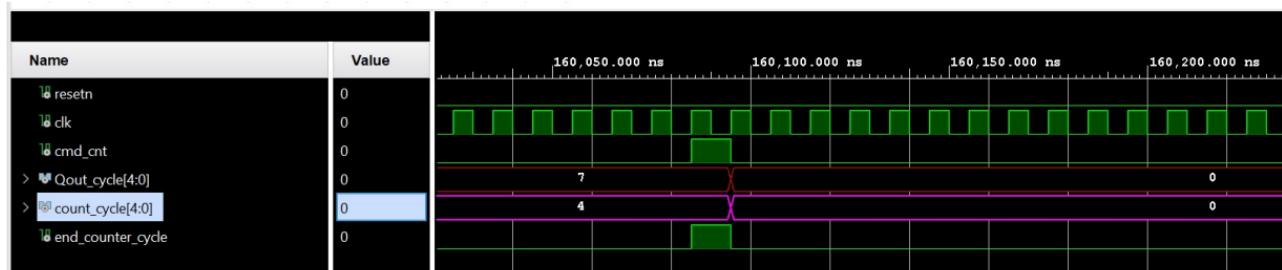


Mise à jour de Qout_cycle suite à un front montant de cmd_cnt.



Les signaux cmd_cnt et end_counter_cycle sont bien en phase.

En effet, le signal end_counter_cycle passe à 1 lorsque le signal cmd_cnt est à 1 ET count_cycle à 4.



```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity tb_tp_fsm is
5 end tb_tp_fsm;
6
7 architecture behavioral of tb_tp_fsm is
8
9     signal clk      : std_logic := '0';
10    signal resetn   : std_logic := '0';
11    signal end_counter_cycle : std_logic;
12
13    -- Les constantes suivantes permettent de définir la fréquence de l'horloge
14    constant hp : time := 5 ns;           -- demi période de 5ns
15    constant period : time := 2*hp;       -- période de 10ns, soit une fréquence de 100Hz
16    constant count_max : natural := 2000; -- nombre de périodes max correspondant au compteur de période
17    constant count_max_cycle : natural := 4;  -- nombre de cycle max correspondant au compteur de cycle
18
19
20 component tp_fsm
21     generic(
22         count_max_cycle : natural
23     );
24     port (
25         clk          : in std_logic;
26         resetn      : in std_logic;
27         end_counter_cycle : out std_logic
28     );
29 end component;
30
31
32
33
34 begin
35 dut: tp_fsm
36     generic map(
37         count_max_cycle => 4
38     )
39     port map (
40         clk => clk,
41         resetn => resetn,
42         end_counter_cycle => end_counter_cycle
43     );

```

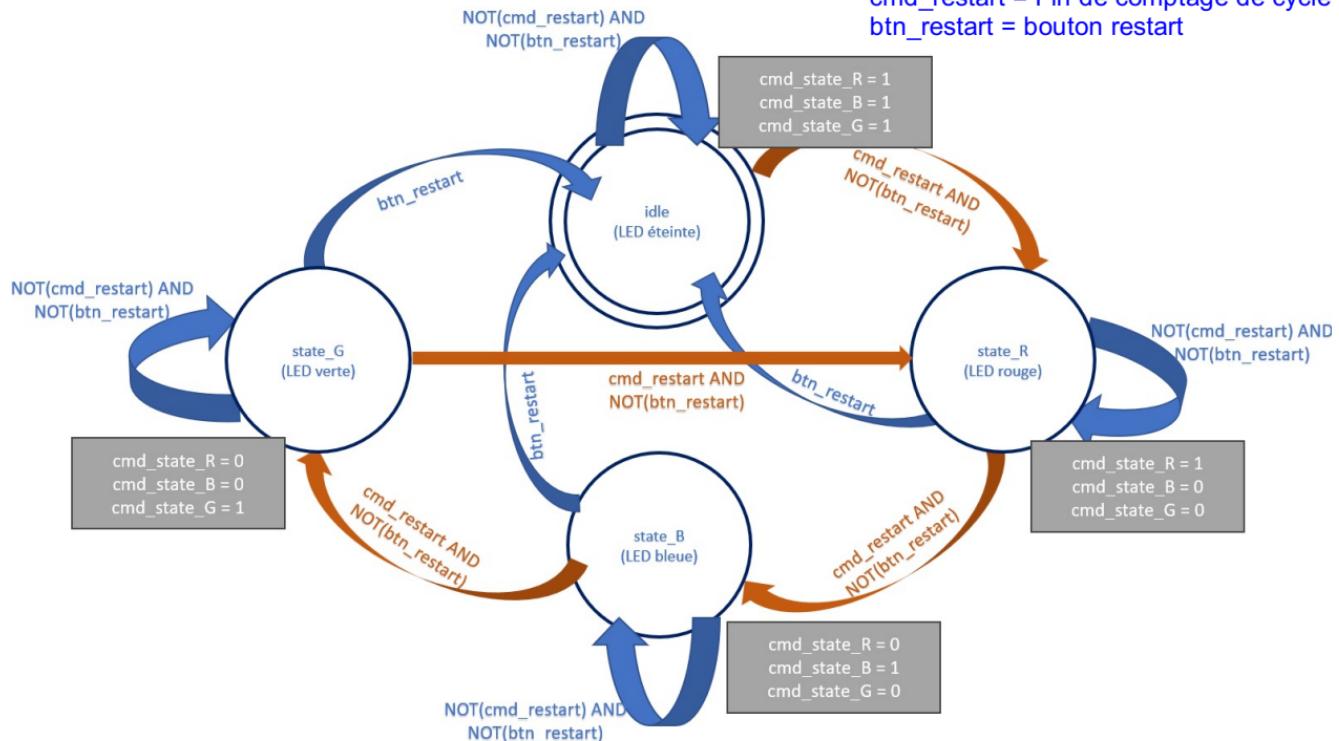
```
-- Simulation du signal d'horloge en continue
45
46 process
47 begin
48     wait for hp;
49     clk <= not clk;
50 end process;
51
52
53 process
54 begin
55     -- INITIALISATION
56     resetn <= '1';
57     wait for period*10;
58     resetn <= '0';
59
60
61     -- Valeurs des sorties attendues
62     -- end_counter_cycle = 0
63     assert end_counter_cycle = '0'
64     report "ERROR: end_counter_cycle not equals to 0" severity failure;
65
66
67
68     -- PREMIER TEST: PASSAGE DU SIGNAL end_counter_cycle A 1
69     -- ATTENTE DE [(2 * count_max) * count_max_tb] periodes
70
71     -- counting for count_max_cycle cycles
72     wait for ((2 * (count_max) * count_max_cycle) - 1) * period;
73     -- Valeurs des sorties attendues
74     -- end_counter_cycle = 1
75     assert end_counter_cycle = '1'
76     report "ERROR: end_counter_cycle not equals to 1" severity failure;
77
```

```
79
80    -- DEUXIEME TEST: PASSAGE DU SIGNAL end_counter_cycle A 1      --
81    -- ATTENTE DE [(2 * count_max) * count_max_tb] periodes supplémentaires --
82
83    -- counting for count_max_cycle cycles
84    wait for (2 * count_max * count_max_cycle * period);
85    -- Valeurs des sorties attendues
86    -- end_counter_cycle = 1
87    assert end_counter_cycle = '1'
88    report "ERROR: end_counter_cycle not equals to 1" severity failure;
89
90
91
92    -- TROISIEME TEST: PASSAGE DU SIGNAL end_counter_cycle A 1      --
93    -- ATTENTE DE [(2 * count_max) * count_max_tb] periodes supplémentaires --
94
95    -- counting for count_max_cycle cycles
96    wait for (2 * count_max * count_max_cycle * period);
97    -- Valeurs des sorties attendues
98    -- end_counter_cycle = 1
99    assert end_counter_cycle = '1'
100   report "ERROR: end_counter_cycle not equals to 1" severity failure;
101
102
103
104    -- QUATRIEME TEST
105    -- ATTENTE DE [(count_max) * count_max_tb] periodes supplémentaires --
106
107    -- counting for (count_max_cycle / 2) cycles
108    wait for (count_max * count_max_cycle * period);
109
110    -- counting for 1 additionnal period (Total: (count_max) periods)
111    wait for period;
112    -- Valeurs des sorties attendues
113    -- end_counter_cycle = 0
114    assert end_counter_cycle = '0'
115    report "ERROR: end_counter_cycle not equals to 0" severity failure;
116
117
118
119    wait;
120
121 end process;
```

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).

LEGENDE

cmd_restart = Fin de comptage de cycle
btn_restart = bouton restart



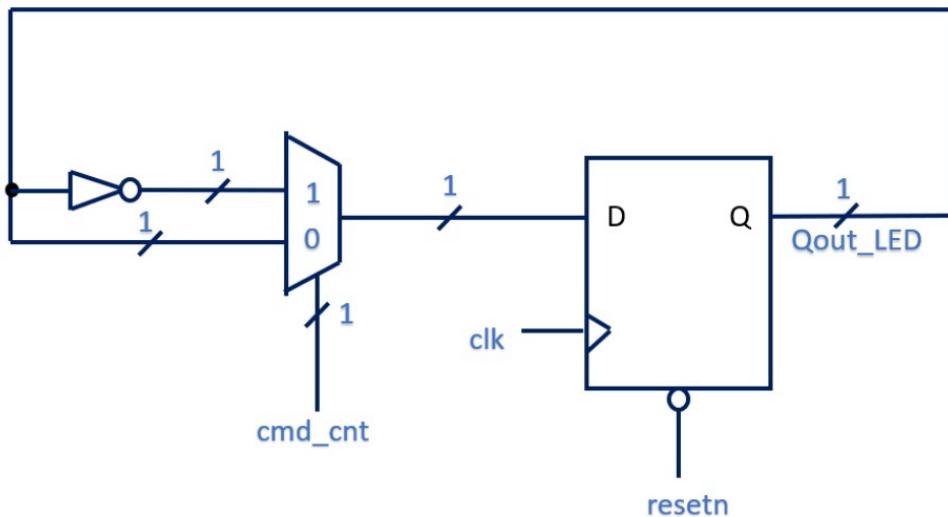
COMMENT FAIRE CLIGNOTER UNE LED AVEC LES SIGNAUX INTERNES ?

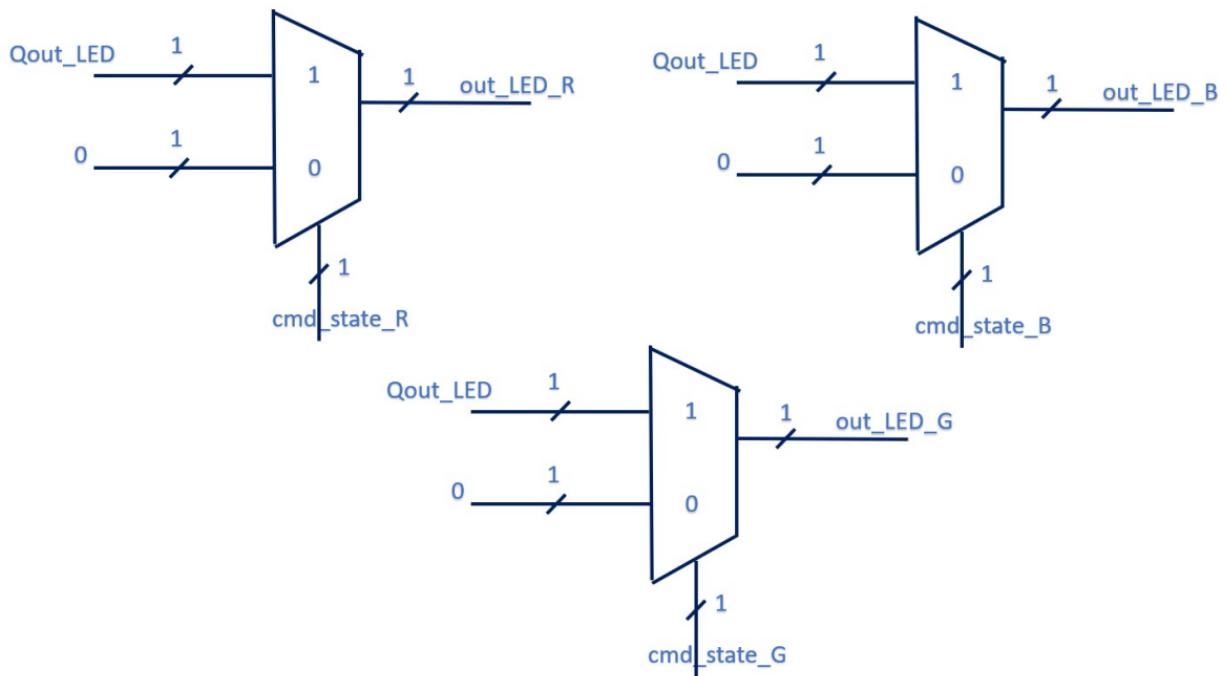
end_counter (cmd_cnt)	nb_cycle	Qout_cycle	Qout_cycle_dec	count_cycle	end_counter_cycle (cmd_restart)
0	0	0	1 = "0001"	0	0
1	1	0	1 = "0001"	0	0
0	1	1	2 = "0010"	1	0
0	1	1	2 = "0010"	1	0
1	2	1	2 = "0010"	1	0
0	2	2	3 = "0011"	1	0
0	2	2	3 = "0011"	1	0
1	3	2	3 = "0011"	1	0
0	3	3	4 = "0100"	2	0
0	3	3	4 = "0100"	2	0
1	4	3	4 = "0100"	2	0
0	4	4	5 = "0101"	2	0
0	4	4	5 = "0101"	2	0
1	5	4	5 = "0101"	2	0
0	5	5	6 = "0110"	3	0
0	5	5	6 = "0110"	3	1
1	6	5	6 = "0110"	3	1
0	0	0	1 = "0001"	0	0

On souhaite que la LED clignote 3 fois avant de passer à l'état suivant.

Par conséquent, on doit se servir du signal cmd_cnt. A chaque réception du signal cmd_cnt, la LED prend l'état précédent.

SCHEMA RTL POUR FAIRE CLIGNOTER UNE LED





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

SIGNAUX D'ENTREE

- clk: Horloge
- resetn: Bouton de reset
- btn_restart: Bouton de redémarrage

SIGNAUX INTERNES

- count_max: Valeur du nombre de périodes à compter (\Rightarrow end_counter)
- count_max_cycle: Valeur du nombre de cycle à compter (\Rightarrow end_counter_cycle)
- Qout_cycle: Signal en sortie du registre du compteur de cycle
- Qout_cycle_dec: Signal en sortie du registre du compteur de cycle + 1
- count_cycle: Compteur de cycle
- cmd_cnt: Commande du multiplexeur de comptage
- cmd_restart: Commande du multiplexeur de restart
- cmd_state_R: Commande indiquant l'état de la LED Rouge
- cmd_state_B: Commande indiquant l'état de la LED Bleue
- cmd_state_G: Commande indiquant l'état de la LED Verte
- Qout_LED: Etat de la LED, allumée ou éteinte
- current_state: Etat actuel (idle, state_R, state_B, state_G)
- next_state: Prochain état

SIGNAUX DE SORTIE

- end_counter_cycle: Fin de cycle
- out_LED_R: Sortie LED Rouge
- out_LED_B: Sortie LED Bleue
- out_LED_G: Sortie LED Verte

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

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.std_logic_arith.all;
4 use ieee.std_logic_unsigned.all;
5
6
7 entity tp_fsm is
8     generic( count_max_cycle : natural := 3 ); -- Valeur du nombre de cycle à compter
9     port (
10         clk           : in std_logic;
11         resetn        : in std_logic;
12         btn_restart   : in std_logic;
13         end_counter_cycle : out std_logic;
14         out_LED_R    : out std_logic;
15         out_LED_B    : out std_logic;
16         out_LED_G    : out std_logic
17     );
18 end tp_fsm;
19
20
21 architecture behavioral of tp_fsm is
22
23     type state is (idle, state_R, state_B, state_G); -- Définition des états du FSM
24
25     signal current_state : state; -- etat dans lequel on se trouve actuellement
26     signal next_state : state; -- etat dans lequel on passera au prochain coup d'horloge
27
28     signal Qout_LED : std_logic := '0'; -- etat de la LED (allumee ou eteinte)
29
30     signal Qout_cycle      : std_logic_vector (2 downto 0) := (others => '0'); -- Signal en sortie du registre du compteur de cycle
31     signal Qout_cycle_dec : std_logic_vector (2 downto 0) := (others => '0'); -- Signal en sortie du registre du compteur de cycle + 1
32     signal count_cycle : std_logic_vector (1 downto 0) := (others => '0'); -- Compteur de cycle
33     signal cmd_cnt       : std_logic := '0'; -- Commande du multiplexeur de comptage
34     signal cmd_restart   : std_logic := '0'; -- Commande du multiplexeur de restart
35     signal cmd_state_R   : std_logic := '0'; -- Commande de pilotage de la LED Rouge
36     signal cmd_state_B   : std_logic := '0'; -- Commande de pilotage de la LED Bleue
37     signal cmd_state_G   : std_logic := '0'; -- Commande de pilotage de la LED Verte
38
```

```
40      --Declaration de l'entite counter_unit (compteur générant le signal end_counter)
41      component counter_unit
42          generic(
43              count_max      : natural
44          );
45          port (
46              clk           : in std_logic;
47              resetn        : in std_logic;
48              end_counter   : out std_logic
49          );
50      end component;
51
52
53      begin
54
55          --Affectation des signaux du compteur de cycle avec ceux de counter_unit
56          uut: counter_unit
57              generic map(
58                  count_max => 100000000
59              )
60              port map (
61                  clk => clk,
62                  resetn => resetn,
63                  end_counter => cmd_cnt
64              );
65      
```

```
00 :  
67 :      -- PARTIE SEQUENTIELLE  
68 Y     process(clk, resetn)  
69 begin  
70 D         if(resetn = '1') then  
71             current_state <= idle;           -- Retour de la FSM à l'état initial  
72  
73             Qout_cycle <= (others => '0');    -- Réinitialisation du compteur  
74             Qout_LED <= '0';                  -- Initialisation de l'ete de la LED  
75  
76         elsif(rising_edge(clk)) then  
77             current_state <= next_state;       -- Passage à l'état suivant  
78  
79 D                 if ((btn_restart = '1') OR (cmd_restart = '1')) then  
80                     Qout_cycle <= (others => '0');    -- Réinitialisation du compteur  
81                     Qout_LED <= '0';                  -- Initialisation de l'ete de la LED  
82                 elsif (cmd_cnt = '1') then  
83                     Qout_cycle <= Qout_cycle + 1;  
84                     Qout_LED <= NOT(Qout_LED);  
85 D                 end if;  
86  
87  
88 D             end if;  
89 D end process;
```

```
--  
91      -- PARTIE COMBINATOIRE  
92      -- Comptage de cycles  
93      Qout_cycle_dec <= Qout_cycle + 1;  
94      --  
95      count_cycle <= Qout_cycle_dec(2 downto 1);  
96      cmd_restart <= '1' when (count_cycle = count_max_cycle AND cmd_cnt = '1')  
97          else '0';  
98      end_counter_cycle <= cmd_restart;  
99  
100     -- Clignotement des LEDs  
101     out_LED_R <= Qout_LED when cmd_state_R = '1'  
102         else '0';  
103  
104     out_LED_B <= Qout_LED when cmd_state_B = '1'  
105         else '0';  
106  
107     out_LED_G <= Qout_LED when cmd_state_G = '1'  
108         else '0';  
--
```

```

---  

110    -- FSM  

111    -- Fonctionnement:  

112    -- Si on appuie sur le bouton_restart, on revient à l'état initial  

113    -- sinon si le signal cmd_restart est à 1 (fin de compatge de cycle), alors on passe à l'état suivant  

114    -- sinon on ne change pas d'état  

115    process(current_state, cmd_restart, btn_restart)  

116    begin  

117        -- Etat initial  

118        case current_state is  

119            when idle =>  

120                -- Signaux de commandes des 3 LEDs  

121                cmd_state_R <= '1';  

122                cmd_state_B <= '1';  

123                cmd_state_G <= '1';  

124                if (btn_restart = '1') then  

125                    next_state <= idle;  

126                elsif (cmd_restart = '1') then  

127                    next_state <= state_R; --prochain etat  

128                else  

129                    next_state <= current_state;  

130                end if;  

131  

132                -- LED rouge  

133            when state_R =>  

134                -- Signaux de commandes des 3 LEDs  

135                cmd_state_R <= '1';  

136                cmd_state_B <= '0';  

137                cmd_state_G <= '0';  

138                if (btn_restart = '1') then  

139                    next_state <= idle;  

140                elsif (cmd_restart = '1') then  

141                    next_state <= state_B;  

142                else  

143                    next_state <= current_state;  

144                end if;  

***
```

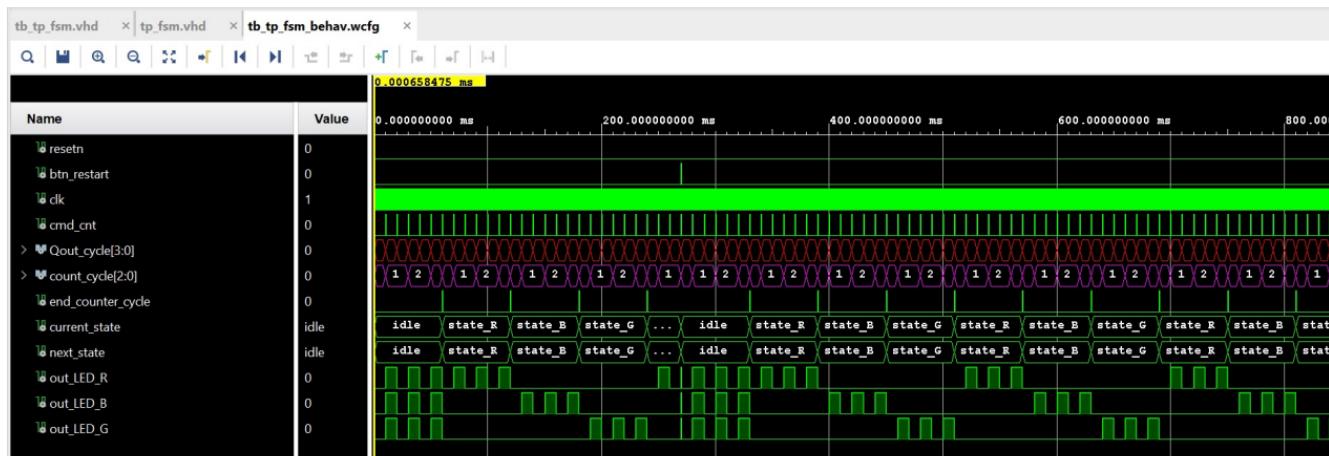
Même fonctionnement pour les autres couleurs.

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

Afin de tester le code, j'ai utilisé:

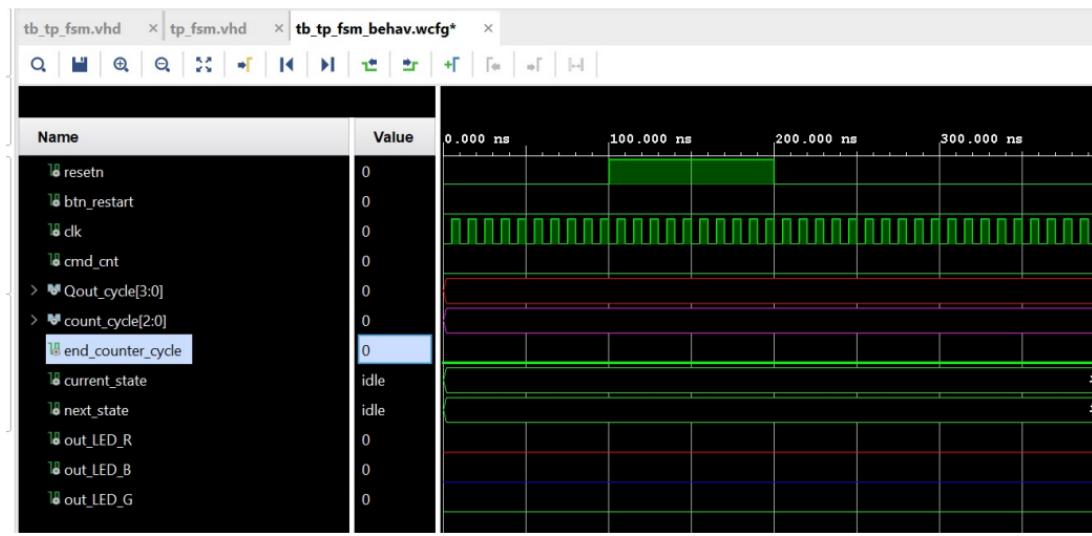
- Période: 10ns
- LED allumée pendant 10ms
- Comptage de 3 cycles

Vue d'ensemble:



Test suite au reset

```
66 --
67 -- INITIALISATION
68 --
69   btn_restart <= '0';
70   wait for period*10;
71   resetn <= '1';
72   wait for period*10;
73   resetn <= '0';
74 --
75 --
76   -- Valeurs des sorties attendues
77   -- end_counter_cycle = 0
78   assert end_counter_cycle = '0'
79     report "ERROR: end_counter_cycle not equals to 0" severity failure;
80   -- LED R, B, et G allumées
81   assert out_LED_R = '0'
82     report "ERROR: out_LED_R not equals to 0" severity failure;
83   assert out_LED_B = '0'
84     report "ERROR: out_LED_B not equals to 0" severity failure;
85   assert out_LED_G = '0'
86     report "ERROR: out_LED_G not equals to 0" severity failure;
```



Test état idle

```
87
88      -- PREMIER TEST: ETAT IDLE
89      -- Les 3 LED doivent clignoter 3 fois en même temps
90
91      -- counting for count_max_cycle cycles
92      wait for ((count_max - 1) * period);
93      -- Valeurs des sorties attendues
94      -- end_counter_cycle = 0
95      assert end_counter_cycle = '0'
96          report "ERROR: end_counter_cycle not equals to 0" severity failure;
97      -- LED R, B, et G allumées
98      assert out_LED_R = '0'
99          report "ERROR: out_LED_R not equals to 0" severity failure;
100     assert out_LED_B = '0'
101         report "ERROR: out_LED_B not equals to 0" severity failure;
102     assert out_LED_G = '0'
103         report "ERROR: out_LED_G not equals to 0" severity failure;
104
105
106      -- counting for count_max_cycle cycles
107      wait for (count_max * period);
108      -- Valeurs des sorties attendues
109      -- end_counter_cycle = 0
110      assert end_counter_cycle = '0'
111          report "ERROR: end_counter_cycle not equals to 0" severity failure;
112      -- LED R, B, et G allumées
113      assert out_LED_R = '1'
114          report "ERROR: out_LED_R not equals to 1" severity failure;
115      assert out_LED_B = '1'
116          report "ERROR: out_LED_B not equals to 1" severity failure;
117      assert out_LED_G = '1'
118          report "ERROR: out_LED_G not equals to 1" severity failure;
119
120
121      -- counting for count_max_cycle cycles
122      wait for (count_max * period);
123      -- Valeurs des sorties attendues
124      -- end_counter_cycle = 0
125      assert end_counter_cycle = '0'
126          report "ERROR: end_counter_cycle not equals to 0" severity failure;
127      -- LED R, B, et G allumées
128      assert out_LED_R = '0'
129          report "ERROR: out_LED_R not equals to 0" severity failure;
130      assert out_LED_B = '0'
131          report "ERROR: out_LED_B not equals to 0" severity failure;
132      assert out_LED_G = '0'
133          report "ERROR: out_LED_G not equals to 0" severity failure;
```

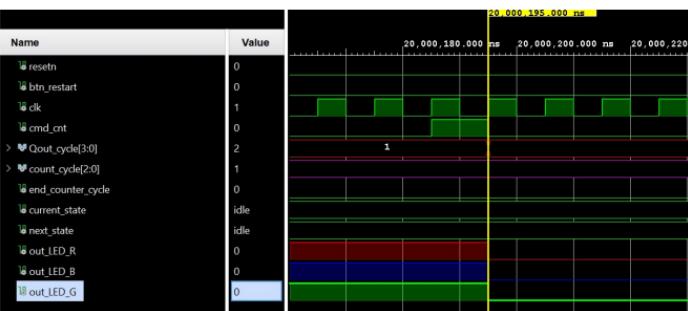


Alternance LED à 0
puis à 1

```
---  
151      -- counting for count_max_cycle cycles  
152      wait for (count_max * period);  
153      -- Valeurs des sorties attendues  
154      -- end_counter_cycle = 0  
155      assert end_counter_cycle = '0'  
156          report "ERROR: end_counter_cycle not equals to 0" severity failure;  
157      -- LED R, B, et G allumées  
158      assert out_LED_R = '0'  
159          report "ERROR: out_LED_R not equals to 0" severity failure;  
160      assert out_LED_B = '0'  
161          report "ERROR: out_LED_B not equals to 0" severity failure;  
162      assert out_LED_G = '0'  
163          report "ERROR: out_LED_G not equals to 0" severity failure;  
164  
165  
166      -- counting for count_max cycle cycles  
167      wait for (count_max * period);  
168      -- Valeurs des sorties attendues  
169      -- end_counter_cycle = 1  
170      assert end_counter_cycle = '1'  
171          report "ERROR: end_counter_cycle not equals to 1" severity failure;  
172      -- LED R, B, et G allumées  
173      assert out_LED_R = '1'  
174          report "ERROR: out_LED_R not equals to 1" severity failure;  
175      assert out_LED_B = '1'  
176          report "ERROR: out_LED_B not equals to 1" severity failure;  
177      assert out_LED_G = '1'  
178          report "ERROR: out_LED_G not equals to 1" severity failure;  
---
```



Passage de 0 à 1 en fin de cycle

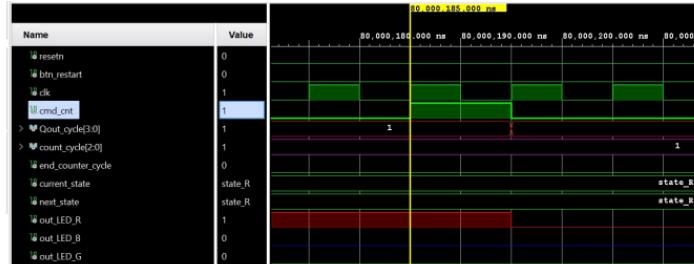
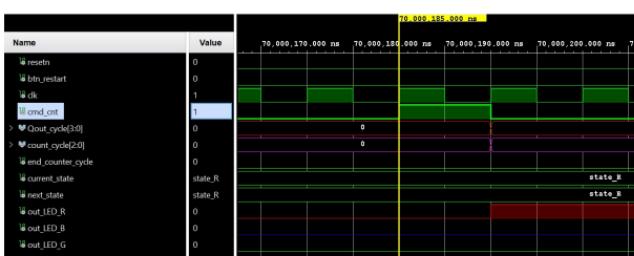


Passage à 1 de end_counter_cycle lors du dernier cmd_cnt à 1



Test état Rouge

Même principe pour le codage du testbench pour le clignotement de chaque LED.

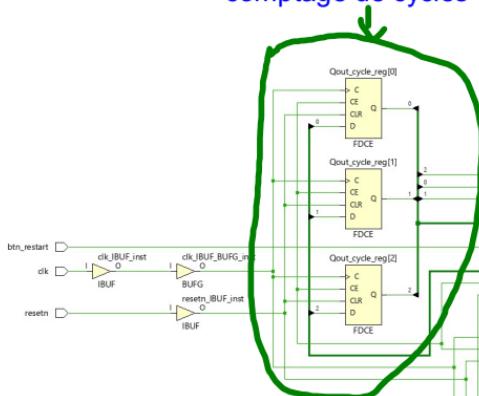


Passage à 1 de end_counter_cycle lors du dernier cmd_cnt à 1



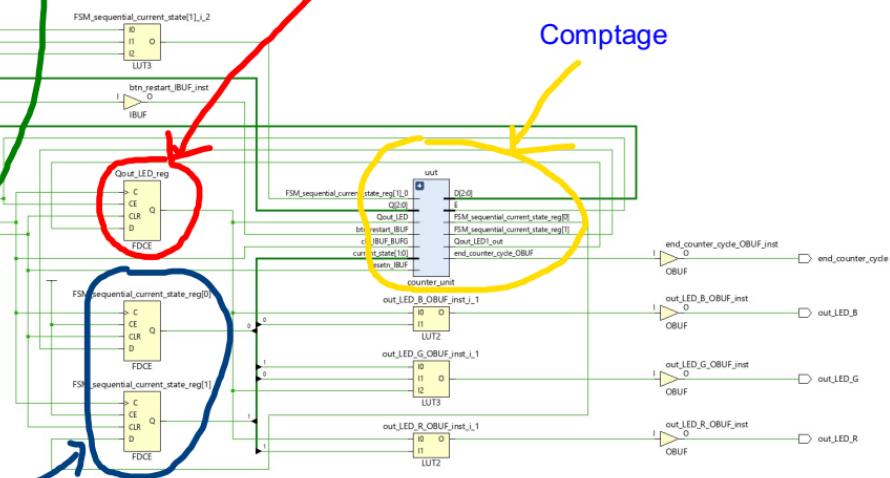
9. Exécutez la synthèse et relevez les ressources utilisées (y compris la FSM). Sur la schématique, identifiez où se situe votre compteur de cycle.

Registres utilisés pour le comptage de cycles



Registre utilisé pour le clignotement des LEDs

Comptage

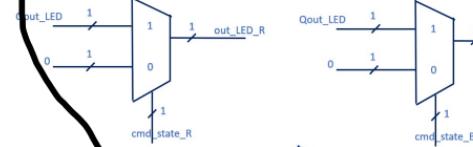
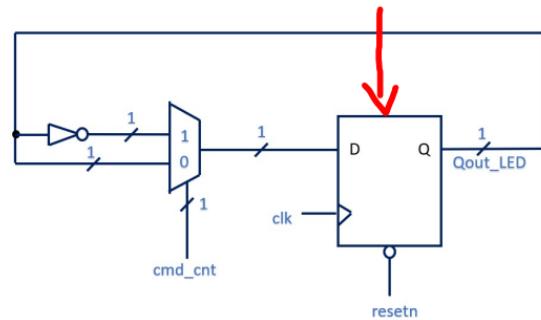


Registres utilisés pour la machine à états

```

89
90 Start RTL Component Statistics
91
92 Detailed RTL Component Info :
93 +---Adders :
94     2 Input   3 Bit      Adders := 1
95 +---Registers :
96     3 Bit      Registers := 1
97     1 Bit      Registers := 1
98 +---Muxes :
99     2 Input   3 Bit      Muxes := 1
100    4 Input   2 Bit      Muxes := 1
101    2 Input   2 Bit      Muxes := 5
102    2 Input   1 Bit      Muxes := 3
103    4 Input   1 Bit      Muxes := 3
104

```



Codage des états

	State	New Encoding	Previous Encoding
77	idle	00	00
78	state_r	01	01
79	state_b	10	10
80	state_g	11	11
81			
82			
83			
84			

=> Codage des états sur bits, soit 2 registres à 4 bit

198	Report Cell Usage:		
199	+-----+-----+-----+		
200	Cell	Count	
201	+-----+-----+-----+		
202	1 BUFG	1	Composant permettant l'optimisation de fonctions de comptage
203	2 CARRY4	7	
204	3 LUT2	2	
205	4 LUT3	4	Look Up Tables
206	5 LUT4	2	
207	6 LUT5	31	
208	7 LUT6	11	Registres (27 utilisés pour le module counter_unit, 3 utilisés pour le comptage de cycle, 1 utilisé pour le clignotement des LEDs et 2 pour les états de la FSM).
209	8 FDCE	33	
210	9 IBUF	3	
211	10 OBUF	4	3 entrées (clk, resetn, btn_restart)
212	+-----+-----+-----+		4 sorties (end_counter_cycle, out_LED_R, out_LED_G, out_LED_B)

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.

$$f = 100 \text{ MHz} = 10^{**8} \text{ Hz}$$

$$T = 1/f \Rightarrow T = 10\text{ns}$$

$$\text{timer} = 1\text{s}$$

$$\text{timer} / T = 100\ 000\ 000 \text{ périodes}$$

Fichier de contrainte

```
 6 # PL System Clock
 7 set_property -dict {PACKAGE_PIN H16 IOSTANDARD LVCMOS33} [get_ports clk]
 8 create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add [get_ports clk]
 9
10 # RGB LEDs
11 set_property -dict {PACKAGE_PIN L15 IOSTANDARD LVCMOS33} [get_ports out_LED_B]
12 set_property -dict {PACKAGE_PIN G17 IOSTANDARD LVCMOS33} [get_ports out_LED_G]
13 set_property -dict {PACKAGE_PIN N15 IOSTANDARD LVCMOS33} [get_ports out_LED_R]
14 set_property -dict {PACKAGE_PIN G14 IOSTANDARD LVCMOS33} [get_ports end_counter_cycle]
15 #set_property -dict { PACKAGE_PIN L14 IOSTANDARD LVCMOS33 } [get_ports { ledl_g }]; #IO_L22P_T3_AD7P_35 Sch=ledl_g
16 #set_property -dict { PACKAGE_PIN M15 IOSTANDARD LVCMOS33 } [get_ports { ledl_r }]; #IO_L23N_T3_35 Sch=ledl_r
17
18 # Buttons
19 set_property -dict {PACKAGE_PIN D20 IOSTANDARD LVCMOS33} [get_ports resetn]
20 set_property -dict {PACKAGE_PIN D19 IOSTANDARD LVCMOS33} [get_ports btn_restart]
```

Bien assigné les couleurs au bon port

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

```
137 | -----  
138 | Clock Summary  
139 | -----  
140 -----  
141 -----  
142 | Clock      Waveform(ns)      Period(ns)      Frequency (MHz)  
143 | -----      -----  
144 | sys_clk_pin {0.000 5.000}    10.000        100.000
```

```
124 |-----  
125 | Design Timing Summary  
126 |-----  
127 |-----  
128 |-----  
129 | WNS(ns)      TNS(ns)      TNS Failing Endpoints  TNS Total Endpoints      WHS(ns)      THS(ns)      THS Failing Endpoints  THS Total Endpoints      WFWS(ns)      TPWS(ns)      TPWS Failing Encp  
130 |-----  
131 | 5.353       0.000          0                  45           0.265       0.000          0                  45           4.500       0.000
```

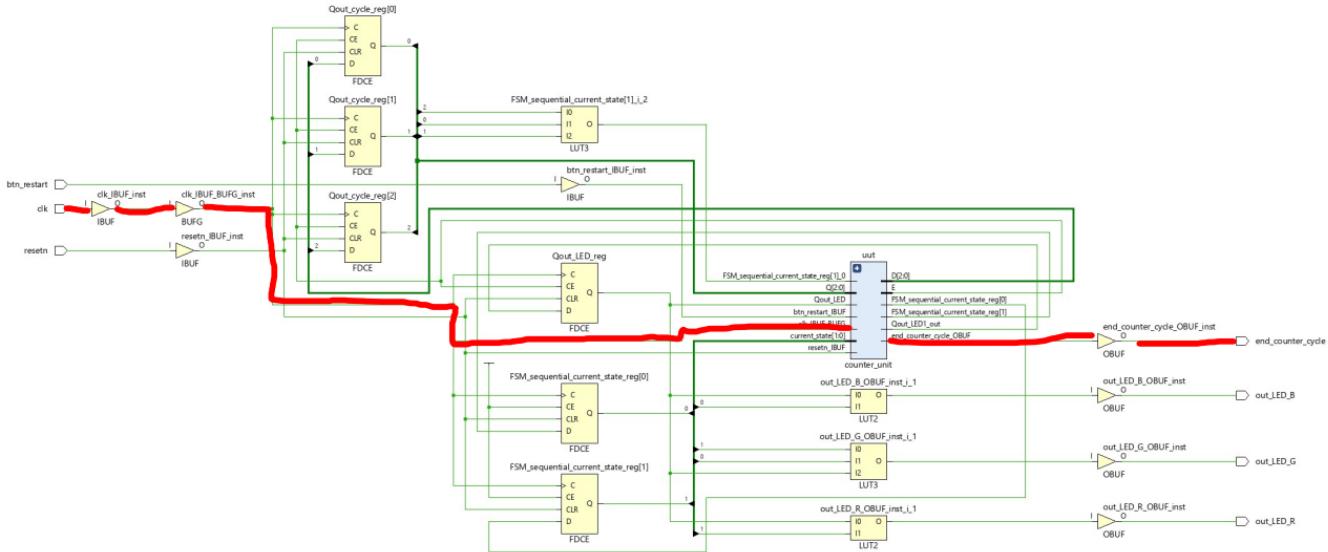
Total Negative Slack = 0
=> Pas de violation de set up

Total Hold Slack = 0
=> Pas de violation de hold

Source et destination du chemin critique

```
191 | Max Delay Paths
192 |
193 | Slack (MET) : 4.775ns (required time - arrival time)
194 |   Source: uut/Q_out_cnt_reg[19]/C
195 |           (rising edge-triggered cell FDCE clocked by sys_clk_pin (rise@0.000ns fall@5.000ns period=10.000ns))
196 |   Destination: uut/Q_out_cnt_reg[25]/D
197 |           (rising edge-triggered cell FDCE clocked by sys_clk_pin (rise@0.000ns fall@5.000ns period=10.000ns))
198 | Path Group: sys_clk_pin
199 | Path Type: Setup (Max at Slow Process Corner)
200 | Requirement: 10.000ns (sys_clk_pin rise@10.000ns - sys_clk_pin rise@0.000ns)
201 | Data Path Delay: 5.280ns (logic 2.331ns (44.147%) route 2.949ns (55.853%))
202 | Logic Levels: 10 (CARRY4=7 LUT4=1 LUT5=2)
203 | Clock Path Skew: -0.019ns (DCD - SCD + CPR)
204 | Destination Clock Delay (DCD): 4.910ns = ( 14.910 - 10.000 )
205 | Source Clock Delay (SCD): 5.357ns
206 | Clock Pessimism Removal (CPR): 0.429ns
207 | Clock Uncertainty: 0.035ns ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
208 | Total System Jitter (TSJ): 0.071ns
209 | Total Input Jitter (TIJ): 0.000ns
210 | Discrete Jitter (DJ): 0.000ns
211 | Phase Error (PE): 0.000ns
212 |
213 | Location      Delay type      Incr(ns)  Path(ns)  Netlist Resource(s)
214 |
215 | (clock sys_clk_pin rise edge)
216 |          0.000    0.000 r
217 | H16          0.000    0.000 r clk (IN)
218 |          net (fo=0) 0.000    0.000 clk
219 | H16          IBUF (Prop_ibuf_I_O) 1.457    1.457 r clk_IBUF_inst/o
220 |          net (fo=1, routed) 2.076    3.533 clk_IBUF
221 | BUFGCTRL_X0Y16 BUFG (Prop_bufg_I_O) 0.101    3.634 r clk_IBUF_BUFG_inst/o
222 |          net (fo=33, routed) 1.723    5.357 uut/clk_IBUF_BUFG
223 | SLICE_X42Y77 FDCE              r uut/Q_out_cnt_reg[19]/C
224 |
225 | SLICE_X42Y77 FDCE (Prop_fdce_C_Q) 0.518    5.875 r uut/Q_out_cnt_reg[19]/Q
226 |          net (fo=3, routed) 0.819    6.695 uut/Q_out_cnt_reg[19]
227 | SLICE_X43Y78 LUT4 (Prop_lut4_I0_O) 0.124    6.819 f uut/end_counter_cycle_OBUF_inst_i_10/o
228 |          net (fo=1, routed) 0.544    7.363 uut/end_counter_cycle_OBUF_inst_i_10_n_0
229 | SLICE_X43Y79 LUT5 (Prop_lut5_I4_O) 0.124    7.487 f uut/end_counter_cycle_OBUF_inst_i_3/o
230 |          net (fo=36, routed) 1.576    9.063 uut/end_counter_cycle_OBUF_inst_i_3_n_0
231 | SLICE_X42Y73 LUT5 (Prop_lut5_I4_O) 0.124    9.187 r uut/Q_out_cnt[0].i_5/o
232 |          net (fo=1, routed) 0.000    9.187 uut/Q_out_cnt[0].i_5_n_0
233 | SLICE_X42Y73 CARRY4 (Prop_carry4_S[1].CO[3])
```

232		net (fo=1, routed)	0.000	9.187	uut/Q_out_cnt[0]_i_5_n_0
233	SLICE_X42Y73	CARRY4 (Prop_carry4_S[1]_CO[3])		9.533	
234		net (fo=1, routed)	0.000	9.720	r uut/Q_out_cnt_reg[0]_i_1/co[3]
235		CARRY4 (Prop_carry4_CI_CO[3])		9.720	uut/Q_out_cnt_reg[0]_i_1_n_0
236	SLICE_X42Y74		0.117	9.837	r uut/Q_out_cnt_reg[4]_i_1/co[3]
237		net (fo=1, routed)	0.009	9.846	uut/Q_out_cnt_reg[4]_i_1_n_0
238		CARRY4 (Prop_carry4_CI_CO[3])		9.963	r uut/Q_out_cnt_reg[8]_i_1/co[3]
239	SLICE_X42Y75		0.117	9.963	uut/Q_out_cnt_reg[8]_i_1_n_0
240		net (fo=1, routed)	0.000	10.080	r uut/Q_out_cnt_reg[12]_i_1/co[3]
241		CARRY4 (Prop_carry4_CI_CO[3])		10.080	uut/Q_out_cnt_reg[12]_i_1_n_0
242	SLICE_X42Y76		0.117	10.197	r uut/Q_out_cnt_reg[16]_i_1/co[3]
243		net (fo=1, routed)	0.000	10.197	uut/Q_out_cnt_reg[16]_i_1_n_0
244		CARRY4 (Prop_carry4_CI_CO[3])		10.314	r uut/Q_out_cnt_reg[20]_i_1/co[3]
245	SLICE_X42Y77		0.117	10.314	uut/Q_out_cnt_reg[20]_i_1_n_0
246		net (fo=1, routed)	0.000	10.637	r uut/Q_out_cnt_reg[24]_i_1/o[1]
247		CARRY4 (Prop_carry4_CI_CO[3])		10.637	uut/Q_out_cnt_reg[24]_i_1_n_6
248	SLICE_X42Y78		0.117		r uut/Q_out_cnt_reg[25]/D
249		net (fo=1, routed)	0.000		
250		CARRY4 (Prop_carry4_CI_O[1])			
251	SLICE_X42Y79		0.323		
252		net (fo=1, routed)	0.000		
253		FDCE			
254	SLICE_X42Y79				
255					
256					
257		(clock sys_clk_pin rise edge)			
258			10.000	10.000	r
259	H16		0.000	10.000	r clk (IN)
260		net (fo=0)	0.000	10.000	clk
261	H16	IBUF (Prop_ibuf_I_O)	1.387	11.387	r clk_IBUF_inst/o
262		net (fo=1, routed)	1.880	13.267	clk_IBUF
263	BUFGCTRL_X0Y16	BUFG (Prop_bufg_I_O)	0.091	13.358	r clk_IBUF_BUFG_inst/o
264		net (fo=33, routed)	1.552	14.910	uut/clk_IBUF_BUFG
265	SLICE_X42Y79	FDCE			r uut/Q_out_cnt_reg[25]/C
266		clock pessimism	0.429	15.338	
267		clock uncertainty	-0.035	15.303	
268	SLICE_X42Y79	FDCE (Setup_fdce_C_D)	0.109	15.412	uut/Q_out_cnt_reg[25]
269					
270		required time		15.412	
271		arrival time		-10.637	
272					
273		slack		4.775	



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

Système vérifié.

A ce stade, je me suis aperçu d'un problème de couleur dans l'enchainement des couleurs des clignotements.
J'ai résolu le problème dans le fichier de contrainte.
Erreur dans l'assignement des ports...