

Segment 2

TP4 : Led driver

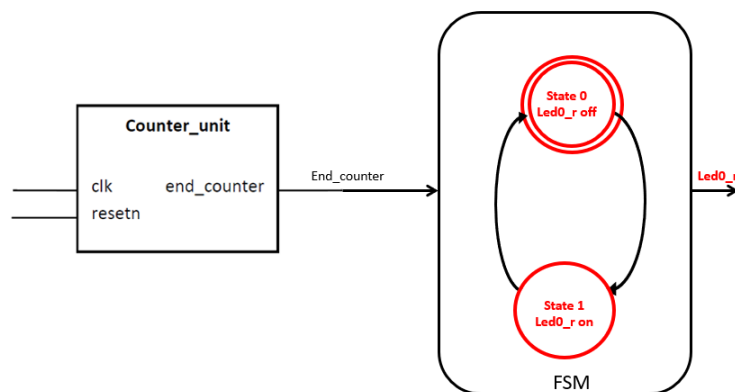
Objectif

L'objectif de cet TP est de réaliser un design permettant de faire clignoter une LED RGB avec une séquence de couleurs entrées à l'aide des boutons. Dans cette partie, nous définirons un module LED_driver .

Questions

1. 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.

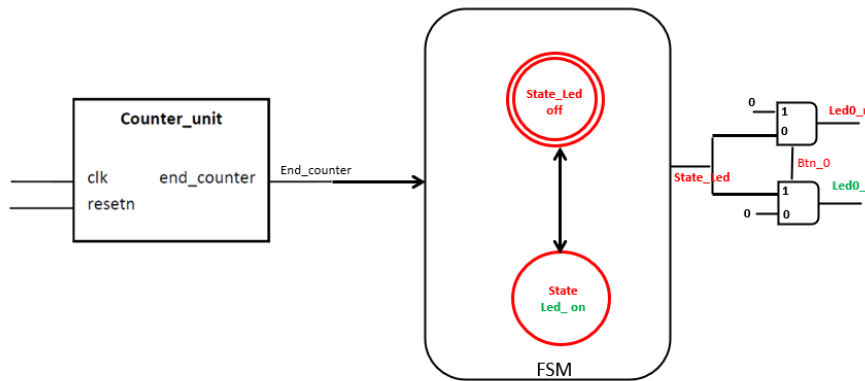
Pour répondre à cette question, nous allons utiliser le même schéma RTL du module counter_unit qui permet de faire clignoter une led. Pour ce faire, nous utilisons le bloc counter_unit et un bloc FSM qui gère les états de la led rouge allumée et éteinte. Notre état initial est la led éteinte. L'état suivant est la led allumée. On a idle=OFF, et State1=ON.



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.

Pour ce faire nous rajoutons un bouton_0(btn_0) qui permet de piloter la led verte et la led rouge. Lorsque le btn_0=1, la led verte est allumée et sinon la led rouge est allumée. Nous utilisons deux multiplexeurs qui nous permette d'envoyer la commande du signal btn_0.

Pour allumer la led verte, il faut que state_led='1' et btn_0='1'. Pour allumer la led rouge, il faut que state_led='1' et btn_0='0'. Et lorsque state_led='0', les deux leds sont éteintes.



3. Rédigez le code VHDL de votre architecture.

Pour cette partie, je conserve toujours le module counter_unit et je crée un code FSM pour l'état de la led. Et un multiplexeur pour gérer les états allumés/éteints des leds.

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  use ieee.std_logic_arith;
5
6  entity tp_leddriver is
7      generic (
8          --max_count : integer :=20000000;
9          --nb_bit : integer := 28;
10         max_count : integer :=4;
11         nb_bit : integer := 3;
12         nbre_cycle : positive :=2 --cycle de clignotement allumé et éteint
13     );
14     port (
15         clk : in std_logic; -- Signal d'horloge
16         resetn : in std_logic; -- Signal de reset
17         --a completer
18         btn_0 : in std_logic; -- Bouton d'entrée
19         led_out_R : out std_logic; -- LED rouge
20         led_out_V : out std_logic; -- LED verte
21     );
22 end tp_leddriver;
23
24
25 architecture behavioral of tp_leddriver is
26
27     type state is (idle, statel); --a modifier avec vos etats
28
29     signal current_state : state; --etat dans lequel on se trouve actuellement
30     signal next_state : state; --etat dans lequel on passera au prochain coup d'horloge
31     signal s_led_out_V : std_logic;
32     signal s_led_out_R : std_logic;
33     signal end_count : std_logic;
34     signal state_led: std_logic; -- état des leds
35
36     --Declaration de l'entite a tester
37     component counter_unit
38     generic (
39         --max_count : integer :=20000000;
40         -- nb_bit : integer := 28
41         max_count : integer :=4;
42         nb_bit : integer := 3
43     );
44     port (
45         clk : in std_logic;
46         resetn : in std_logic;
47         end_counter : out std_logic
48     );
49 end component;
50
51 begin

```

```

53  compreur : counter_unit
54  generic map (
55      --max_count => 20000000,
56      -- nb_bit => 28
57      max_count => 4,
58      nb_bit => 3
59  )
60  port map (
61      clk => clk,
62      resetn => resetn,
63      end_counter => end_count
64  );
65
66  -- Process séquentielle Logique de la machine à états
67  process(clk, resetn, end_count)
68  begin
69      if resetn = '1' then
70          current_state <= idle;
71      elsif rising_edge(end_count) then
72          current_state <= next_state;
73      end if;
74  end process;
75
76  -- FSM Logique de contrôle de la machine à états
77  process(current_state, btn_0, state_led, end_count) --à compléter avec vos signaux
78  begin
79      --signaux pilotes par la fsm état ou les leds sont éteintes
80
81      case current_state is
82      when idle => -- idle= état de led éteinte
83          state_led <= '0';
84          next_state <= state1; -- prochain état idle= état de led allumée
85
86      --signaux pilotes par la fsm état ou les leds sont allumées
87      when state1 => -- state1= état de led allumée
88          state_led <= '1';
89          next_state <= idle; -- prochain état idle= état de led éteinte
90
91      end case;
92  end process;
93
94  --Partie combinatoire logique de controle des Leds
95
96  s_led_out_R <= '1' when (state_led='1' and btn_0='0')
97  else '0';
98
99  s_led_out_V <= '1' when (state_led='1' and btn_0='1')
100 else '0';
101
102 led_out_R <= s_led_out_R; -- led rouge
103 led_out_V <= s_led_out_V; -- led verte
104
105
106 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 ?

```

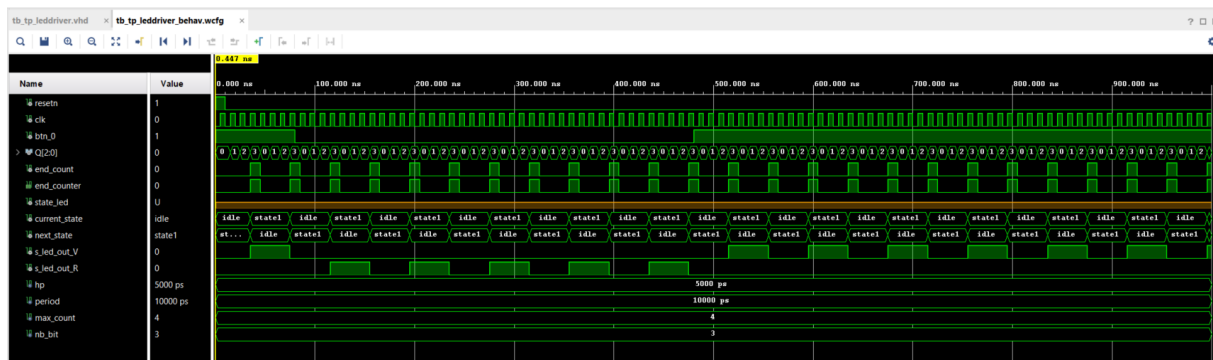
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  use ieee.std_logic_arith;
5
6
7  entity tb_tp_leddriver is
8  end tb_tp_leddriver;
9
10 architecture behavioral of tb_tp_leddriver is
11
12     signal resetn : std_logic := '0';
13     signal clk : std_logic := '0';
14     signal btn_0 : std_logic := '0';
15     signal s_led_out_V : std_logic;
16     signal s_led_out_R : std_logic;
17     signal end_count : std_logic;
18     signal state_led : std_logic; -- état des leds
19     -- Les constantes suivantes permette de definir la frequence de l'horloge
20     constant hp : time := 5 ns; --demi periode de 5ns
21     constant period : time := 2*hp; --periode de 10ns, soit une frequence de 100Hz
22     --constant max_count : integer := 20000000;
23     --constant nb_bit : integer := 28;
24     constant max_count : integer := 4;
25     constant nb_bit : integer := 3;
26
27     --Déclaration du composant fsm
28     component tp_leddriver
29     port (
30         clk : in std_logic; -- Signal d'horloge
31         resetn : in std_logic; -- Signal de reset
32         --à compléter
33         btn_0 : in std_logic; -- Bouton d'entrée
34         led_out_R : out std_logic; -- LED rouge
35         led_out_V : out std_logic; -- LED verte
36     );
37 end component;
38
39 begin
40     dut: tp_leddriver
41     port map (
42         clk => clk, -- Signal de reset
43         resetn => resetn,
44         --à compléter
45         btn_0 => btn_0, -- Bouton d'entrée
46         led_out_R => s_led_out_R, -- LED rouge
47         led_out_V => s_led_out_V -- LED verte
48     );
49

```

```

50      --Simulation du signal d'horloge en continue
51      process
52      begin
53          wait for hp;
54          clk <= not clk;
55      end process;
56      --Simulation du signal rstn en continue avec affichage de end_counter qui fonctionne après le cycle allumée éteint
57      process
58      begin
59          resetn <= '1';
60          --wait for hp;
61          wait for period*1;
62          resetn <= '0';
63          assert end_count='0'
64          report "end_counter : test failed";
65          wait for period*max_count;
66          assert end_count='1'
67          report "end_counter : test failed";
68          wait;
69      end process;
70      --simulation de leds et olignement
71      process
72      begin
73          -- btn_0
74          btn_0 <= '1';
75          -- Verifions l'état initial idle (LED blanche)
76          assert (s_led_out_R = '0' and s_led_out_V = '1')
77          report "Initial state mismatch" severity error;
78          wait for 80 ns;
79          btn_0 <= '0';
80          wait for period*4*max_count;
81          -- Verifions l'état initial idle (LED rouge allumée)
82          assert (s_led_out_R = '1' and s_led_out_V = '0')
83          report "Initial state mismatch" severity error;
84          wait for period*6*max_count;
85          btn_0 <= '1';
86          --Pour finir la simulation
87          wait;
88      end process;
89      end behavioral;

```

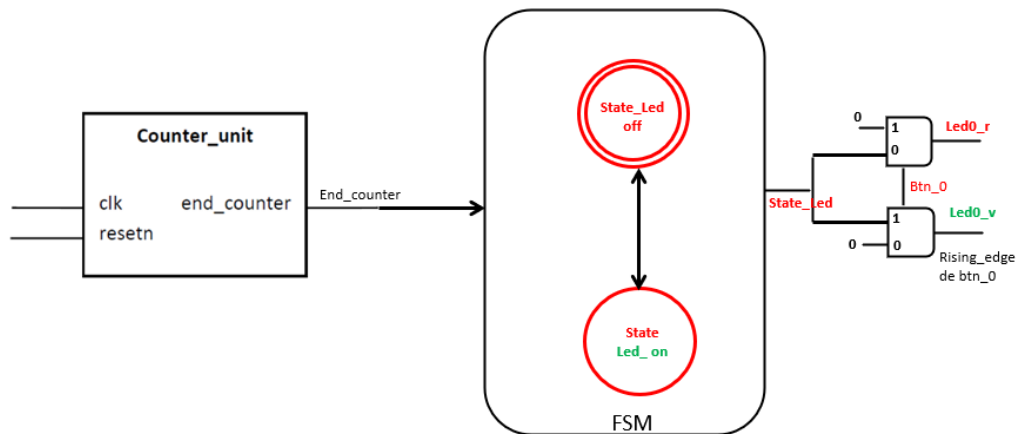


Lorsque le bouton est pressé pendant plus d'un cycle d'horloge, on observe que la led verte est allumée et s'éteint en fonction d'End_count. Lorsque le bouton passe à zéro, n'est plus pressée, la led rouge s'allume et s'éteint en fonction d'End_count.

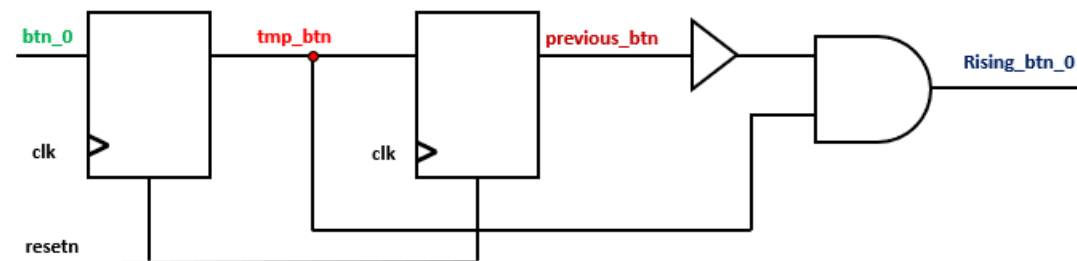
Si le bouton est longtemps pressé (plusieurs cycles d'horloges) alors la led verte clignote durant tout le cycle.

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

Pour que la led verte clignote une seule fois même si le bouton est maintenu, il faut que lors d'un front montant du bouton, la led verte clignote et sinon la led rouge clignote.



Le schéma RTL de rising_btn_0 est donné ci-dessous. Avec deux registres et une partie combinatoire, on sort le signal qui servira à piloter les leds rouge et verte.



6. Ajoutez cette solution à votre architecture RTL.

Dans la partie combinatoire, nous récrivons la condition sur du front montant de btn_0 qui est rising_btn_0

```

---Process séquentielle de risingedgebtn

process (clk, resetn)
begin
    if (resetn = '1') then
        tmp_btn      <= '0';
        previous_btn <= '0';
    elsif (rising_edge(clk) and end_count='1') then
        tmp_btn      <= btn_0;
        previous_btn <= tmp_btn;
    end if;
end process;
--Partie combinatoire
rising_btn_0 <= tmp_btn and (not( previous_btn));

```

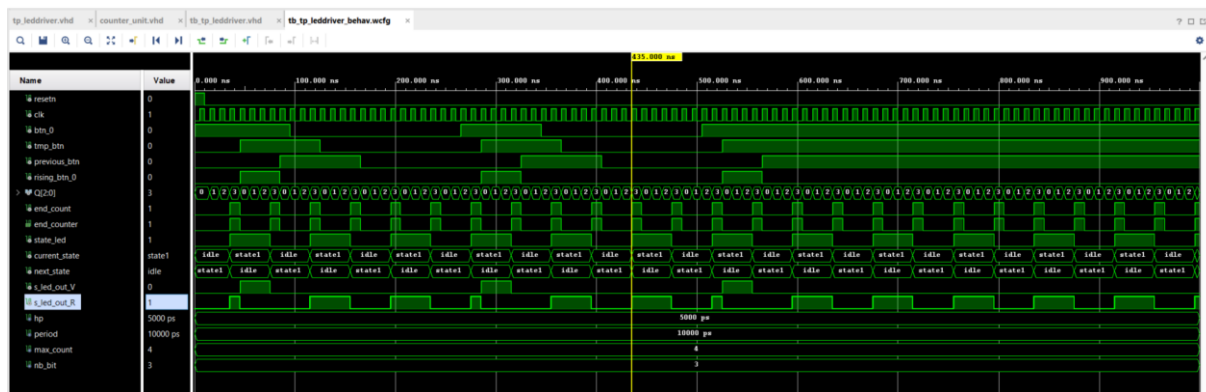
7. Mettez à jour votre code VHDL et vérifiez votre résultat à la simulation.

```

--gestion de la led rouge et verte avec la nouvelle condition avec rising_edge btn_0

s_led_out_v <= '1' when (state_led = '1' and rising_btn_0 = '1') else
    '0';
s_led_out_R <= '1' when (state_led = '1' and rising_btn_0 = '0')
else '0';

```

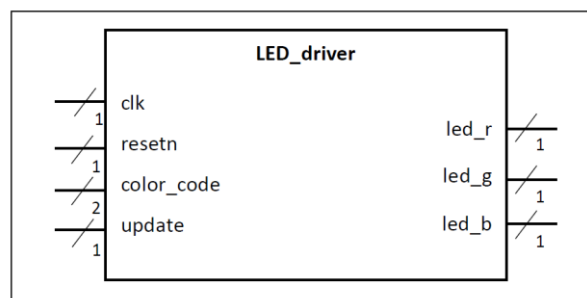


Dans cette logique combinatoire, lorsque `state_led` est '1'. Si `rising_btn_0` est '1', cela signifie que le front montant du signal `btn_0` a été détecté, donc la LED verte est allumée (`s_led_out_V` est '1') et la LED rouge est éteinte (`s_led_out_R` est '0').

Si `rising_btn_0` n'est pas actif (`rising_btn_0`='0'), mais que `btn_0` est '1', cela signifie que le bouton est maintenu enfoncé, donc la LED rouge est allumée (`s_led_out_R` est '1') et la LED verte est éteinte (`s_led_out_V` est '0'). Enfin, si `rising_btn_0` n'est pas actif et que `btn_0` est '0', cela signifie que le bouton est relâché, donc la LED rouge est allumée (`s_led_out_R` est '1') et la LED verte est éteinte (`s_led_out_V` est '0').

Lorsque `state_led` est '0', les deux LEDs sont éteintes (`s_led_out_R` et `s_led_out_V` sont '0').

- 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.

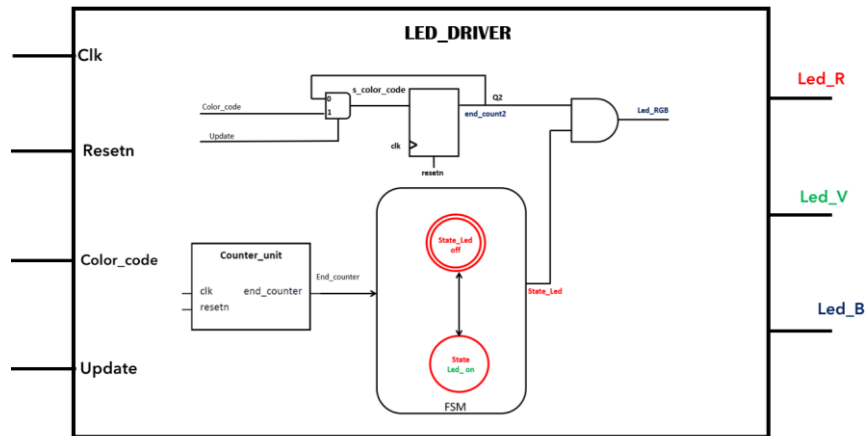


A titre d'exemple voici une table mettant en relation un code couleur avec une couleur de la LED RGB :

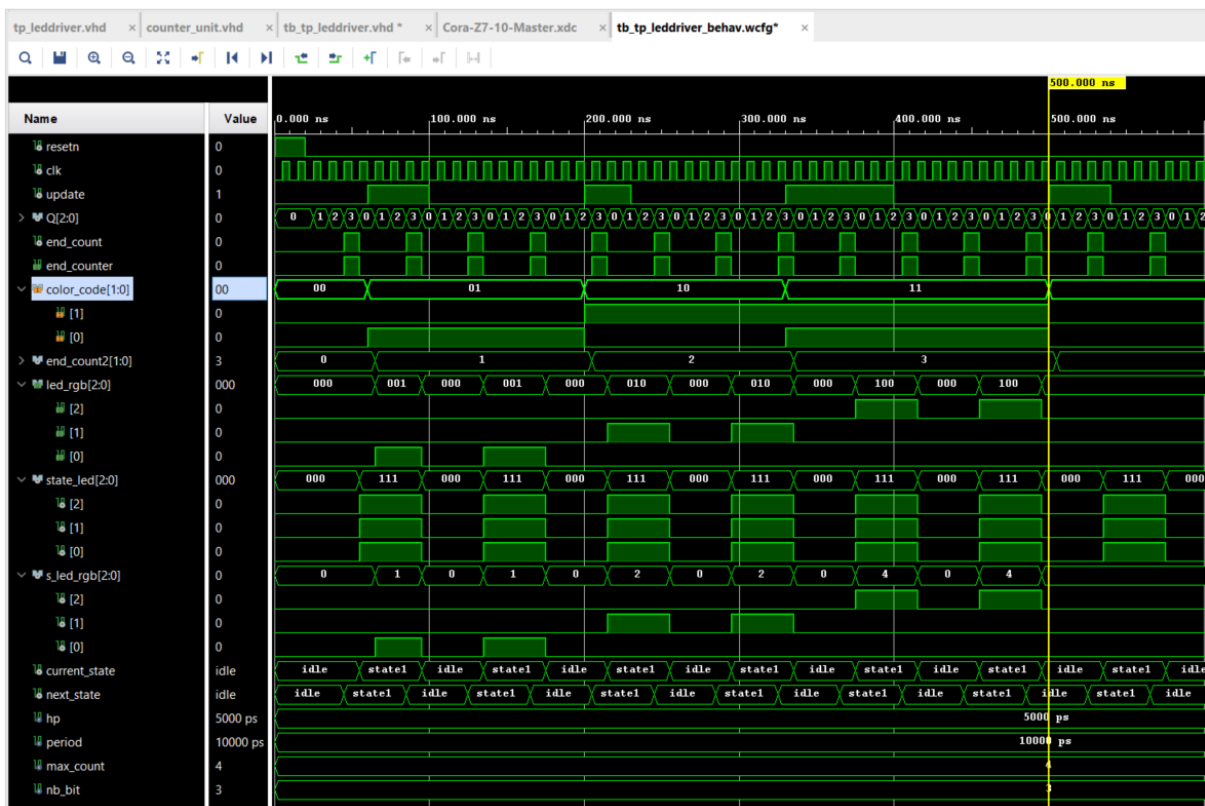
Code couleur	Couleur de la LED RGB
01	Rouge
10	Verte
11	Bleue
00	Eteinte

Pour créer le module de pilotage d'une LED RGB qui permet de clignoter une led RGB connectée en sortie d'une couleur définie par un code couleur, nous utiliserons un compteur pour contrôler le clignotement de la LED. Nous utiliserons le code couleur fourni : "00" pour

éteint, "01" pour rouge, "10" pour verte, et "11" pour bleue. La partie de la logique combinatoire mapperà les différents codes couleur sur les valeurs correspondantes de la LED RGB (rgb_led). Le clignotement de la LED est ajouté en utilisant un compteur (counter2). Le compteur compte jusqu'à une valeur max_count pour allumer la LED, puis l'éteint. Cette valeur peut être ajustée en fonction de la fréquence souhaitée du clignotement.



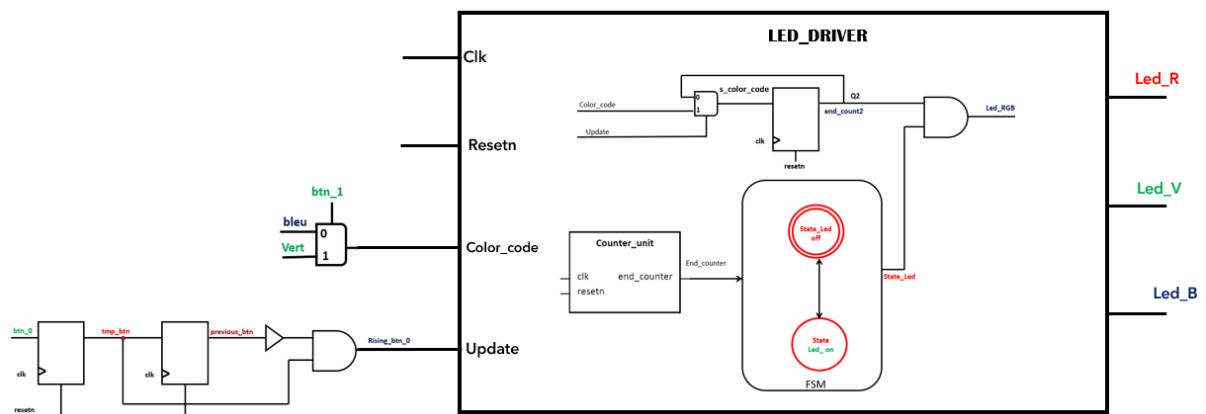
Son code vhld+testbench donne le résultat suivant :



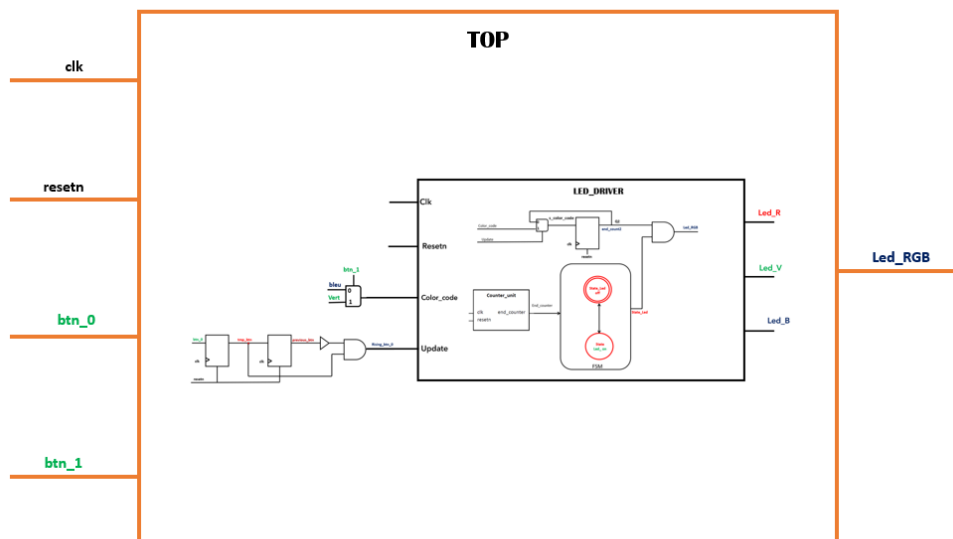
Sur le chronogramme, on voit bien le changement de color_code qui évolue de 00,01, 10, 11. On remarque que la led rouge est allumé durant end_count=01 et color_code=01. La led verte est allumé durant end_count=10 et color_code=10. La led bleue est allumé durant end_count=11 et color_code=11. La led éteinte lorsque color_code=00

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*



Pour cette partie, on ajoute un bloc pour le contrôle de l'update, ce bloc est relié au bouton *btn_0*. Avec un multiplexeur, on gère le signal *color_code* qui doit recevoir soit le code couleur « vert » si le bouton *btn_1* est pressé, et il doit recevoir le code couleur « bleu » sinon. Ce multiplexeur est relié au bouton *btn_1*. Le module top représente l'ensemble avec le *led_driver*. Ses signaux d'entrées sont : *clk*, *resetn*, *btn_0* et *btn_1*. Le signal de sortie est la led RGB.



10. Ecrivez le code VHDL correspondant à votre architecture.

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  use ieee.std_logic_arith;
5
6  entity top is
7
8      port (
9          clk          : in std_logic;  -- Signal d'horloge
10         resetn       : in std_logic;  -- Signal de reset
11         btn_0        : in std_logic;  -- Signal de mise à jour
12         btn_1        : in std_logic;  -- Signal de mise à jour
13         led_out_RGB  : out std_logic_vector(2 downto 0) -- LED RGB
14     );
15
16 end top;
17
18 architecture behavioral of top is
19
20     signal update: std_logic;
21     signal color_code : std_logic_vector(1 downto 0);
22     signal s_led_out_RGB : std_logic_vector(2 downto 0);
23
24     ----RisingEdgebtn
25     signal tmp_btn      : std_logic;
26     signal previous_btn : std_logic;
27     signal rising_btn_0 : std_logic;
28
29     --Declaration de led_driver
30     component led_driver
31
32     port (
33         clk          : in std_logic;  -- Signal d'horloge
34         resetn       : in std_logic;  -- Signal de reset
35         update       : in std_logic;  -- Signal de mise à jour
36         color_code   : in std_logic_vector(1 downto 0);
37         led_out_RGB  : out std_logic_vector(2 downto 0) -- LED Red
38     );
39
40 end component;
41
42 begin
43
44     Leddriver : led_driver
45     port map (
46         clk => clk,
47         resetn=>resetn,
48         update => update,
49         color_code => color_code,
50         led_out_RGB => s_led_out_RGB
51     );
52
53     --- process sÃ©quentielle du risingbtn_0
54
55     --- process sÃ©quentielle du risingbtn_0
56
57     process (clk, resetn)
58     begin
59         if (resetn = '1') then
60             tmp_btn      <= '0';
61             previous_btn <= '0';
62         elsif (rising_edge(clk)) then
63             tmp_btn      <= btn_0;
64             previous_btn <= tmp_btn;
65         end if;
66     end process;
67
68     --Partie de la partie combinatoire
69     rising_btn_0 <= not(tmp_btn) and ( previous_btn);
70     update<= rising_btn_0;
71
72     -- partie combinatoire des couleurs color_code
73
74     color_code<="10" when (btn_1='1')
75     else
76         "11";
77
78     --- gestion des couleurs de sortie avec le signal interne s_led_out entre la FSM et la couleur
79
80     led_out_RGB <=s_led_out_RGB;
81
82 end behavioral;

```

11. Ecrivez un testbench pour tester votre design.

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  use ieee.std_logic_arith;
5
6
7  entity tb_top is
8  end tb_top ;
9
10 architecture behavioral of tb_top is
11
12     -- signaux de top
13     signal resetn      : std_logic := '0';
14     signal clk          : std_logic := '0';
15     signal btn_0        : std_logic := '0';
16     signal btn_1        : std_logic := '0';
17     signal led_out_RGB : std_logic_vector(2 downto 0); -- état des leds
18
19
20     -- Les constantes suivantes permette de definir la frequence de l'horloge
21     constant hp : time := 5 ns;      --demi periode de 5ns
22     constant period : time := 2*hp; --periode de 10ns, soit une frequence de 100Hz
23
24     constant max_count : integer :=4;
25     constant nb_bit : integer := 3;
26
27     --Déclaration du composant fsm
28     component top
29     port (
30         clk          : in std_logic; -- Signal d'horloge
31         resetn       : in std_logic; -- Signal de reset
32         --a completer
33         btn_0        : in std_logic; -- Signal de mise à jour
34         btn_1 : in std_logic; -- Signal de mise à jour
35         led_out_RGB  : out std_logic_vector(2 downto 0) -- LED RGB
36     );
37
38 end component;
39
40 begin
41     dut: top
42     port map (
43         clk => clk,
44         resetn => resetn, -- Signal de reset
45         --a completer
46         btn_0 => btn_0 , -- Bouton d'entrée
47         btn_1 => btn_1 ,
48         led_out_RGB => led_out_RGB -- LED rouge
49     );
50
51
52     --Simulation du signal d'horloge en continue
53     process
54     begin

```

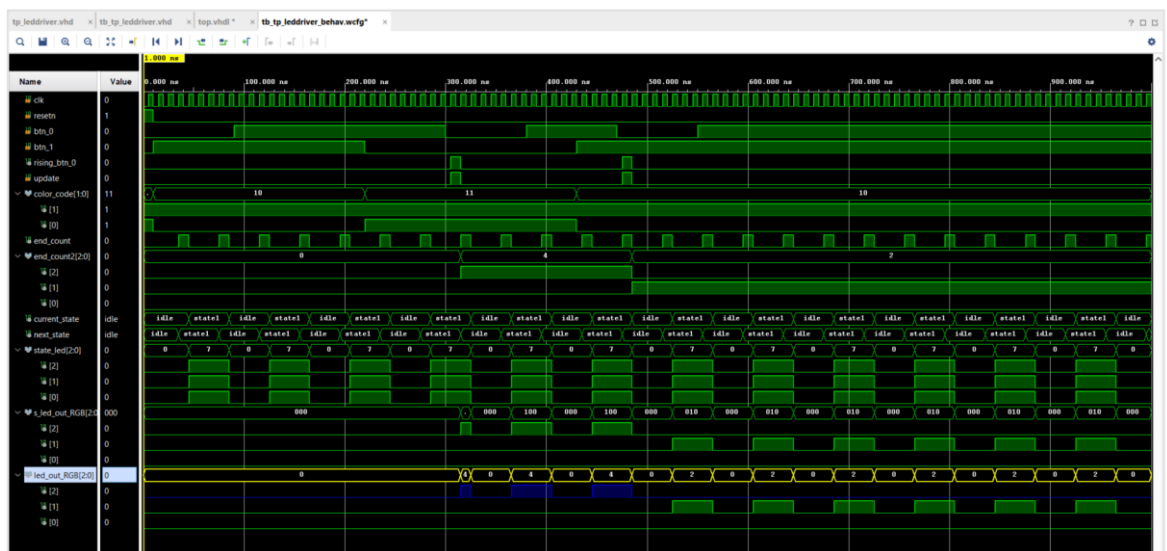
```

52      --Simulation du signal d'horloge en continue
53      process
54      begin
55          wait for hp;
56          clk <= not clk;
57
58      end process;
59
60      process
61      begin
62
63      -- Resetn
64          resetn <= '1';
65          wait for 10 ns;
66          resetn<='0';
67          -- Verifions l'état initial idle (LED éteinte)
68
69
70          btn_1<='1';
71          wait for 80ns;
72          btn_0<='1';
73          wait for 80ns;
74          -- Verifions l'état state 1 (LED verte
75
76          wait for 50 ns;
77
78          btn_1<='0';
79          wait for 80ns;
80          btn_0<='0';
81          wait for 80ns;
82          -- Verifions l'état state 1 (LED bleue)
83
84          btn_0<='1';
85
86          wait for 50 ns;
87
88          btn_1<='1';
89          wait for 40ns;
90          btn_0<='0';
91          wait for 80ns;
92          btn_0<='1';
93
94          --Pour finir la simulation
95          wait;
96      end process;
97
98  end behavioral;

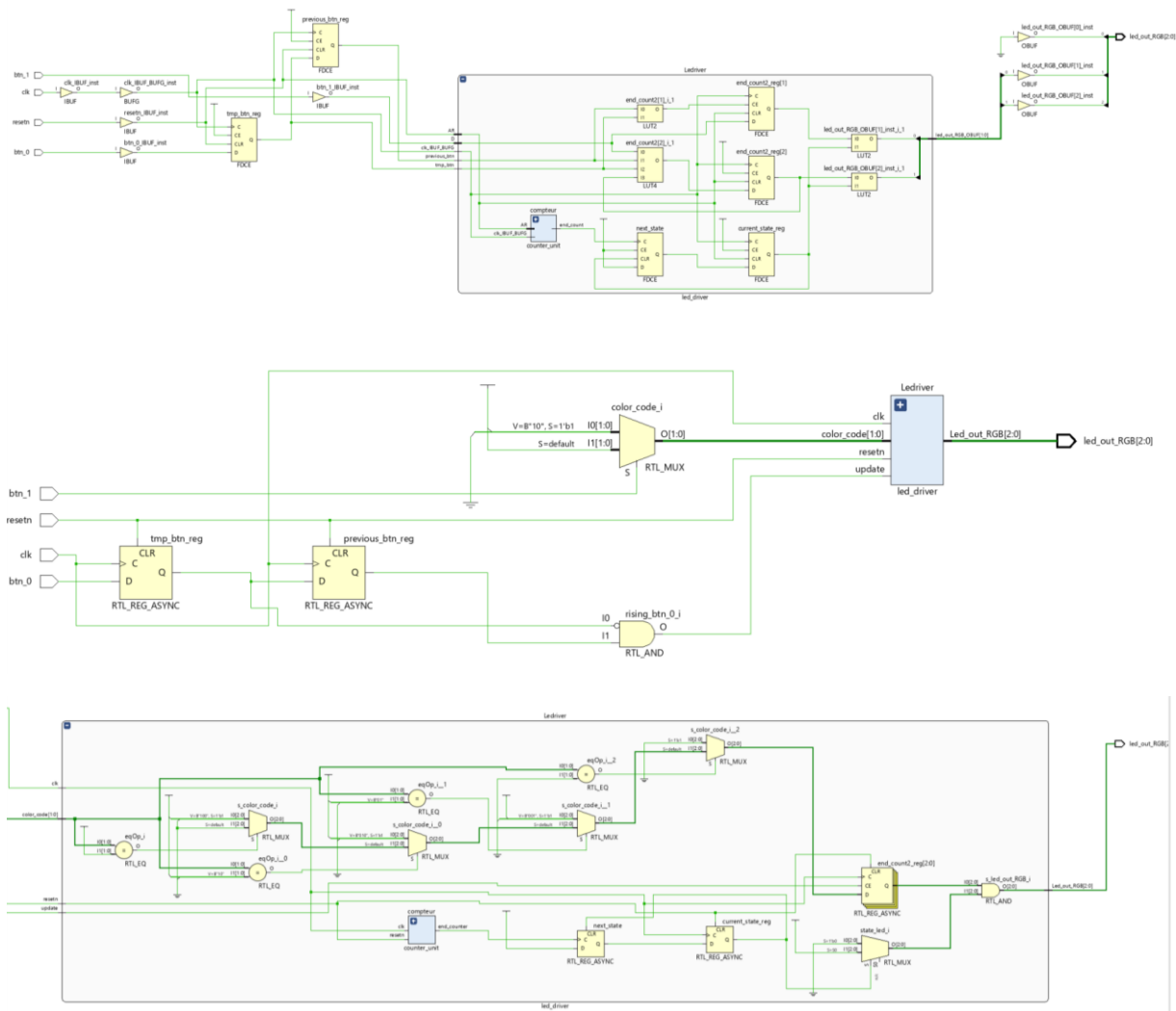
```

12. Vérifier votre résultat à la simulation.

On observe bien le rising_btn qui évolue lorsqu'on appui sur le btn_1 et btn_0. Le signal *update* doit recevoir 1 uniquement lorsque le rising_btn vient d'être appuyé. Le signal *color_code* doit reçoit bien le code couleur « vert » si le btn_1 est pressé, et le code couleur « bleu » sinon. Le chronogramme ci-dessous prouve le bon fonctionnement de la simulation.



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



Le schéma est représenté ci-dessus. On retrouve bien notre module Led_driver dans le quel se trouve notre counter_unit et le compteur de la gestion du clignotement.

Les ressources correspondent bien à ce que nous avons utilisé pour notre schéma RT. Les éléments utilisés sont répertoriés ici.

<pre> Start RTL Component Statistics ----- Detailed RTL Component Info : +---Adders : 2 Input 3 Bit Adders := 1 +---Registers : 3 Bit Registers := 2 1 Bit Registers := 4 +---Muxes : 2 Input 3 Bit Muxes := 2 5 Input 3 Bit Muxes := 1 2 Input 2 Bit Muxes := 1 ----- Finished RTL Component Statistics </pre>	<pre> Start Writing Synthesis Report ----- Report BlackBoxes: +-----+ BlackBox name Instances +-----+ +-----+ Report Cell Usage: +-----+ Cell Count +-----+ 1 BUFG 1 2 LUT1 1 3 LUT2 5 4 LUT4 1 5 FDCE 8 6 IBUF 4 7 OBUF 3 +-----+ </pre>
---	---

Tous les ressources sont répertoriées ci-dessus. Pour un générique de 4, on a 8 registres (2 pour la gestion des boutons, 1 pour la machine d'état, 3 pour le counter2(pour 3 leds) et 2 pour le counter_unit).

Lorsque le générique est de 200000000, on a donc 33 registres (2 pour la gestion des boutons, 1 pour la machine d'état, 2 pour le counter2(2leds) et 28 pour le counter_unit).

<pre> Start RTL Component Statistics ----- Detailed RTL Component Info : +---Registers : 3 Bit Registers := 1 1 Bit Registers := 3 +---Muxes : 5 Input 3 Bit Muxes := 1 2 Input 2 Bit Muxes := 1 2 Input 1 Bit Muxes := 1 ----- Finished RTL Component Statistics </pre>	<pre> BlackBox name Instances +-----+ +-----+ Report Cell Usage: +-----+ Cell Count +-----+ 1 BUFG 1 2 CARRY4 7 3 LUT2 33 4 LUT4 5 5 LUT6 3 6 FDCE 33 7 IBUF 4 8 OBUF 3 +-----+ </pre>
--	--

14. Effectuez le placement routage et étudiez les rapports.

Connexion des différents ports dans le fichier de contraintes.

```

1  ## This file is a general .xdc for the Cora 27-07S Rev. B
2  ## To use it in a project:
3  ## - uncomment the lines corresponding to used pins
4  ## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project
5
6  # PL System Clock
7  set_property -dict { PACKAGE_PIN H16  IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L13P_T2_MRCC_35 Sch=sysclk
8  create_clock -add -name sys_clk_pin -period 10.00 -waveform { 0 5 } [get_ports { clk }];#set
9
10 # RGB LEDs
11 set_property -dict { PACKAGE_PIN L15  IOSTANDARD LVCMOS33 } [get_ports { led_out_RGB[2] }]; #IO_L22N_T3_AD7N_35 Sch=led0_b
12 set_property -dict { PACKAGE_PIN G17  IOSTANDARD LVCMOS33 } [get_ports { led_out_RGB[1] }]; #IO_L16P_T2_35 Sch=led0_g
13 set_property -dict { PACKAGE_PIN M15  IOSTANDARD LVCMOS33 } [get_ports { led_out_RGB[0] }]; #IO_L21P_T3_DQS_AD14P_35 Sch=led0_r
14 #set_property -dict { PACKAGE_PIN G14  IOSTANDARD LVCMOS33 } [get_ports { led1_b }]; #IO_0_35 Sch=led1_b
15 #set_property -dict { PACKAGE_PIN L14  IOSTANDARD LVCMOS33 } [get_ports { led1_g }]; #IO_L22P_T3_AD7P_35 Sch=led1_g
16 #set_property -dict { PACKAGE_PIN M15  IOSTANDARD LVCMOS33 } [get_ports { led1_r }]; #IO_L23N_T3_35 Sch=led1_r
17
18 # Buttons
19 set_property -dict { PACKAGE_PIN D20  IOSTANDARD LVCMOS33 } [get_ports { btn_0 }]; #IO_L4N_T0_35 Sch=btn[0]
20 set_property -dict { PACKAGE_PIN D19  IOSTANDARD LVCMOS33 } [get_ports { btn_1 }]; #IO_L4P_T0_35 Sch=btn[1]
21
22 -----
60 ## ChipKit Outer Analog Header - as Digital I/O
61 ## NOTE: The following constraints should be used when using these ports as digital I/O.
62 set_property -dict { PACKAGE_PIN F17  IOSTANDARD LVCMOS33 } [get_ports { resetn }]; #IO_L6N_T0_VREF_35 Sch=ck_a[0]

```

Le rapport de temps est le suivant :

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

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

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

Design Timing Summary											
WNS(ns)	TNS(ns)	TNS Failing Endpoints	TNS Total Endpoints	WHS(ns)	THS(ns)	THS Failing Endpoints	THS Total Endpoints	WFS(ns)	TFWS(ns)	TFWS Failing Endpoints	TFWS Total Endpoints
4.270	0.000	0	3635	0.037	0.000	0	3619	3.750	0.000	0	2003

All user specified timing constraints are met.

Le chemin critique est :

Slack (MET) : 25.899ns (required time - arrival time)

Source:

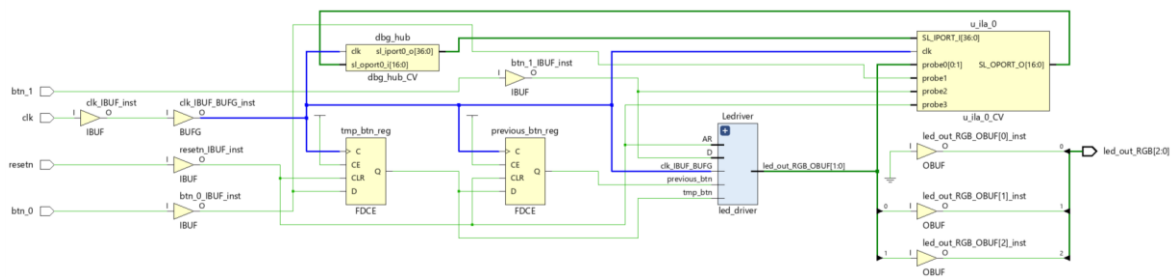
dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_switch/state_reg[2]/C(rising edge-triggered cell FDRE clocked by dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_inst/SERIES7_BSCAN.bscan_inst/TCK {rise@0.000ns fall@16.500ns period=33.000ns})

Destination:

dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_switch/portno_temp_reg[0]/D

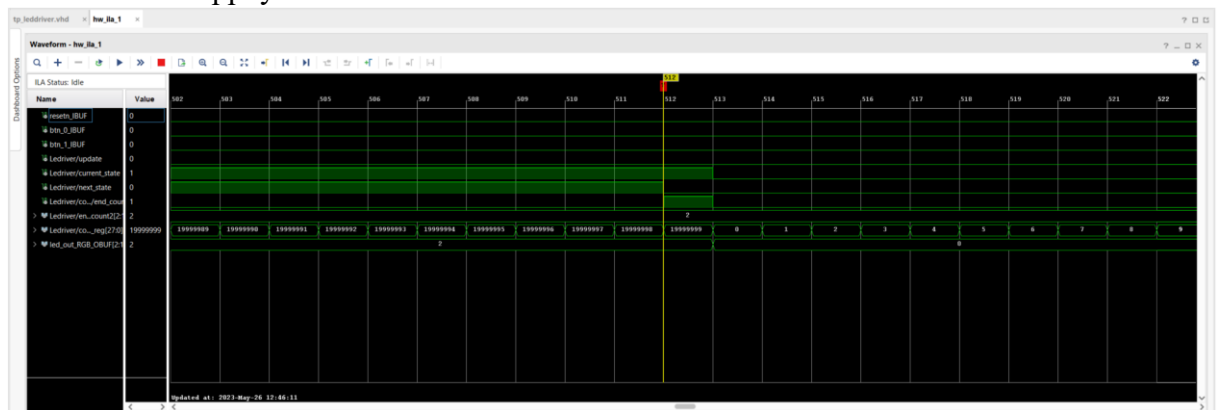
Max Delay Paths	
Slack (MET) :	25.899ns (required time - arrival time)
Source:	dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_switch/state_reg[2]/C (rising edge-triggered cell FDRE clocked by dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_inst/SERIES7_BSCAN.bscan_inst/TCK {rise@0.000ns fall@16.500ns period=33.000ns})
Destination:	dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_switch/portno_temp_reg[0]/D (rising edge-triggered cell FDRE clocked by dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_inst/SERIES7_BSCAN.bscan_inst/TCK {rise@0.000ns fall@16.500ns period=33.000ns})

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

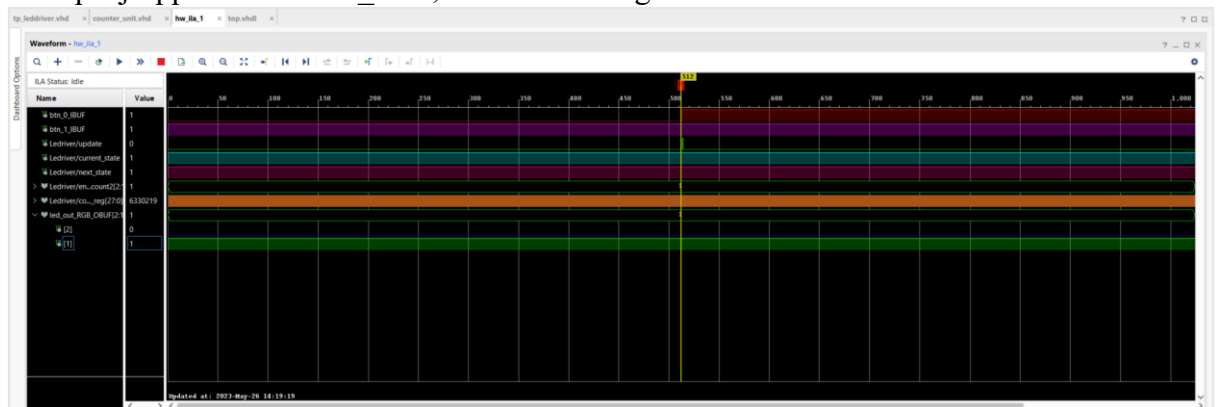


La simulation du bitstream donne les résultats suivants.

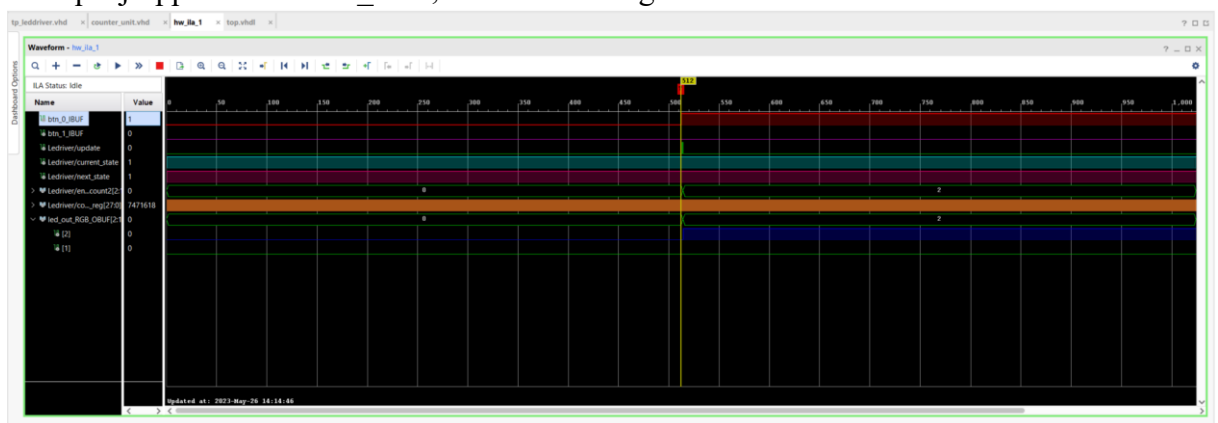
Aucun bouton appuyé.



Lorsque j'appuie sur le btn_1 =1, la led verte clignote.



Lorsque j'appuie sur le btn_0 =1, la led bleue clignote.



La vidéo de démonstration pour l'illustrer.