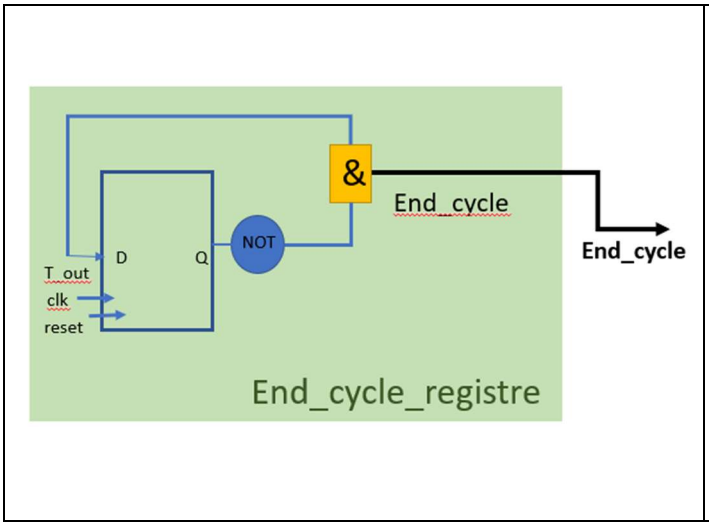
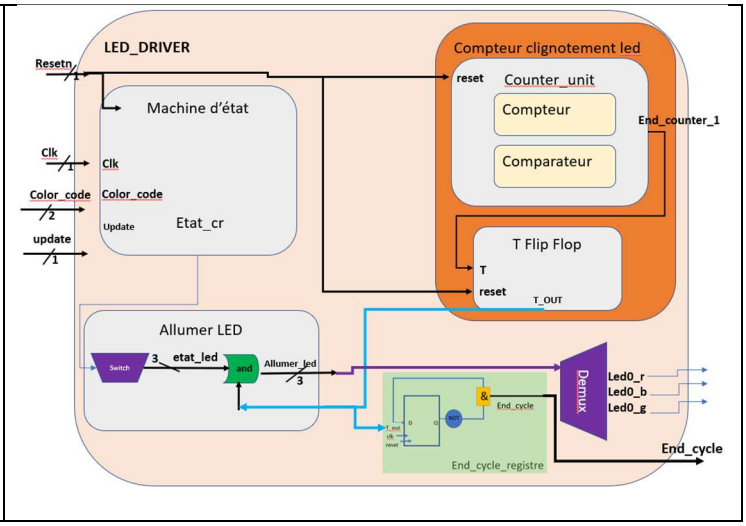
Compte rendu de – TP04 - Pilotage de LED et mémoire Partie 2

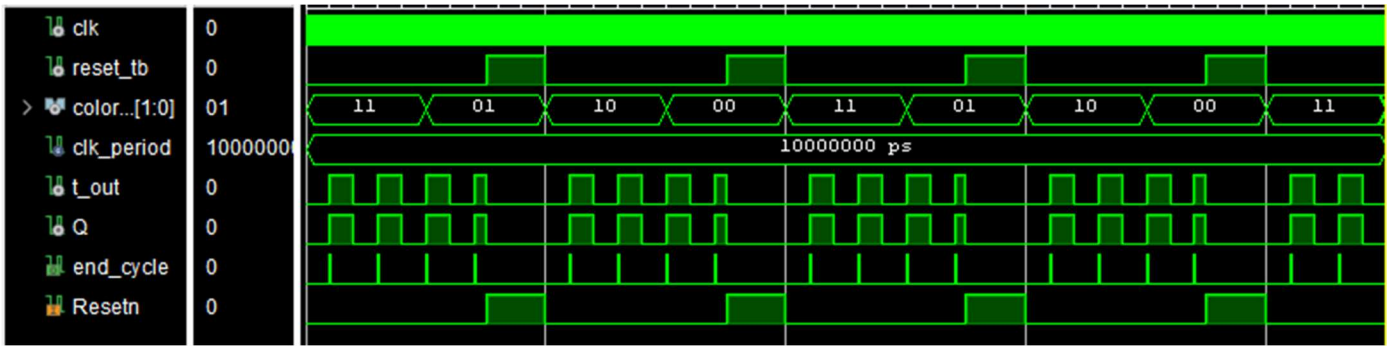
1.1 Objectif de ce TP

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.

1) Question 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.





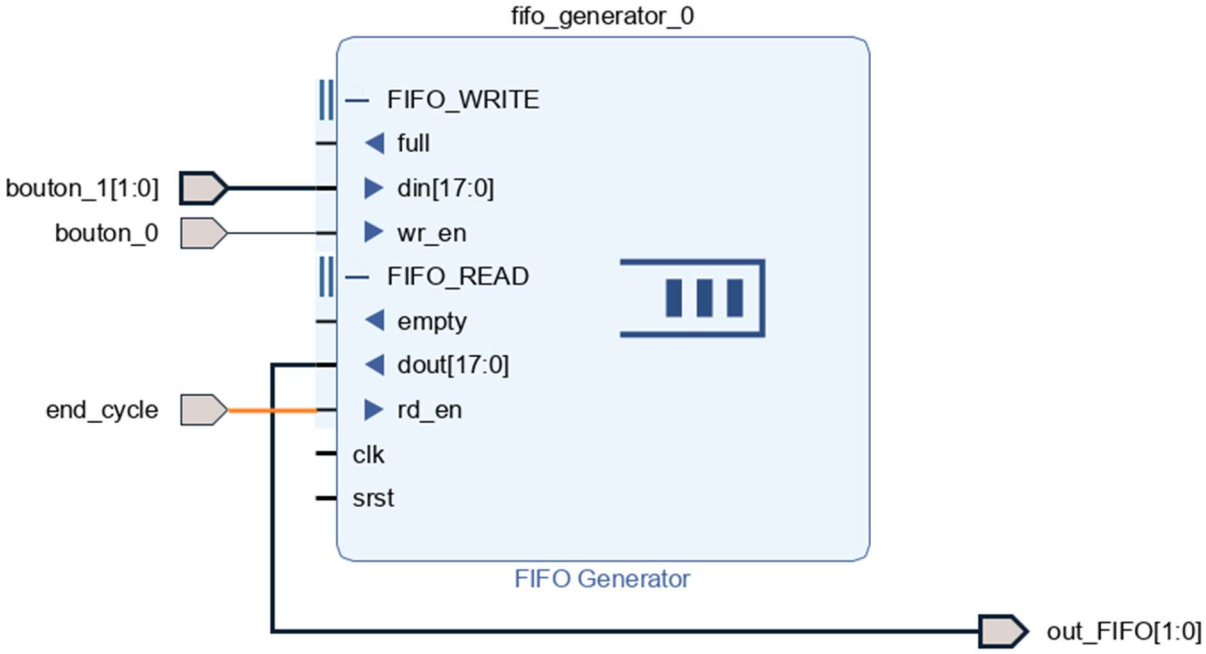


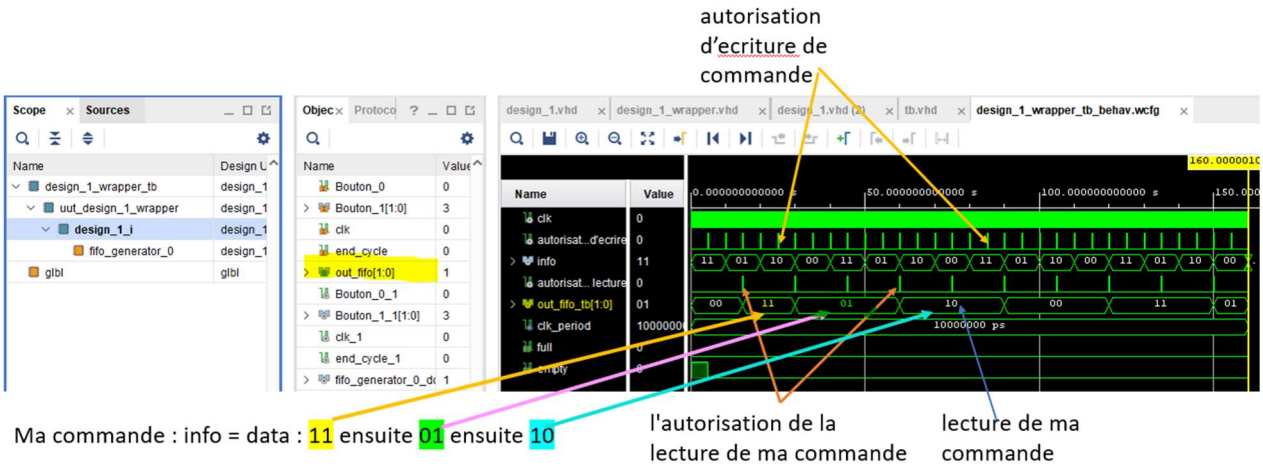
2. Modifiez la logique en entrée du module pour ajouter une FIFO. Cette FIFO doit prendre en entrée le code couleur « vert » ou « bleu » suivant l'état du bouton_1 et est connectée en sortie à l'entrée color_code du module LED_driver. La donnée est écrite dans la FIFO lorsqu'il y a un front montant du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

+ Question3 et question 4

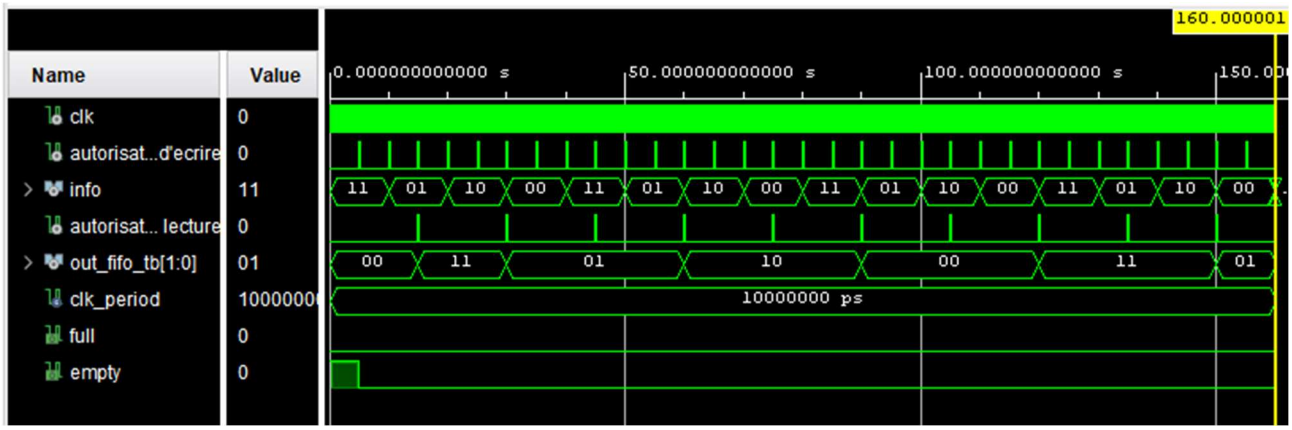
- je commence par comprendre le fonctionnement de mon FIFO pour pouvoir l'intégrer dans mon tp :

- J'ai créé mon composant IP avec des entrées sorties adaptées (clk =100MHz)





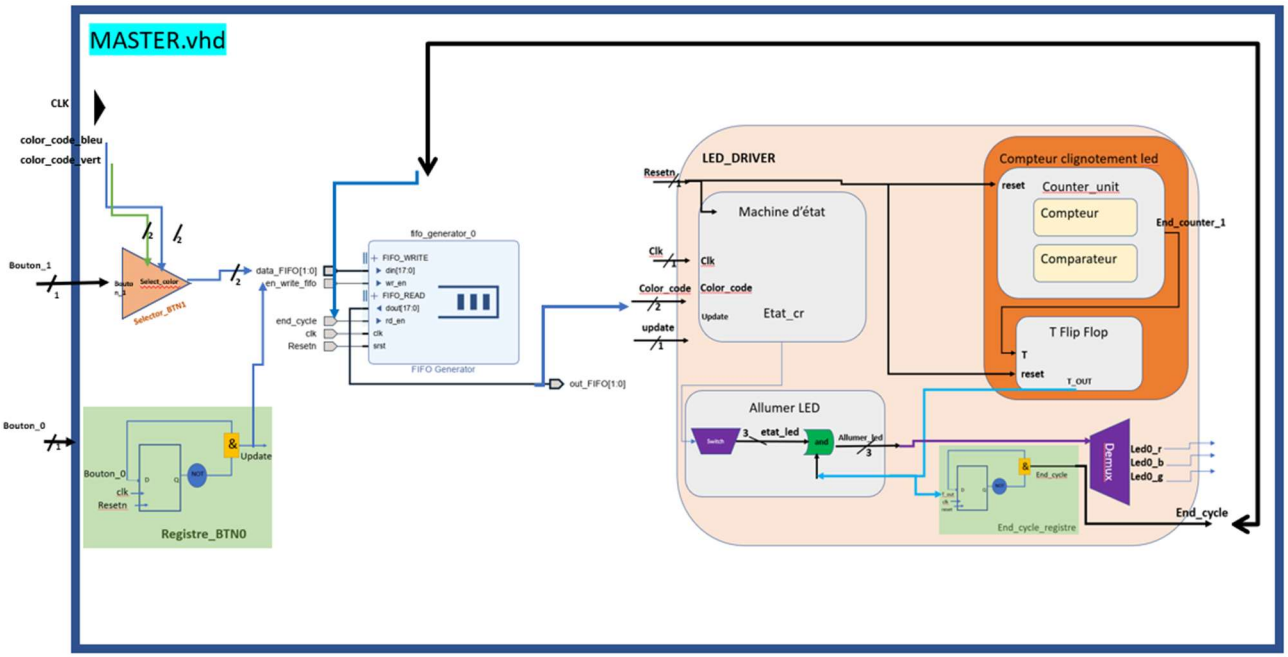
Compte rendu Eve CHAR du TP4-partie2 - 19/05/2023 – 24/05/2023

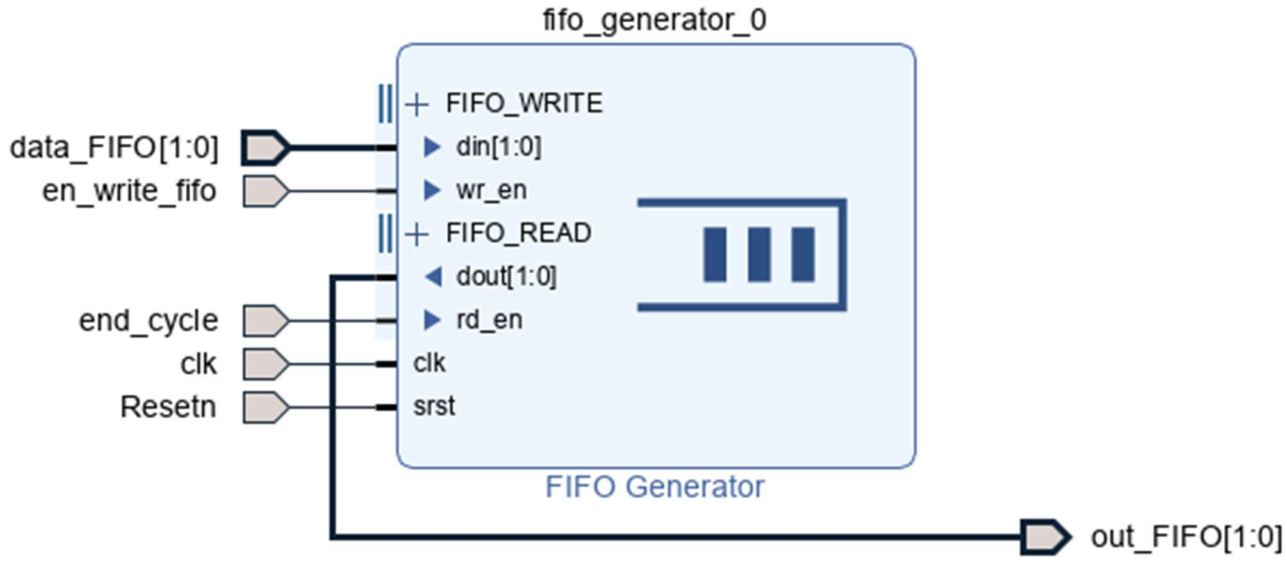


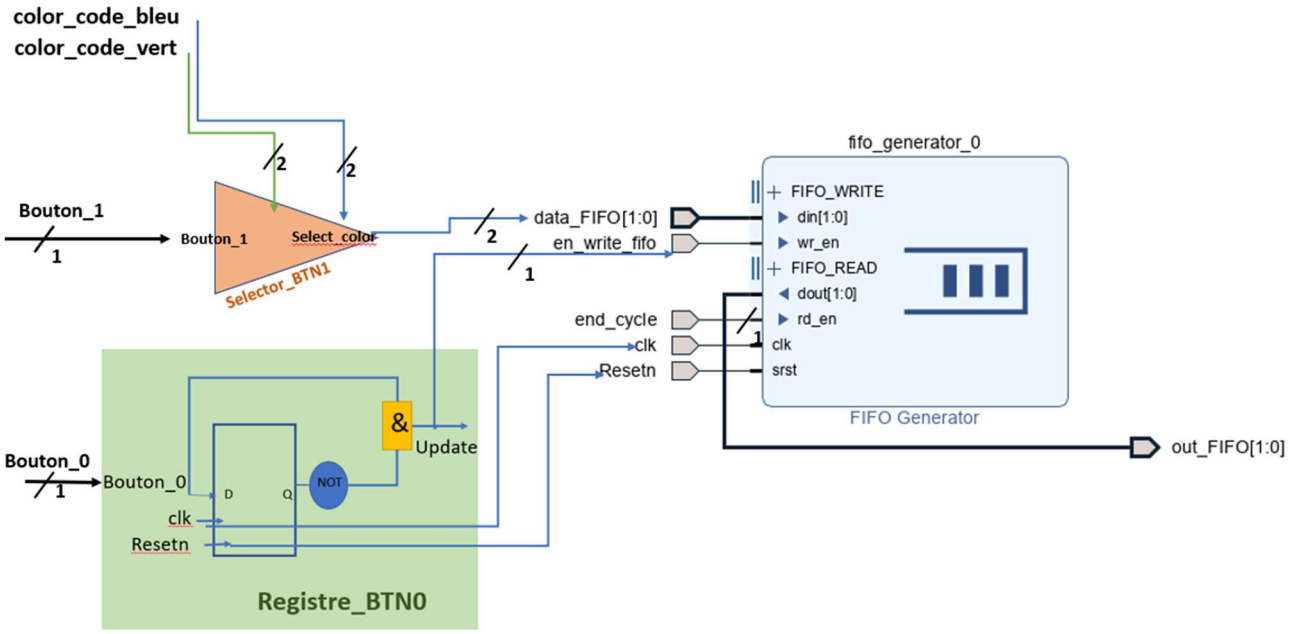

```
test_bouton_1_process : process -- tester ma commande
begin
Bouton_1_tb <= "11";
wait for 10 sec;
Bouton_1_tb <= "01";
wait for 10 sec;
Bouton_1_tb <= "10";
wait for 10 sec;
Bouton_1_tb <= "00";
wait for 10 sec;
end process;

test_bouton_0_process : process -- tester l'autorisation de mon ecriture de commande
begin
Bouton_0_tb <= '0';
wait for 5 sec;
Bouton_0_tb <= '1';
wait for 20000 ns;
Bouton_0_tb <= '0';
wait for 10 sec;
Bouton_0_tb <= '1';
wait for 20 sec;
end process;


test_end_cycle_process : process -- tester l'autorisation de la lecture de ma commande
begin
end_cycle_tb <= '0';
wait for 15 sec;
end_cycle_tb <= '1';
wait for 20000 ns;
end process;
```

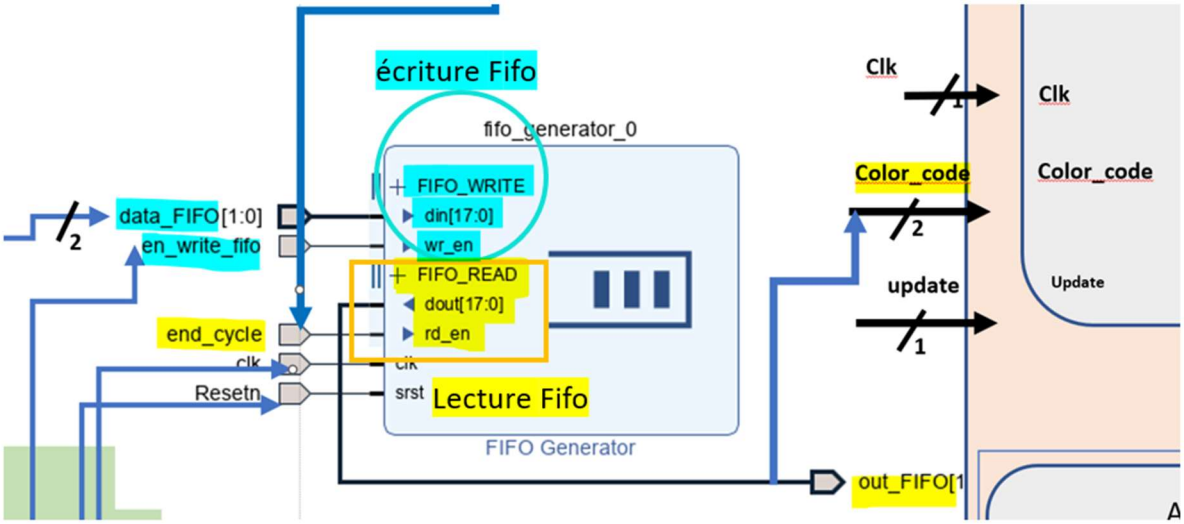
Ensuite je reviens à mon projet MASTER pour rassembler mon architecture globale, maintenant que j'ai compris le fonctionnement d'un composant FIFO :

MASTER GLOBAL qui englobe : nouveau LED_DRIVER (Question 1) & composant FIFO









Une fois notre IP est bien importé et intégré dans notre code puis instancié on va simuler le comportement fonctionnel de l'ensemble de mon architecture.

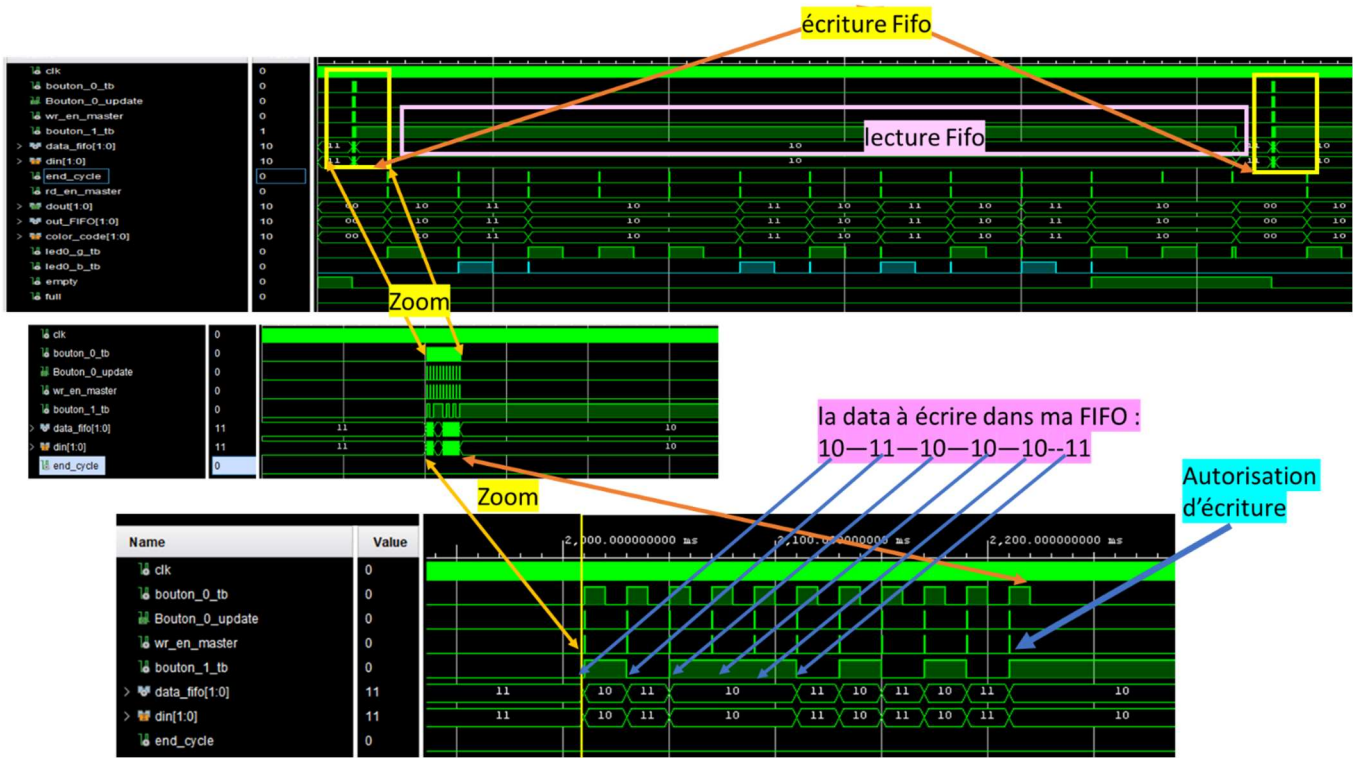
- **<u>Importation de mon IP</u>** FIFO (fifo_generator_0) : ici j'ai 2 bits en entrée et sortie de ma fifo

```
component fifo_generator_0 is
PORT (
        clk                     : IN  std_logic := '0';
        srst                    : IN  std_logic := '0';
        wr_en                   : IN  std_logic := '0';
        rd_en                   : IN  std_logic := '0';
        din                     : IN  std_logic_vector(1 DOWNTO 0) := (OTHERS => '0');
        dout                    : OUT std_logic_vector(1 DOWNTO 0) := (OTHERS => '0');
        full                    : OUT std_logic := '0';
        empty                   : OUT std_logic);-- := '1');

end component;
```

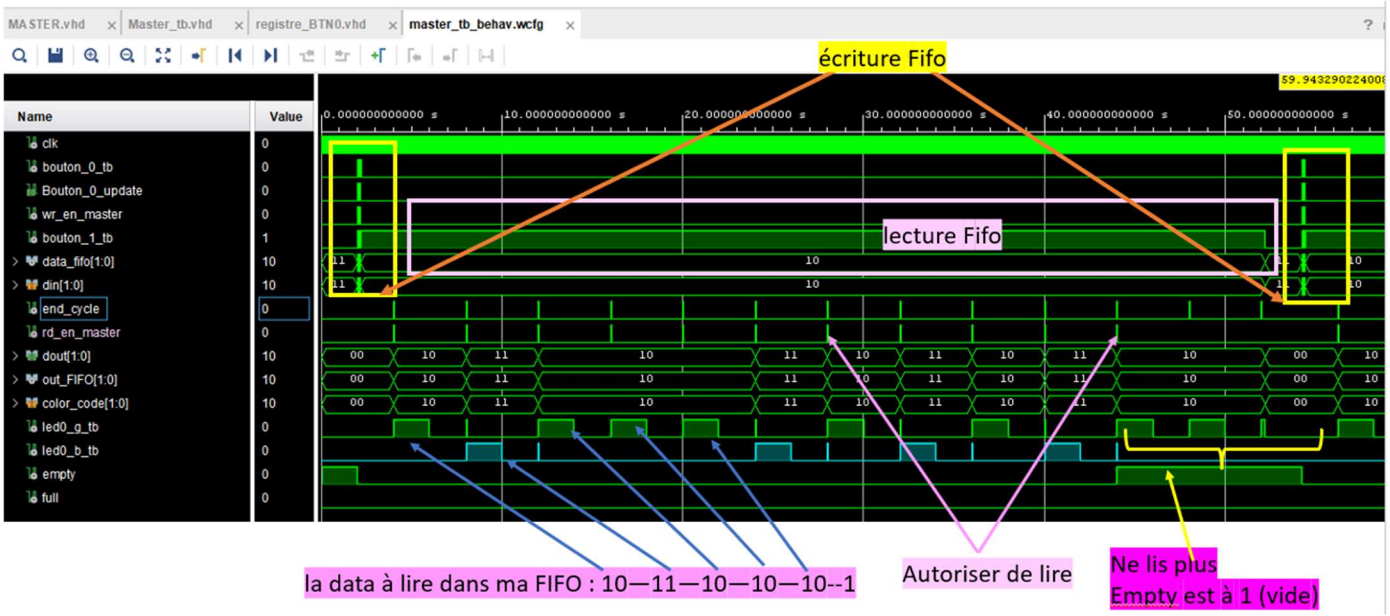
- **<u>Instanciation de mon IP</u>** FIFO (fifo_generator_0) : ici dans mon module MASTER GLOBAL :

```
inst_fifo_generator_0: fifo_generator_0 port map(
       clk => clk, --
       srst => Reset_master, ---
       din =>  data_fifo,
       wr_en=> wr_en_master, --
       rd_en => rd_en_master,---
       dout => out_FIFO,
      full =>  full, --
      empty => empty );
```

- **<u>Explication de l'écriture de la fifo dans mon architecture</u>**



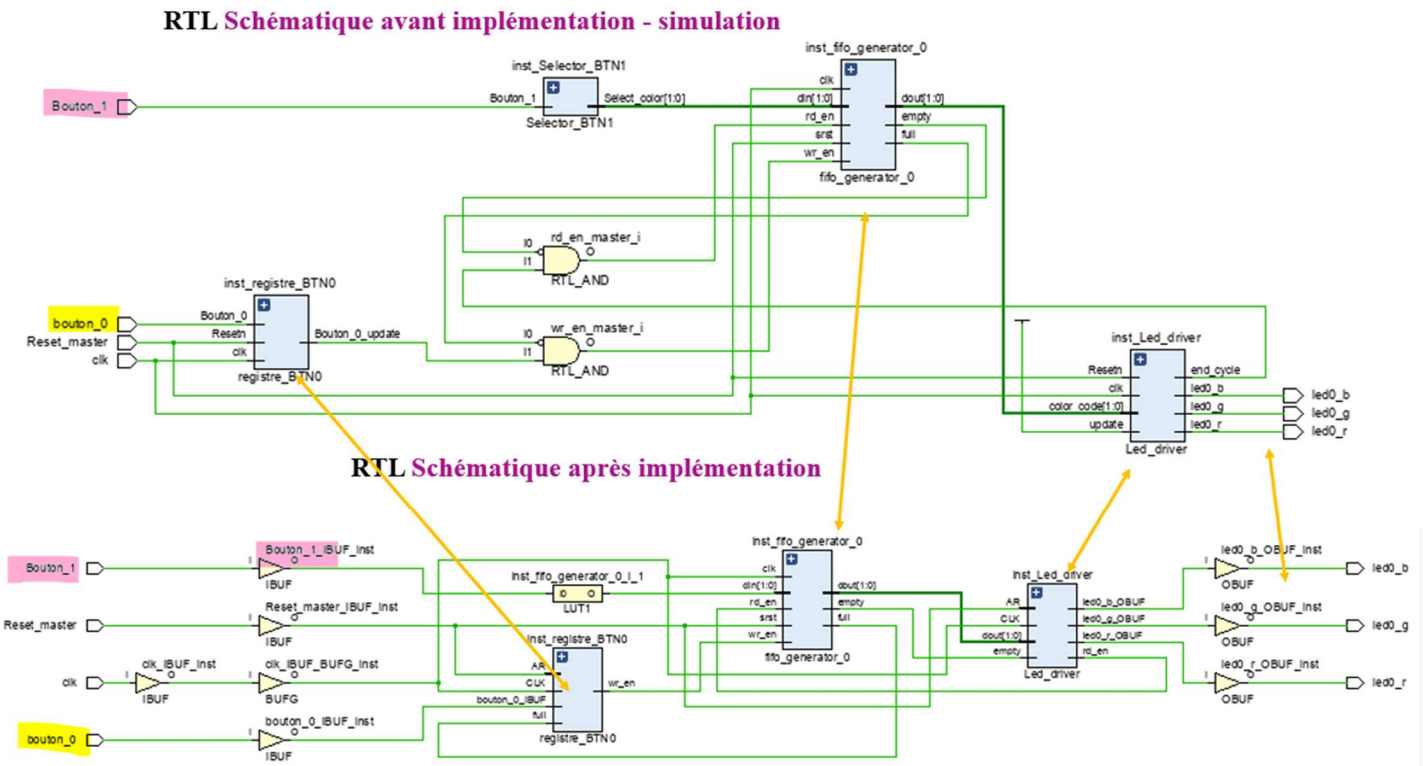
- **Explication de la lecture de la fifo dans mon architecture**



- **Zoom sur le pic bleu juste avant le changement de bit s en lecture de data: Explication**

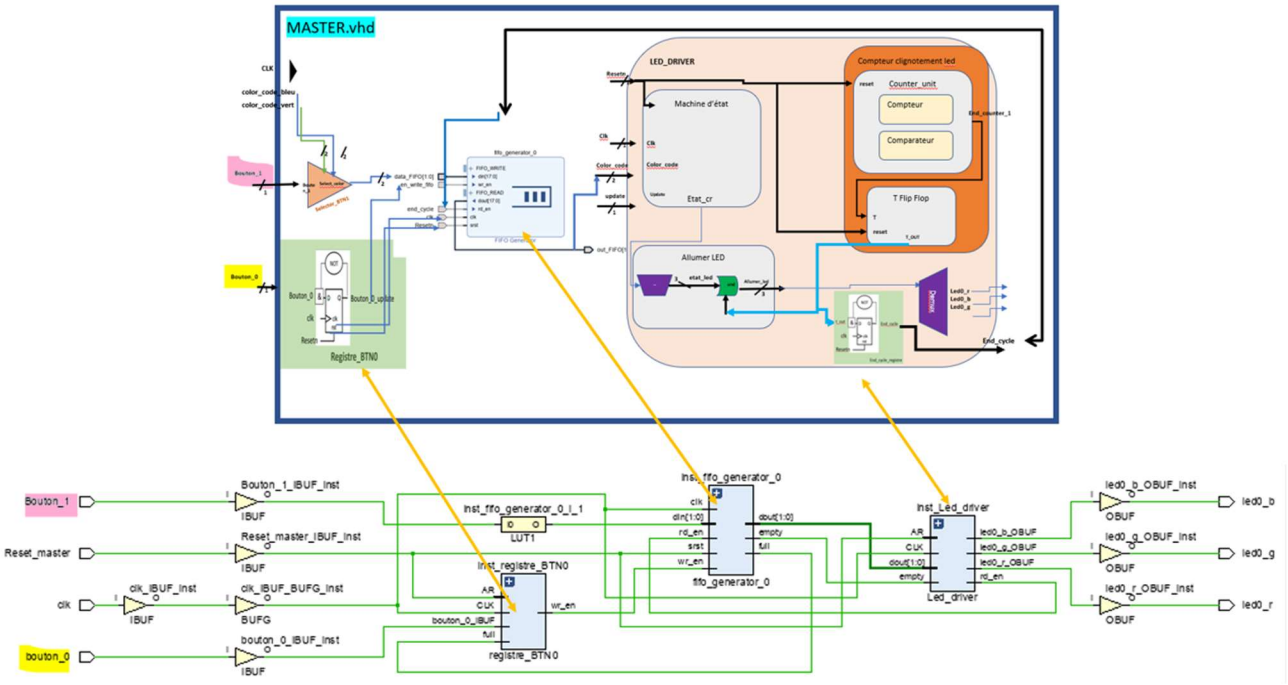
Question 5 : Réalisez une synthèse et étudiez le rapport de synthèse, les ressources utilisées doivent

correspondre à votre schéma RTL.



1.2 Schématique :

Ma schématique après implémentation en comparaison avec mon architecture :



Question6 : Effectuez le placement routage et étudiez les rapports..

1.3 Rapport de synthèse

```
Start RTL Component Statistics
---------------------------------------------
Detailed RTL Component Info :
+---Adders :
      2 Input   10 Bit       Adders := 2
+---XORs :
      2 Input      1 Bit         XORs := 40
+---Registers :
                  10 Bit    Registers := 4
                   3 Bit    Registers := 1
                   1 Bit    Registers := 9
+---Muxes :
      6 Input     3 Bit        Muxes := 1
      5 Input     3 Bit        Muxes := 1
      2 Input     2 Bit        Muxes := 1
      2 Input     1 Bit        Muxes := 1
---------------------------------------------
Finished RTL Component Statistics
```

```
Report Cell Usage:
+------+---------+------+
|      |Cell     |Count |
+------+---------+------+
|1     |BUFG     |     1|
|2     |CARRY4   |     7|
|3     |LUT1     |     7|
|4     |LUT2     |    12|
|5     |LUT3     |     6|
|6     |LUT4     |    26|
|7     |LUT5     |     5|
|8     |LUT6     |    11|
|9     |MUXCY    |    20|
|10    |RAMB18E1 |     1|
|11    |FDRE     |    80|
|12    |IBUF     |     3|
|13    |OBUF     |     3|
+------+---------+------+
```

1.4 Rapport de timing

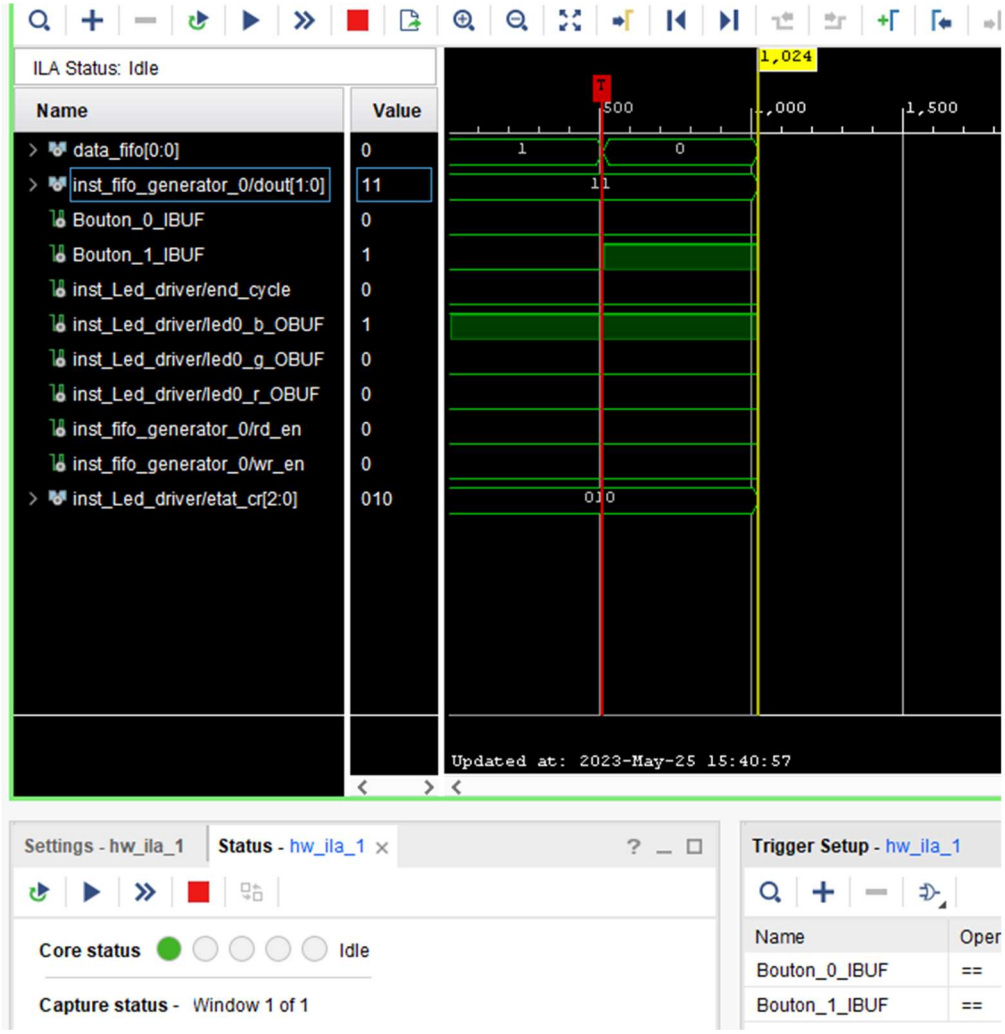
```
------------------------------------------------------------------------------------------------
 Design Timing Summary
 ---------------------
------------------------------------------------------------------------------------------------

    WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints      WHS(ns)      THS(ns)
    -------      -------  ---------------------  -------------------      -------      -------
      3.740        0.000                      0                 4601        0.031        0.000


Setup :            0  Failing Endpoints,  Worst Slack       26.013ns,  Total Violation        0.000ns
Hold  :            0  Failing Endpoints,  Worst Slack        0.049ns,  Total Violation        0.000ns
PW    :            0  Failing Endpoints,  Worst Slack       15.250ns,  Total Violation        0.000ns
```

Et le chemin critique :

```
Max Delay Paths
--------------------------------------------------------------------------------------
Slack (MET) :             26.013ns  (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_BS
  Destination:            dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_switch/portno_temp_reg[5]/D
                            (rising edge-triggered cell FDRE clocked by dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BS
  Path Group:             dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_inst/SERIES7_BSCAN.bscan_inst/TCK
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            33.000ns  (dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_inst/SERIES7_BSCAN.bscan_
  Data Path Delay:        6.921ns  (logic 2.241ns (32.382%)  route 4.680ns (67.618%))
  Logic Levels:           6  (CARRY4=2 LUT3=1 LUT4=1 LUT5=1 LUT6=1)
  Clock Path Skew:        -0.063ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    3.065ns = ( 36.065 - 33.000 )
    Source Clock Delay      (SCD):    3.503ns
    Clock Pessimism Removal (CPR):    0.375ns
  Clock Uncertainty:      0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.000ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns
```

Question7 : Générez le bitstream et vérifiez que vous avez le comportement attendu sur carte.

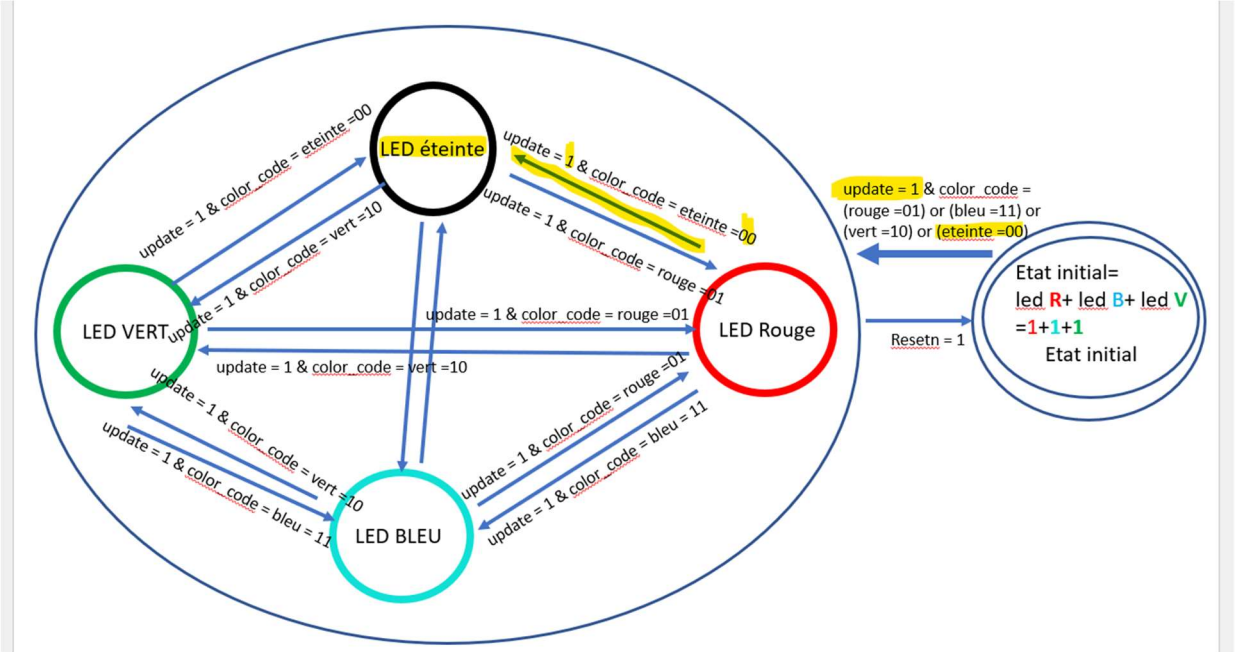
Simulation ILA :



1.5 observation sur carte CORAZ7 : voir vidéo aussi

Explication pourquoi que j'ai un etat initial eteint et non pas blanc dès la programmation de ma carte . :

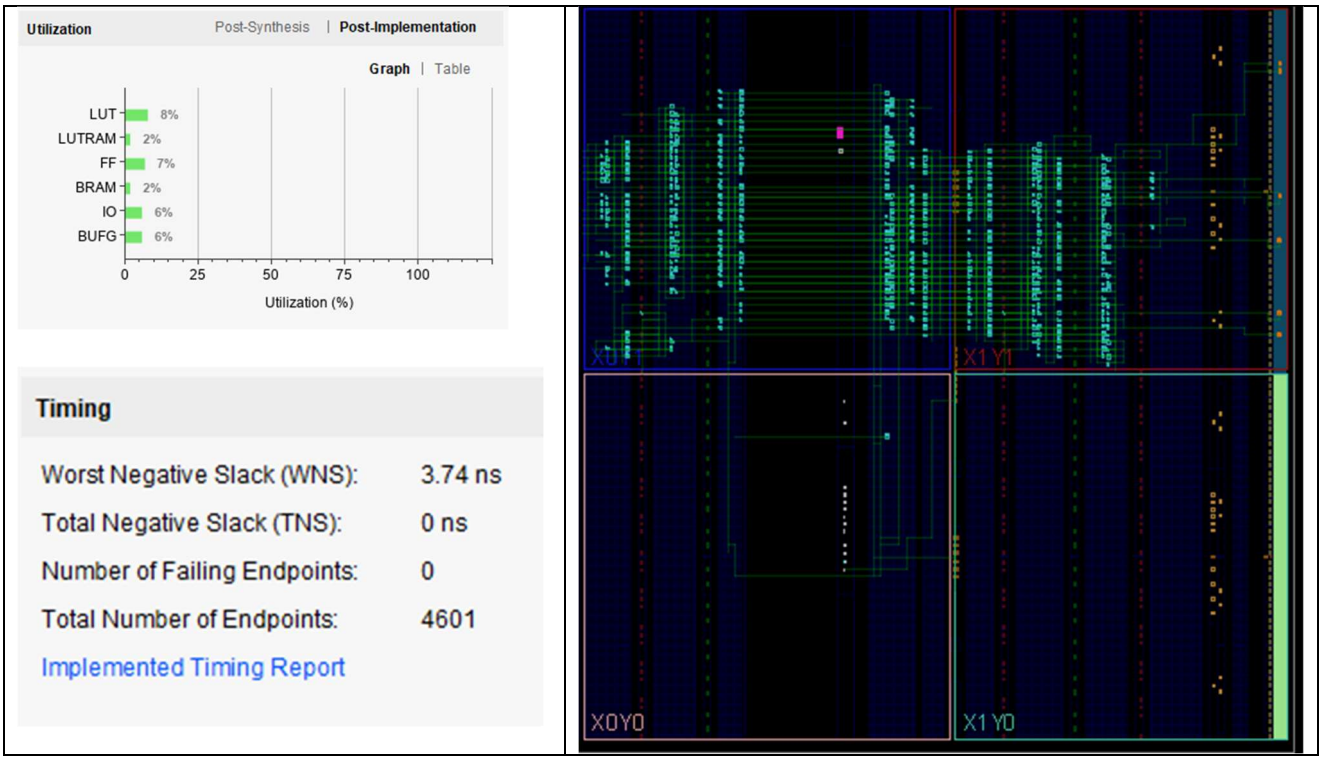
➔la machine à etat dans led_driver.vhd est ci_dessous, et comme j'ai un update à '1' la carte va par defaut sauter au premier etat suivant qui est selon lui le 00 = eteinte, valeur par defaut de color_code.




```
-- Définition de ma machine à etat--
process(color_code,update,etat_cr)
begin
--initialisation des etats:
if update = '0' then
    etat_sv <= etat_cr;
elsif color_code = color_code_eteinte then --& update = '1'
    etat_sv <= led_eteinte;
elsif color_code = color_code_rouge then --& update = '1'
    etat_sv <= led_rouge;
elsif color_code = color_code_bleu then --& update = '1'
    etat_sv <= led_bleu;
elsif color_code = color_code_vert then --& update = '1'
    etat_sv <= led_vert;
end if;
end process;
```

```
  inst_Led_driver : entity Led_driver port map(
      clk => clk,
      Resetn => Reset_master,
      color_code => out_FIFO,
      update => '1', --ici mon update est tjs à 1 donc il saute au premier etat suivant qui est eteint selon ma machine à état
      end_cycle => end_cycle,
      led0_r => led0_r,
      led0_b => led0_b,
      led0_g => led0_g);
```

Annexe



Un aperçu sur mon test bench pour tester ma data à l'entrée de mon architecture avec le composant FIFO :

```
-- Declaration de ma bibliothèque
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
Use work.all;

entity master_tb is
end master_tb;

architecture arch_master_tb of master_tb is
   signal clk : std_logic ;
   --signal Resetn :std_logic ;
   --signal reset_tb :std_logic :='0';
   signal led0_r_tb : std_logic :='0';
   signal led0_b_tb : std_logic :='0';
   signal led0_g_tb : std_logic :='0';
   signal bouton_1_tb : std_logic :='0';
   signal bouton_0_tb : std_logic :='0';
   signal Reset_master_tb : std_logic :='0';
   constant clk_period : time :=  10000 ns;--10 ns;
--  signal color_code : STD_LOGIC_vector (1 downto 0):= "00";
--  signal out_FIFO :  STD_LOGIC_vector (1 downto 0):= "00";

begin
    uut_master : entity Master Port map (
            clk => clk,
            Bouton_1=> Bouton_1_tb,
            Bouton_0 => Bouton_0_tb,
            Reset_master => Reset_master_tb,
            led0_r => led0_r_tb,
            led0_b => led0_b_tb,
            led0_g => led0_g_tb);

    ---Clock process definitions
    clk_process : process
    begin
    clk <= '0';
    wait for clk_period/2;
    clk <= '1';
    wait for clk_period/2;
    end process;
```

```
tester_mes_Boutons_process : process
begin
--initialiser mes registres
    Reset_master_tb <= '1';
    Bouton_0_tb <= '0';
    Bouton_1_tb <= '0';
wait for 2 sec;
   Reset_master_tb <= '0';
   -- simu entrée fifo bouton_1 : 1 0 1 1 1 0 1 0 1 0 1
--1
wait for 10 ms;
   Bouton_1_tb <= '1';
   Bouton_0_tb <= '1';--autorise l'écriture dans le fifo
wait for 10 ms;
   Bouton_1_tb <= '1';
   Bouton_0_tb <= '0';
--0
wait for 10 ms;
   Bouton_1_tb <= '0';
   Bouton_0_tb <= '1';--autorise l'écriture dans le fifo
wait for 10 ms;
   Bouton_1_tb <= '0';
   Bouton_0_tb <= '0';
--1
wait for 10 ms;
   Bouton_1_tb <= '1';
   Bouton_0_tb <= '1';--autorise l'écriture dans le fifo
wait for 10 ms;
   Bouton_1_tb <= '1';
   Bouton_0_tb <= '0';
--1
wait for 10 ms;
   Bouton_1_tb <= '1';
   Bouton_0_tb <= '1';--autorise l'écriture dans le fifo
wait for 10 ms;
   Bouton_1_tb <= '1';
   Bouton_0_tb <= '0';
--1
```

```
        _ _
    --1
    wait for 10 ms;
       Bouton_1_tb <= '1';
       Bouton_0_tb <= '1';--autorise l'écriture dans le fifo
    wait for 10 ms;
       Bouton_1_tb <= '1';
       Bouton_0_tb <= '0';
    --0
    wait for 10 ms;
       Bouton_1_tb <= '0';
       Bouton_0_tb <= '1';--autorise l'écriture dans le fifo
    wait for 10 ms;
       Bouton_1_tb <= '0';
       Bouton_0_tb <= '0';
    --1
    wait for 10 ms;
       Bouton_1_tb <= '1';
       Bouton_0_tb <= '1';--autorise l'écriture dans le fifo
    wait for 10 ms;
       Bouton_1_tb <= '1';
       Bouton_0_tb <= '0';
    --0
    wait for 10 ms;
       Bouton_1_tb <= '0';
       Bouton_0_tb <= '1'; --autorise l'écriture dans le fifo
    wait for 10 ms;
       Bouton_1_tb <= '0';
       Bouton_0_tb <= '0';
    --1
    wait for 10 ms;
       Bouton_1_tb <= '1';
       Bouton_0_tb <= '1'; --autorise l'écriture dans le fifo
    wait for 10 ms;
       Bouton_1_tb <= '1';
       Bouton_0_tb <= '0';
    wait for 50 sec;

    end process;

end arch_master_tb;
```