

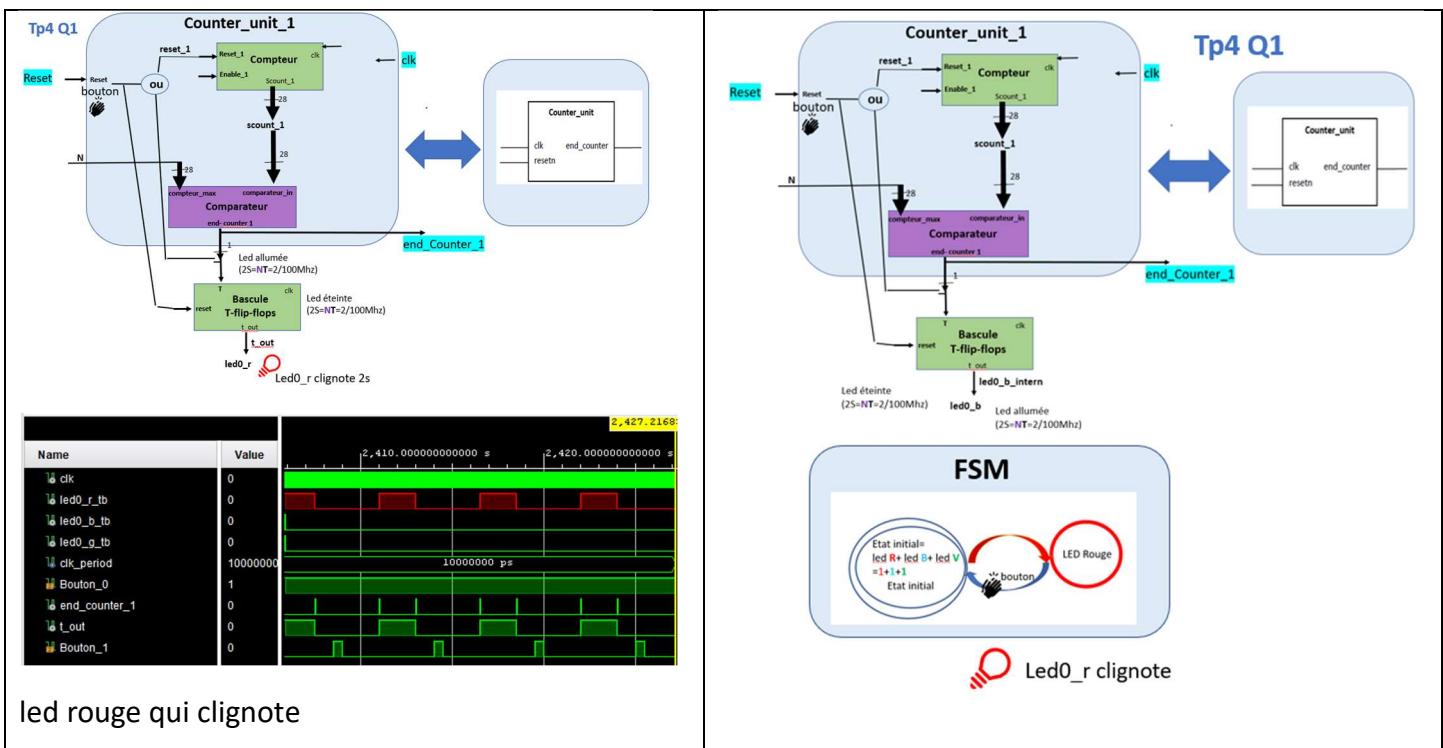
# Compte rendu de – TP04 - Pilotage de LED et mémoire

## Partie 1

### 1.1 Objectif de ce TP

L'objectif de ce TP est de réaliser une architecture permettant de faire clignoter deux LEDs RGB en rouge, vert et bleu. Le pilotage des LEDs se fera à l'aide de machines à états.

- 1) Question 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.



### Coeur de mon code SOURCE FSM pour clignoter led rouge

```

process( clk , Bouton_1)
begin -- définition de l'initialisation de ma machine à etat-- remise
    if bouton_1='1' then etat_cr <= etat_initial;
    elsif rising_edge(clk) then etat_cr <= etat_sv;
    end if;
end process;

-- Définition de ma machine à etat--
process(etat_cr, end_counter_1)
begin
--initialisation des etats:
etat_sv <= etat_cr;
    case etat_cr is
        when etat_initial =>
            if end_counter_1 = '1' then
                etat_sv <= led_rouge;
            end if;
        when others =>
            etat_sv <= led_rouge;
        end case;
end process;

process(etat_cr, bouton_1, end_counter_1)--state_switch
begin
case etat_cr is
    when etat_initial =>
        etat_led <= v_led_blancl;      -- blanc => rouge + bleu + vert
    when led_rouge =>
        etat_led <= v_led_rouge; -- rouge
    when led_bleu =>
        etat_led <= v_led_bleu; -- bleu
    when led_vert =>
        etat_led <= v_led_vert; -- vert
    when others =>
        etat_led <= v_led_rouge; -- rouge
    end case;
end process;

process (clk,etat_led)
begin
if rising_edge(clk) then
    allumer_led <= etat_led and (t_out, t_out, t_out);
    led0_r <= allumer_led(0);
    led0_b <= allumer_led(1);
    led0_g <= allumer_led(2);
end if;
end process;

end arch_TP4_FSM;

```

### Coeur de mon code SOURCE TB pour clignoter led rouge

```

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

entity TP4_tb is
end TP4_tb;

architecture arch_TP4_tb of TP4_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 end_counter : std_logic ;
    constant clk_period : time := 10 ns;--10000 ns;

begin
    uut_TP4_FSM : entity TP4_FSM Port map (
        clk => clk,
        Bouton_1 => reset_tb,
        Bouton_0 => '1',
        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;

    bouton_process : process
    begin
        reset_tb <= '0';
        wait for 5000 ms;
        reset_tb <= '1';
        wait for 1000 ms;
        reset_tb <= '0';
    end process;

```

end arch\_TP4\_tb;

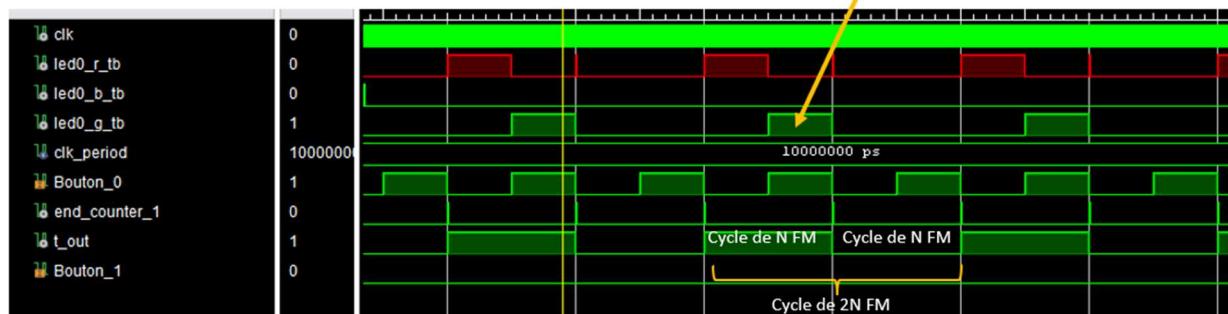
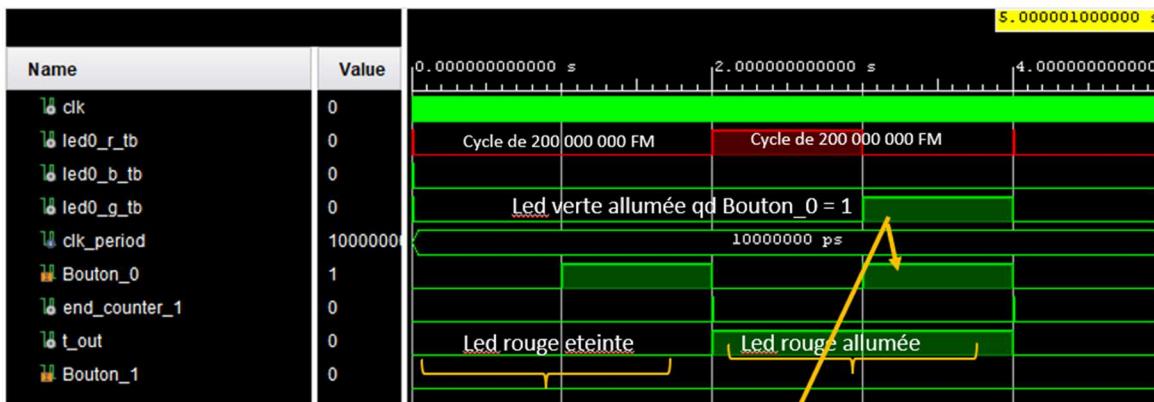
- 2) Modifiez votre architecture pour piloter une LED rouge et une LED verte. Lorsque le bouton\_0 est appuyé, la LED verte est allumée, sinon la LED rouge est allumée.+ Question3

<p>Etat initial= led R+ led B+ led V =1+1+1 Etat initial</p>	<p>Cœur de mon code SOURCE FSM pour clignoter led rouge et allumer led verte, seulement en cas de bouton-0 est égal à 1</p> <pre>-- Définition de ma machine à état-- process(etat_cr, end_counter_1) begin   --initialisation des états:   etat_sv &lt;= etat_cr;   case etat_cr is     when etat_initial =&gt;       if end_counter_1 = '1' then         etat_sv &lt;= led_rouge;       end if;     when led_rouge =&gt;       if bouton_0 = '1' then         etat_sv &lt;= led_vert;       end if;     when led_vert =&gt;       if bouton_0 = '0' then         etat_sv &lt;= led_rouge;       end if;     when others =&gt;       etat_sv &lt;= led_rouge;   end case; end process;  process(etat_cr, bouton_0, bouton_1,end_counter_1)--state_switch begin   case etat_cr is     when etat_initial =&gt;       etat_led &lt;= v_led_blan; -- blanc =&gt; rouge + bleu + vert     when led_rouge =&gt;       etat_led &lt;= v_led_rouge; -- rouge     when led_bleu =&gt;       etat_led &lt;= v_led_bleu; -- bleu     when led_vert =&gt;       etat_led &lt;= v_led_vert; -- vert     when others =&gt;       etat_led &lt;= v_led_rouge; -- rouge   end case; end process;  process (clk,etat_led) begin   if rising_edge(clk) then     allumer_led &lt;= etat_led and (t_out, t_out, t_out);     led0_r &lt;= allumer_led(0);     led0_b &lt;= allumer_led(1);     led0_g &lt;= allumer_led(2);   end if; end process; end arch_TP4_FSM;</pre>
--	--

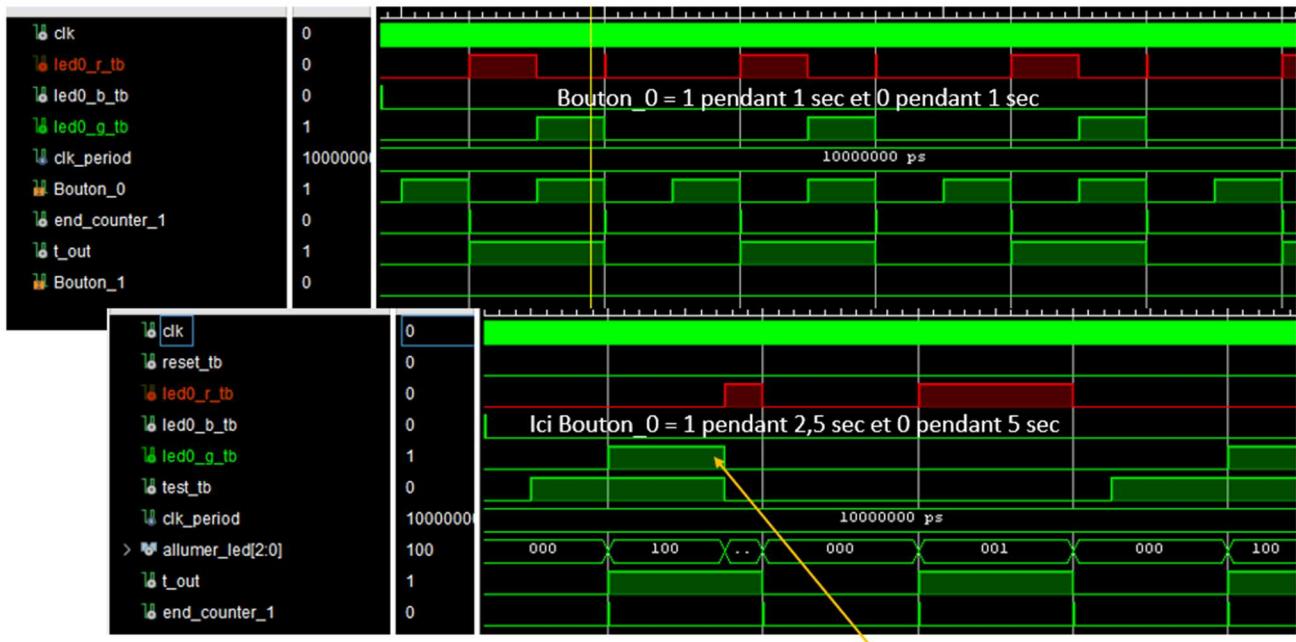
Voir ci-dessous

Explication simulation sur TB :

J'ai laissé 1 seconde bouton\_0 appuyé



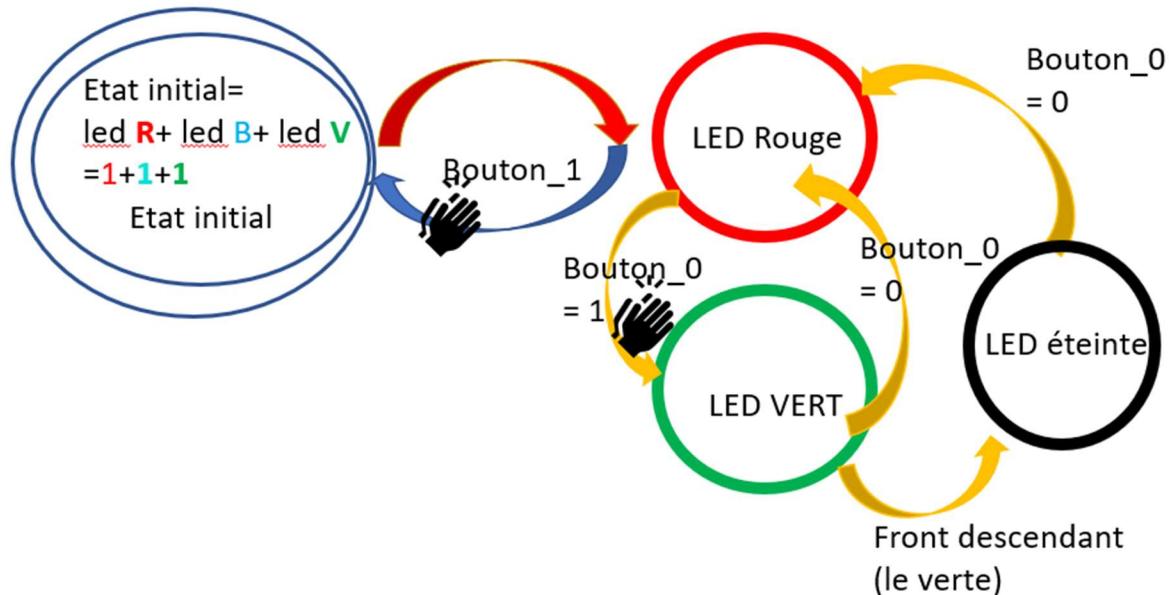
4. Réalisez une simulation en rédigeant un testbench. Que se passe-t-il si le bouton est pressé pendant plus d'un cycle d'horloge ?



Si je reste plus d'un cycle N la led reste allumée **en vert**

5. Que faudrait-il faire pour que la LED ne clignote en vert qu'une seule fois même si le bouton est maintenu ?  
+ Question6 + Question7

J'ai ajouté un état à ma machine à état : **Etat 'led éteinte'**



<pre> -- Définition de ma machine à état -- process(etat_cr, end_counter_1, bouton_0, allumer_led) begin     --initialisation des états:     etat_sv &lt;= etat_cr;     case etat_cr is         when etat_initial =&gt;             if end_counter_1 = '1' then                 etat_sv &lt;= led_rouge;             end if;         when led_rouge =&gt;             if bouton_0 = '1' then                 etat_sv &lt;= led_vert;             end if;         when led_vert =&gt;             if bouton_0 = '0' then                 etat_sv &lt;= led_rouge;             elsif falling_edge(allumer_led(2)) then                 etat_sv &lt;= led_etainte;             end if;         when led_etainte =&gt;             if bouton_0 = '0' then                 etat_sv &lt;= led_rouge;             end if;         when others =&gt;             etat_sv &lt;= led_rouge;     end case; end process; </pre>	<pre> architecture arch_TP4_FSM of TP4_FSM is     -- Définition des signaux     signal end_counter_1 : STD_LOGIC := '0';     signal end_counter_2 : STD_LOGIC := '0';     type state is (etat_initial, led_rouge, led_bleu, led_vert, led_etainte);     signal etat_cr, etat_sv : state; --état dans lequel on se trouve actuellement     signal etat_led : std_logic_vector(2 downto 0);     signal allumer_led : std_logic_vector(2 downto 0);     constant v_led_rouge : std_logic_vector := "001";     constant v_led_bleu : std_logic_vector := "010";     constant v_led_vert : std_logic_vector := "100";     constant v_led_blan : std_logic_vector := "111";     constant v_led_etainte : std_logic_vector := "000";     signal t_out : std_logic;  process(etat_cr)--state_switch begin     case etat_cr is         when etat_initial =&gt;             etat_led &lt;= v_led_blan; -- blanc =&gt;         when led_rouge =&gt;             etat_led &lt;= v_led_rouge; -- rouge         when led_bleu =&gt;             etat_led &lt;= v_led_bleu; -- bleu         when led_vert =&gt;             etat_led &lt;= v_led_vert; -- vert         when others =&gt;             etat_led &lt;= v_led_etainte; -- rouge     end case; end process; </pre>
---	--

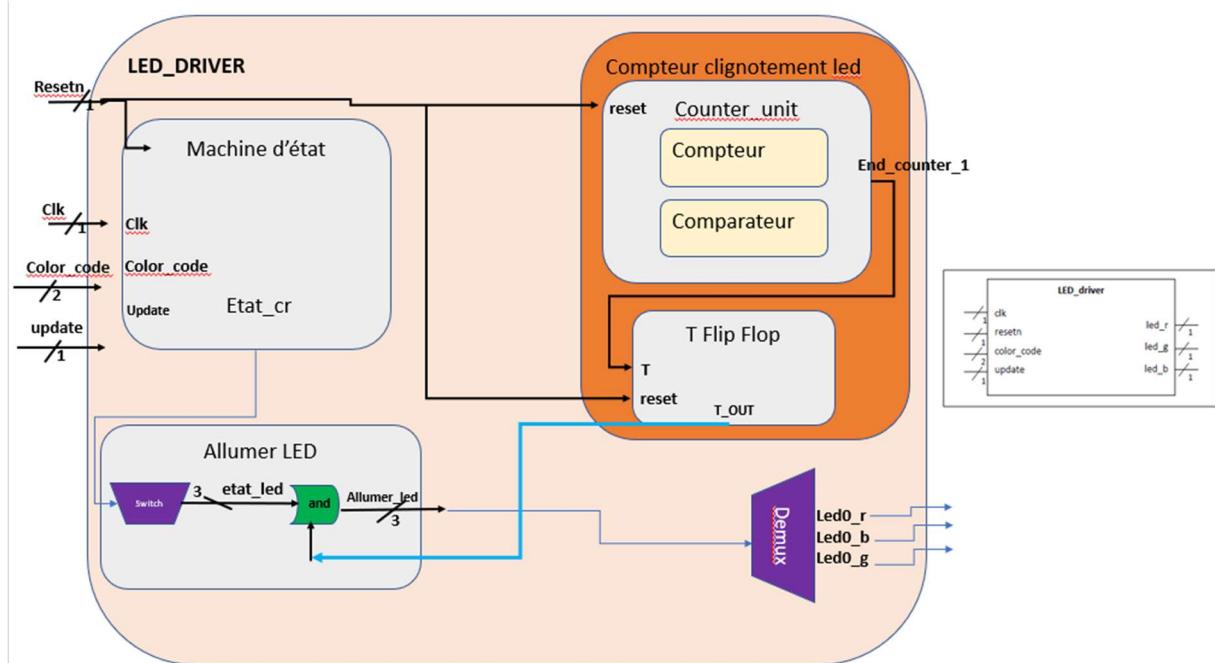


**Question 8 :** Créez un module de pilotage d'une LED RGB en RTL. Ce dernier doit permettre de faire clignoter une LED RGB connectée en sortie d'une couleur définie par un code couleur donné en entrée. Le changement de couleur de la LED RGB n'a lieu que si un signal *update* est reçu.

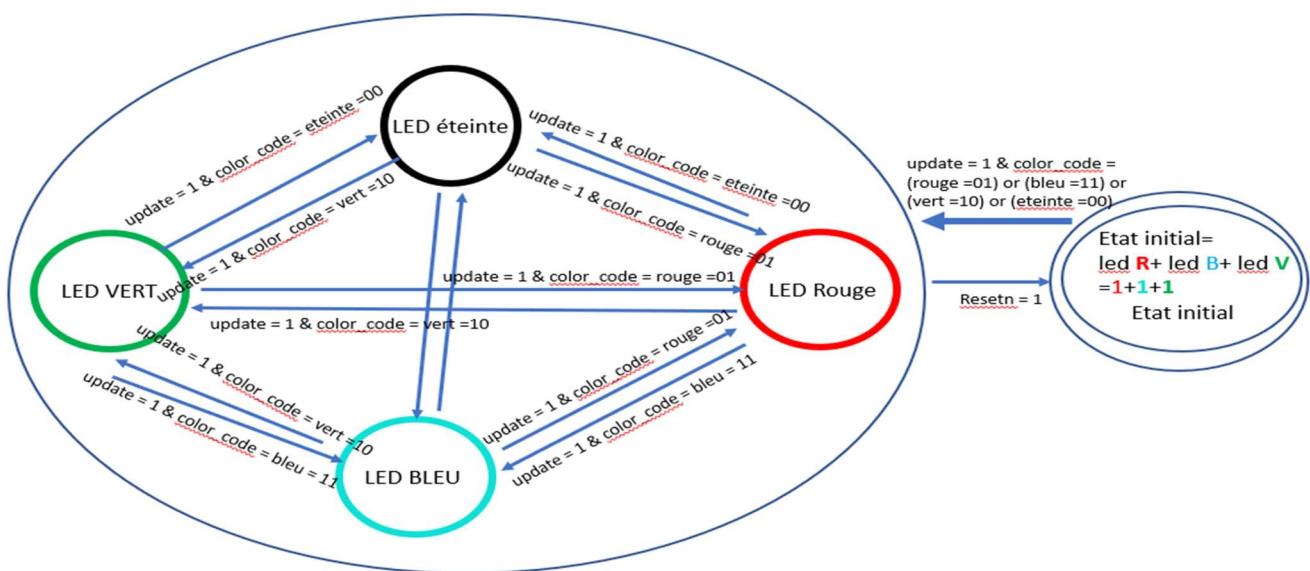
J'ai commencé par comprendre la machine à état et mettre à plat les différentes états, ensuite remettre à jour mon architecture.

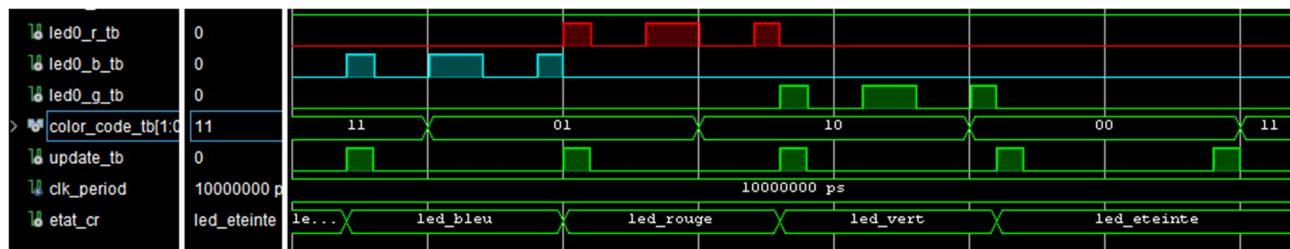
Je remplace mon FSM par mon nouveau module 'LED\_DRIVER' et je re-verifie mes différents signaux de ma machine à état. Et je ne touche à rien dans mon module 'allumer led' ni celui de 'counter unit'.

### **Mon architecture du module de pilotage d'une LED RGB en RTL : LED DRIVER :**



### **Ma machine à état de mon module : LED\_DRIVER**



**Ma simulation TB : LED\_DRIVER**

Question 9 et 10 et 11 et 12 et 13

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

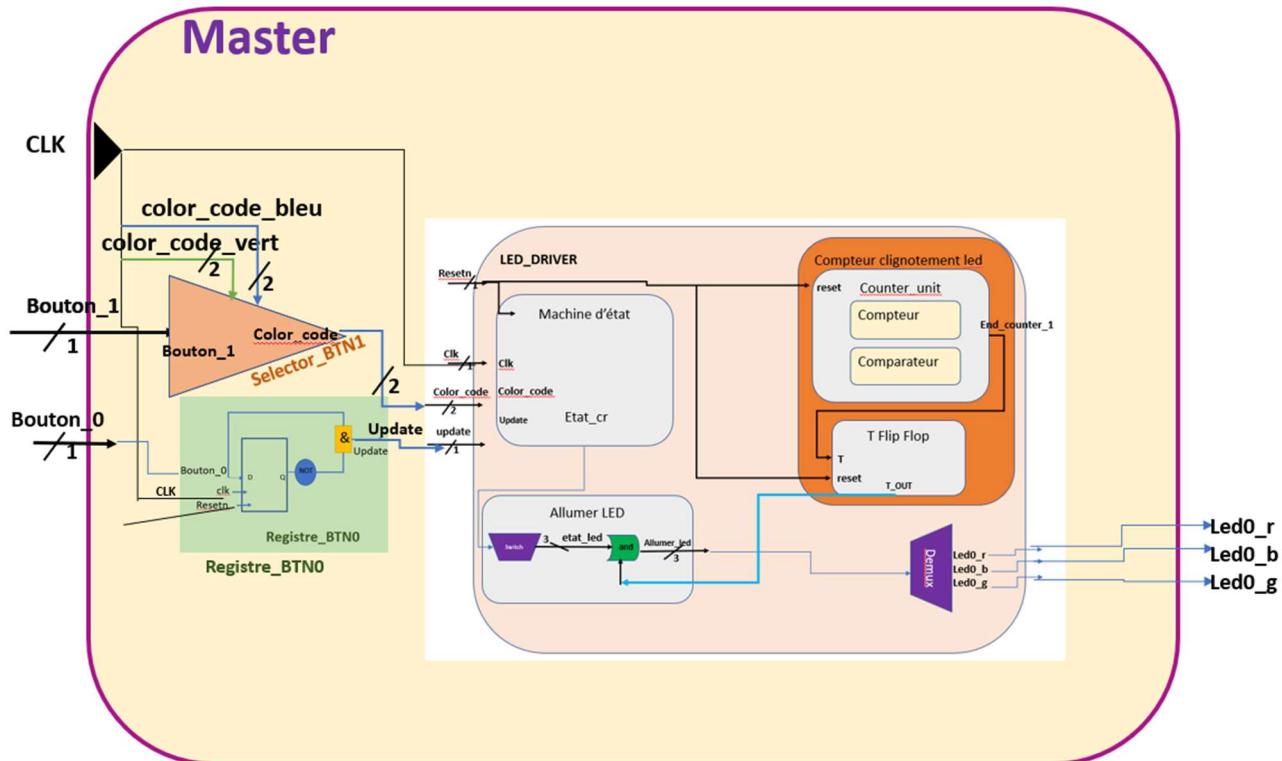
10. Ecrivez le code VHDL correspondant à votre architecture.

11. Ecrivez un testbench pour tester votre design.

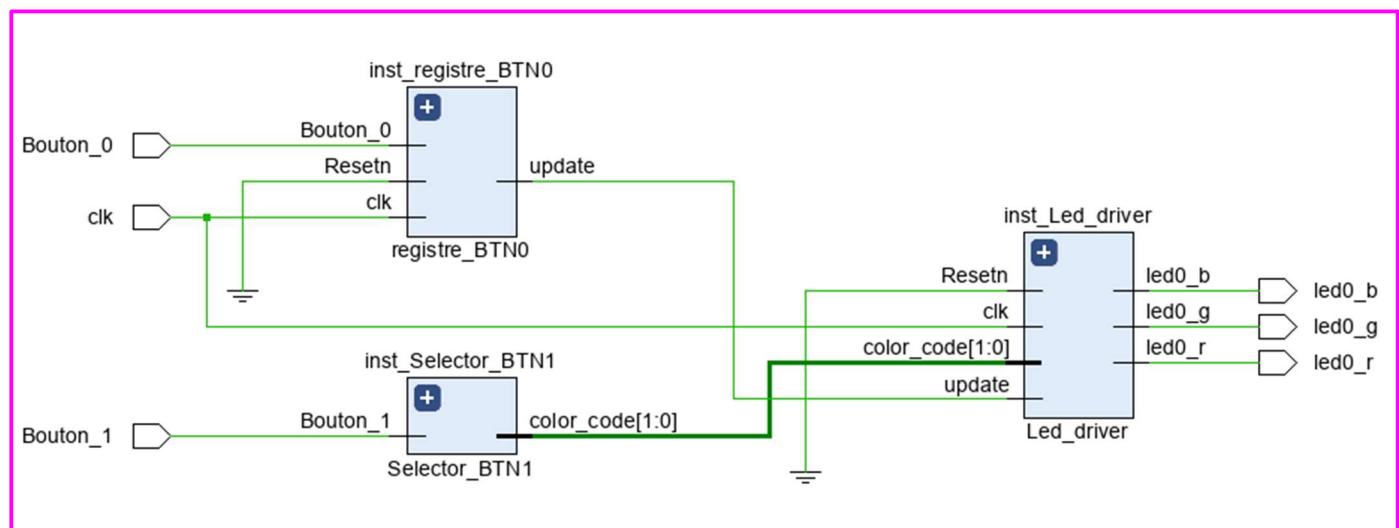
12. Vérifier votre résultat à la simulation.

Je commence par comprendre la nouvelle architecture (que j'ai nommée MASTER) qui englobe mon LED\_DRIVER et identifier ses différents signaux d'entrées et de sortie.

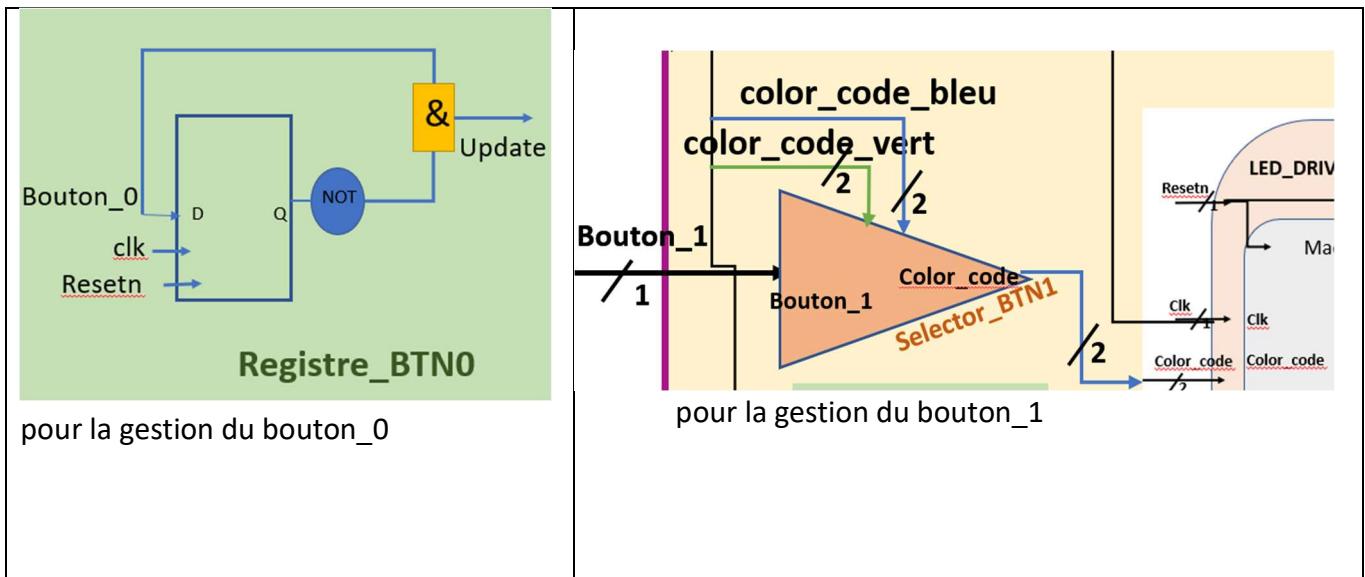
### Mon architecture : MASTER



Mon schéma RTL ANALYSIS vivado :



Explication pour la gestion des boutons (bouton\_0 et bouton\_1) :



- si front montant de clk alors :

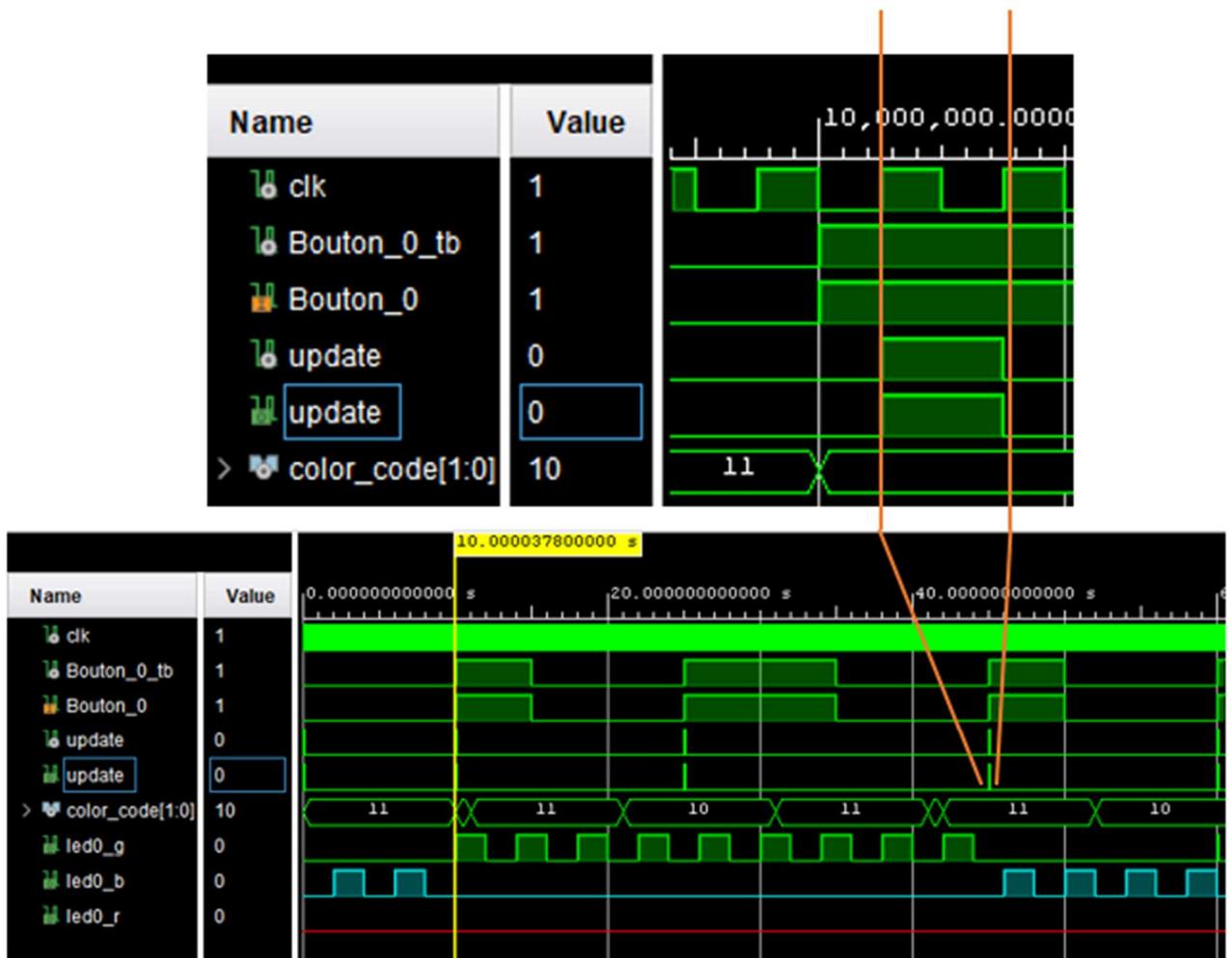
-- - si front montant (bouton\_0) = 1 et si Q =0 en mémoire de la dernière etat de mon entrée (ici D= bouton\_0 et Q= btn0\_prev =0) alors update vaut 1

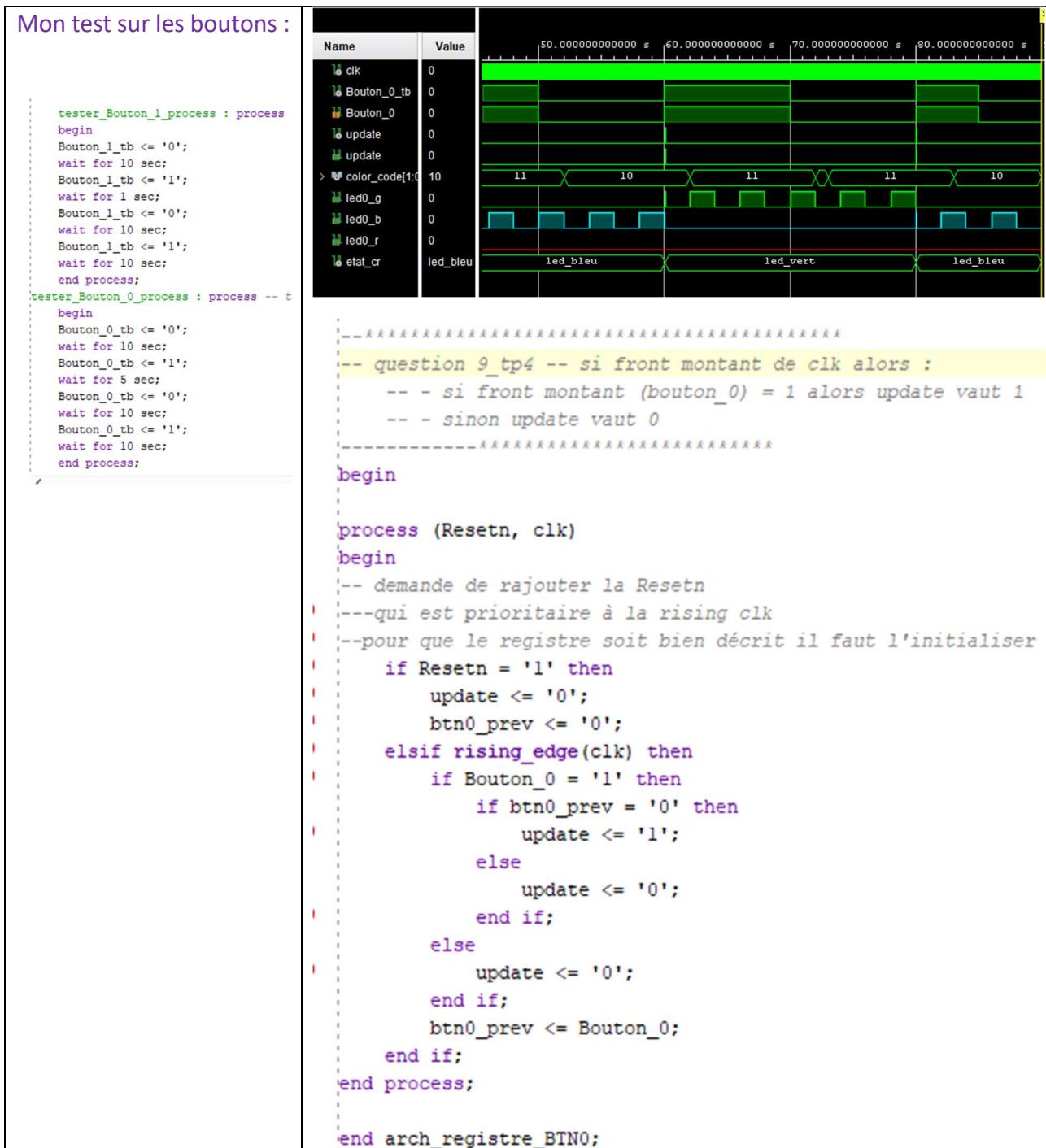
-- - sinon update vaut 0

\* Explication :



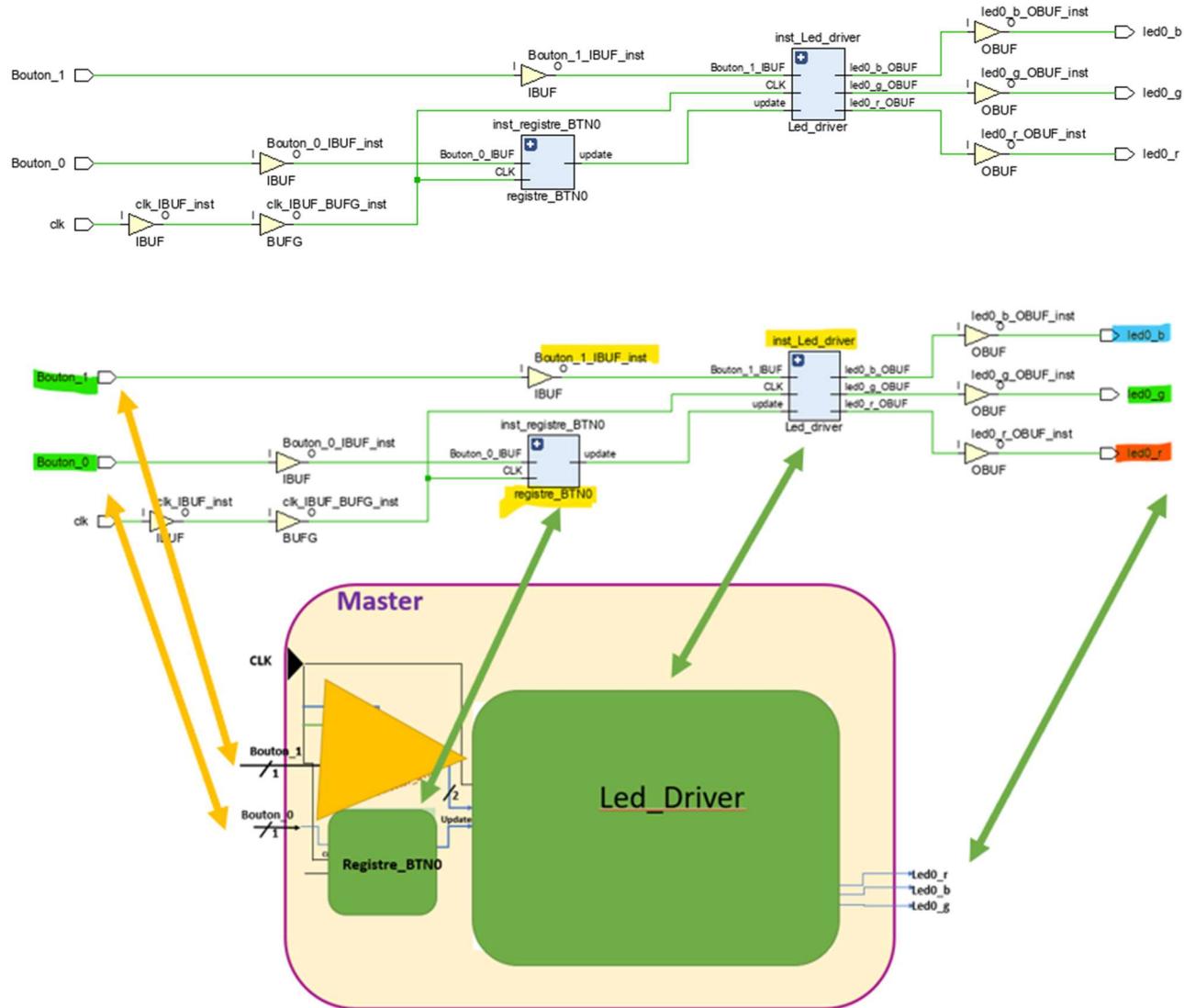
Ma simulation TB : MASTER de la question 9 complète :





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

## 1.2 Schématique : après implémentation



Question14 : Effectuez le placement routage et étudiez les rapports..

## 1.3 Rapport de synthèse

Start RTL Component Statistics		Report Cell Usage:	
-----		-----	
Detailed RTL Component Info :		-----	
+---Registers :			Cell Count
3 Bit	Registers := 2	1	BUFG 1
1 Bit	Registers := 6	2	CARRY4 7
+---Muxes :		3	LUT1 1
6 Input	3 Bit Muxes := 1	4	LUT2 4
5 Input	3 Bit Muxes := 1	5	LUT3 2
2 Input	2 Bit Muxes := 1	6	LUT4 1
-----		7	LUT5 1
Finished RTL Component Statistics		8	LUT6 5
-----		9	FDRE 39
		10	IBUF 3
		11	OBUF 3
-----		+-----+	+-----+

## 1.4 Rapport de timing

Design Timing Summary					
WNS(ns)	TNS(ns)	TNS Failing Endpoints	TNS Total Endpoints	WHS(ns)	THS(ns)
5.934	0.000	0	66	0.182	0.000
Setup :	0 Failing Endpoints, Worst Slack	5.934ns, Total Violation	0.000ns		
Hold :	0 Failing Endpoints, Worst Slack	0.182ns, Total Violation	0.000ns		
PW :	0 Failing Endpoints, Worst Slack	4.500ns, Total Violation	0.000ns		

### Et le chemin critique :

```
Max Delay Paths
Slack (MET) : 5.934ns (required time - arrival time)
Source: inst_Led_driver/inst_Counter_unit/inst_compteur/comptage_1_reg[23]/C
          (rising edge-triggered cell FDRE clocked by sys_clk_pin (rise@0.000ns fall@5.000ns period=10.000ns))
Destination: inst_Led_driver/inst_Counter_unit/inst_compteur/comptage_1_reg[0]/R
          (rising edge-triggered cell FDRE clocked by sys_clk_pin (rise@0.000ns fall@5.000ns period=10.000ns))
Path Group: sys_clk_pin
Path Type: Setup (Max at Slow Process Corner)
Requirement: 10.000ns (sys_clk_pin rise@10.000ns - sys_clk_pin rise@0.000ns)
Data Path Delay: 3.584ns (logic 0.704ns (19.645%) route 2.880ns (80.355%))
Logic Levels: 2 (LUT5=1 LUT6=1)
Clock Path Skew: -0.018ns (DCD - SCD + CPR)
Destination Clock Delay (DCD): 4.916ns = ( 14.916 - 10.000 )
Source Clock Delay (SCD): 5.362ns
Clock Pessimism Removal (CPR): 0.429ns
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
```

## Question15 : Générez le bitstream et vérifiez que vous avez le comportement attendu sur carte.

The screenshot shows the Vivado Project Summary interface for a project named 'TP4\_Question9'. The project is set up for a Cora-Z7-10 device with the file 'Cora-Z7-10-Master.xdc'. The settings include a Zynq-7000 part and an xc7z010clg400-1 device, using VHDL as the target language and Mixed as the simulator language. The Synthesis and Implementation sections both show 'Complete' status with no errors or warnings. The Implementation section also indicates 'No incremental implementation'.

Synthesis		Implementation		Summary   Route Status
Status:	✓ Complete	Status:	✓ Complete	
Messages:	⚠ 2 warnings	Messages:	No errors or warnings	
Part:	xc7z010clg400-1	Part:	xc7z010clg400-1	
Strategy:	Vivado Synthesis Defaults	Strategy:	Vivado Implementation Defaults	
Report Strategy:	Vivado Synthesis Default Reports	Report Strategy:	Vivado Implementation Default Reports	
Incremental synthesis:	None	Incremental implementation:	None	

## Annexe

