

TP4 – Pilotage de LED et mémoire

Partie 1

Rendu

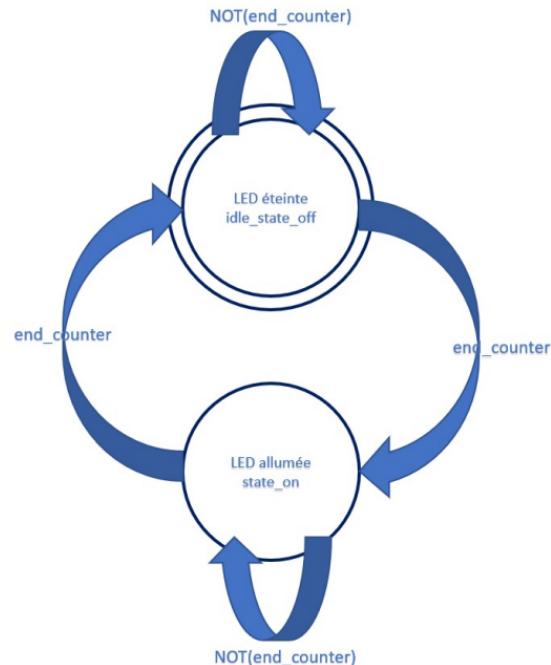
Votre rapport devra contenir :

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

Vous fournirez également vos codes source commentés.

Questions

- Créez une architecture RTL permettant de faire clignoter une LED (par exemple la led0_r) en utilisant le module *Counter_unit* du TP2 et une machine à état.

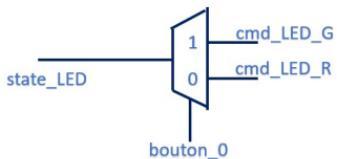


Machine à états

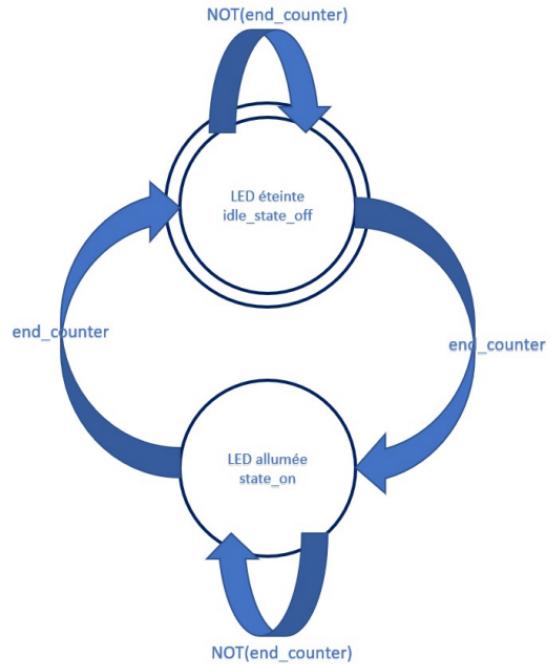
2. Modifiez votre architecture pour piloter une LED rouge et une LED verte. Lorsque le bouton_0 est appuyé, la LED verte est allumée, sinon la LED rouge est allumée.

Schéma RTL

Solution n1



Solution n2



Machine à états

3. Rédigez le code VHDL de votre architecture.

C:/FORMATION_SAFRAN/TP/TP4/_tp_pilotage_LED.vhd

Code implémenté avec la solution n2
Code disponible sous tp_pilotage_LED_Question3.vhd

```
Q | H | ← | → | X | // | ■ | ? |
```

```
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_pilotage_LED is
8     port (
9         clk           : in std_logic;
10        resetn       : in std_logic;
11        bouton_0     : in std_logic;
12        out_LED_R    : out std_logic;
13        out_LED_G    : out std_logic
14    );
15 end tp_pilotage_LED;
16
17
18 architecture behavioral of tp_pilotage_LED is
19
20     type state_LED is (idle_state_off, state_on);      -- Définition des états du FSM
21
22     signal current_state : state_LED;    -- etat dans lequel on se trouve actuellement
23     signal next_state   : state_LED;    -- etat dans lequel on passera au prochain coup d'horloge
24
25     signal end_counter  : std_logic := '0';    -- etat de la LED (allumee ou eteinte)
26     signal cmd_state_LED : std_logic := '0';    -- commande l'état de la LED (allumee ou eteinte)
27
28
29     --Declaration de l'entite counter_unit (compteur générant le signal end_counter)
30 component counter_unit
31     generic(
32         count_max      : natural
33     );
34     port (
35         clk           : in std_logic;
36         resetn       : in std_logic;
37         end_counter   : out std_logic
38     );
39 end component;
```

```
41
42 begin
43
44     --Affectation des signaux du compteur de cycle avec ceux de counter_unit
45     uut: counter_unit
46         generic map(
47             count_max => 1000
48         )
49         port map (
50             clk => clk,
51             resetn => resetn,
52             end_counter => end_counter
53         );
54
55
56     -- PARTIE SEQUENTIELLE
57     process(clk, resetn)
58     begin
59         if(resetn = '1') then
60             current_state <= idle_state_off;           -- Retour de la FSM à l'état initial
61
62         elsif(rising_edge(clk)) then
63             current_state <= next_state;           -- Passage à l'état suivant
64
65         end if;
66     end process;
67
68     -- PARTIE COMBINATOIRE
69     -- Clignotement des LEDs
70     out_LED_R <= cmd_state_LED when (bouton_0 = '0')
71         else '0';
72
73     out_LED_G <= cmd_state_LED when (bouton_0 = '1')
74         else '0';
...
```

```
77      -- FSM
78      process(current_state, end_counter)
79      begin
80          --signaux pilotes par la fsm
81          case current_state is
82              when idle_state_off =>
83                  cmd_state_LED <= '0';
84                  if (end_counter = '1') then
85                      next_state <= state_on; --prochain etat
86                  else
87                      next_state <= current_state;
88                  end if;
89
90
91          when state_on =>
92              cmd_state_LED <= '1';
93              if (end_counter = '1') then
94                  next_state <= idle_state_off; --prochain etat
95              else
96                  next_state <= current_state;
97              end if;
98
99
100         end case;
101
102
103     end process;
104
105
106
107 end behavioral;
```

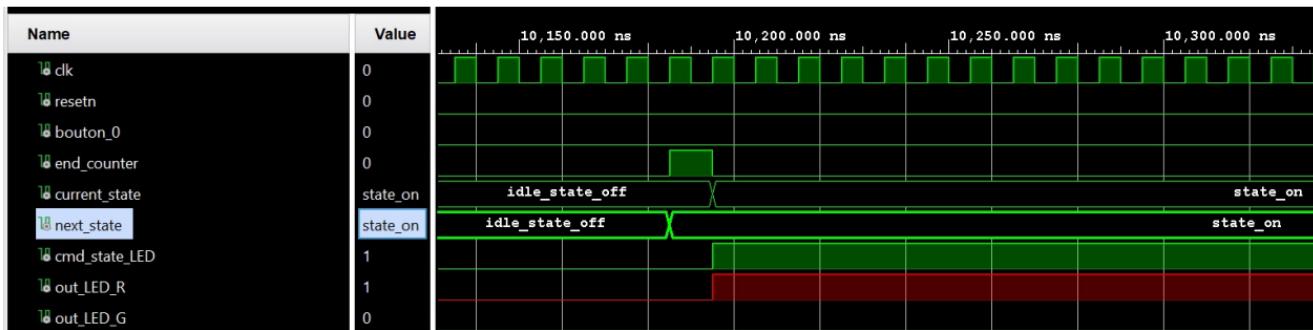
4. Réalisez une simulation en rédigeant un testbench. Que se passe-t-il si le bouton est pressé pendant plus d'un cycle d'horloge ?

Code disponible du testbench dans [tb_tp_pilotage_LED_Question4.vhd](#)

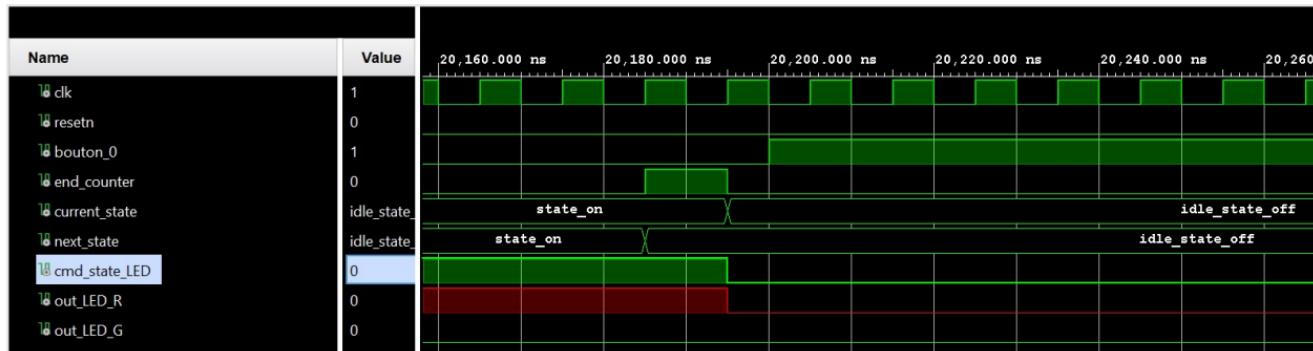
Si le bouton est pressé pendant plus d'un cycle d'horloge, la LED verte continue à clignoter.

-- PREMIER TEST AVEC bouton_0 NON APPUYE --

Suite au signal end_counter, on observe le changement d'état d'idle_state_off à state_on.
Le signal out_LED_R passe à 1.

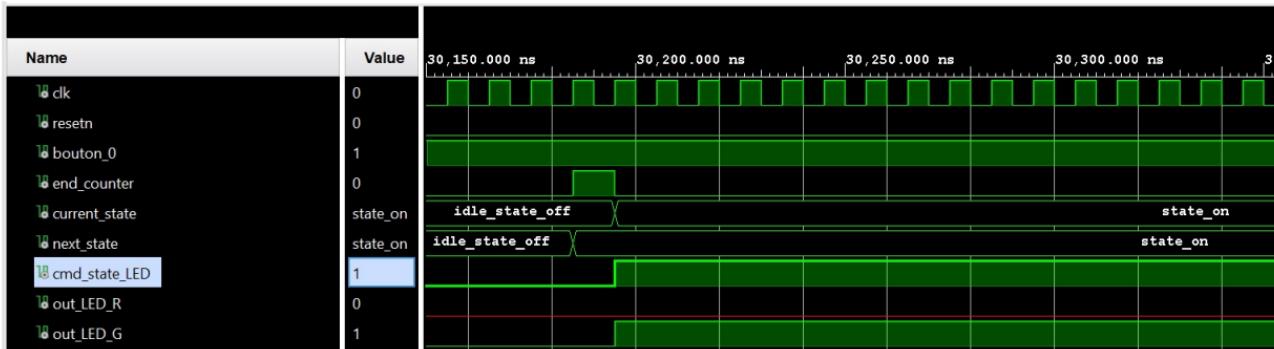


Suite au signal end_counter, on observe le changement d'état de state_on à d'idle_state_off.
Le signal out_LED_R passe à 0.

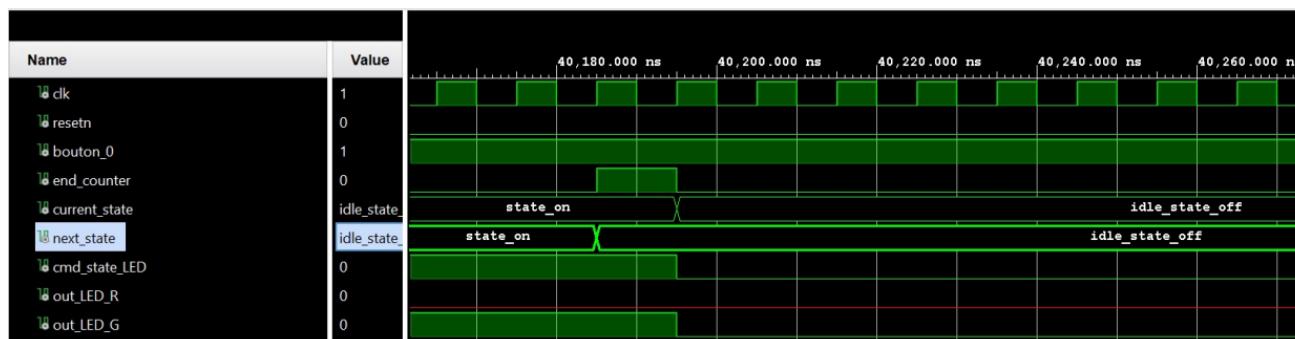


-- DEUXIEME TEST AVEC bouton_0 APPUYE --

Suite au signal end_counter, on observe le changement d'état d'idle_state_off à state_on.
Le signal out_LED_G passe à 1.

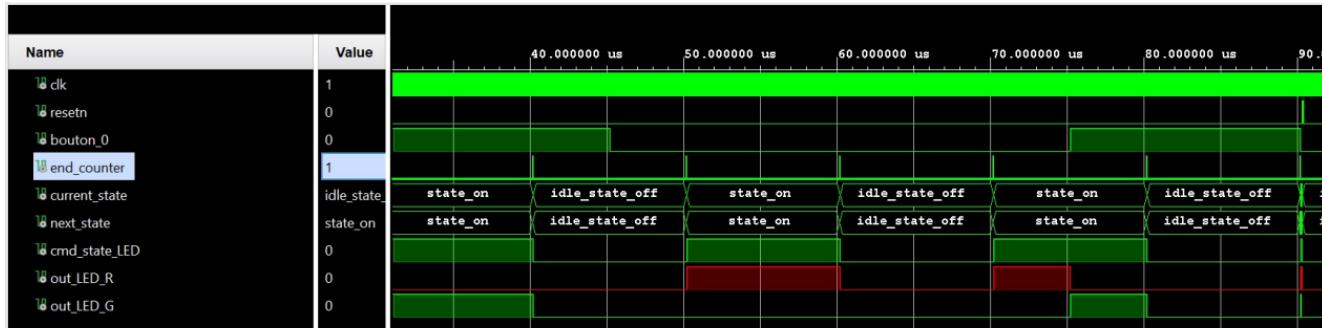


Suite au signal end_counter, on observe le changement d'état de state_on à d'idle_state_off.
Le signal out_LED_G passe à 0.



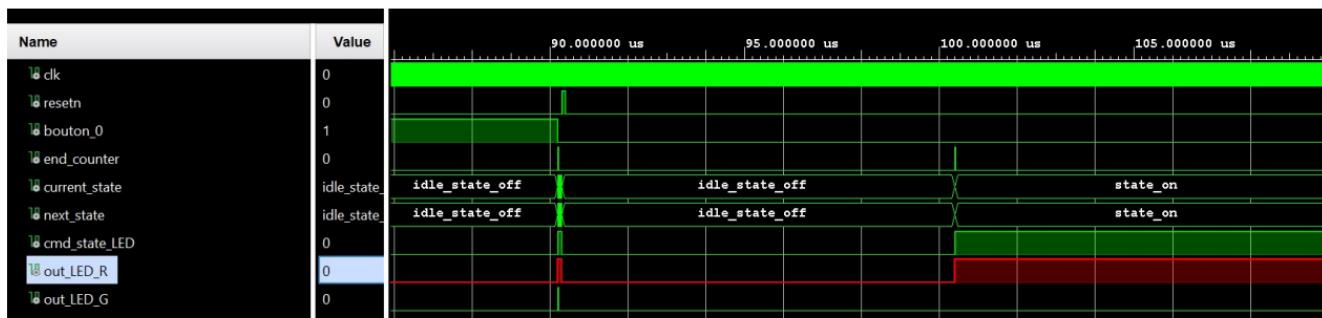
-- TROISIEME ET QUATRIEME TEST AVEC bouton_0 --

Tests similaires en appuyant ou relâchant le bouton bouton_0 en milieu de période.



-- CINQUIEME TEST AVEC APPUI SUR BOUTON resetn --

Suite au reset, l'état revient bien à idle_state_off.



5. Que faudrait-il faire pour que la LED ne clignote en vert qu'une seule fois même si le bouton est maintenu ?

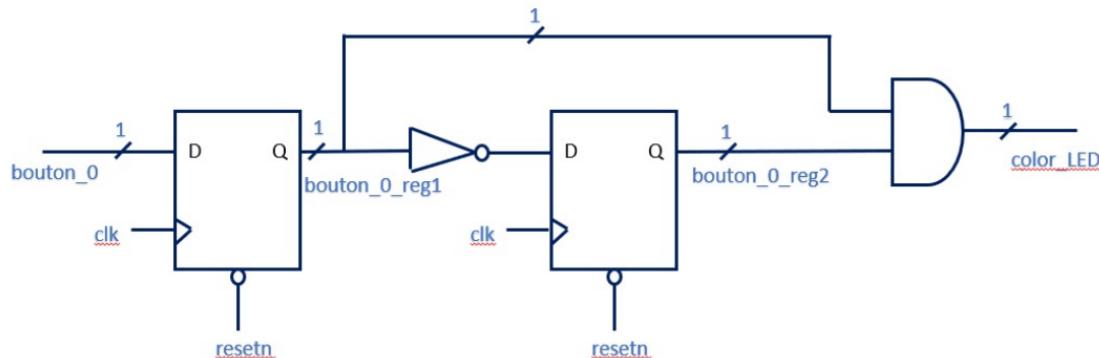
On souhaite que lorsque le bouton bouton_0 est maintenu, la LED recopie l'état signal cmd_state_LED le temps d'un coup d'horloge. Ensuite, la LED rouge continue à clignoter.

Pour cela, on peut conserver notre machine à état de la question précédente afin de gérer les états idle_state_off et state_on.

On ajoute à l'architecture un signal color_LED qui va gérer la couleur de la LED.

Pour détecter un front montant du signal bouton_0, on utilise deux registres. Voir schéma RTL ci-dessous.

Schéma RTL



Explications

Le premier registre permet de récupérer la valeur de bouton_0 sur un front montant d'horloge.

Ainsi, bouton_0_reg1 capture la valeur du signal bouton_0.

bouton_0_reg2 correspond au NOT(bouton_0_reg1) avec un temps de décalage.

Enfin, ET logique pour trouver la valeur du signal color_LED.

bouton_0_reg1	bouton_0_reg2	color_LED
0	0	0
0	1	0
0	1	0
0	1	0
1	1	1
1	0	0
1	0	0
1	0	0

Cela correspond bien à la table d'un ET logique.

6. Ajoutez cette solution à votre architecture RTL.

Le code est disponible dans le fichier tp_pilotage_LED_Question6.vhd

Modifications des parties sequentielles et combinatoires.

```
58          -- PARTIE SEQUENTIELLE
59      process(clk, resetn)
60      begin
61          if resetn = '1' then
62              current_state <= idle_state_off;           -- Retour de la FSM à l'état initial
63
64          elsif rising_edge(clk) then
65              current_state <= next_state;           -- Passage à l'état suivant
66
67          -- Registre utilisé pour créer le signal de commande de la couleur des LEDs
68          bouton_0_reg2 <= NOT(bouton_0_reg1);
69          bouton_0_reg1 <= bouton_0;
70
71          if (bouton_0_reg1 = '1') AND (bouton_0_reg2 = '1') then
72              color_LED <= '1';
73          else
74              color_LED <= '0';
75          end if;
76
77
78          end if;
79      end process;
80
81          -- PARTIE COMBINATOIRE
82          -- Clignotement des LEDs
83      out_LED_R <= cmd_state_LED when color_LED = '0'
84          else '0';
85
86      out_LED_G <= cmd_state_LED when color_LED = '1'
87          else '0';
88
```

7. Mettez à jour votre code VHDL et vérifiez votre résultat à la simulation.

[tb_tp_pilotage_LED_Question7.vhd](#)

-- PREMIER TEST AVEC bouton_0 NON APPUYE --

Name	Value	0.000000 us	20.000000 us	40.000000 us
clk	0	1	0	1
resetn	0	1	0	1
bouton_0	0	1	0	1
bouton_0_reg1	0	1	0	1
bouton_0_reg2	1	0	1	0
color_LED	0	1	0	1
end_counter	0	1	0	1
current_state	idle_state_...	state_on	idle_...	state_on
next_state	idle_state_...	state_on	idle_...	state_on
cmd_state_LED	0	1	0	1
out_LED_R	0	1	0	1
out_LED_G	0	0	0	0

On observe bien la LED rouge qui clignote.

-- DEUXIEME TEST AVEC bouton_0 APPUYE

--

-- Signal cmd_state_LED a l'etat HAUT

--



On observe bien le pilotage de la LED verte le temps d'un coup d'horloge.

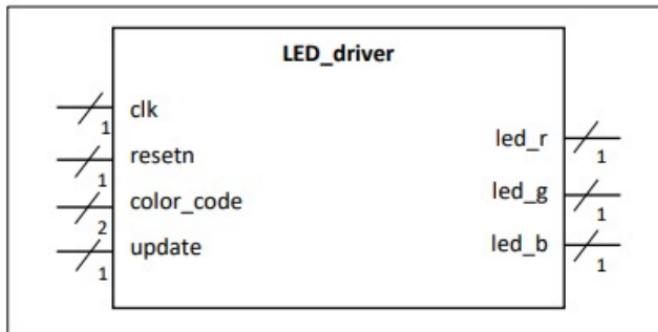
NB: Il y a un décalage d'un coup d'horloge en raison des registres.

-- TROISIEME TEST AVEC bouton_0 APPUYE --
-- Signal cmd_state_LED a l'état BAS --



Dans ce test, le signal color_LED passe bien à 1 le temps d'un coup d'horloge. La LED verte reste à l'état BAS pour autant car le signal cmd_state_LED est à l'état BAS.

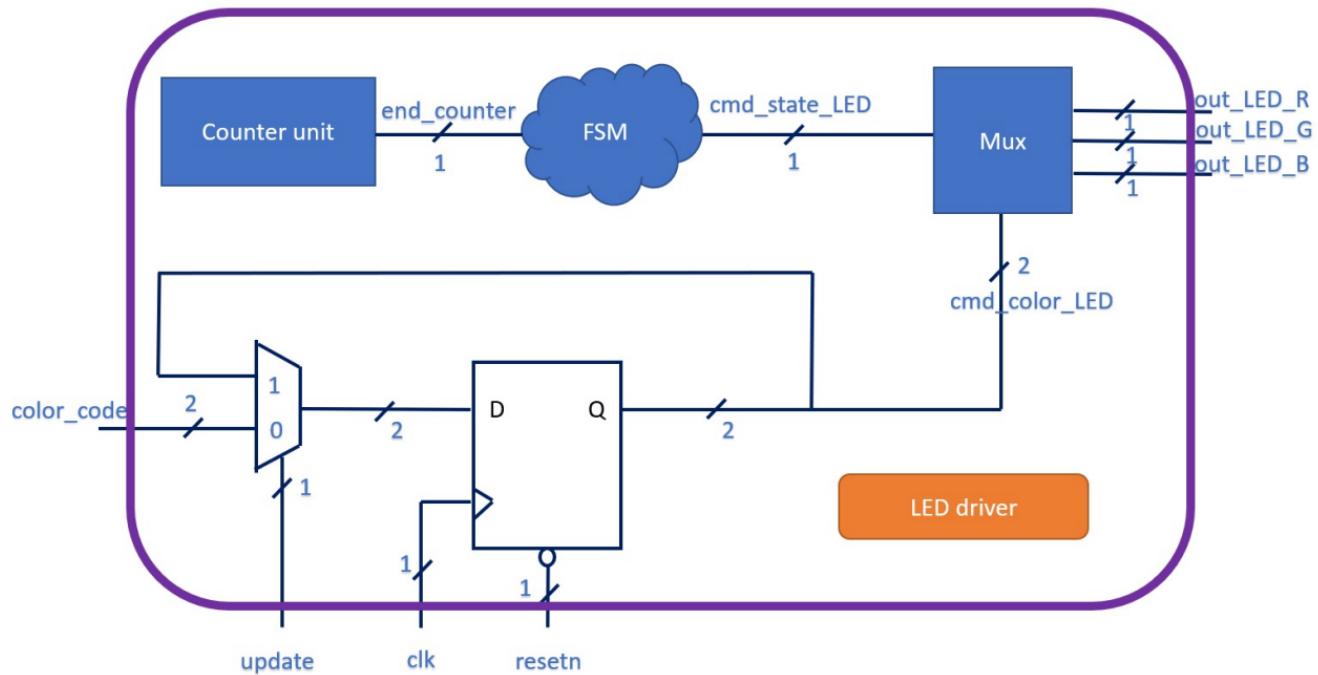
8. Créez un module de pilotage d'une LED RGB en RTL. Ce dernier doit permettre de faire clignoter une LED RGB connectée en sortie d'une couleur définie par un code couleur donné en entrée. Le changement de couleur de la LED RGB n'a lieu que si un signal *update* est reçu.



A titre d'exemple voici une table mettant en relation un code couleur avec une couleur de la LED RGB :

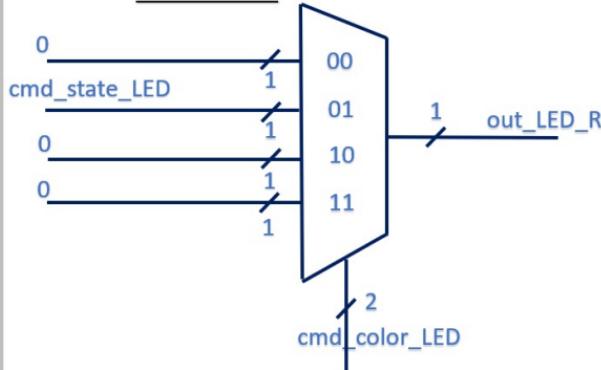
Code couleur	Couleur de la LED RGB
01	Rouge
10	Verte
11	Bleue
00	Eteinte

On peut reprendre une partie du code des exercices précédents. En effet, pour générer le signal cmd_state_LED le principe reste identique. Ainsi, on utilisera le code VHDL du counter unit et de la machine à états de la question 3.



Détails du bloc MUX

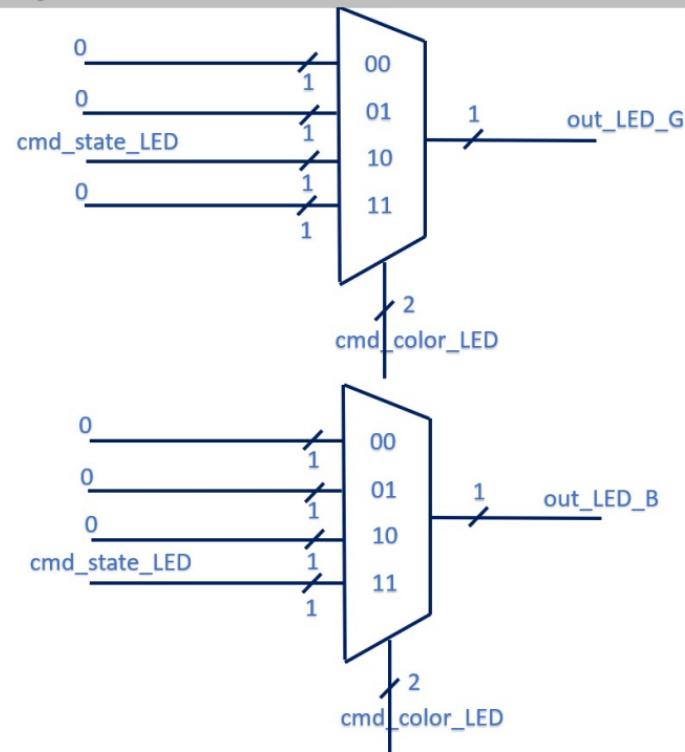
SOLUTION 1



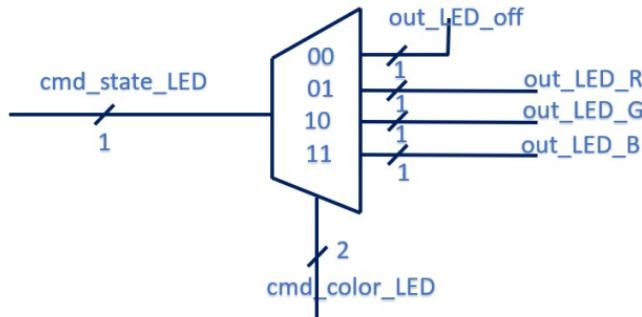
```
-- PARTIE COMBINATOIRE
-- Clignotement des LEDs
out_LED_R <= cmd_state_LED when cmd_color_LED = "01"
    else '0';

out_LED_G <= cmd_state_LED when cmd_color_LED = "10"
    else '0';

out_LED_B <= cmd_state_LED when cmd_color_LED = "11"
    else '0';
```

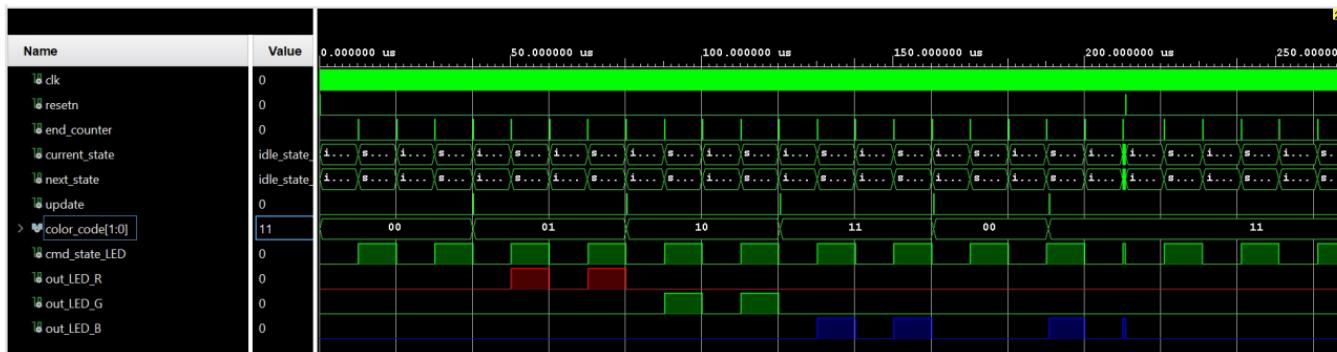


SOLUTION 2

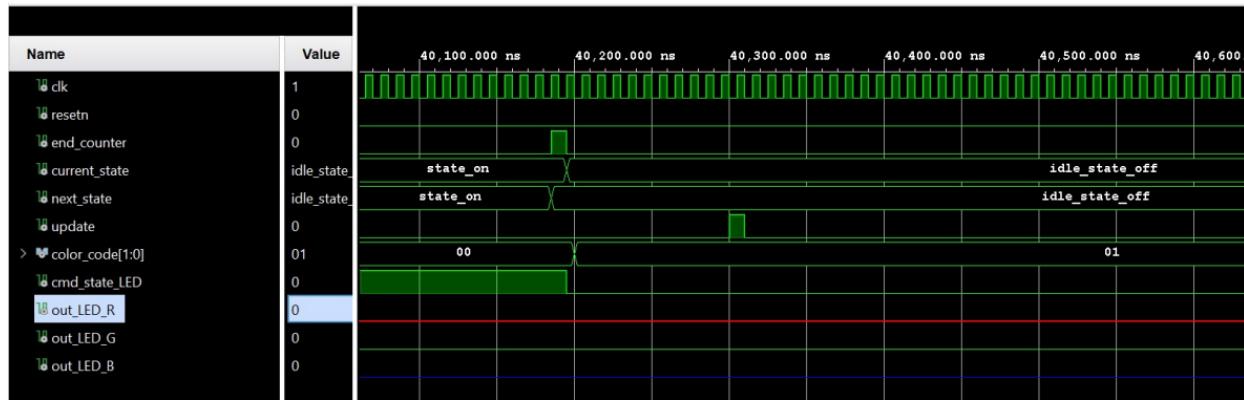


Par la suite, nous utiliserons la solution 1 avec des multiplexeurs.

Vérification au testbench du fonctionnement



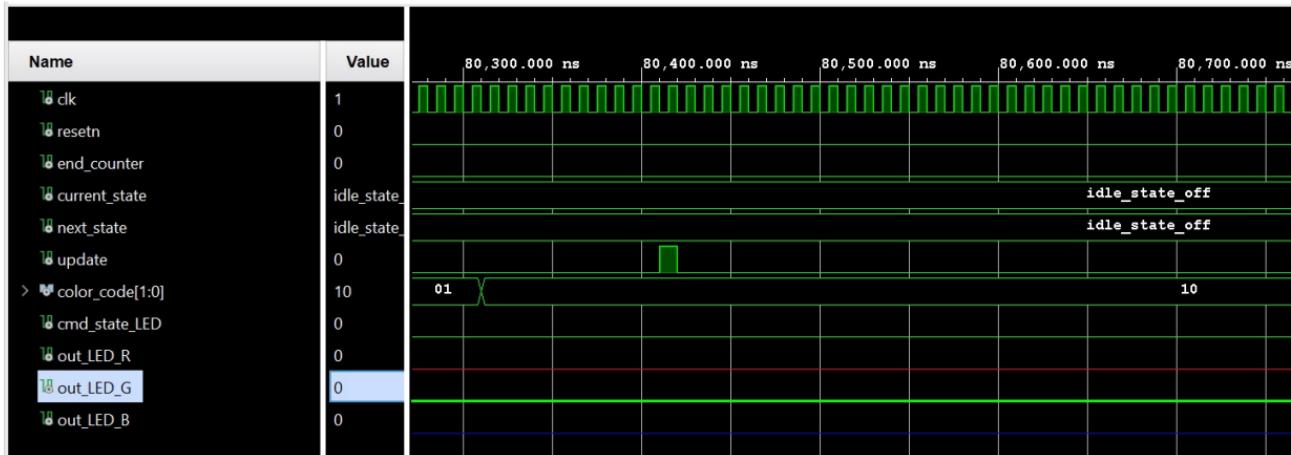
TEST AVEC LA LED ROUGE



ZOOM OUT pour observer le clignotement



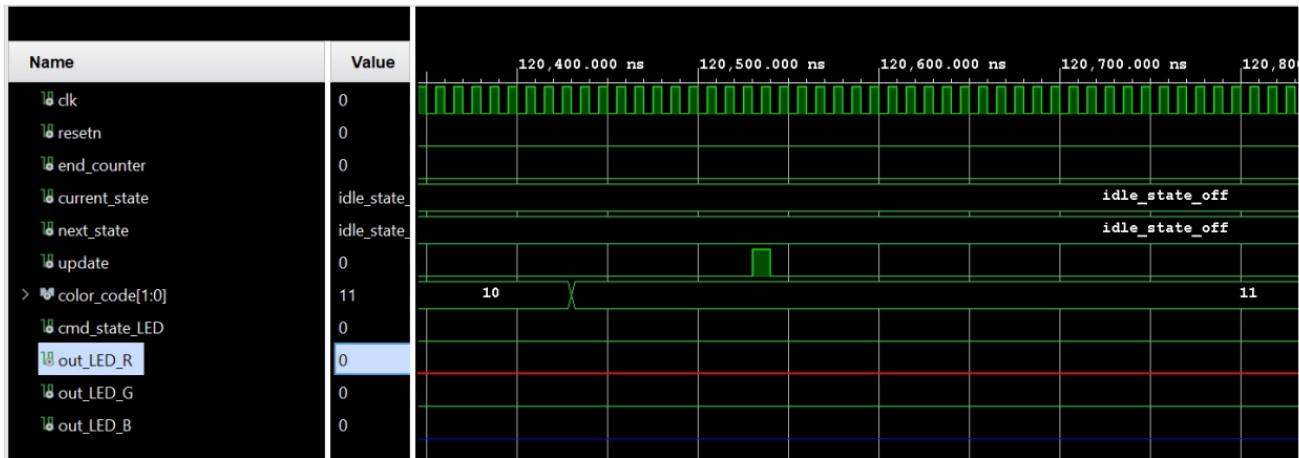
TEST AVEC LA LED Verte



ZOOM OUT pour observer le clignotement



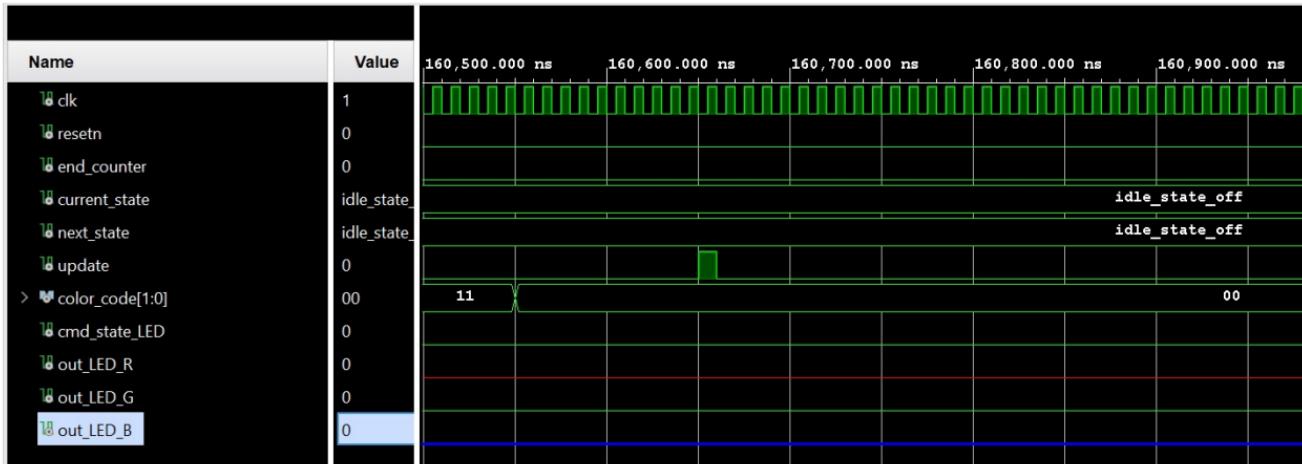
TEST AVEC LA LED BLEUE



ZOOM OUT pour observer le clignotement



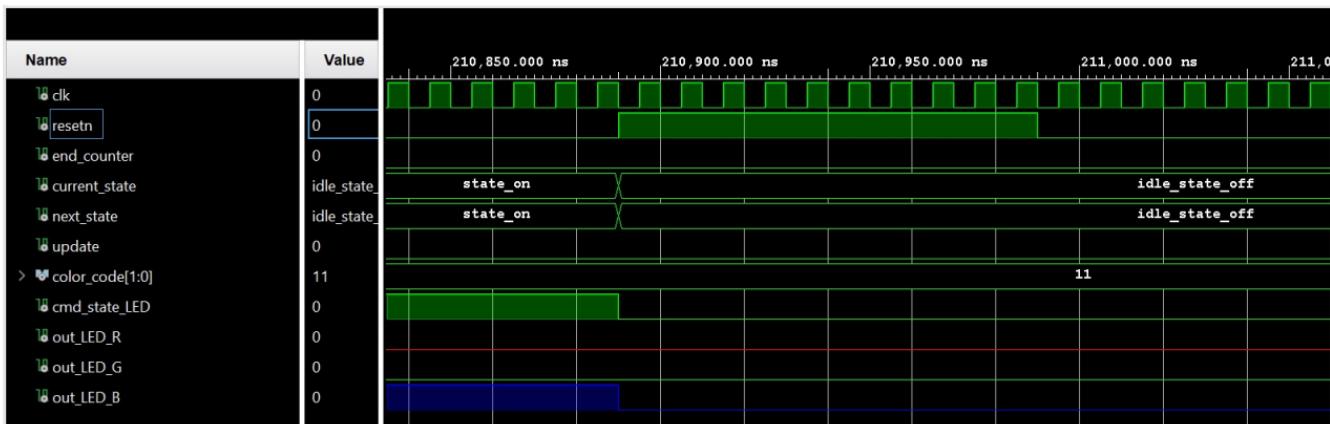
TEST AVEC LE COULEUR '00' => Vérification que les 3 LED s cessaient de clignoter



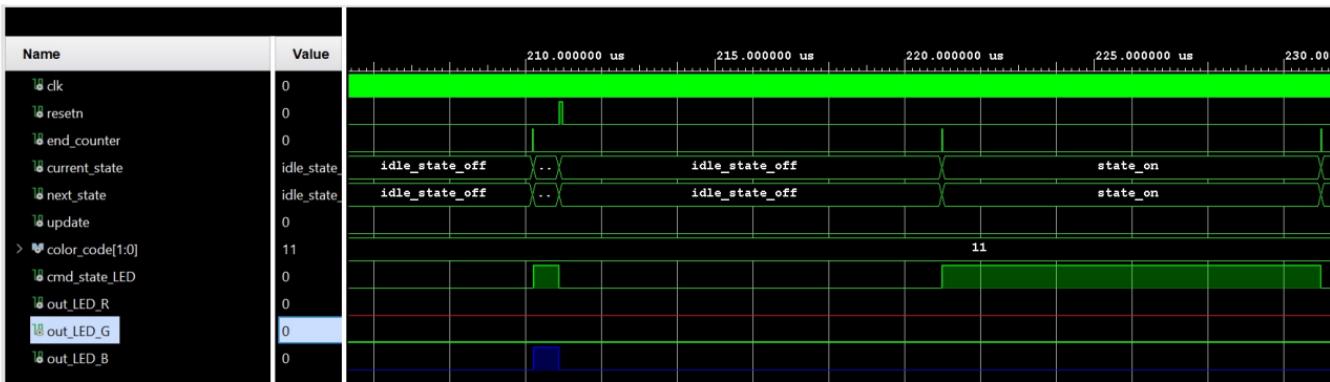
ZOOM OUT pour observer l'arrêt du clignotement



TEST DU RESET



ZOOM OUT pour observer l'arrêt du clignotement

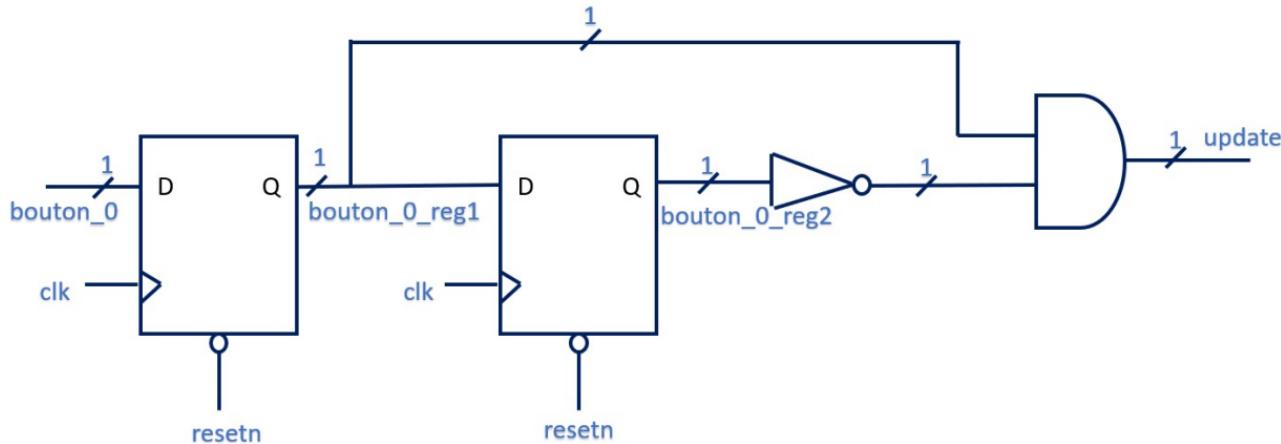


9. Ajouter la logique nécessaire pour piloter les entrées/sorties de votre module.

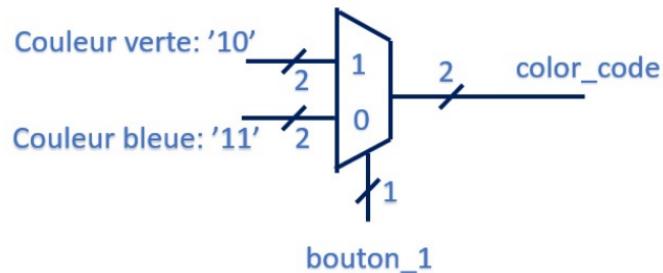
- Le signal *update* doit recevoir 1 uniquement lorsque le bouton *_0* vient d'être appuyé, maintenir le bouton enfoncé ne doit pas maintenir le signal *update* à 1
 - Le signal *color_code* doit recevoir soit le code couleur « vert » si le bouton *_1* est pressé, il doit recevoir le code couleur « bleu » sinon
 - Les LED R, G et B sont connectées avec les couleurs respectives de la LED *_0*
-

- Le signal *update* doit recevoir 1 uniquement lorsque le bouton *_0* vient d'être appuyé, maintenir le bouton enfoncé ne doit pas maintenir le signal *update* à 1

On reprend le principe de fonctionnement de la question 5 en l'adaptant.



- Le signal *color_code* doit recevoir soit le code couleur « vert » si le bouton_1 est pressé, il doit recevoir le code couleur « bleu » sinon



10. Ecrivez le code VHDL correspondant à votre architecture.

Fichier tp_pilotage_LED.vhd

C:/FORMATION_SAFRAN/TP/TP4_PilotageLED_Memoire/tp_pilotage_LED.vhd

```
Q |  ↺ | ↾ | X | ⌂ | // | ⌃ | ? |  
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_pilotage_LED is  
8     port (  
9         clk          : in std_logic;      -- horloge  
10        resetn       : in std_logic;      -- bouton de reset  
11        bouton_0     : in std_logic;      -- bouton autorisant la mise a jour de la couleur  
12        bouton_1     : in std_logic;      -- bouton pilotant la couleur (verte si 1, bleue si 0)  
13        out_LED_G    : out std_logic;     -- etat de la LED verte  
14        out_LED_B    : out std_logic;     -- etat de la LED bleue  
15    );  
16 end tp_pilotage_LED;  
17  
18  
19 architecture behavioral of tp_pilotage_LED is  
20  
21     signal update   : std_logic := '0';           -- signal autorisant la mise a jour de la couleur  
22     signal color_code : std_logic_vector(1 downto 0) := "11"; -- Couleur de la LED  
23     signal bouton_0_regl : std_logic := '0';          -- Signal bouton_0 après le premier registre  
24     signal bouton_0_reg2 : std_logic := '0';          -- Signal bouton_0 après le second registre  
25  
26  
27     --Declaration de l'entite LED_driver_unit  
28 component LED_driver_unit  
29     port (  
30         clk          : in std_logic;  
31         resetn       : in std_logic;  
32         update       : in std_logic := '0';      -- signal autorisant la mise a jour de la couleur  
33         color_code   : in std_logic_vector(1 downto 0) := "11";      -- couleur de LED  
34         out_LED_R    : out std_logic;      -- etat de la LED rouge  
35         out_LED_G    : out std_logic;      -- etat de la LED verte  
36         out_LED_B    : out std_logic;      -- etat de la LED bleue  
37     );  
38 end component;
```

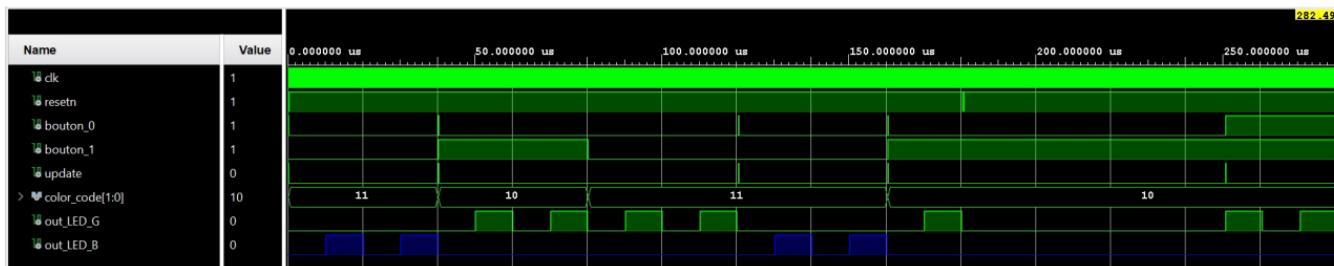
```
54
55    -- PARTIE SEQUENTIELLE
56    process(clk, resetn)
57    begin
58        if resetn = '0' then
59            bouton_0_reg1 <= '0';
60            bouton_0_reg2 <= '0';
61
62        elsif rising_edge(clk) then
63            -- Gestion du bouton_1: s'il est appuyé, couleur verte, sinon bleue.
64            if bouton_1 = '1' then
65                color_code <= "10";
66            else
67                color_code <= "11";
68            end if;
69
70            -- Gestion du bouton_0: registres
71            bouton_0_reg2 <= bouton_0_reg1;
72            bouton_0_reg1 <= bouton_0;
73
74        end if;
75    end process;
76
77    -- PARTIE COMBINATOIRE
78    -- Gestion du bouton_1: s'il est appuyé, couleur verte, sinon bleue.
79    color_code <= "10" when bouton_1 = '1'
80    else "11";
81
82    -- Gestion du bouton_0: s'il est appuyé, le signal update prend la valeur '1' le temps d'un cop d'horloge.
83    update <= '1' when (bouton_0_reg1 = '1') AND NOT(bouton_0_reg2 = '1')
84    else '0';
85
86 end behavioral;
```

11. Ecrivez un testbench pour tester votre design.

Fichier disponible dans tb_tp_pilotage_LED.vhd

12. Vérifier votre résultat à la simulation.

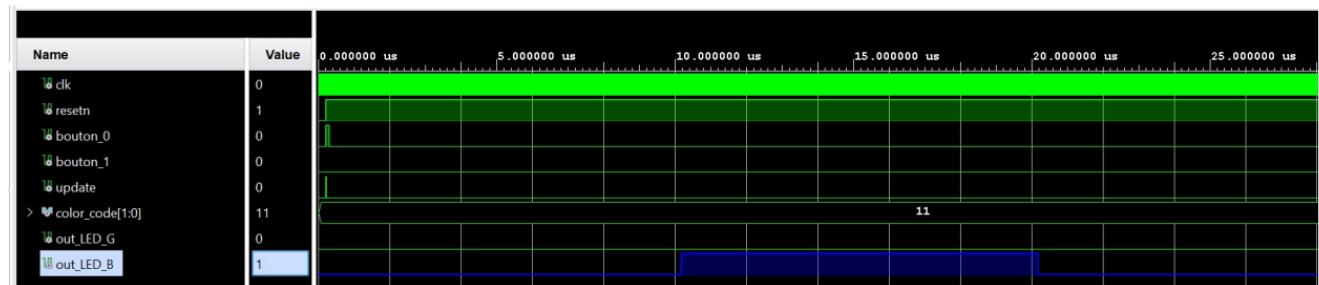
VUE D'ENSEMBLE DES TESTS



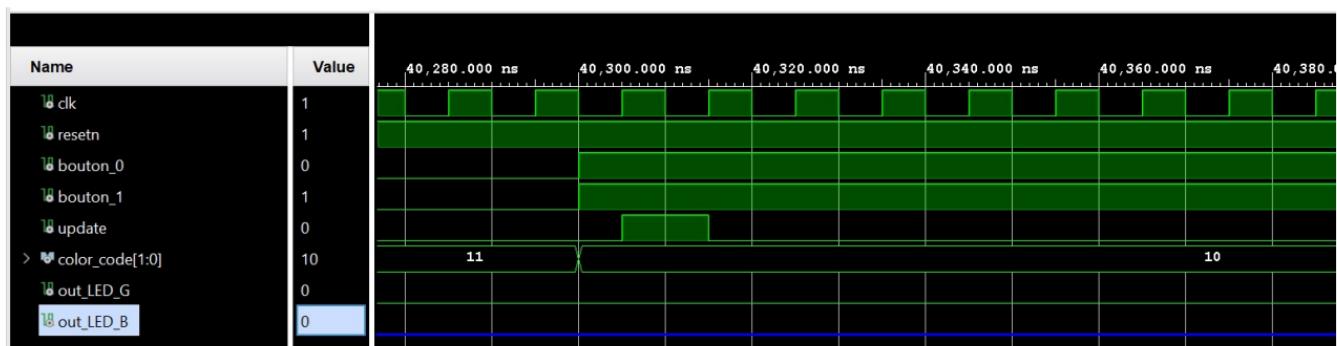
PREMIER TEST: CLIGNOTEMENT DE LA LED BLEUE



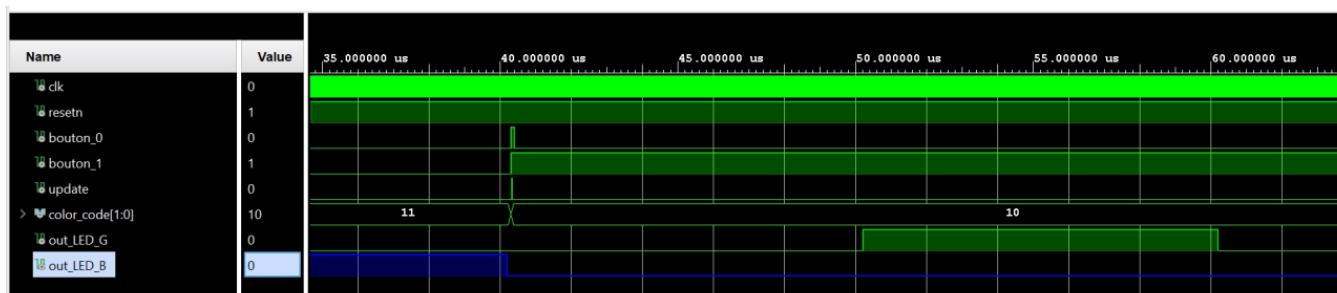
ZOOM OUT pour observer le clignotement



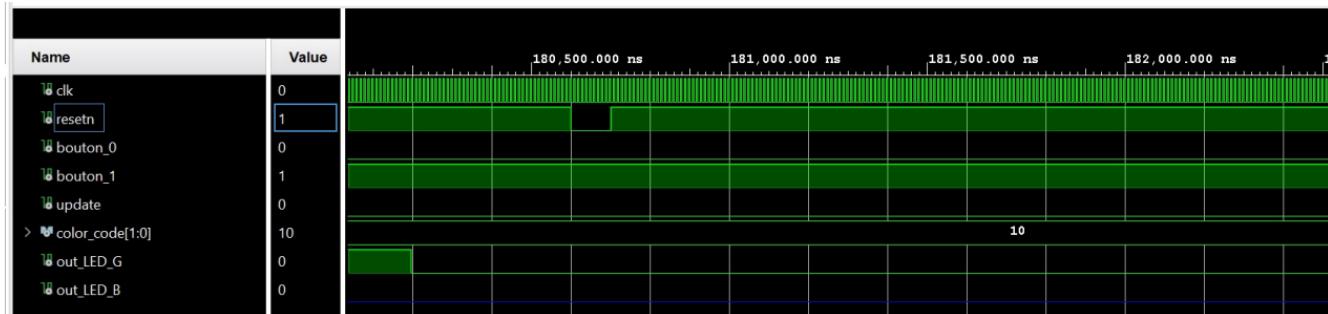
DEUXIEME TEST: CLIGNOTEMENT DE LA LED VERTE



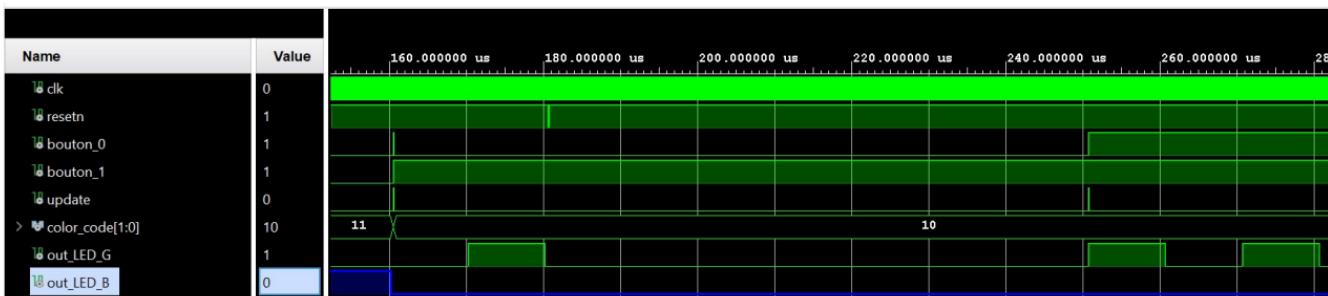
ZOOM OUT pour observer le clignotement



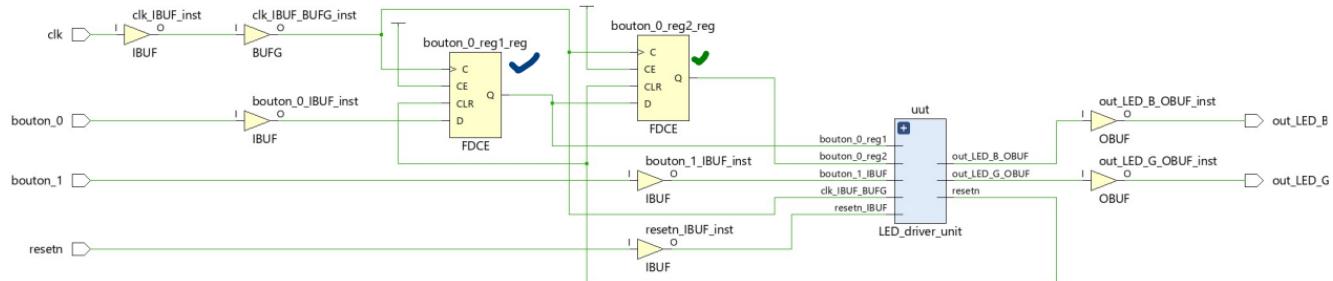
DERNIER TEST: TEST DU RESET



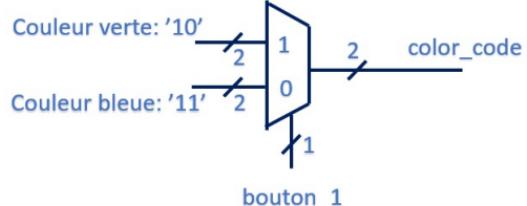
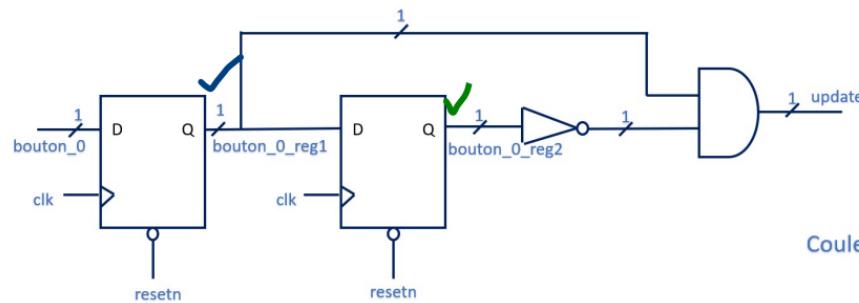
Le choix du fonctionnement du reset a été de stopper le clignotement des LEDs. Au prochain appui sur le bouton_0, les LEDs se remettent à clignoter en fonction de la position du bouton_1.



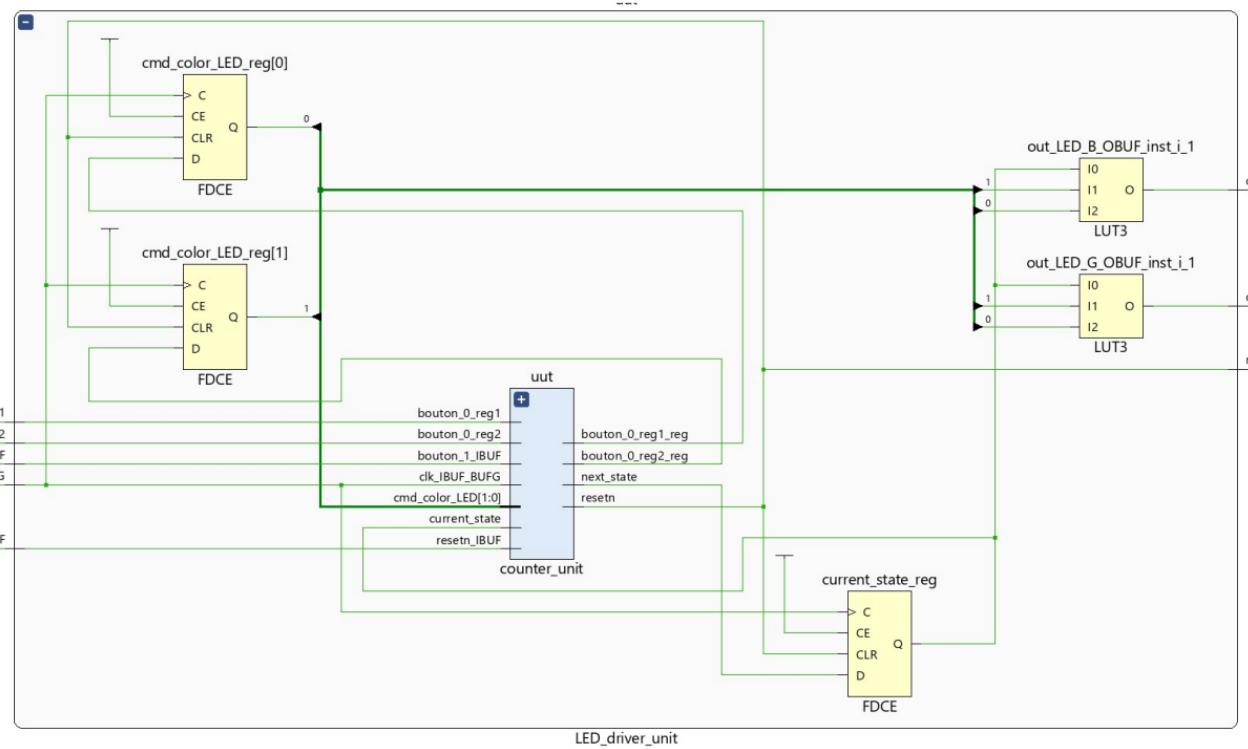
13. Réalisez une synthèse et étudiez le rapport de synthèse, les ressources utilisées doivent correspondre à votre schéma RTL.



On retrouve bien les registres des schémas RTL.



Module LED_driver



```
81 -----
82 Start RTL Component Statistics
83 -----
84 Detailed RTL Component Info :
85 +---Registers :
86           2 Bit      Registers := 1
87           1 Bit      Registers := 3
88 +---Muxes :
89       2 Input    2 Bit      Muxes := 1
90       2 Input    1 Bit      Muxes := 1
91 -----
92 Finished RTL Component Statistics
93 -----
```

2 registres à 1 bit:

- bouton_0_reg1
- bouton_0_reg2

1 registre à 1 bit à l'intérieur du module LED_driver:

- Etat machine à états

1 registre à 2 bits à l'intérieur du module LED_driver:

- cmd_color_LED

```
185 Report Cell Usage:  
186 +-----+-----+-----+  
187 |       |Cell    |Count   |  
188 +-----+-----+-----+  
189 | 1     |BUFG    |      1|  
190 | 2     |CARRY4  |      7|  
191 | 3     |LUT1    |      1|  
192 | 4     |LUT3    |      3|  
193 | 5     |LUT4    |      2|  
194 | 6     |LUT6    |      36|  
195 | 7     |FDCE    |      32|  
196 | 8     |IBUF    |      4|  
197 | 9     |OBUF    |      2|  
198 +-----+-----+-----+
```

7 CARRY4:

Composants permettant l'optimisation
des fonctions de comptage

LUT:

Look Up Tables pour la logique
combinatoire (portes et multiplexeurs)

IBUF - 4 entrées:

- clk
- resetn
- bouton_0
- bouton_1

32 registres FDCE:

- 27 utilisés pour le module counter_unit
- 2 utilisés pour le comptage de cycle
- 2 utilisés pour la sélection de la couleur
- 1 utilisé pour la machine à états

OBUF - 2 sorties:

- out_LED_B
- out_LED_G

14. Effectuez le placement routage et étudiez les rapports.

124	-----						-----							
125	Design Timing Summary						-----							
126	-----						-----							
127	-----						-----							
128	WNS (ns)						TNS (ns) TNS Failing Endpoints TNS Total Endpoints						WFTWS (ns)	
129	-----						-----						TFWTS (ns)	
130	5.005						0.000 0 31						-----	
131	-----						0.235						4.500	
132	-----						0.000						0.000	

Total Negative Slack = 0
=> Pas de violation de setup

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

Source et destination du chemin critique

191 Max Delay Paths
192 -----
193 Slack (MET) : 5.005ns (required time - arrival time)
194 Source: uut/uut/Q_out_cnt_reg[25]/C
195 (rising edge-triggered cell FDCE clocked by sys_clk_pin {rise@0.000ns fall@5.000ns period=10.000ns})
196 Destination: uut/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: 5.068ns (logic 2.224ns (43.879%) route 2.844ns (56.121%))
202 Logic Levels: 9 (CARRY4=7 LUT6=2)
203 Clock Path Skew: 0.000ns (DCD - SCD + CPR)
204 Destination Clock Delay (DCD): 4.911ns = (14.911 - 10.000)
205 Source Clock Delay (SCD): 5.363ns
206 Clock Pessimism Removal (CPR): 0.453ns
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 H16 0.000 0.000 r
217 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=32, routed) 1.729 5.363 uut/uut/clk_IBUF_BUFG
223 SLICE_X42Y68 FDCE r uut/uut/Q_out_cnt_reg[25]/C
224 SLICE_X42Y68 FDCE (Prop_fdce_C_Q) 0.518 5.881 r uut/uut/Q_out_cnt_reg[25]/Q
225 net (fo=3, routed) 0.849 6.730 uut/uut/Q_out_cnt_reg[25]
226 SLICE_X43Y67 LUT6 (Prop_lut6_I4_O) 0.124 6.854 f uut/uut/current_state_i_6/o
227 net (fo=29, routed) 1.807 8.661 uut/uut/current_state_i_6_n_0
228 SLICE_X43Y62 LUT6 (Prop_lut6_I5_O) 0.124 8.785 r uut/uut/Q_out_cnt[0].i_2/o
229 net (fo=1, routed) 0.189 8.974 uut/uut/Q_out_cnt[0].i_2_n_0
230 SLICE_X42Y62 CARRY4 (Prop_carry4_DI[0].CO[3])
231 0.550 9.524 r uut/uut/Q_out_cnt_reg[0].i_1/co[3]
232 net (fo=1, routed) 0.000 9.524 uut/uut/Q_out_cnt_reg[0].i_1_n_0
233 SLICE_X42Y63 CARRY4 (Prop_carry4_CI_CO[3])

235			0.117	9.641	r	uut/uut/Q_out_cnt_reg[4]_i_1/co[3]
236		net (fo=1, routed)	0.000	9.641		uut/uut/Q_out_cnt_reg[4]_i_1_n_0
237	SLICE_X42Y64	CARRY4 (Prop_carry4_CI_CO[3])				
238			0.117	9.758	r	uut/uut/Q_out_cnt_reg[8]_i_1/co[3]
239		net (fo=1, routed)	0.000	9.758		uut/uut/Q_out_cnt_reg[8]_i_1_n_0
240	SLICE_X42Y65	CARRY4 (Prop_carry4_CI_CO[3])				
241			0.117	9.875	r	uut/uut/Q_out_cnt_reg[12]_i_1/co[3]
242		net (fo=1, routed)	0.000	9.875		uut/uut/Q_out_cnt_reg[12]_i_1_n_0
243	SLICE_X42Y66	CARRY4 (Prop_carry4_CI_CO[3])				
244			0.117	9.992	r	uut/uut/Q_out_cnt_reg[16]_i_1/co[3]
245		net (fo=1, routed)	0.000	9.992		uut/uut/Q_out_cnt_reg[16]_i_1_n_0
246	SLICE_X42Y67	CARRY4 (Prop_carry4_CI_CO[3])				
247			0.117	10.109	r	uut/uut/Q_out_cnt_reg[20]_i_1/co[3]
248		net (fo=1, routed)	0.000	10.109		uut/uut/Q_out_cnt_reg[20]_i_1_n_0
249	SLICE_X42Y68	CARRY4 (Prop_carry4_CI_O[1])				
250			0.323	10.432	r	uut/uut/Q_out_cnt_reg[24]_i_1/o[1]
251		net (fo=1, routed)	0.000	10.432		uut/uut/Q_out_cnt_reg[24]_i_1_n_6
252	SLICE_X42Y68	FDCE			r	uut/uut/Q_out_cnt_reg[25]/D
253						-----
254						
255		(clock sys_clk_pin rise edge)				
256			10.000	10.000	r	
257	H16		0.000	10.000	r	clk (IN)
258		net (fo=0)	0.000	10.000		clk
259	H16	IBUF (Prop_ibuf_I_O)	1.387	11.387	r	clk_IBUF_inst/O
260		net (fo=1, routed)	1.880	13.267		clk_IBUF
261	BUFGCTRL_X0Y16	BUFG (Prop_bufg_I_O)	0.091	13.358	r	clk_IBUF_BUFG_inst/O
262		net (fo=32, routed)	1.553	14.911		uut/uut/clk_IBUF_BUFG
263	SLICE_X42Y68	FDCE			r	uut/uut/Q_out_cnt_reg[25]/C
264		clock pessimism	0.453	15.363		
265		clock uncertainty	-0.035	15.328		
266	SLICE_X42Y68	FDCE (Setup_fdce_C_D)	0.109	15.437		uut/uut/Q_out_cnt_reg[25]
267						-----
268		required time		15.437		
269		arrival time		-10.432		
270						-----
271		slack		5.005		

15. Générez le bitstream et vérifiez que vous avez le comportement attendu sur carte.

A ce stade, aucune LED n'était allumée. En menant l'enquête avec ILA, je me suis aperçu que le signal était en permanence à 1. En effet, n'ayant pas suffisamment de bouton sur la carte, j'ai assigné le resetn au pin Y18.

```
22 ## Pmod Header JA
23 set_property -dict {PACKAGE_PIN Y18 IOSTANDARD LVCMOS33} [get_ports resetn]
24 #set_property -dict { PACKAGE_PIN Y19 IOSTANDARD LVCMOS33 } [get_ports { ja[1] }]; #IO_L17N_T2_34 Sch=ja_n[1]
```

Ainsi, j'ai revu mon code en modifiant le fonctionnement du resetn dans les différents fichiers de source.

```
54      -- PARTIE SEQUENTIELLE
55 process(clk, resetn)
56 begin
57     if resetn = '0' then
```

Suite à cette modification, le fonctionnement est bien celui attendu.