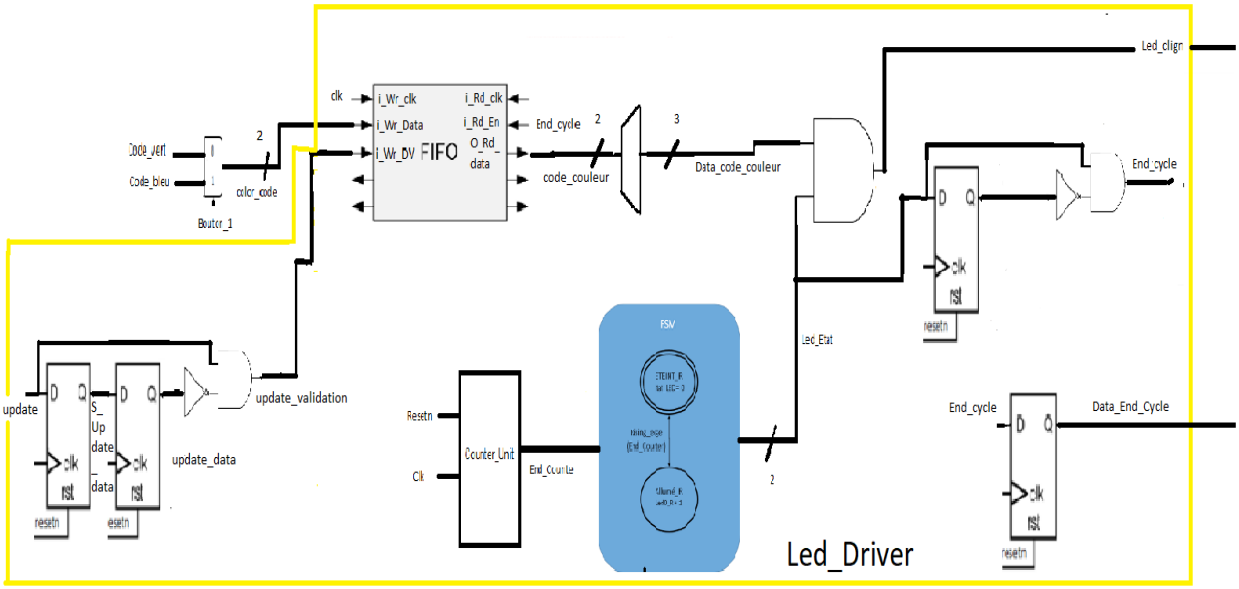
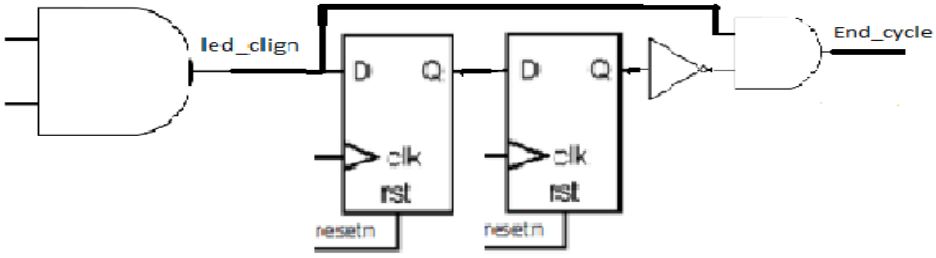
1. **Sur l'architecture RTL, modifiez le module LED_driver en ajoutant une sortie end_cycle. Cette sortie vaudra 1 à la fin d'un cycle allumé/éteint de la LED RGB.**





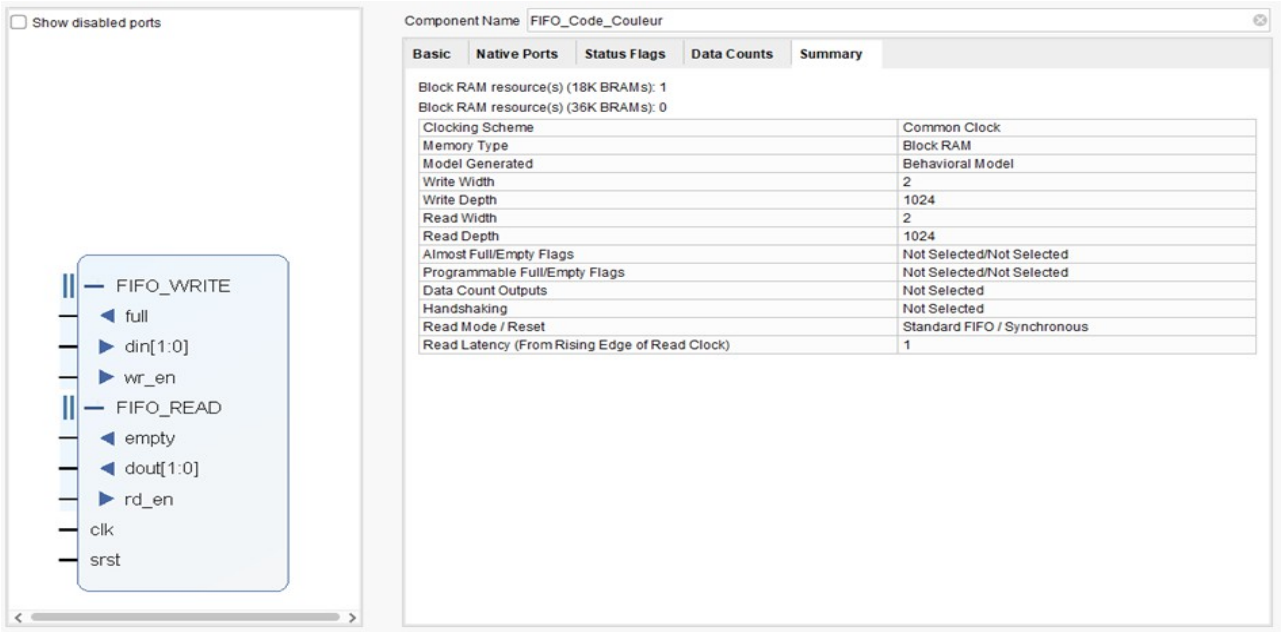
Cela permet d'avoir un front montant à chaque fin de cycle ainsi End_cycle prendra la valeur 1 à chaque fin d'un cycle.

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.**

La Fifo créée un décalage entre l'entrée End_Cycle et la sortie Code_Couleur. Pour régler ce

problème, il faut ajouter un registre sur le signal End_Cycle afin d'avoir les signaux Led_Clignotement et Data_End_Cycle synchronisé.

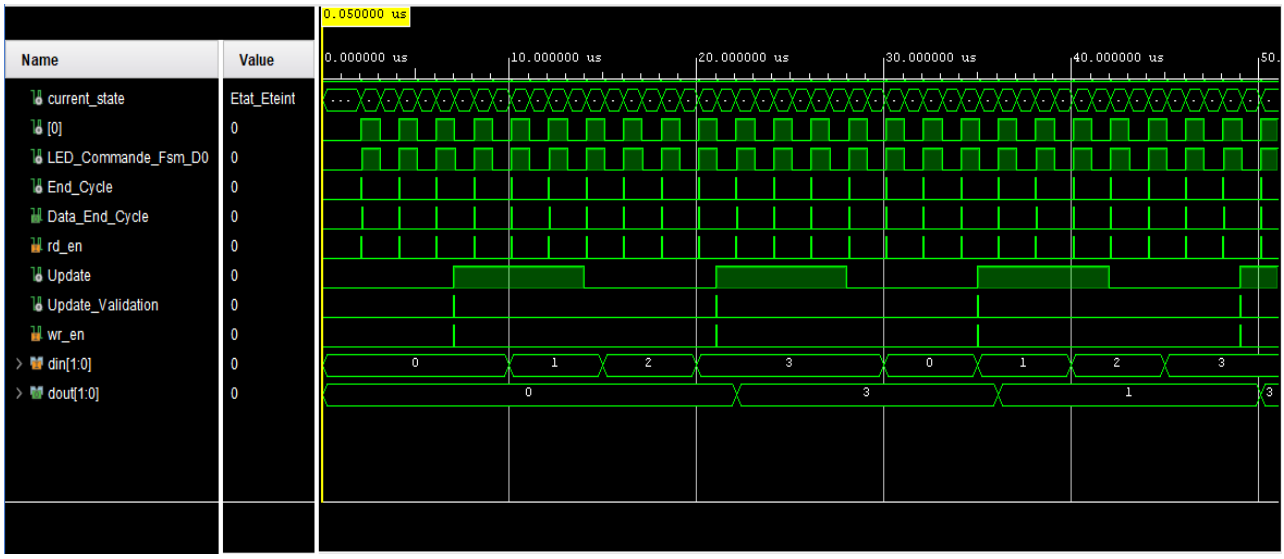
3. **Modifiez vos codes de la partie 1 pour y ajouter les nouveaux éléments de votre architecture**




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


entity LedDriver is
    port (
        clk         : in std_logic;
        resetn      : in std_logic;
        Color_Code  : in std_logic_Vector (1 downto 0);
        Update      : in std_logic;
        LED_Output  : out std_logic_vector (2 downto 0);
        Data_End_Cycle : out std_logic
    );
end LedDriver;

architecture behavioral of LedDriver is

    -- Lié à la machine d'état.
    type state is (Etat_Eteint, Etat_Allume);
    signal current_state : state;  -- Etat dans lequel on se trouve actuellement
    signal next_state : state;     -- Etat dans lequel on passera au prochain coup d'horloge
    signal LED_Commande_Fsm : std_logic_vector (2 downto 0); -- Signal indiquant le status de notre FSM (Leds ON ou Leds OFF)
    signal LED_Commande_Fsm_D0 : std_logic;
    signal End_Cycle : std_logic;
    signal S_Data_End_Cycle : std_logic;

    -- Lié au Compteur.
    signal End_Counter  : std_logic;

    -- Signaux interne
    signal S_LED_Output      : std_logic_vector (2 downto 0); -- Signal de sortie porte AND entre la FIFO et FSM

    -- Signal lié à Update
    signal Update_data       : std_logic;
    signal S_Update_data       : std_logic;
    signal Update_Validation : std_logic;

    signal Code_Couleur      : std_logic_vector (1 downto 0); -- Signal de sortie de la Fifo sur 2 bits.
    signal Data_Code_Couleur : std_logic_vector (2 downto 0); -- traduction de Code_Couleur sur 3 bits.
```

```
-- Composant FIFO_Code_Couleur
component FIFO_Code_Couleur
        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);
            full  : out std_logic;
            empty : out std_logic
        );
    end component;


begin
--Affectation des signaux du testbench avec ceux de l'entite a tester
Compteur : Counter_unit
    port map (
        clk => clk,
        resetn => resetn,
        End_Counter => End_Counter
    );
Fifo : FIFO_Code_Couleur
        port map (
            clk => clk,
            srst => resetn,
            din => Color_Code,
            wr_en => Update_Validation,
            rd_en => End_Cycle,
            dout => Code_Couleur
        );

    -- ************** Gestion du reset ************** --
    process(resetn, clk)
    begin
        if(resetn = '1') then
            -- Lors du reset on réinitialise la machine d'état.
            current_state <= Etat_Eteint;
        elsif(rising_edge(clk)) then
            -- On change d'état à chaque front montant de End_Counter.
            current_state <= next_state;
        end if;
    end process;
    -- ***************** Fin du reset ***************** --

    -- **************  Début Process detection de front sur le signal Update ***************** --
    process(resetn, clk)
    begin
        if(resetn = '1') then
            Update_data <= '0';
        elsif(rising_edge(clk)) then
            -- On change d'état à chaque front montant de End_Counter.
            if(Update = '1') then
                S_Update_data <= Update;
                Update_data <= S_Update_data;
            else
                Update_data <= '0';
                S_Update_data<='0';
            end if;
        end if;
    end process;
    -- **************  Fin Process detection de front sur le signal Update ***************** --
    Update_Validation <= not (Update_data) and Update;


    -- ************** Début de processe de la Machine d'états ************** --
    process(current_state, next_state, END_Counter) -- Liste des signaux de sensibilité du processe
    begin
       case current_state is
            -- L'indicateur Led_Commande passe à 0.
            when Etat_Eteint =>
                LED_Commande_Fsm <= "000";
                if(rising_edge(END_Counter)) then
                    next_state <= Etat_Allume;
                end if;
```

```
            -- L'indicateur Led_Commande passe à 1.
            when Etat_Allume =>
                LED_Commande_Fsm <= "111";
            if(rising_edge(END_Counter)) then
                    next_state <= Etat_Eteint;
                end if;

        end case;
    end process;
    -- ************** FIN de processe de la Machine d'états ************** --

    -- ************** Partie combinatoire de la gestion de la traduction 2 bits vers 3 bits ************** --
    Data_Code_Couleur <= "000" when (Code_Couleur = "00")
        else "001" when (Code_Couleur = "01")
          else "010" when (Code_Couleur = "10")
            else "100" when (Code_Couleur = "11")
              else "000";
    -- ************** Fin combinatoire de la gestion de la traduction 2 bits vers 3 bits ************** --

    -- ************** DEB Partie process de la gestion de END_CYCLE ************** --
    process(resetn, clk)
    begin
        if(resetn = '1') then
            -- Lors du reset on réinitialise les signaux lié au Code Couleur.
            --End_Cycle   <= '0';
            LED_Commande_Fsm_D0 <= '0';
        elsif(rising_edge(clk)) then
            -- On change d'état à chaque front montant de End_Counter.
            if(LED_Commande_Fsm(0) = '1') then
                LED_Commande_Fsm_D0 <= LED_Commande_Fsm(0);
            else
                LED_Commande_Fsm_D0 <= '0';
            end if;
        end if;
    end process;
    -- ************** FIN Partie process de la gestion de END_CYCLE ************** --
    -- ************** DEB Partie Combinatoire de la gestion de END_CYCLE ************** --
    End_cycle <= not(LED_Commande_Fsm_D0) and LED_Commande_Fsm(0);
    -- ************** DEB Partie Combinatoire de la gestion de END_CYCLE ************** --

    -- ************** DEB Partie process de la gestion de DATA_END_CYCLE ************** --
    process(resetn, clk)
    begin
        if(resetn = '1') then
            -- Lors du reset on réinitialise les signaux lié au Code Couleur.
            S_Data_End_Cycle <= '0';
        elsif(rising_edge(clk)) then
            S_Data_End_Cycle <= End_Cycle;
        --else
            --S_Data_End_Cycle <= S_Data_End_Cycle;
        end if;
    end process;
    -- ************** FIN Partie process de la gestion de DATA_END_CYCLE ************** --
    Data_End_Cycle <= S_Data_End_Cycle;

    -- ************** Partie Combinatoire entre Fsm et Code Couleur ************** --
    S_LED_Output <= Data_Code_Couleur and LED_Commande_Fsm;
    -- Signal de sortie prends le signal interne.
    LED_Output <= S_LED_Output;
    -- ************** Fin Combinatoire entre Fsm et Code Couleur ************** --

end behavioral;
```



Cette simulation nous permet de valider notre composant Led_Driver avec utilisation d'une FIFO. Lorsque le signal wr_en de la FIFO détecte un front montant elle enregistre la valeur présente sur

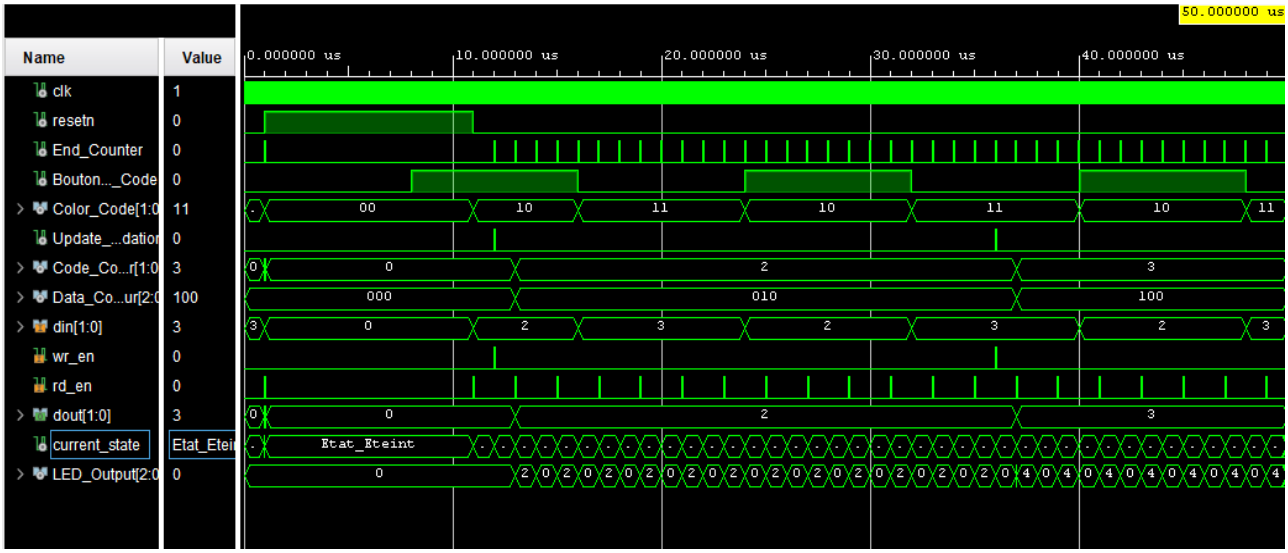
l’entre Din de cette dernière. Et lorsque le signal re_en passe à 1 cette valeur est envoyé à la sortie Dout de la FIFO.

Ensuite j'ai modifié mon fichier top pour y ajouter mon nouveau module Led_Driver.

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

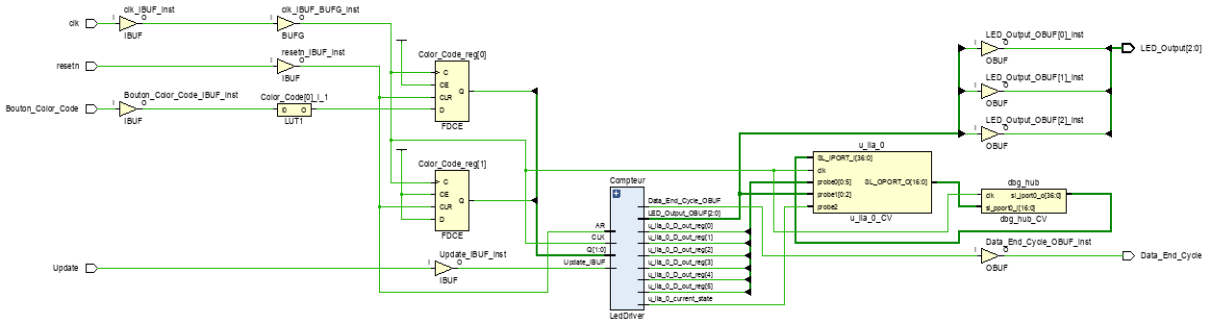
J'ai réutilisé le même testbench que lors de la partie 1 du TP4.

```
    -- Process Gestion du resetn
    process
       begin
            wait for 100*period;
            resetn <= '1';
            wait for 4000*period;
            resetn <= '0';
            wait for 4000*period;
    end process;

    -- Process Gestion du Bouton_1 : Code Couleur
    process
       begin
            for i in 1 to 4 loop
                wait for 800*period;
                Bouton_Color_Code <= not Bouton_Color_Code;
            end loop;
    end process;

    -- Process Gestion du Bouton_0 : Update
    process
       begin
            for i in 1 to 3 loop
                wait for 1200*period;
                Buton_Update <= not Buton_Update;
            end loop;
    end process;

end behavioral;
```



La dernière simulation montre que notre système fonctionne bien avec la mise en place de la Fifo dans le fichier top. En fonction du code couleur et le la sortie de le Fifo on fait clignoter la LED désiré.

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




```
---------------------------------------------------------------------
Start RTL Component Statistics
---------------------------------------------------------------------
Detailed RTL Component Info :
+---Registers :
                   2 Bit    Registers := 1
                   1 Bit    Registers := 6
+---Muxes :
       2 Input    3 Bit        Muxes := 1
       5 Input    3 Bit        Muxes := 1
       2 Input    2 Bit        Muxes := 1
---------------------------------------------------------------------
Finished RTL Component Statistics
---------------------------------------------------------------------
Report Cell Usage:
+------+------------------+------+
|      |Cell              |Count |
+------+------------------+------+
|1     |FIFO_Code_Couleur |     1|
|2     |BUFG              |     1|
|3     |CARRY4            |     7|
|4     |LUT1              |     3|
|5     |LUT2              |     2|
|6     |LUT3              |     3|
|7     |LUT4              |     1|
|8     |LUT5              |     1|
|9     |LUT6              |    35|
|10    |FDCE              |    35|
|11    |IBUF              |     4|
|12    |OBUF              |     4|
+------+------------------+------+
```

On voit bien que l'ensemble, semble conforme à notre design et notre schéma RTL. Avec le bon nombre de registre, de multiplexeurs, d'adders et d'entrées / sortie de notre système. La synthèse correspond bien à notre schéma RTL. De plus, on a bien un registre en moins que sur la partie 1 car

il est remplacé par le registre. On a bien nos 3 muxes, on a 4 entrées/4 sorties.

6. **Effectuez le placement routage et étudiez les rapports.**

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

    WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints      WHS(ns)      THS(ns)  T
    -------      -------  ---------------------  -------------------      -------      -------  -
      1.993        0.000                      0                 3825        0.031        0.000


All user specified timing constraints are met.
```

Notre système ne devrait pas rencontrer de métastabilité car on ne retrouve pas de slack (TNS et THS = 0ns)

Chemin critique :

```
Max Delay Paths
--------------------------------------------------------------------------------------
Slack (MET) :           25.453ns  (required time - arrival time)
  Source:               dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_switch/state_reg[0]/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@
  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@
```

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

Voir la vidéo