Segment 2

TP4 : Led driver-FIFO

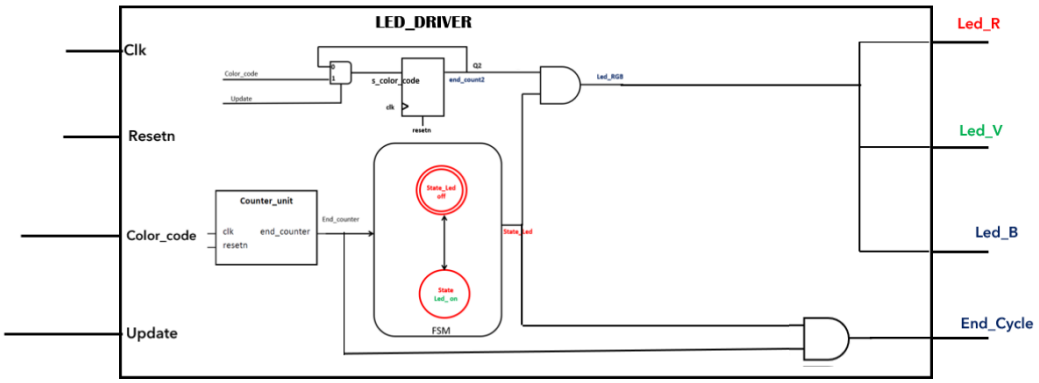
Objectif

L'objectif de cette partie 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, vous utiliserez le module LED_driver de la partie 1 et vous ajouterez l'utilisation d'un composant mémoire : la FIFO.

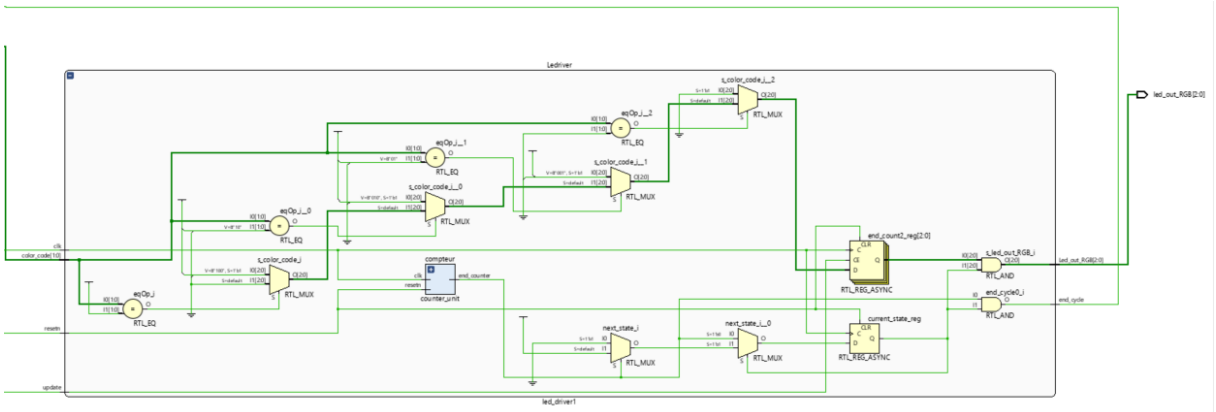
Questions

1. Sur l'architecture RTL, modifiez le module *LED_driver* en ajoutant une sortie *end_cycle*. Cette sortie vaudra 1 à la fin d'un cycle allumé/éteint de la LED RGB.

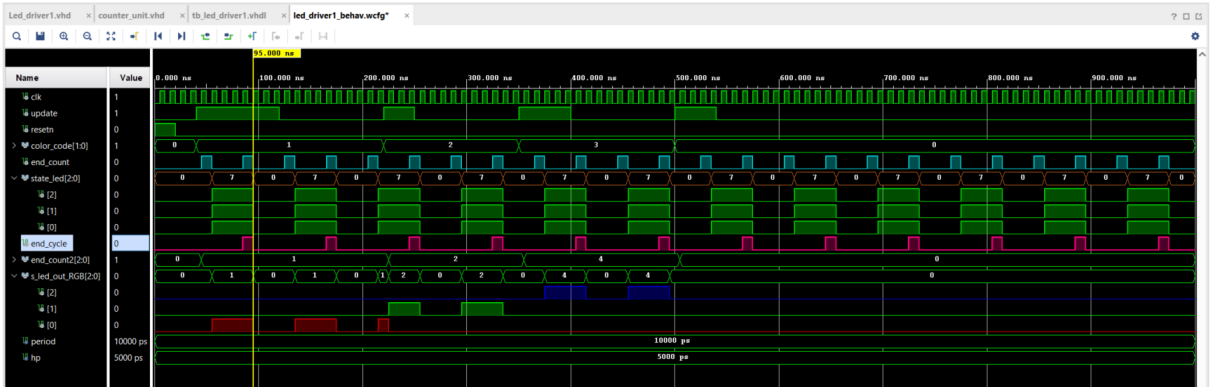
Pour ce faire j'ajoute une porte AND entre ed_count et state_led. Quelque soit la valeur de state_led(allumé ou éteint), et end_count à 1on a end_cycle à 1 sinon à 0.




```
--Partie combinatoire logique de controle de End_cycle

end_cycle <= '1' when  end_count= '1' and state_led(0)='1'      --- state_led(0) choix de bit de state_led sont à 1 ou 0
                    else '0';
```



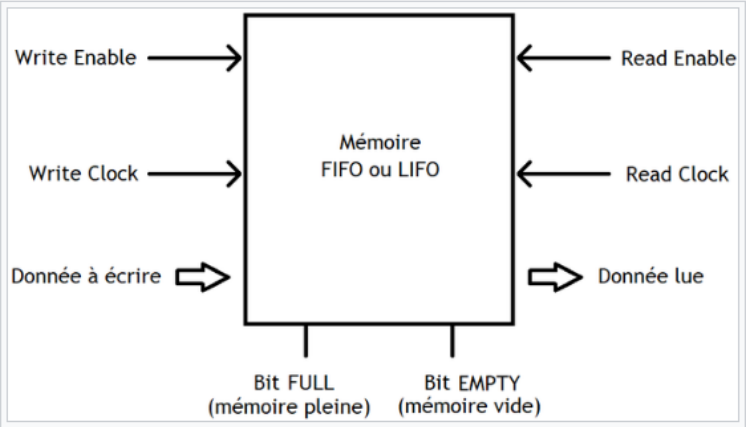
Bout de code rajouter à notre code Led_driver.



2. Modifiez la logique en entrée du module pour ajouter une FIFO. Cette FIFO doit prendre en entrée le code couleur « vert » ou « bleu » suivant l'état du bouton_1 et est connectée en sortie à l'entrée *color_code* du module *LED_driver*. La donnée est écrite dans la FIFO lorsqu'il y a un front montant du bouton_0. La donnée de la FIFO est lue lorsque le signal *end_cycle* du module LED_driver vaut 1.

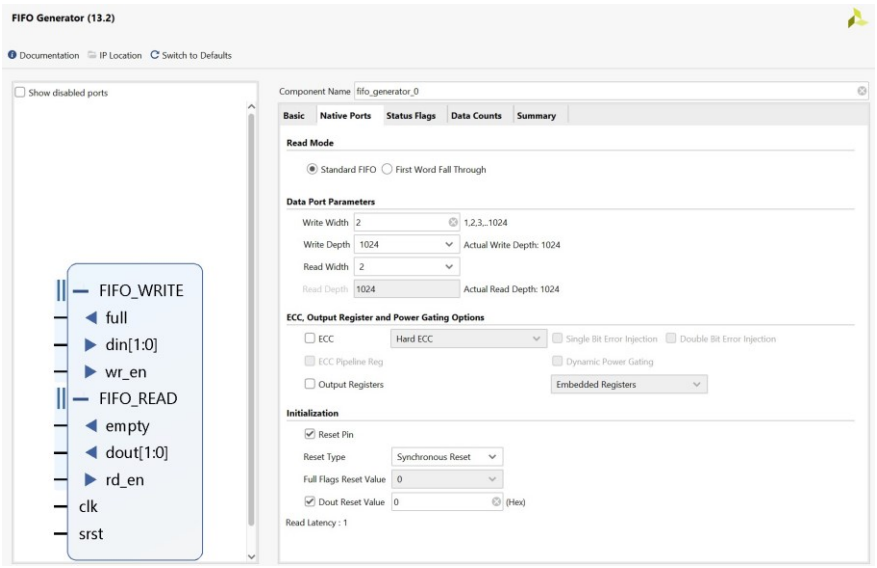
Pour ajouter une FIFO, ouvrez le catalog d'IP de Vivado (onglet *Project Manager -> IP Catalog*) et recherchez l'IP qui vous semble la plus pertinente. Les documentations des IPs de Xilinx sont disponibles en ligne, cela peut faciliter vos recherches.

Une fois que vous avez choisi une IP dans le catalogue, double cliquez dessus et choisissez les paramètres. Cliquez ensuite sur *OK*. Pour vous aider pour l'implémentation de cette IP, vous pouvez ouvrir un design d'exemple. Pour cela, faites un clic droit sur l'IP dans l'onglet *Sources*, puis *Open IP Example Design*.



Interface des mémoires FIFO et LIFO

J'ai choisi la FIFO Generator pour notre cas.

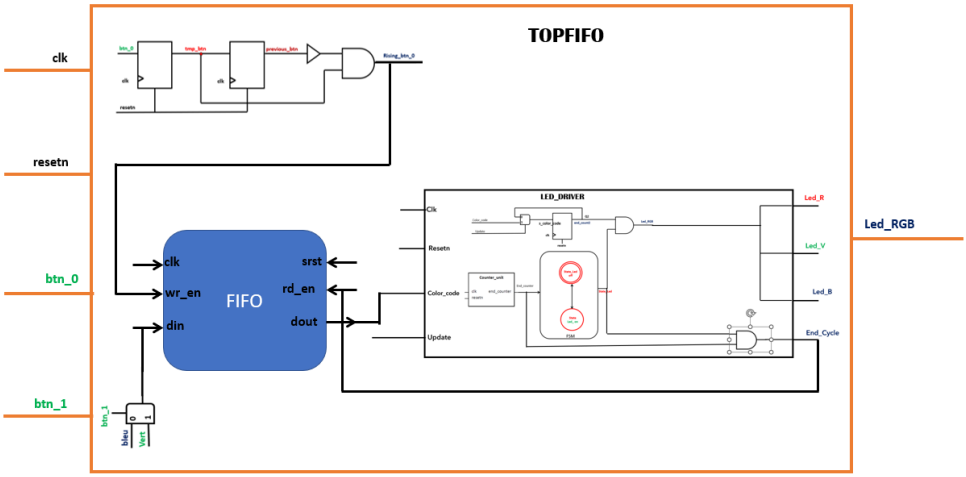


Ses paramètres d'entrées et sortie sont les suivants :

component FIFO_LED

port (

clk : in std_logic; srst : in std_logic; din : in std_logic_vector(1 downto 0); wr_en : in std_logic; rd_en : in std_logic; dout : out std_logic_vector(1 downto 0));



3. Modifiez vos codes de la partie 1 pour y ajouter les nouveaux éléments de votre architecture.

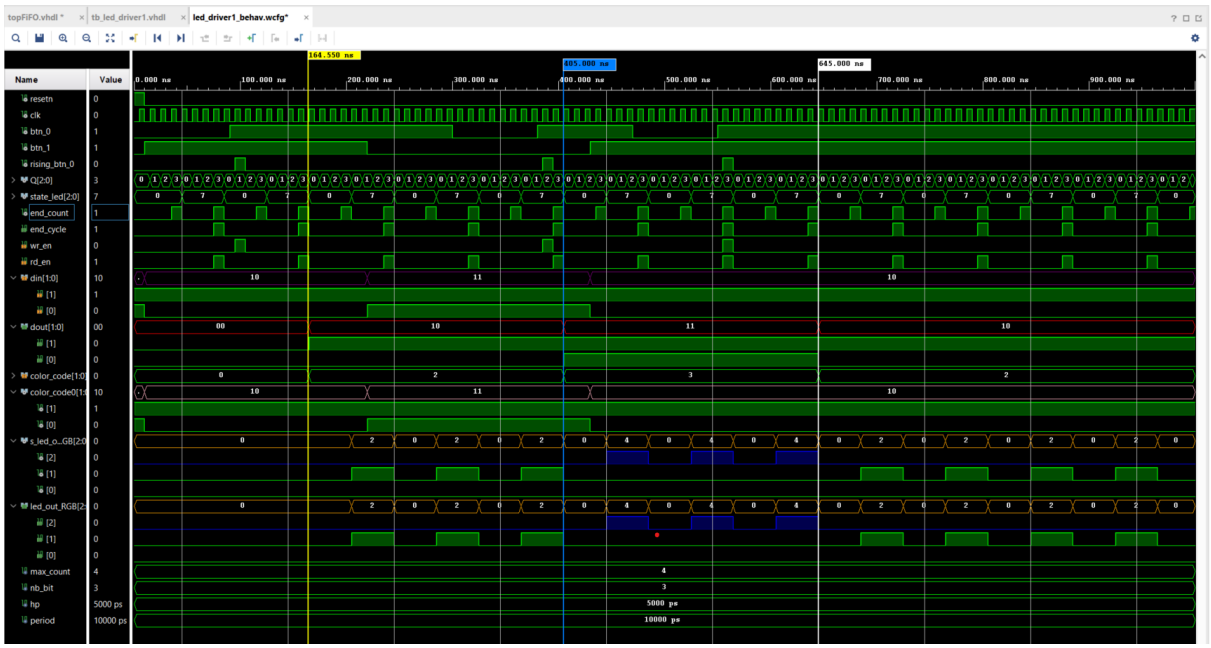
```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_unsigned.all;
4   use ieee.std_logic_arith;
5
6   entity topFIFO is
7
8       port (
9           clk         : in std_logic;   -- Signal d'horloge
10          resetn      : in std_logic;   -- Signal de reset
11          btn_0       : in std_logic;   -- Signal de mise a  jour
12          btn_1       : in std_logic;   -- Signal de mise a  jour
13          led_out_RGB : out std_logic_vector(2 downto 0)  -- LED RGB
14
15       );
16  end topFIFO;
17
18  architecture behavioral of topFIFO is
19
20      signal update: std_logic;
21      signal color_code0 : std_logic_vector(1 downto 0);
22      signal color_code : std_logic_vector(1 downto 0);
23      signal s_led_out_RGB  : std_logic_vector(2 downto 0);
24      signal end_cycle  : std_logic;  -- End_cycle
25
26      ----RisingEdgebtn
27      signal tmp_btn      : std_logic;
28      signal previous_btn : std_logic;
29      signal rising_btn_0 : std_logic;
30
31
32   --Declaration de led_driver
33  component led_driver1
34
35      port (
36          clk         : in std_logic;   -- Signal d'horloge
37          resetn      : in std_logic;   -- Signal de reset
38          update      : in std_logic;   -- Signal de mise a  jour
39          color_code  : in std_logic_vector(1 downto 0);
40          end_cycle  : out std_logic;  -- End_cycle
41          led_out_RGB : out std_logic_vector(2 downto 0)  -- LED Red
42       );
43
44  end component;
45
46   -- Declaration de la FIFO LED
47
48  component FIFO_LED
49     port  (
50      clk : in std_logic;
51      srst : in std_logic;
52      din : in std_logic_vector(1 downto 0);
53      wr_en : in std_logic;
54      rd_en : in std_logic;
55      dout : out std_logic_vector(1 downto 0)
```

```
    dout : out std_logic_vector(1 downto 0)

  );
end component;

begin

Ledriver : led_driver1
port map (
            clk => clk,
            resetn=>resetn,
            update => '1',
            color_code => color_code,
            led_out_RGB => s_led_out_RGB,
            end_cycle => end_cycle
        );

FIFO : FIFO_LED
port map  (

     clk => clk,
     srst => resetn,
     din => color_code0 ,
     wr_en => rising_btn_0,
     rd_en => end_cycle,
     dout => color_code
    );

--- process séquentielle du risingbtn_0

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

    -- partie combinatoire des couleurs color_code pour le vert et le bleu

        color_code0<="10" when (btn_1='1')
            else
                "11";
--- gestion des couleurs de sortie avec le signal interne s_led_out entre la FSM et la couleur

led_out_RGB <=s_led_out_RGB;

end behavioral;
```

4. Mettez à jour le testbench et réalisez une simulation pour vérifier votre design.

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


entity tb_topFifo_led is
end tb_topFifo_led ;

architecture behavioral of tb_topFifo_led  is

    signal resetn      : std_logic := '0';
    signal clk         : std_logic := '0';
    signal update         : std_logic := '0';
    signal btn_0       : std_logic := '0';
    signal btn_1        : std_logic := '0';
    signal color_code : std_logic_vector(1 downto 0);
    signal s_led_out_RGB  : std_logic_vector(2 downto 0);
    signal end_cycle : std_logic:= '0';
    signal end_count : std_logic;    -- signal counter unit
    signal end_count2 : std_logic_vector(2 downto 0);  -- signal counter2
    signal s_color_code : std_logic_vector(2 downto 0);
    signal state_led: std_logic_vector(2 downto 0);    -- Ã©tat des leds

    -- Les constantes suivantes permette de definir la frequence de l'horloge
    constant hp : time := 5 ns;       --demi periode de 5ns
    constant period : time := 2*hp;  --periode de 10ns, soit une frequence de 100Hz
    --constant  max_count : integer :=20000000;
    --constant   nb_bit : integer := 28;
     constant  max_count : integer :=4;
     constant   nb_bit : integer := 3;

     --Dolaration du composant fsm
    component topFIFO
         port (
        clk          : in std_logic;    -- Signal d'horloge
        resetn       : in std_logic;    -- Signal de reset
        btn_0        : in std_logic;    -- Signal de mise a  jour
        btn_1        : in std_logic;    -- Signal de mise a  jour
        led_out_RGB : out std_logic_vector(2 downto 0)   -- LED RGB

     );
    end component;

    begin
    dut: topFIFO
        port map (
          clk   =>  clk,
          resetn => resetn,    -- Signal de reset
                --a completer
         btn_0 => btn_0 ,      -- Bouton d'entrÃ©e
         btn_1 => btn_1 ,

        led_out_RGB => s_led_out_RGB  -- LED rouge
```

```
54          led_out_RGB => s_led_out_RGB  -- LED rouge
55          );
56
57      --Simulation du signal d'horloge en continue
58      process
59      begin
60          wait for hp;
61          clk <= not clk;
62
63      end process;
64       --Simulation du signal restn en continue avec affichage de end_counter qui fonctionne aprÃ¨s le cycle allumÃ©e Ã©teint
65
66  process
67
68      begin
69
70  -- Resetn
71      resetn <= '1';
72       wait for 10 ns;
73          resetn<='0';
74
75     --Verifions l'état initial idle (LED éteinte)
76
77
78          btn_1<='1';
79            wait for 80ns;
80          btn_0<='1';
81          wait for 80ns;
82  -- Verifions l'état state 1  (LED verte
83
84          wait for 50 ns;
85
86          btn_1<='0';
87           wait for 80ns;
88          btn_0<='0';
89          wait for 80ns;
90  -- Verifions l'état state 1 (LED bleue)
91
92          btn_0<='1';
93
94          wait for 50 ns;
95
96           btn_1<='1';
97           wait for 40ns;
98          btn_0<='0';
99          wait for 80ns;
100         btn_0<='1';
101
102 --Pour finir la simulation
103 wait;
104 end process;
105
106 end behavioral;
```



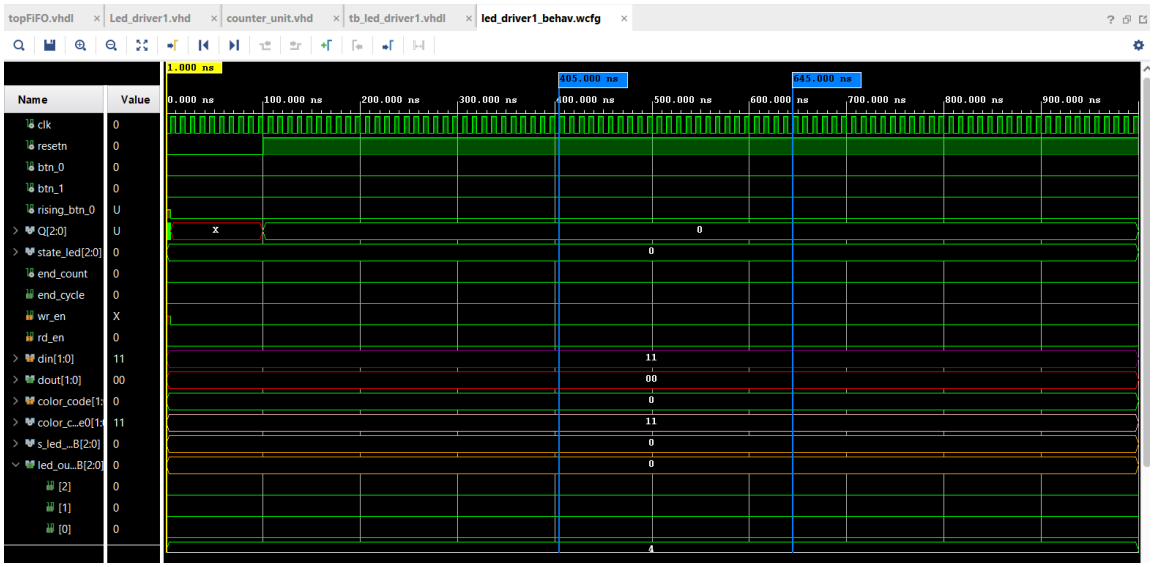
On valide ainsi la simulation de la simulation de la FIFO et du Led driver. Lorsque Write enable wr_en détecte un front montant, elle enregistre la valeur (donnée à écrire) présente sur

l'entrée din de cette dernière. Et lorsque le red enable rd_en passe à 1, la valeur enregistrée dans din est envoyée sur la sortie dout de la FIFO (donnée lue).

Test de vérification du fonctionnement de la mémoire.

On vérifie ainsi le bon fonctionnement de btn_0 et btn_1.

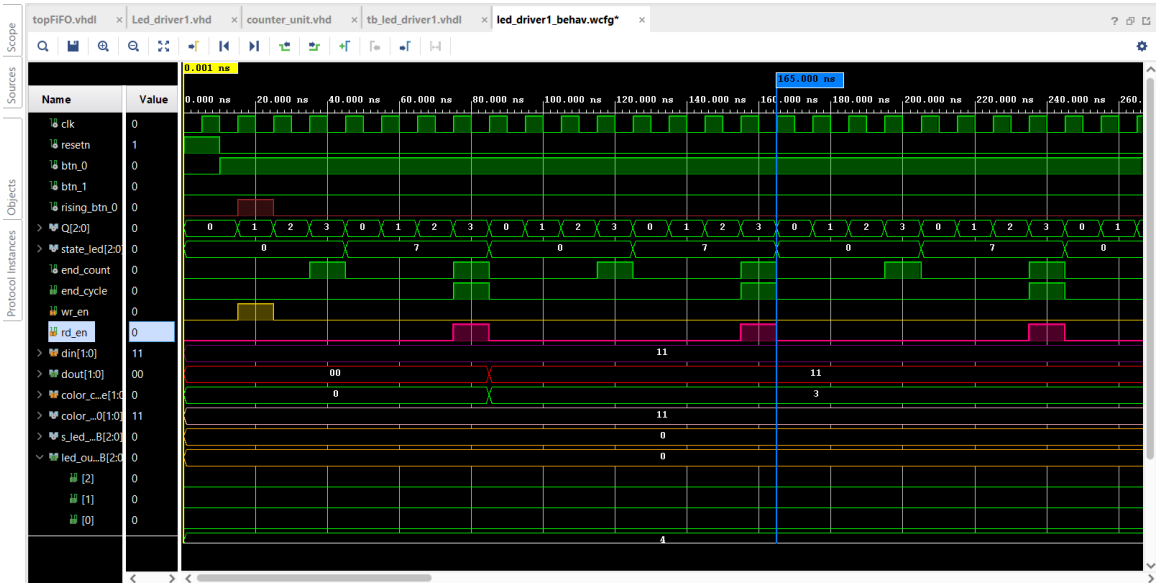
1er test : Initialisation du reset



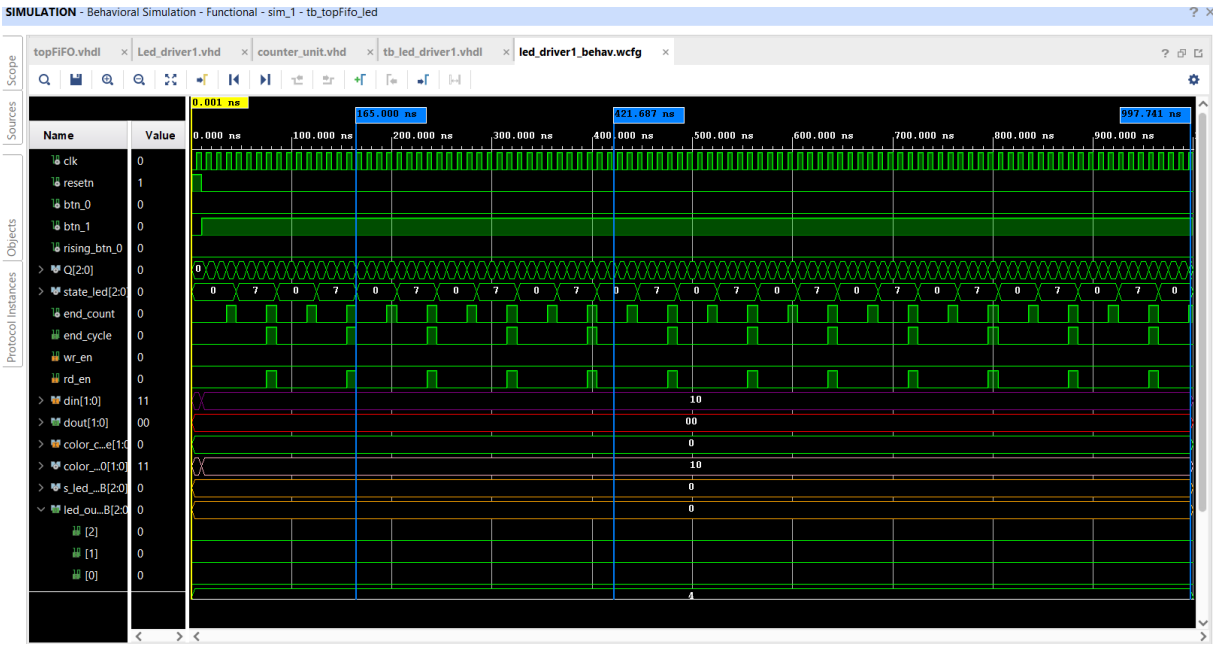
On remarque que tout est à zéro, la FIFO est vide.

2e test : btn_1=0, btn_0=1.

On écrit dans la FIFO.Rising_btn et Wr_en sont à1, la FIFO ecrit le code couleur du signal din qui est bleu. La led clignote dans cette couleur. Ce qu'on peut voir en dout, qui prend cette valeur au prochain front montant de end_cycle. Rd_en est aussi à 1.



3e test : btn_1=1, btn_0=0.



4e test : MEMORISATION/CLIGNOTEMENT D'UNE SEQUENCE

-- Appui sur le bouton_1 => chgt de couleur (verte) --

-- Appui sur le bouton_0 => écriture dans la FIFO (vert) -- Relâchement du bouton_0 --

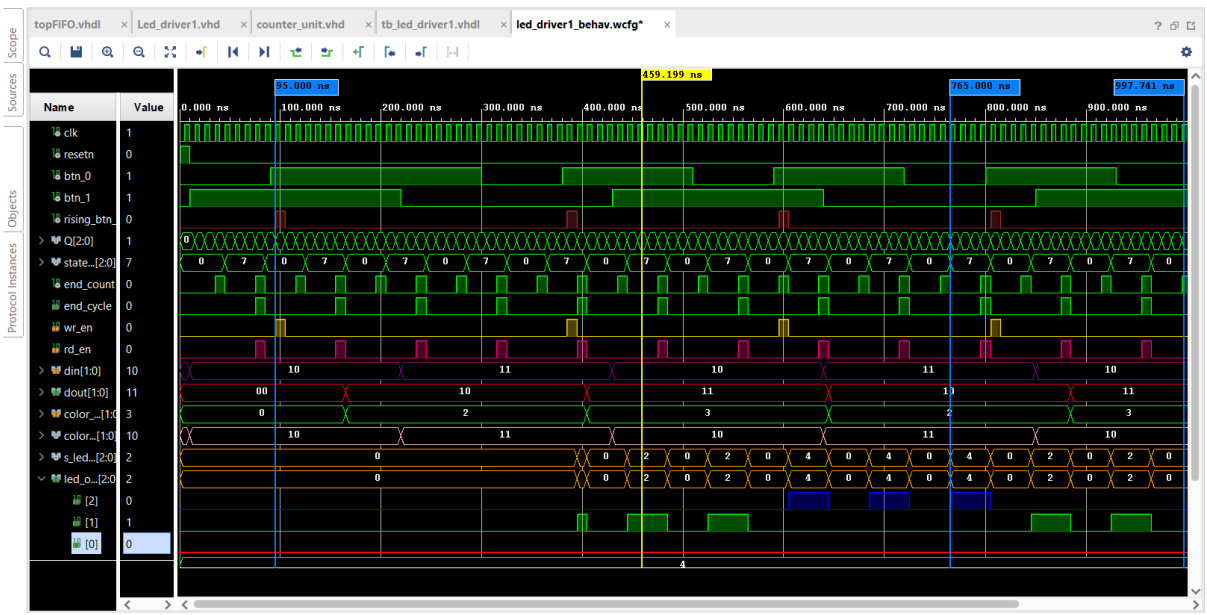
-- Relâchement du bouton_1 => chgt de couleur (bleue) --

-- Appui sur le bouton_0 => écriture dans la FIFO (bleu) -- Relâchement du bouton_0 --

-- Appui sur le bouton_1 => chgt de couleur (verte) --

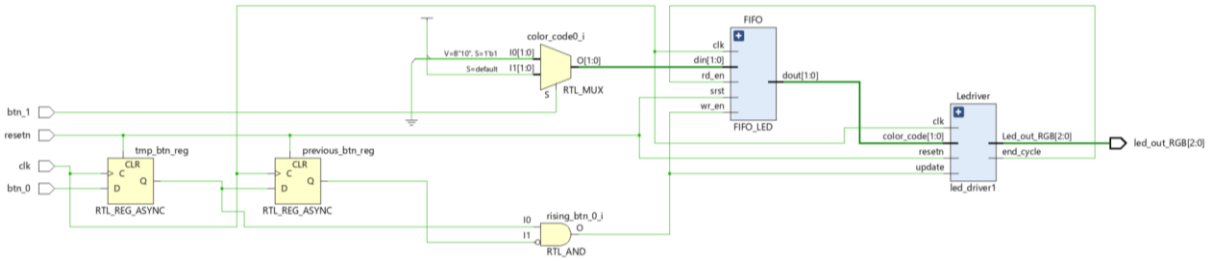
-- Appui sur le bouton_0 => écriture dans la FIFO (vert) --

-- Attente de 5 cycles => lecture de la FIFO –

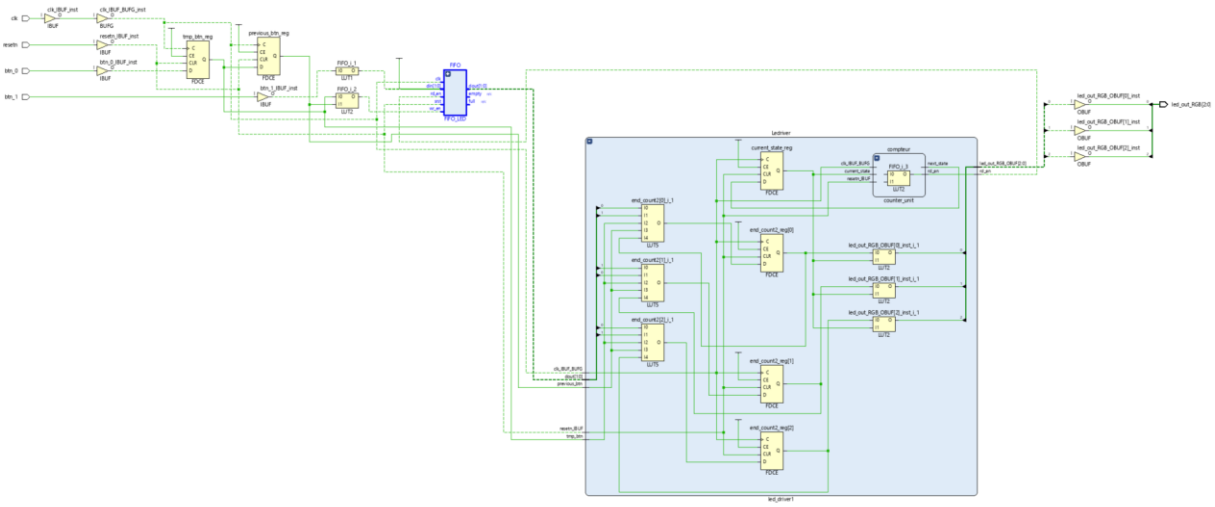


=> On s'attend à observer la séquence "vert/bleu/vert"

5. 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, mais aussi la FIFO.



Les ressources correspondent bien à ce que nous avons utilisé pour notre schéma RTL.

```
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
---------------------------------------------
```

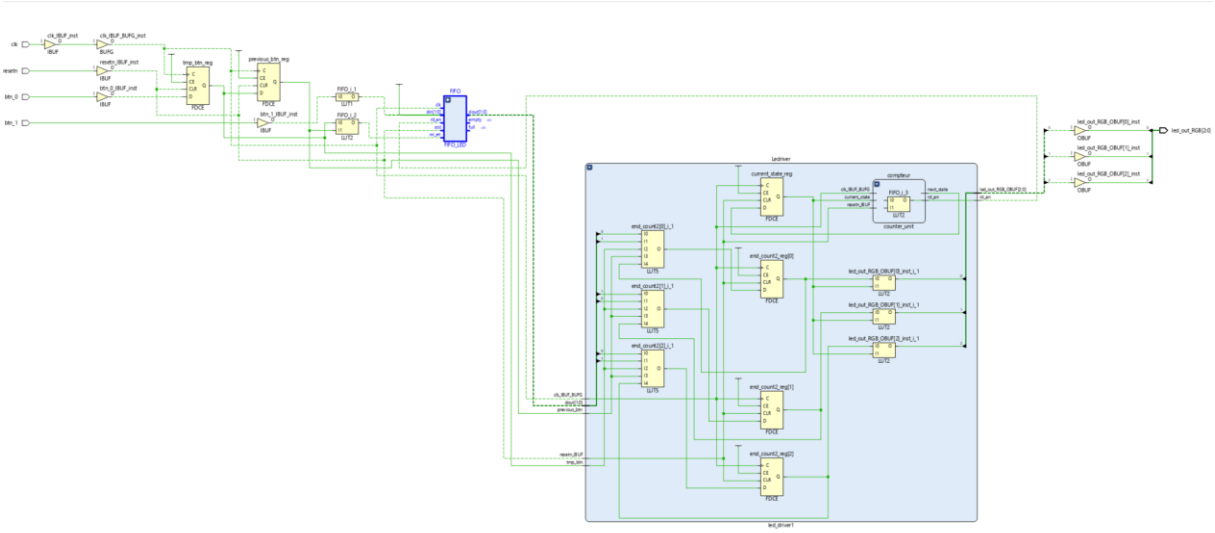
```
Report BlackBoxes:
+------+--------------+----------+
|      |BlackBox name |Instances |
+------+--------------+----------+
|1     |FIFO_LED      |         1|
+------+--------------+----------+

Report Cell Usage:
+------+--------------+------+
|      |Cell          |Count |
+------+--------------+------+
|1     |FIFO_LED_bbox |     1|
|2     |BUFG          |     1|
|3     |CARRY4        |     7|
|4     |LUT1          |     1|
|5     |LUT2          |    35|
|6     |LUT4          |     4|
|7     |LUT5          |     3|
|8     |LUT6          |     3|
|9     |FDCE          |    34|
|10    |IBUF          |     4|
|11    |OBUF          |     3|
+------+--------------+------+
```

Les éléments utilisés sont répertoriés ici. Lorsque le générique est de 200000000, on a donc 33 registres (2 pour la gestion des boutons, 1pour la machine d'état, 3 pour le counter2 et 28 pour le counter_unit).

6. Effectuez le placement routage et étudiez les rapports.

Connexion des différents ports dans le fichier de contraintes.

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

# PL System Clock
set_property -dict { PACKAGE_PIN H16   IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L13P_T2_MRCC_35 Sch=sysclk
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports { clk }];#set

# RGB LEDs
set_property -dict { PACKAGE_PIN L15   IOSTANDARD LVCMOS33 } [get_ports { led_out_RGB[2] }]; #IO_L22N_T3_AD7N_35 Sch=led0_b
set_property -dict { PACKAGE_PIN G17   IOSTANDARD LVCMOS33 } [get_ports { led_out_RGB[1] }]; #IO_L16P_T2_35 Sch=led0_g
set_property -dict { PACKAGE_PIN N15   IOSTANDARD LVCMOS33 } [get_ports { led_out_RGB[0]}]; #IO_L21P_T3_DQS_AD14P_35 Sch=led0_r
#set_property -dict { PACKAGE_PIN G14   IOSTANDARD LVCMOS33 } [get_ports { led1_b }]; #IO_0_35 Sch=led1_b
#set_property -dict { PACKAGE_PIN L14   IOSTANDARD LVCMOS33 } [get_ports { led1_g }]; #IO_L22P_T3_AD7P_35 Sch=led1_g
#set_property -dict { PACKAGE_PIN M15   IOSTANDARD LVCMOS33 } [get_ports { led1_r }]; #IO_L23N_T3_35 Sch=led1_r

# Buttons
set_property -dict { PACKAGE_PIN D20   IOSTANDARD LVCMOS33 } [get_ports { btn_0 }]; #IO_L4N_T0_35 Sch=btn[0]
set_property -dict { PACKAGE_PIN D19   IOSTANDARD LVCMOS33 } [get_ports { btn_1 }]; #IO_L4P_T0_35 Sch=btn[1]

## ChipKit Outer Analog Header - as Digital I/O
## NOTE: The following constraints should be used when using these ports as digital I/O.
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 :

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	WPWS(ns)	TPWS(ns)	TPWS Failing Endpoints	TPWS Total Endpoints
3.591	0.000	0	5047	0.023	0.000	0	5031	3.750	0.000	0	2916

Clock

```
------------------------------------------------------------------------------------------------
| 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


------------------------------------------------------------------------------------------------
```

Chemin critique

Slack (MET) : 25.539ns (required time - arrival time)

Source:
dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_switch/state_reg[1]/C

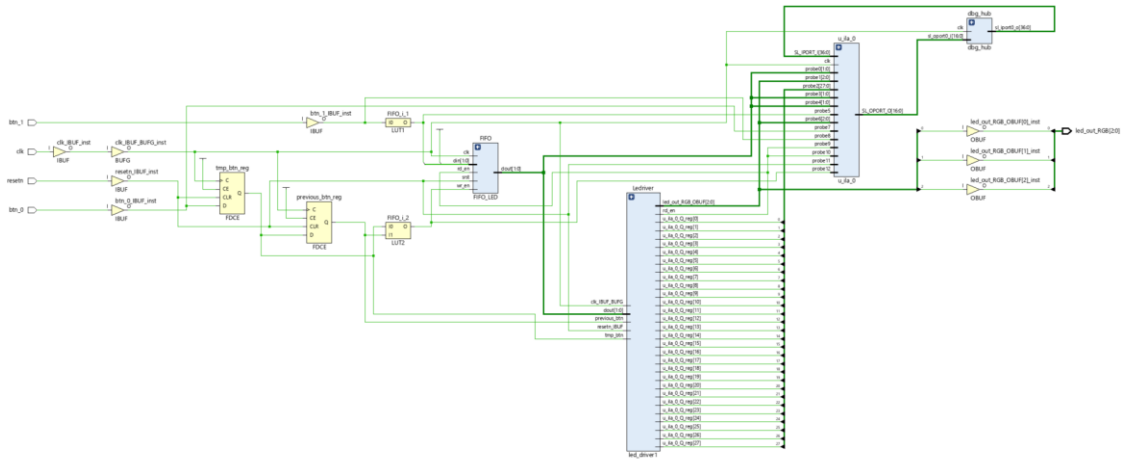
(rising edge-triggered cell FDRE clocked by dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_inst/SERIES7_BSCAN .bscan_inst/TCK {rise@0.000ns fall@16.500ns period=33.000ns})

Destination:
dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_switch/portno_temp_r eg[4]/D

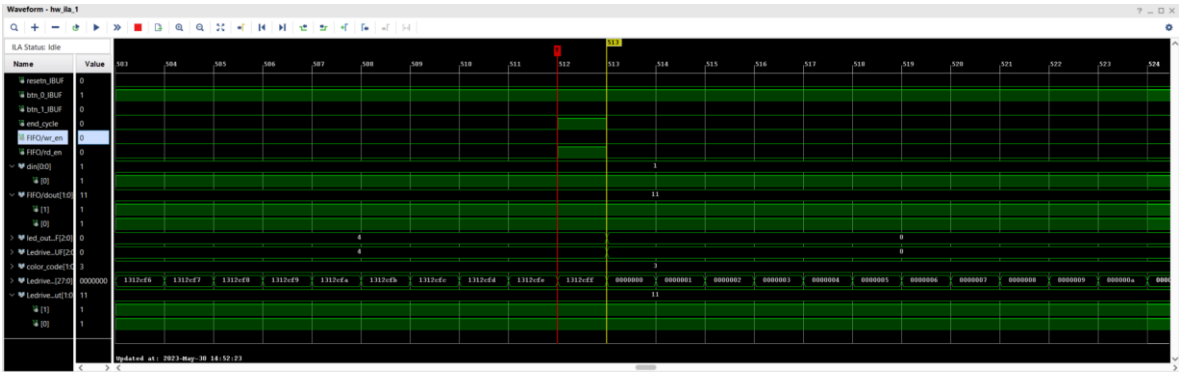
(rising edge-triggered cell FDRE clocked by dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_inst/SERIES7_BSCAN .bscan_inst/TCK {rise@0.000ns fall@16.500ns period=33.000ns}).

```
Max Delay Paths
--------------------------------------------------------------------------------------
Slack (MET) :             25.539ns  (required time - arrival time)
  Source:                 dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_switch/state_reg[1]/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[4]/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})
```

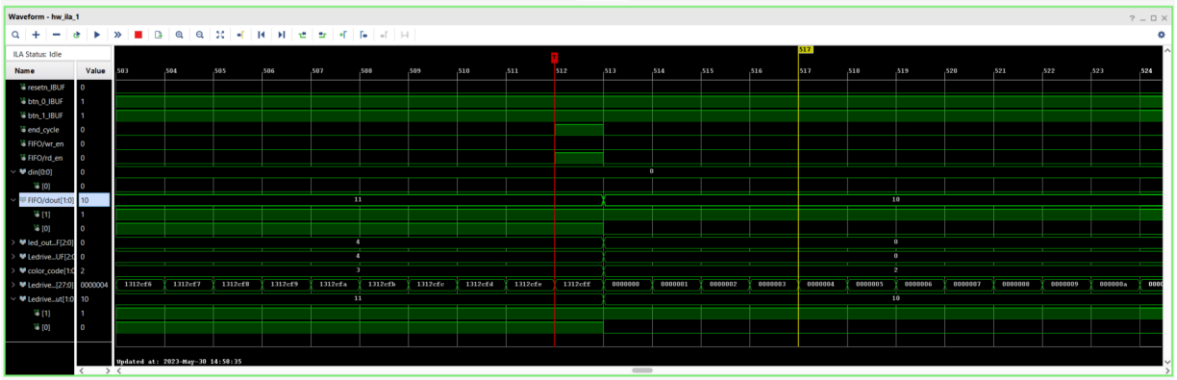
7. Générez le bitstream et vérifiez que vous avez le comportement attendu sur carte.



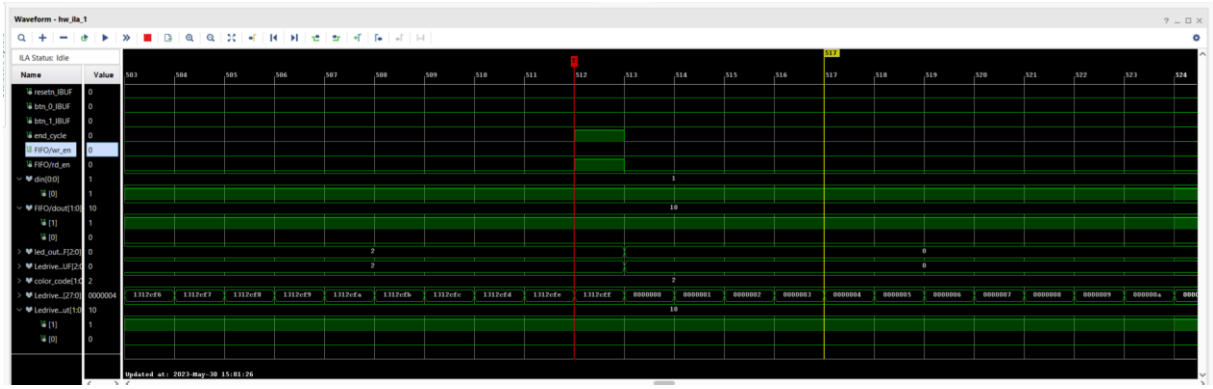
Au démarrage la led est à 0. Lorsque j'appuie sur Btn_0=1, la valeur envoyée est bleue. La led clignote alors en bleue.



Btn_1 et btn_0 sont à 1. La valeur actuelle est la bleue et la suivante est la verte. On passe d'un clignotement bleu à vert.

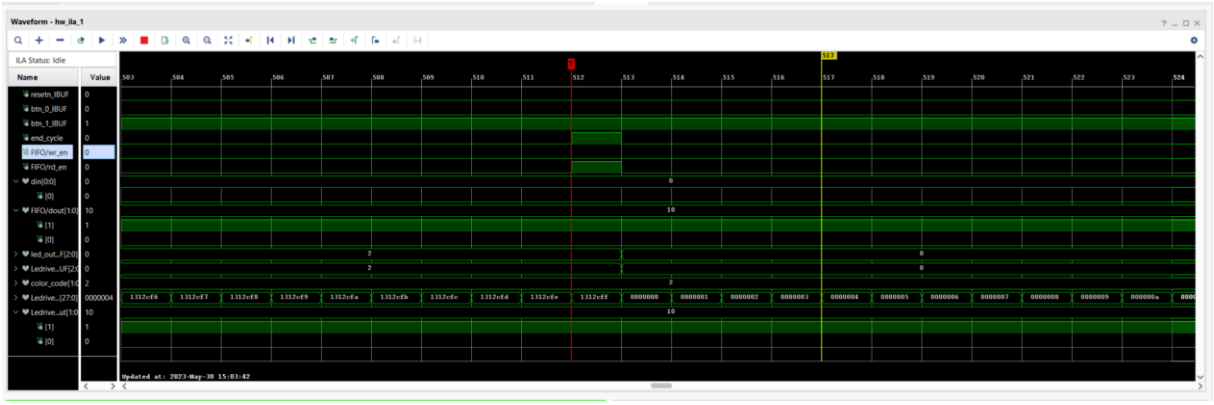


Btn_0 et btn_1 sont à 0. Il garde la dernière valeur en mémoire qui est 10. La led sur la carte clignote en vert.



Btn_1= 1

La valeur reste inchangée dans la FIFO. La led verte clignote toujours



Btn_0=1, la valeur passe envoyée est la couleur bleue. La led clignote en bleue.