

TP5 – Domaines d’horloge

Rendu

Votre rapport devra contenir :

- Vos schéma RTL
- Vos résultats de simulation avec vos chronogrammes commentés
- Vos résultats de synthèse (analyse de vos ressources)
- Vos résultats de STA
- Vos résultats de mesure ILA
- Une démonstration de votre design

Vous fournirez également vos codes sources commentés.

Objectif

L’objectif de ce TP est de mettre en place une architecture utilisant plusieurs domaines d’horloge. Pour cela, vous utiliserez deux LEDs RGB qui clignoteront avec des fréquences différentes grâce aux horloges. Vous apprendrez également à utiliser une PLL pour générer des horloges.

Questions

1. A l'aide du module *LED_driver* du TP4, créez une architecture RTL permettant de piloter les deux LED RGB. Les LEDs RGB devront clignoter 10 fois en rouge puis 10 fois en bleu et 10 fois en vert avant de recommencer à partir du rouge. En entrée des modules *LED_driver* le signal *update* ne devra pas être à 1 pendant plus d'un coup d'horloge. Il doit s'agir d'une impulsion.

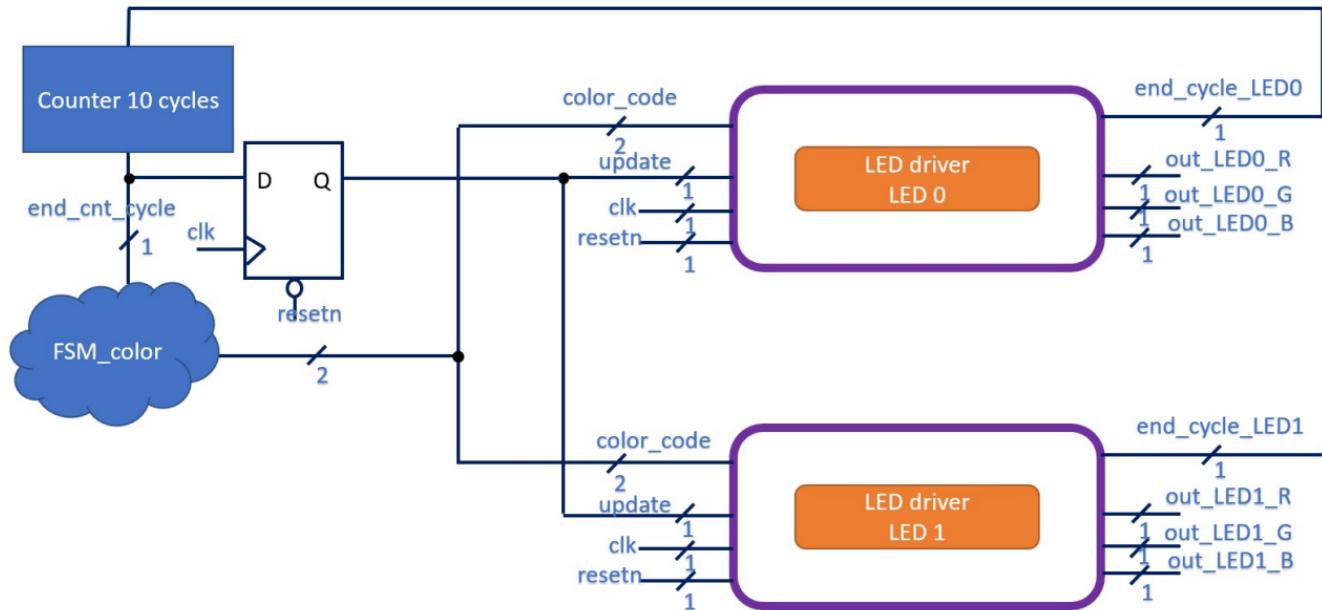
Le compteur de temporisation devra compter 100 000 000 coups d'horloge (pour une fréquence d'horloge à 100MHz, cela correspond à 1s). **Dans la suite de ce TP, la valeur du compteur de temporisation ne sera pas modifiée, même lorsque les fréquences d'horloges changeront.**

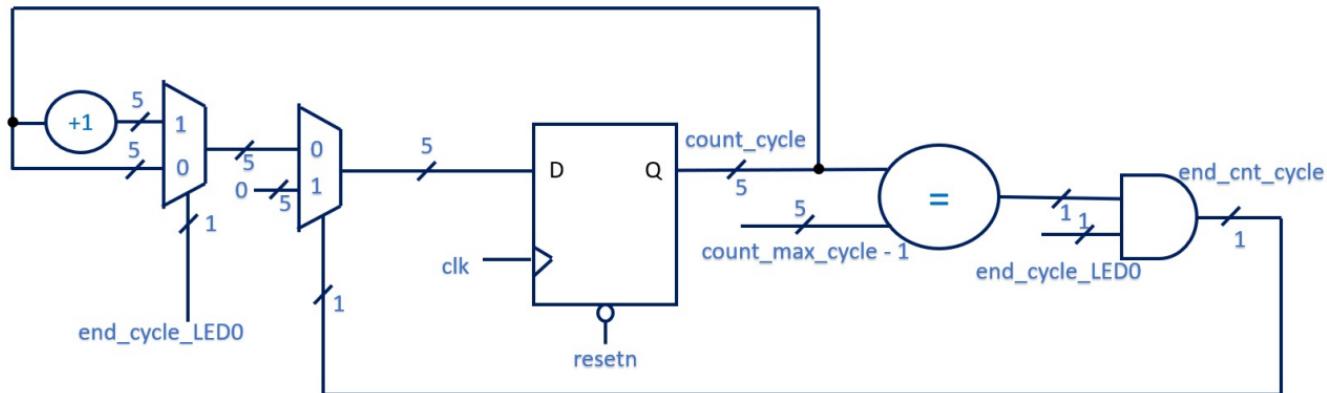


Le signal `end_cycle_LED0` est envoyé à chaque fin de cycle allumé/éteint.

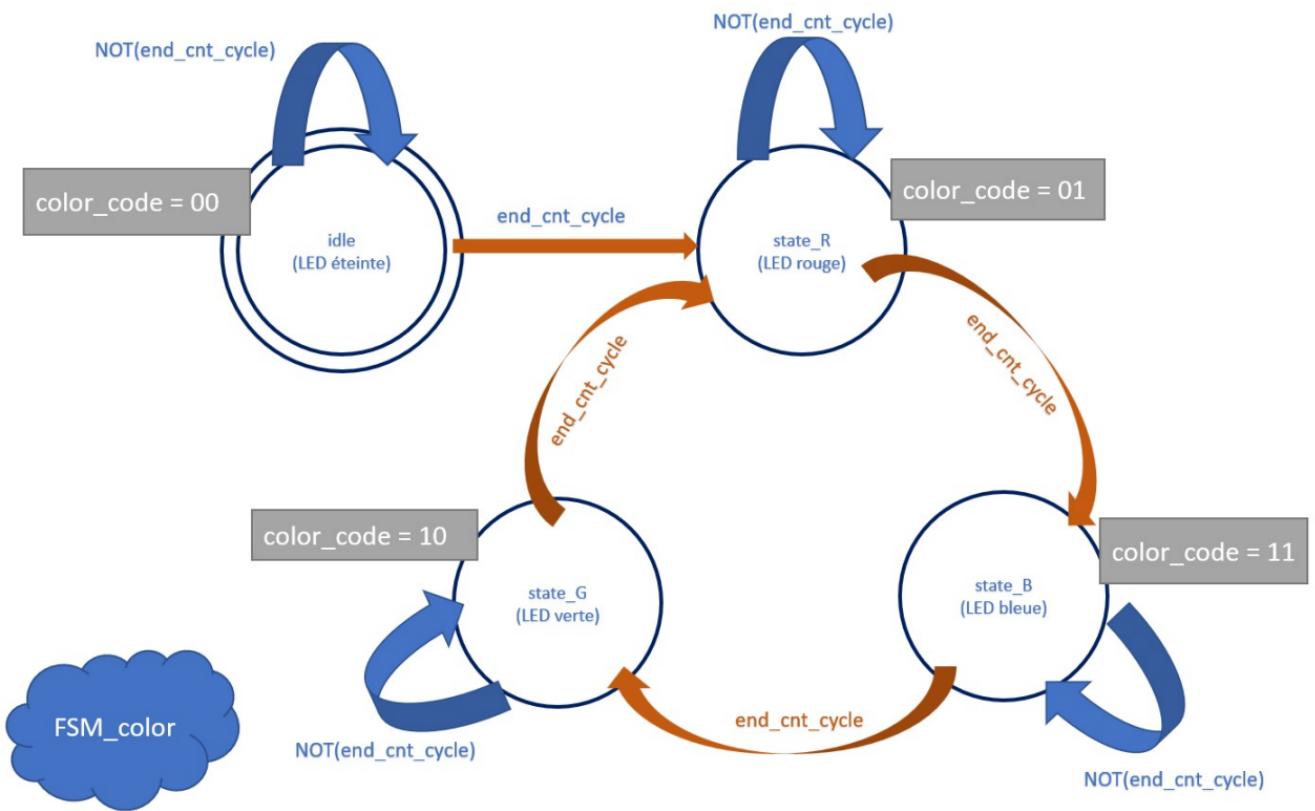
On ajoute un module "Counter 10 cycles" qui envoie un signal (front montant) tous les 10 cycles "allumé/éteint".

NB: Nous devons insérer un registre entre les signaux `end_cnt_cycle` et `update` afin de synchroniser les signaux `color_code` et `update`.

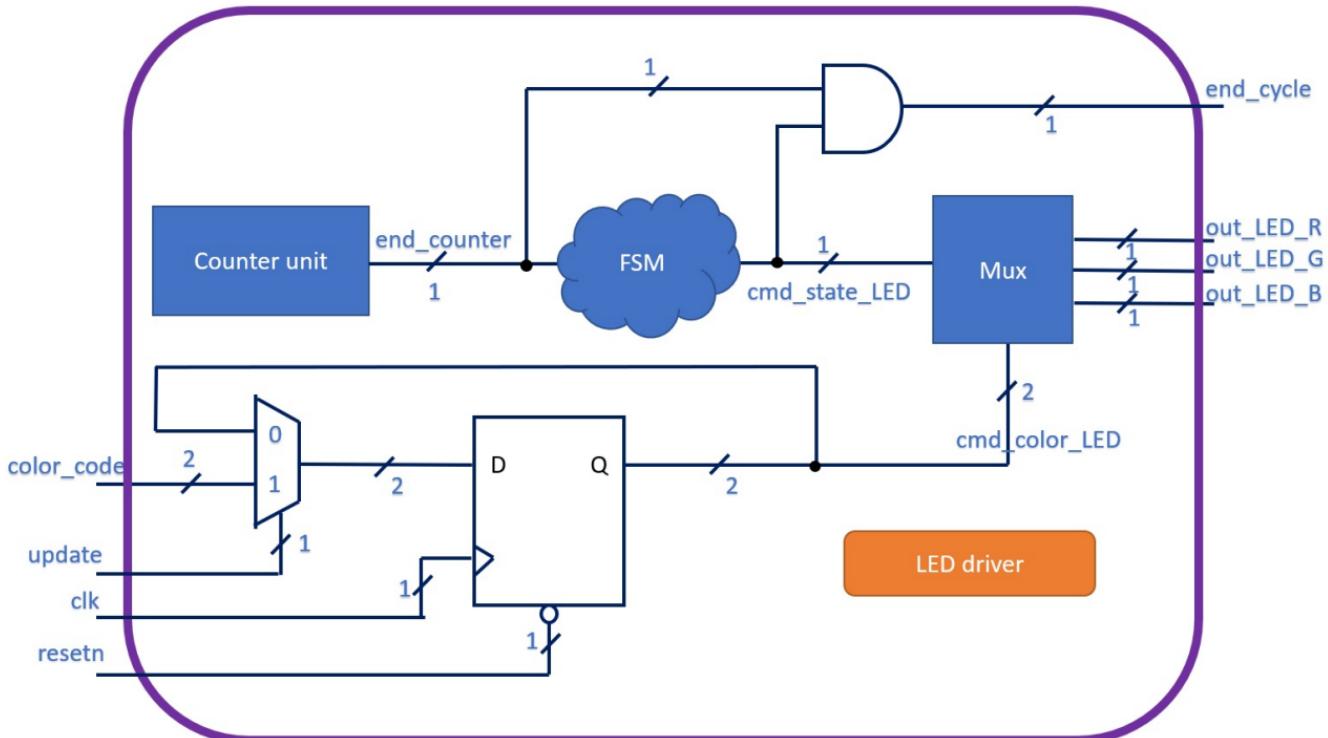




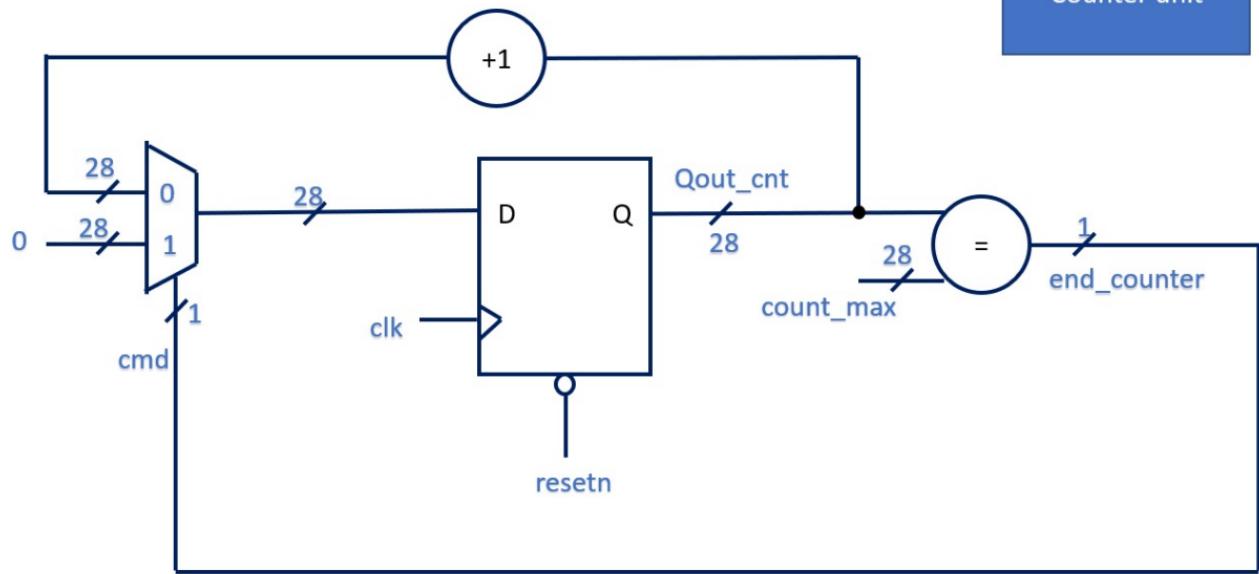
Counter 10 cycles



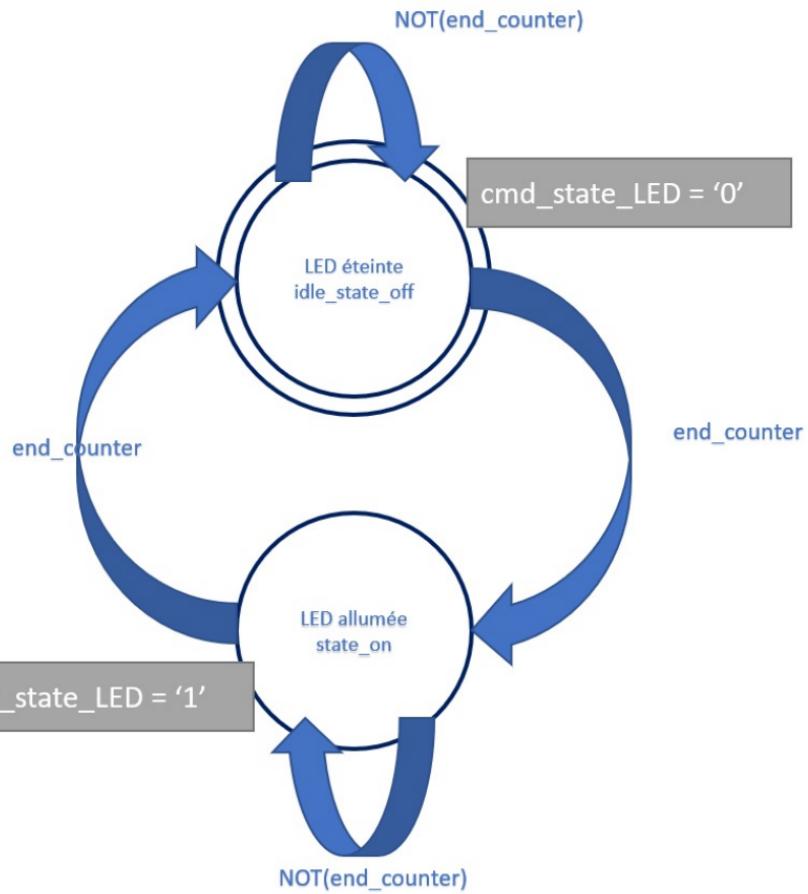
Module LED_driver (version 3)

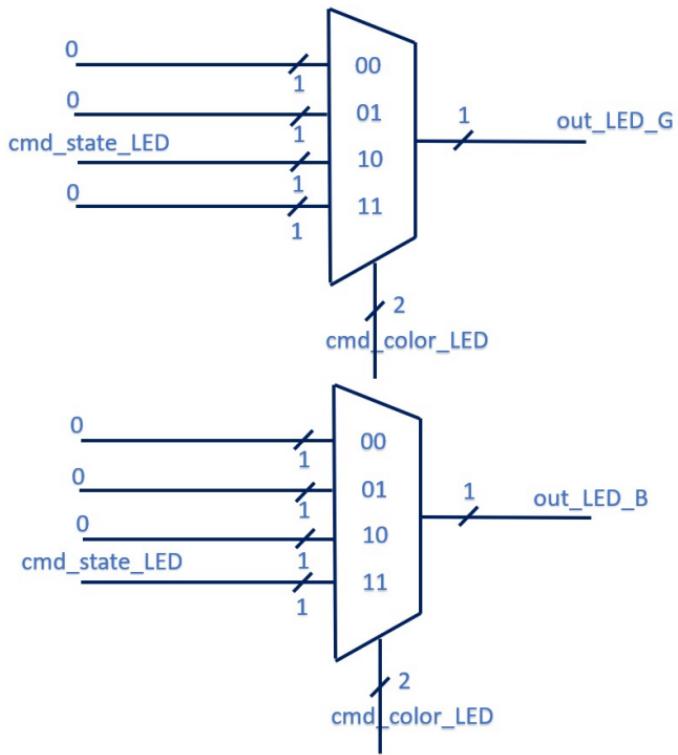
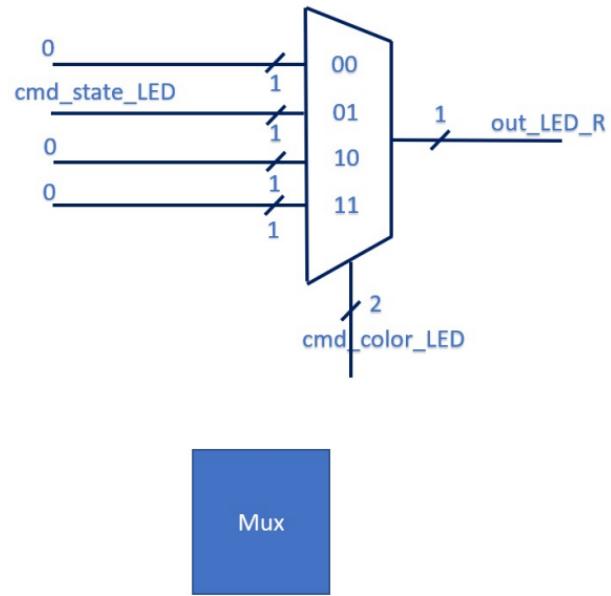


Counter unit



FSM





2. Rédigez le code VHDL correspondant à votre architecture.

Le code est disponible dans le fichier "tp_domaineHorloge_Q1-3.vhd".

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4
5
6  entity top is
7      port (
8          clk           : in std_logic;          -- horloge
9          resetn        : in std_logic;          -- bouton de reset
10         out_LED0_R   : out std_logic;         -- etat de la LED0 rouge
11         out_LED0_G   : out std_logic;         -- etat de la LED0 verte
12         out_LED0_B   : out std_logic;         -- etat de la LED0 bleue
13         out_LED1_R   : out std_logic;         -- etat de la LED1 rouge
14         out_LED1_G   : out std_logic;         -- etat de la LED1 verte
15         out_LED1_B   : out std_logic;         -- etat de la LED1 bleue
16     );
17 end top;
18
19
20 architecture behavioral of top is
21
22
23     -----
24     -- SIGNAUX INTERNES MODULES "LED_DRIVER"
25
26     -- Signaux internes pour les fins de cycle "allume/eteint"
27     signal end_cycle_LED0 : std_logic := '0';      -- Sortie end_cycle du LED_driver 0
28     signal end_cycle_LED1 : std_logic := '0';      -- Sortie end_cycle du LED_driver 1 (sortie non utilisée)
29
30     -- Signaux internes pour la gestion de la couleur
31     signal color_code :      std_logic_vector (1 downto 0) := "00";    -- Couleur de la LED
32     -- "00" si eteinte
33     -- "01" si rouge
34     -- "10" si vert
35     -- "11" si bleu
36
37     signal update :      std_logic := '0';          -- signal autorisant la mise a jour de la couleur
```

```
39
40      -- SIGNAUX INTERNES MODULE "Counter 10 cycles"
41
42      constant count_max_cycle : natural := 10;           -- Nombre de cycles "allume/eteint" a compter
43      signal count_cycle : std_logic_vector (4 downto 0) := (others => '0');           -- Compteur de cycle
44      signal end_cnt_cycle : std_logic := '0';           -- Signal indiquant la fin de 10 cycles
45
46
47      -- SIGNAUX INTERNES DE LA MACHINE A ETATS "FSM_color"
48
49      type state_LED is (idle, state_R, state_B, state_G);    -- Définition des états du FSM
50
51
52      signal current_state : state_LED;   --etat dans lequel on se trouve actuellement
53      signal next_state : state_LED;     --etat dans lequel on passera au prochain coup d'horloge
54
55
56
57      -- DECLARATION DE L'ENTITE LED_driver_unit
58
59
60      component LED_driver_unit
61          port (
62              clk           : in std_logic;
63              resetn        : in std_logic := '0';
64              update         : in std_logic := '0';           -- signal autorisant la mise a jour de la couleur
65              color_code    : in std_logic_vector (1 downto 0) := "00";       -- couleur de LED
66              out_LED_R     : out std_logic;    -- etat de la LED rouge
67              out_LED_G     : out std_logic;    -- etat de la LED verte
68              out_LED_B     : out std_logic;    -- etat de la LED bleue
69              end_cycle     : out std_logic      -- fin de cycle
70          );
71      end component;
```

```
73 begin
74
75
76 -----
77 -- AFFECTATION DES SIGNALS
78 -----
79 --Affectation des signaux du module LED_driver de la LED0
80 mapping_LED0_driver: LED_driver_unit
81     port map (
82         clk => clk,
83         resetn => resetn,
84         update => update,
85         color_code => color_code,
86         out_LED_R => out_LED0_R,
87         out_LED_G => out_LED0_G,
88         out_LED_B => out_LED0_B,
89         end_cycle => end_cycle_LED0
90 );
91
92 --Affectation des signaux du module LED_driver de la LED1
93 mapping_LED1_driver: LED_driver_unit
94     port map (
95         clk => clk,
96         resetn => resetn,
97         update => update,
98         color_code => color_code,
99         out_LED_R => out_LED1_R,
100        out_LED_G => out_LED1_G,
101        out_LED_B => out_LED1_B,
102        end_cycle => end_cycle_LED1
103 );
104
```

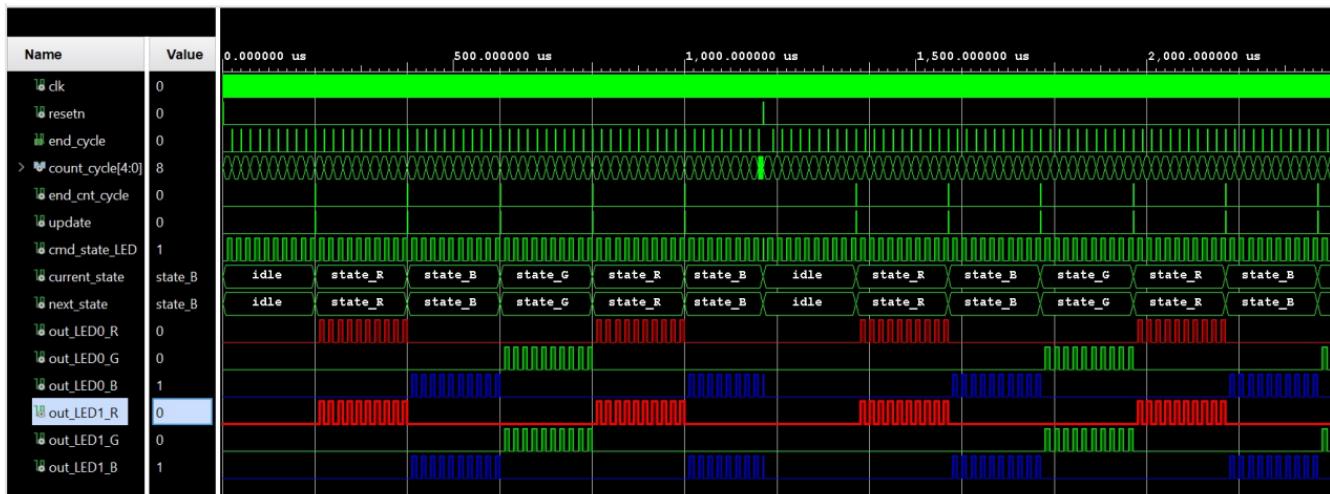
```
107 -----
108 -- PARTIE SEQUENTIELLE
109 -----
110 process(clk, resetn)
111 begin
112   if resetn = '1' then
113     current_state <= idle;           -- Retour de la FSM à l'état initial
114
115   count_cycle <= (others => '0'); -- Réinitialisation du compteur
116
117   elsif rising_edge(clk) then
118     current_state <= next_state;    -- Passage à l'état suivant
119
120     -- Gestion du compteur de cycle
121     if end_cnt_cycle = '1' then
122       count_cycle <= (others => '0'); -- Réinitialisation du compteur
123     elsif end_cycle_LED0 = '1' then
124       count_cycle <= count_cycle + 1;
125     end if;
126
127     -- Ajout d'un registre pour synchroniser le chgt de couleur et le signal update
128     update <= end_cnt_cycle;
129
130   end if;
131 end process;
132
133
134
135 -----
136 -- PARTIE COMBINATOIRE
137 -----
138 -- Signal de fin des 10 cycles (on compte de 0 à 9)
139 end_cnt_cycle <= '1' when (count_cycle = (count_max_cycle - 1) AND end_cycle_LED0 = '1')
140   else '0';
141
```

```
143
144
145
146    -- Machine a etats "FSM_color" pour la gestion des couleurs
147    -- Si on a compté 10 cycles, on change de d'état (ie de couleur) sinon on reste dans le même état.
148    process(current_state, end_cnt_cycle)
149    begin
150        --signaux pilotes par la fsm
151        case current_state is
152            when idle =>
153                color_code <= "00";
154                if end_cnt_cycle = '1' then
155                    next_state <= state_R; --prochain état
156                else
157                    next_state <= current_state;
158                end if;
159
160
161            when state_R =>
162                color_code <= "01";
163                if end_cnt_cycle = '1' then
164                    next_state <= state_B; --prochain état
165                else
166                    next_state <= current_state;
167                end if;
168
169
170            when state_B =>
171                color_code <= "11";
172                if end_cnt_cycle = '1' then
173                    next_state <= state_G; --prochain état
174                else
175                    next_state <= current_state;
176                end if;
177
178
179            when state_G =>
180                color_code <= "10";
181                if end_cnt_cycle = '1' then
182                    next_state <= state_R; --prochain état
183                else
184                    next_state <= current_state;
185                end if;
186
187
188        end case;
189
190    end process;
191
192 end behavioral;
```

3. Rédigez le testbench et simulez votre design. Vérifiez que les modules réceptionnent correctement le signal *update*.

Le code est disponible dans le fichier "tb_domaineHorloge_Q1-3.vhd".

Vue d'ensemble des tests



-- PREMIER TEST - ETAT IDLE

--

-- Test des 10 cycles "allume/eteint"

-- Attente: LEDs eteintes tout le temps



-- SECOND TEST - ETAT ROUGE

--

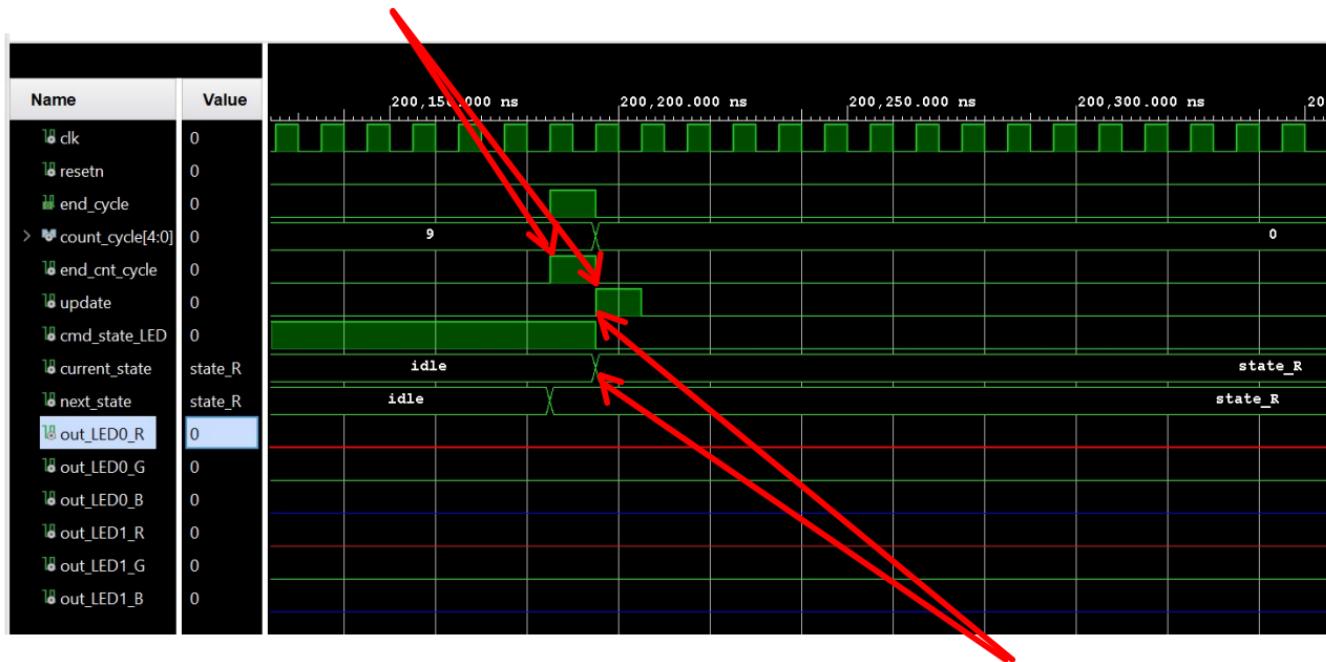
-- Test des 10 cycles "allume/eteint"

-- Attente: Clignotement des LEDs en rouge



Si on zoom pour s'intéresser au changement d'état, nous observons le bon comportement.

Le signal update a bien été décalé d'une période en raison du registre par rapport au signal end_cnt_cycle.



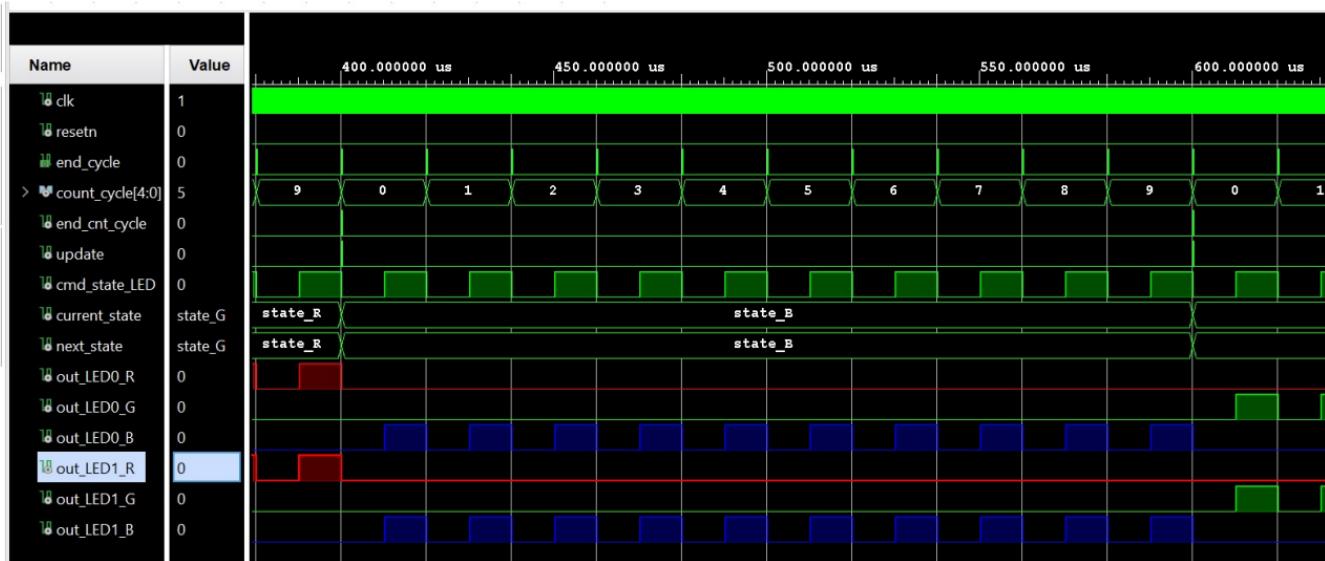
Le signal update est bien en phase avec le signal current_state.

-- TROISIEME TEST - ETAT BLEU

--

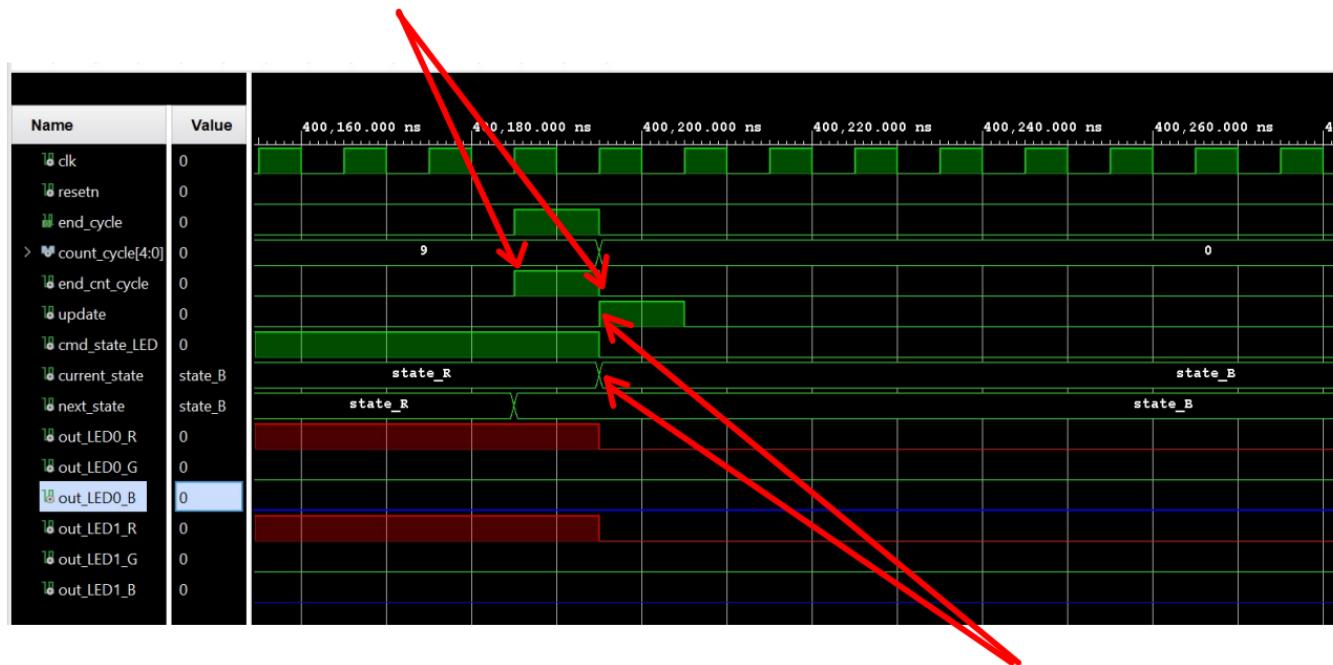
-- Test des 10 cycles "allume/eteint"

-- Attente: Clignotement des LEDs en bleu



Si on zoom pour s'intéresser au changement d'état, nous observons le bon comportement.

Le signal update a bien été décalé d'une période en raison du registre par rapport au signal end_cnt_cycle.



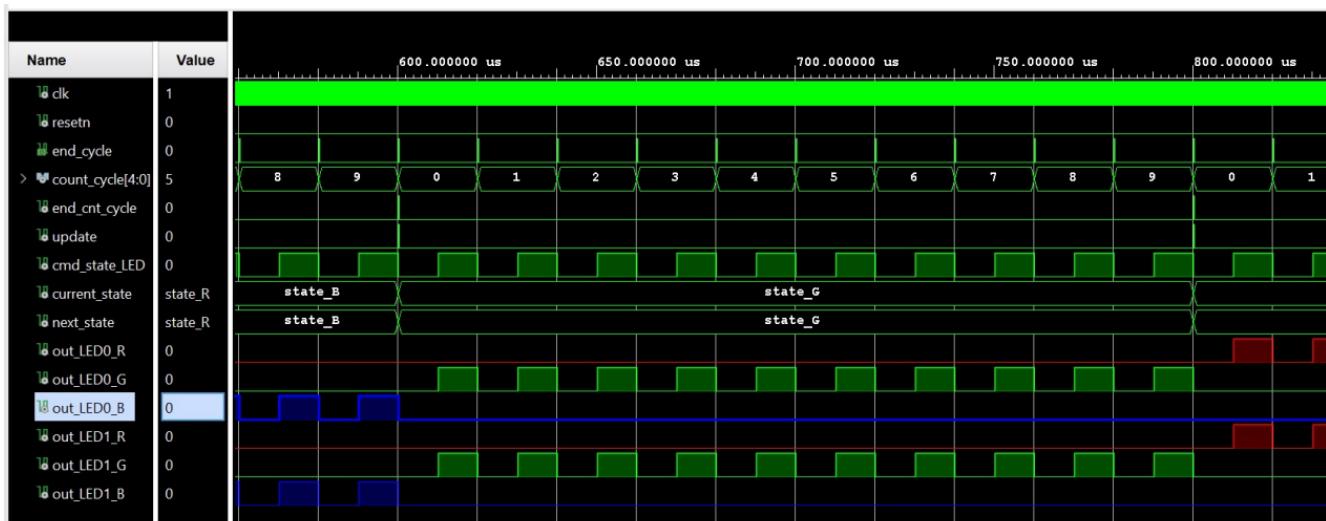
Le signal update est bien en phase avec le signal current_state.

-- QUATRIEME TEST - ETAT VERT

--

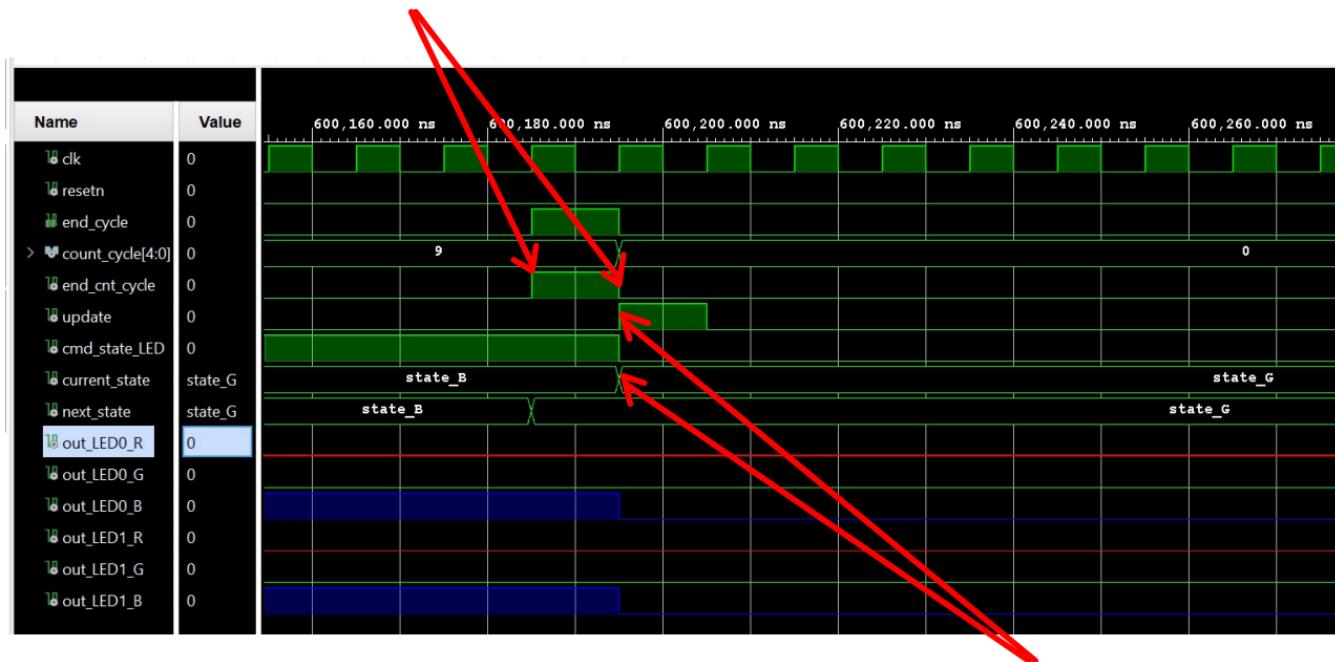
-- Test des 10 cycles "allume/eteint"

-- Attente: Clignotement des LEDs en vert



Si on zoom pour s'intéresser au changement d'état, nous observons le bon comportement.

Le signal update a bien été décalé d'une période en raison du registre par rapport au signal end_cnt_cycle.



Le signal update est bien en phase avec le signal current_state.

-- FIN DE TEST AVEC un RESET

--

-- Test des 10 cycles "allume/eteint"

-- Attente: LEDs eteintes tout le temps



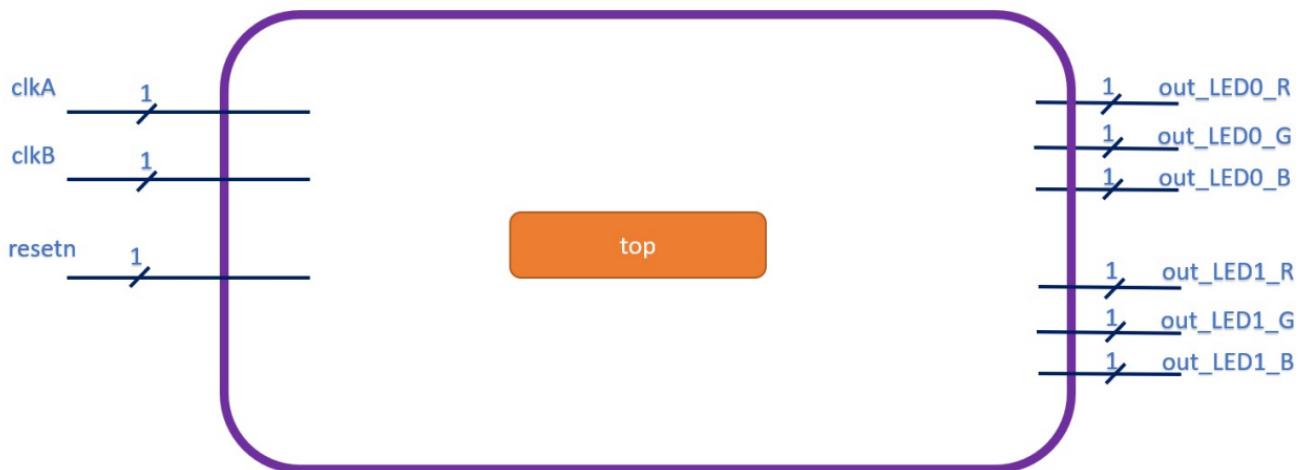
Si on zoom pour s'intéresser au comportement lors du resetn, les LEDs arrêtent de clignoter en bleu, la FSM passe bien à l'état "idle".



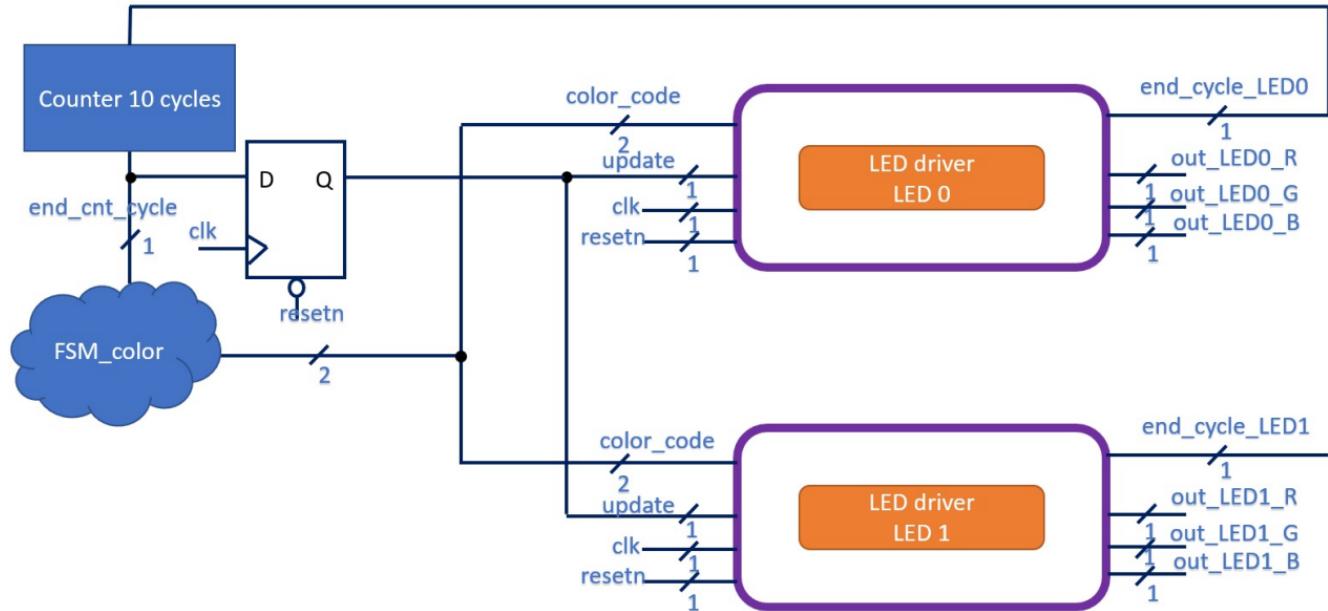
4. Modifiez votre design pour gérer deux signaux d'horloge différents. La première horloge, *clkA*, est associé à la logique en dehors des modules *LED_driver* et au module *LED_driver* de la LED0. La deuxième horloge, *clkB*, est associé au module *LED_driver* de la LED1.

Le changement de couleur des deux LEDs à lieu lorsque la LED0 à clignoté 10 fois.

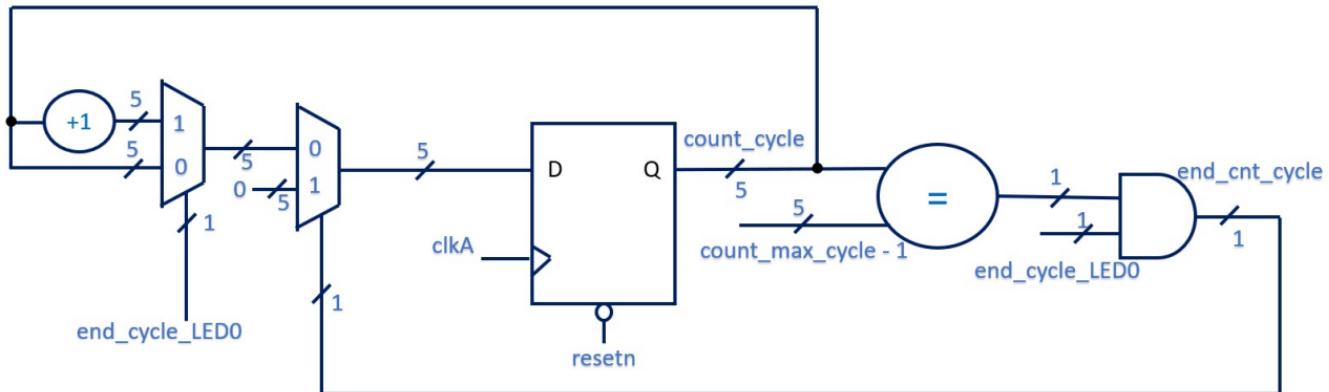
On conserve le design des questions 1 à 3 en séparant les horloges des composants *LED_driver*.



L'horloge clkA est utilisée pour le registre externe aux composants LED_driver.



L'horloge clkA est utilisée pour le registre du module "Counter 10 cycles".



Counter 10 cycles

Modifications dans le code du fichier principal

```
entity top is
  port (
    clkA      : in std_logic;      -- horloge A
    clkB      : in std_logic;      -- horloge B
    resetn    : in std_logic;      -- bouton de reset
    out_LED0_R : out std_logic;    -- etat de la LED0 rouge
    out_LED0_G : out std_logic;    -- etat de la LED0 verte
    out_LED0_B : out std_logic;    -- etat de la LED0 bleue
    out_LED1_R : out std_logic;    -- etat de la LED1 rouge
    out_LED1_G : out std_logic;    -- etat de la LED1 verte
    out_LED1_B : out std_logic;    -- etat de la LED1 bleue
  );
end top;

-----  
-- AFFECTATION DES SIGNAUX  
-----  
--Affectation des signaux du module LED_driver de la LED0
mapping_LED0_driver: LED_driver_unit
  port map (
    clk => clkA,
    resetn => resetn,
    update => update,
    color_code => color_code,
    out_LED_R => out_LED0_R,
    out_LED_G => out_LED0_G,
    out_LED_B => out_LED0_B,
    end_cycle => end_cycle_LED0
  );

--Affectation des signaux du module LED_driver de la LED1
mapping_LED1_driver: LED_driver_unit
  port map (
    clk => clkB,
    resetn => resetn,
    update => update,
    color_code => color_code,
    out_LED_R => out_LED1_R,
    out_LED_G => out_LED1_G,
    out_LED_B => out_LED1_B,
    end_cycle => end_cycle_LED1
  );
```

```
-----  
-- PARTIE SEQUENTIELLE  
-----  
  
process(clkA, resetn)  
begin  
    if resetn = '1' then  
        current_state <= idle;           -- Retour de la FSM à l'état initial  
  
        count_cycle <= (others => '0'); -- Réinitialisation du compteur  
  
    elsif rising_edge(clkA) then  
        current_state <= next_state;      -- Passage à l'état suivant  
  
        -- Gestion du compteur de cycle  
        if end_cnt_cycle = '1' then  
            count_cycle <= (others => '0'); -- Réinitialisation du compteur  
        elsif end_cycle_LED0 = '1' then  
            count_cycle <= count_cycle + 1;  
        end if;  
  
        -- Ajout d'un registre pour synchroniser le chgt de couleur et le signal update  
        update <= end_cnt_cycle;  
  
    end if;  
end process;
```

5. Modifiez votre testbench tel que l'horloge *clkA* ait une fréquence de 250MHz et *clkB* 50MHz.

Modifications du testbench

```
-- Les constantes suivantes permettent de définir la fréquence de l'horloge
constant hp_A : time := 2 ns;                      -- demi période de 2ns
constant period_A : time := 2 * hp_A;                -- période de 4ns, soit une fréquence de 250MHz
constant hp_B : time := 10 ns;                       -- demi période de 10ns
constant period_B : time := 2 * hp_B;                -- période de 20ns, soit une fréquence de 50MHz

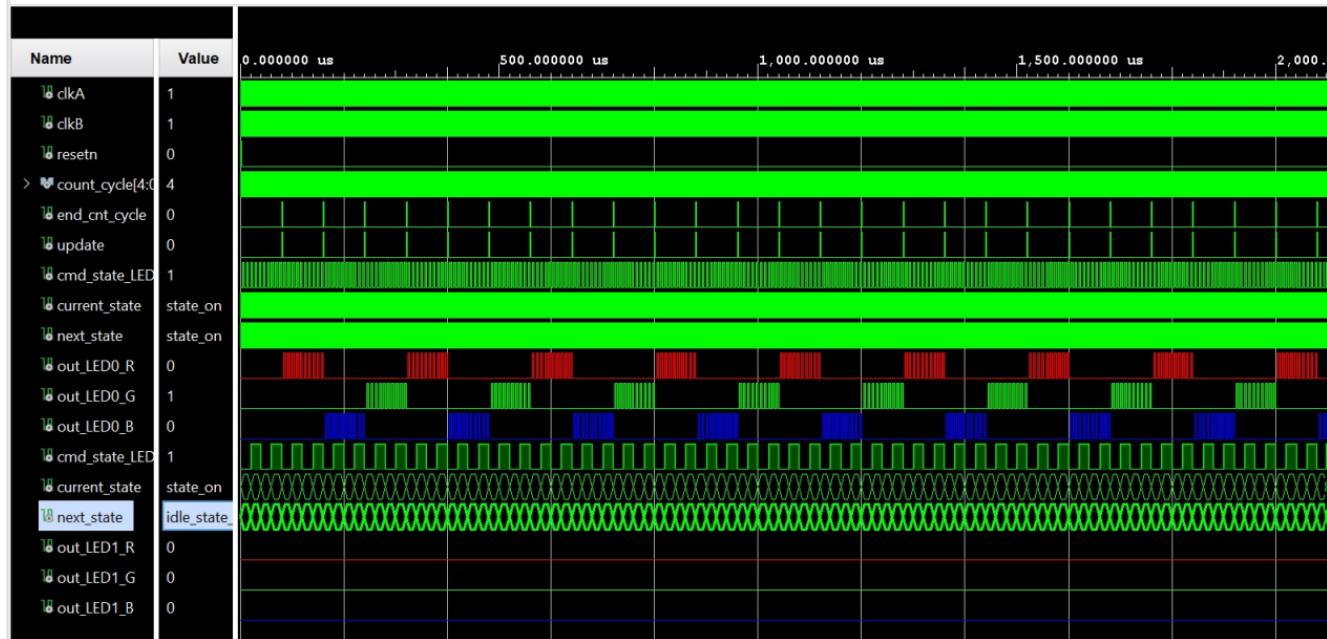
-----
-- SIMULATION DES SIGNAUX D'HORLOGE EN CONTINU
-----

process
begin
    wait for hp_A;
    clkA <= not clkA;
end process;

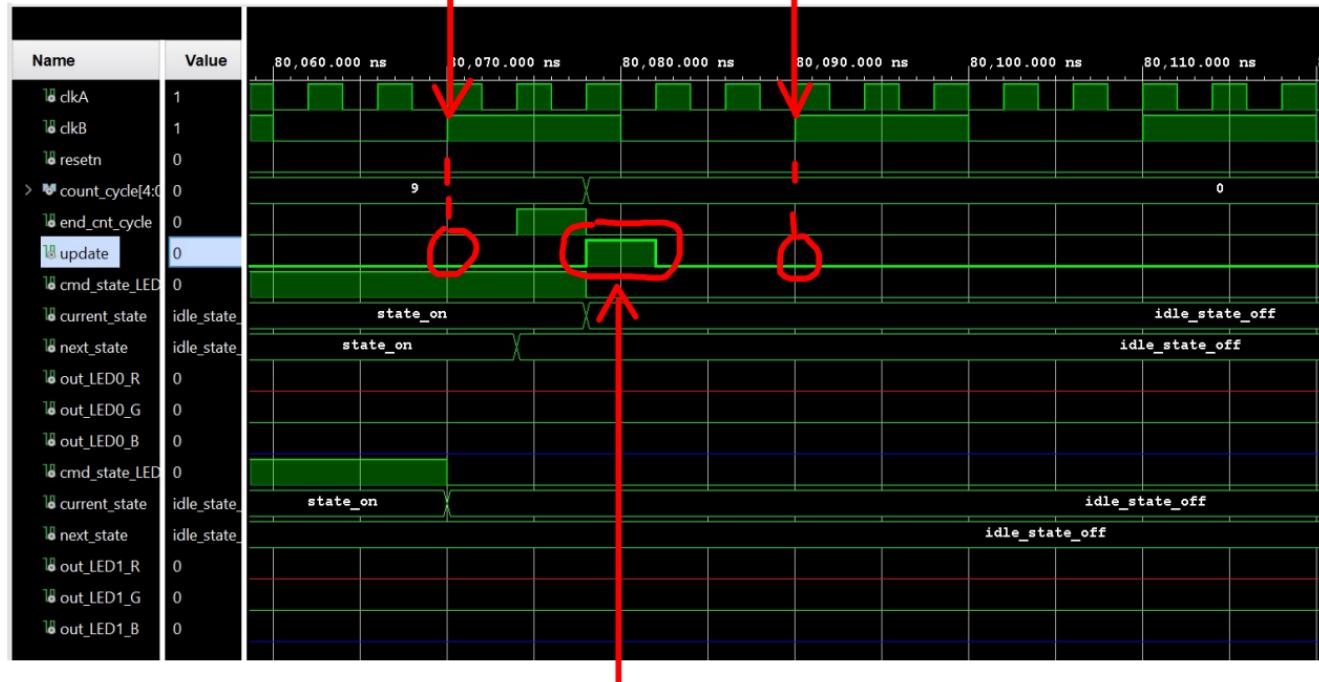
process
begin
    wait for hp_B;
    clkB <= not clkB;
end process;
```

6. Lancer une simulation. Que se passe-t-il au niveau des signaux *update* des modules *LED_driver* ?
Quelle incidence cela a-t-il sur les LEDs ?

On observe que la LED1 ne clignote pas.



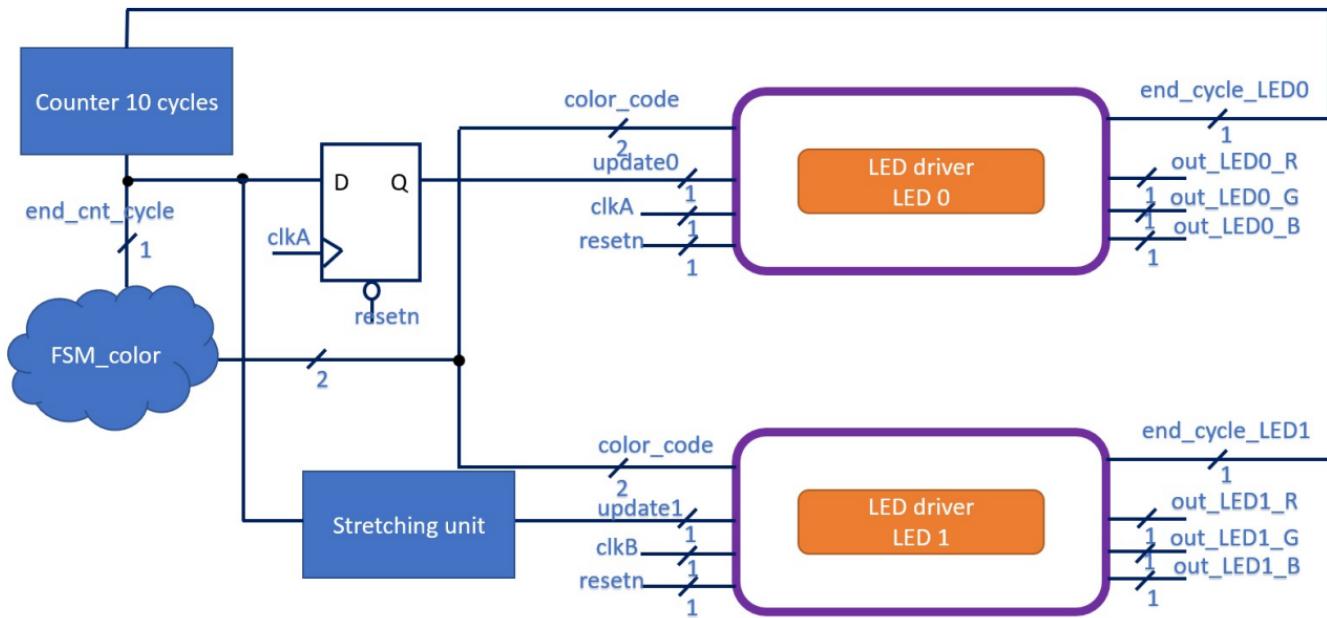
Si on regarde plus en détail, on s'aperçoit que le passage à 1 du signal update n'est pas détecté par la clock B.

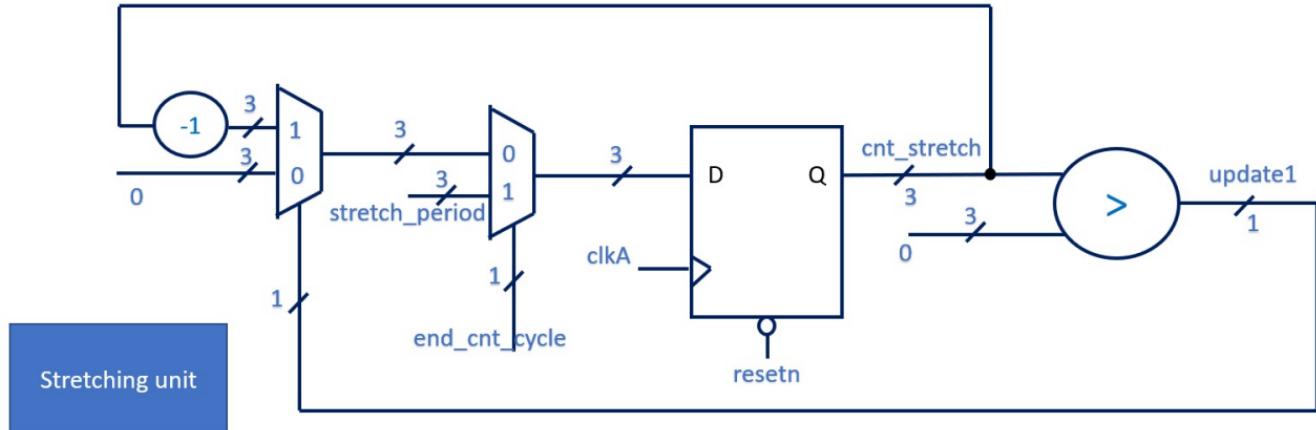


Signal update non détecté

7. Proposez une solution pour corriger le problème lié au changement de domaine d'horloge, vous pourrez vous aider du lien <https://nandland.com/lesson-14-crossing-clock-domains/>.

Afin de palier à ce problème, on va essayer d'allonger le signal end_cnt_cycle sur une période de clock B (ie 5 périodes de clock A). On parle alors de stretching.





8. Mettez en place votre solution et testez-la en simulation. Si votre résultat de simulation n'est toujours pas valide, proposez une autre solution.

Ajout des signaux internes et affectation pour chaque LED

```
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113

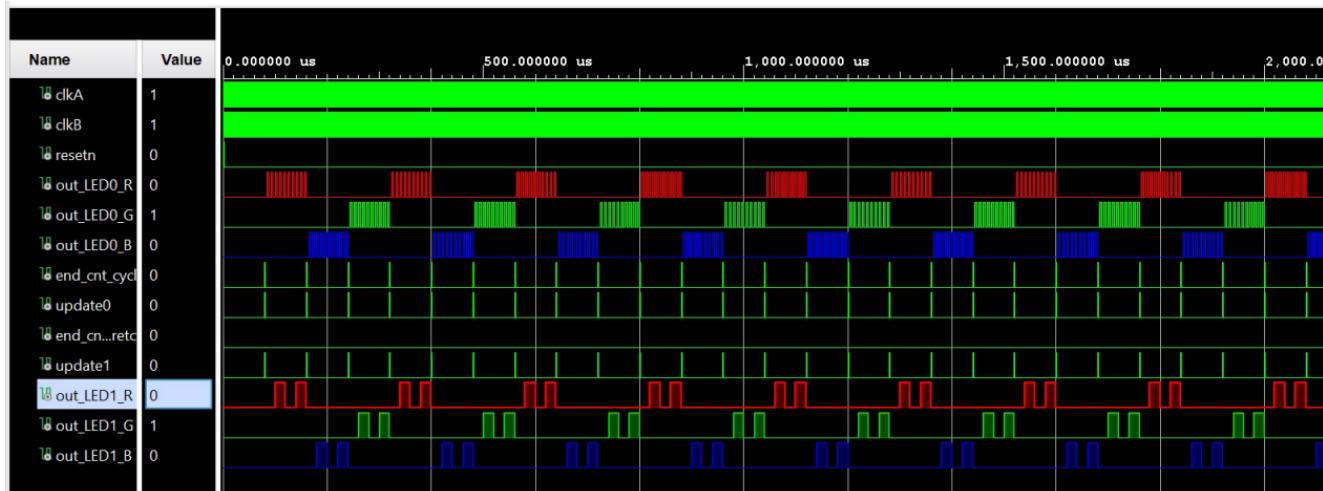
-----  
-- AFFECTATION DES SIGNAUX  
-----  
--Affectation des signaux du module LED_driver de la LED0  
mapping_LED0_driver: LED_driver_unit  
    port map (  
        clk => clkA,  
        resetn => resetn,  
        update => update0,  
        color_code => color_code,  
        out_LED_R => out_LED0_R,  
        out_LED_G => out_LED0_G,  
        out_LED_B => out_LED0_B,  
        end_cycle => end_cycle_LED0  
    );  
  
--Affectation des signaux du module LED_driver de la LED1  
mapping_LED1_driver: LED_driver_unit  
    port map (  
        clk => clkB,  
        resetn => resetn,  
        update => update1,  
        color_code => color_code,  
        out_LED_R => out_LED1_R,  
        out_LED_G => out_LED1_G,  
        out_LED_B => out_LED1_B,  
        end_cycle => end_cycle_LED1  
    );
```

```

117
118      -- PARTIE SEQUENTIELLE
119
120      process(clkA, resetn)
121      begin
122          if resetn = '1' then
123              current_state <= idle;           -- Retour de la FSM à l'état initial
124
125          count_cycle <= (others => '0'); -- Réinitialisation du compteur
126
127          elsif rising_edge(clkA) then
128              current_state <= next_state;   -- Passage à l'état suivant
129
130              -- Gestion du compteur de cycle
131          if end_cnt_cycle = '1' then
132              count_cycle <= (others => '0'); -- Réinitialisation du compteur
133          elsif end_cycle_LED0 = '1' then
134              count_cycle <= count_cycle + 1;
135          end if;
136
137          -- Ajout d'un registre pour synchroniser le chgt de couleur et le signal update0
138          -- Signal pilotant la mise à jour des couleurs de la LED0
139          update0 <= end_cnt_cycle;
140
141          -- Stretching du signal end_cnt_cycle pour le composant LED_driver de la LED1
142          if end_cnt_cycle = '1' then
143              cnt_stretch <= stretch_period;
144          elsif cnt_stretch > 0 then
145              cnt_stretch <= cnt_stretch - 1;
146          end if;
147      end if;
148
149      end process;
150
151      -- PARTIE COMBINATOIRE
152
153      -- Signal de fin des 10 cycles (on compte de 0 à 9)
154      end_cnt_cycle <= '1' when (count_cycle = (count_max_cycle - 1) AND end_cycle_LED0 = '1')
155          else '0';
156
157
158      -- Mise à jour du signal pilotant la mise à jour des couleurs de la LED1
159      update1 <= '1' when cnt_stretch > 0
160          else '0';
161

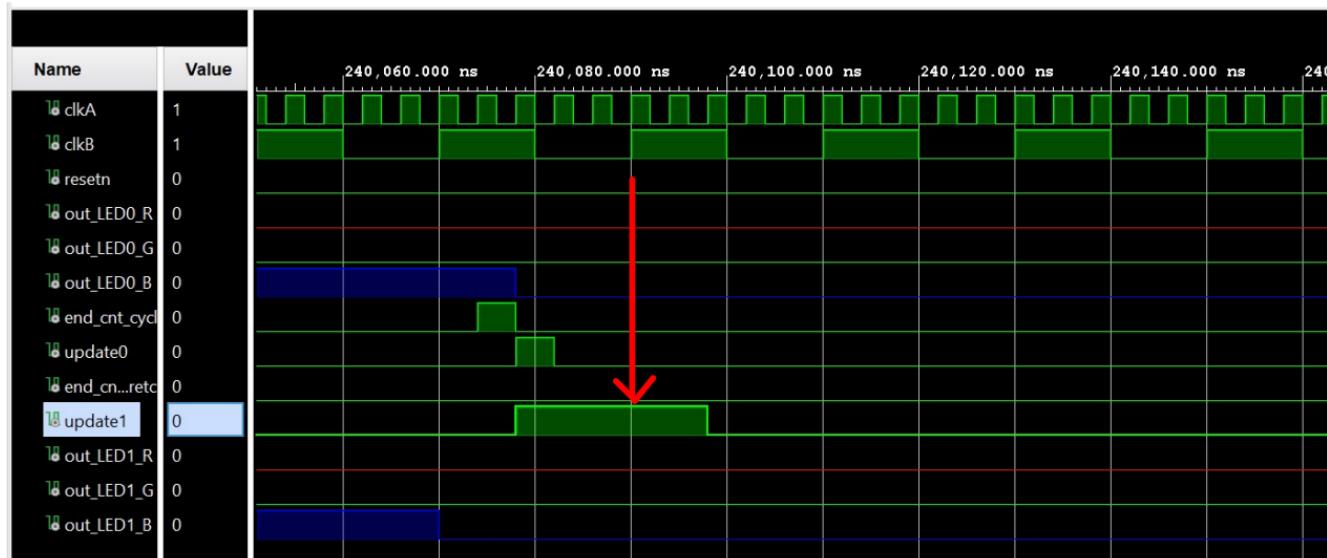
```

On observe que les LEDs clignotent chacune à leur rythme.
Le problème est bien résolu.



Si on regarde plus en détail, le signal update1 est bien un stretch du signal update0 sur 5 périodes de clockA (ie une période de clockB).

Ainsi, cela permet de détecter l'update lors d'un front montant de la clockB.



9. Pour générer plusieurs horloges vous aurez besoin d'une PLL. Trouvez quel est le nom de l'IP PLL de l'IP Catalog de Vivado puis ajoutez là à votre architecture.

La fréquence de l'horloge en entrée de la PLL est de 100MHz. Les deux horloges en sortie doivent être à 250MHz et 50MHz.

The screenshot shows the Xilinx Vivado IP Catalog interface. At the top, there are tabs for Project Summary, tp_domaineHorloge_Q1-3.vhd, LED_driver_unit_v3.vhd, tb_domaineHorloge_Q1-3.vhd, and IP Catalog. Below the tabs, there are filter icons for search, file type, and license status. A search bar is present. The main area displays a table of IP cores with columns for Name, Status, License, and VLVN. The table includes entries for Dynamic Function eXchange, Embedded Processing, and a expanded section for FPGA Features and Design. Under FPGA Features and Design, the Clocking Wizard is highlighted with a red box. The Clocking Wizard entry has a status of AXI4, Production, Included, and a VLVN of xilinx.com:ip:clk_wiz:6.0.

Name	Status	License	VLVN
> Dynamic Function eXchange			
> Embedded Processing			
▼ FPGA Features and Design			
▼ Clocking			
Clocking Wizard	AXI4	Production	Included xilinx.com:ip:clk_wiz:6.0
> IO Interfaces			
> Soft Error Mitigation			
> XADC			
▼ Processor			

Clocking Wizard (6.0)



[Documentation](#) [IP Location](#) [Switch to Defaults](#)

[IP Symbol](#) [Resource](#)

Show disabled ports

Component Name PLL

Clocking Options

[Output Clocks](#)

[Port Renaming](#)

[PLLE2 Settings](#)

[Summary](#)

Clock Monitor

Enable Clock Monitoring

Primitive

MMCM PLL

Clocking Features

Frequency Synthesis Minimize Power

Phase Alignment

Dynamic Reconfig

Safe Clock Startup

Jitter Optimization

Balanced

Minimize Output Jitter

Maximize Input Jitter filtering

Dynamic Reconfig Interface

Options

Phase Duty Cycle Config Write DRP registers

AXI4Lite DRP

Input Clock Information

Input Clock	Port Name	Input Frequency(MHz)	^ 1	Jitter Options	Input Jitter	Source
Primary	clk_in1	100.000	19.000 - 800.000	UI	0.010	Single ended clock capable pin
Secondary	clk_in2	100.000	80.000 - 160.000		0.010	Single ended clock capable pin

Clocking Wizard (6.0)



[Documentation](#) [IP Location](#) [Switch to Defaults](#)

IP Symbol Resource
 Show disabled ports

Component Name: PLL

Clocking Options

Output Clocks

Port Renaming

PLLE2 Settings

Summary

Output Clock	Port Name	Requested	Actual	Requested	Actual	Requested	Actual	Drives
<input checked="" type="checkbox"/> clk_out1	clk_out1	50.000	50.00000	0.000	0.000	50.000	50.0	BUFG
<input checked="" type="checkbox"/> clk_out2	clk_out2	250.000	250.00000	0.000	0.000	50.000	50.0	BUFG
<input type="checkbox"/> clk_out3	clk_out3	100.000	N/A	0.000	N/A	50.000	N/A	BUFG
<input type="checkbox"/> clk_out4	clk_out4	100.000	N/A	0.000	N/A	50.000	N/A	BUFG
<input type="checkbox"/> clk_out5	clk_out5	100.000	N/A	0.000	N/A	50.000	N/A	BUFG
<input type="checkbox"/> clk_out6	clk_out6	100.000	N/A	0.000	N/A	50.000	N/A	BUFG

USE CLOCK SEQUENCING

Clocking Feedback

Source

Signaling

- Automatic Control On-Chip Single-ended
 Automatic Control Off-Chip Differential
 User-Controlled On-Chip
 User-Controlled Off-Chip



Output Clock	Sequence Number
clk_out1	1
clk_out2	1
clk_out3	1
clk_out4	1
clk_out5	1
clk_out6	1

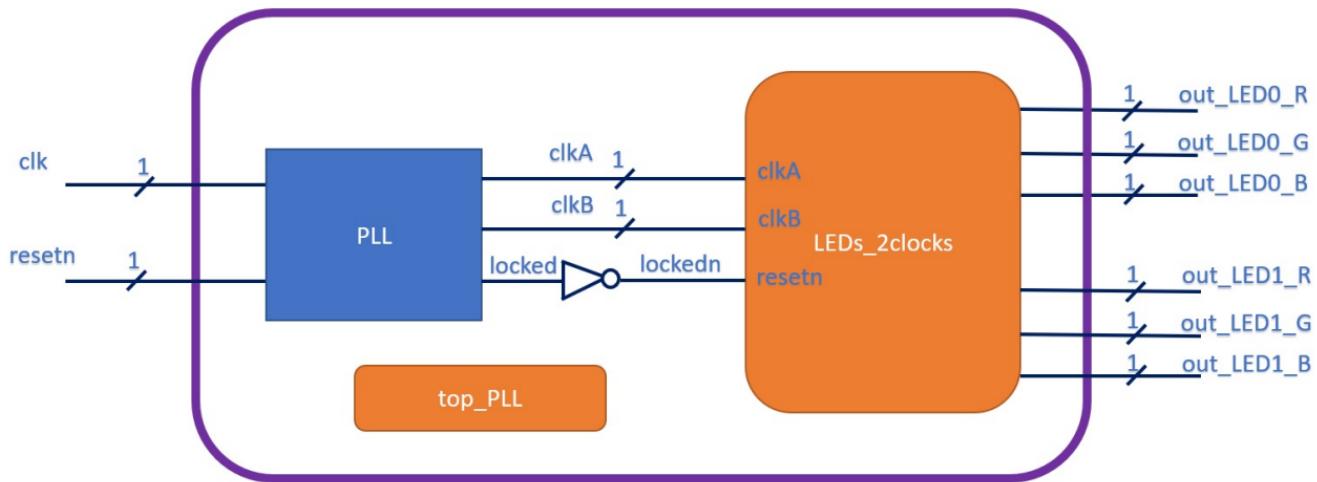
Enable Optional Inputs / Outputs for MMCM/PLL

- reset power_down
 locked

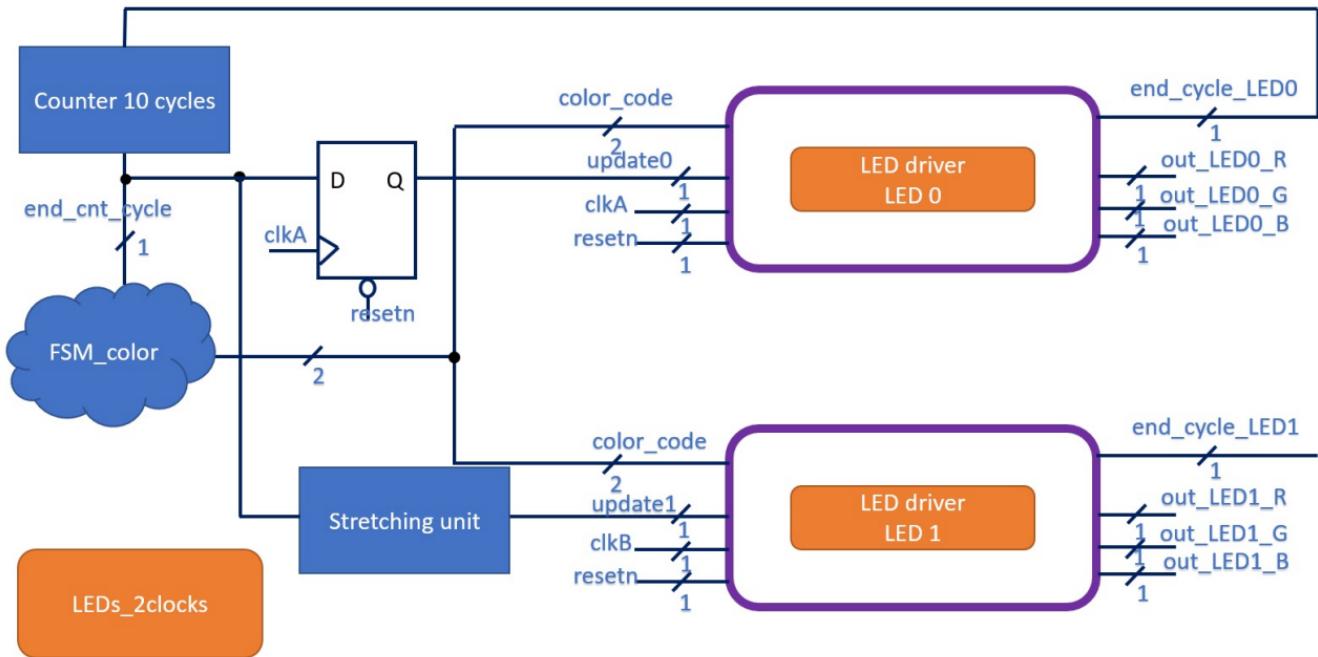
Reset Type

- Active High Active Low

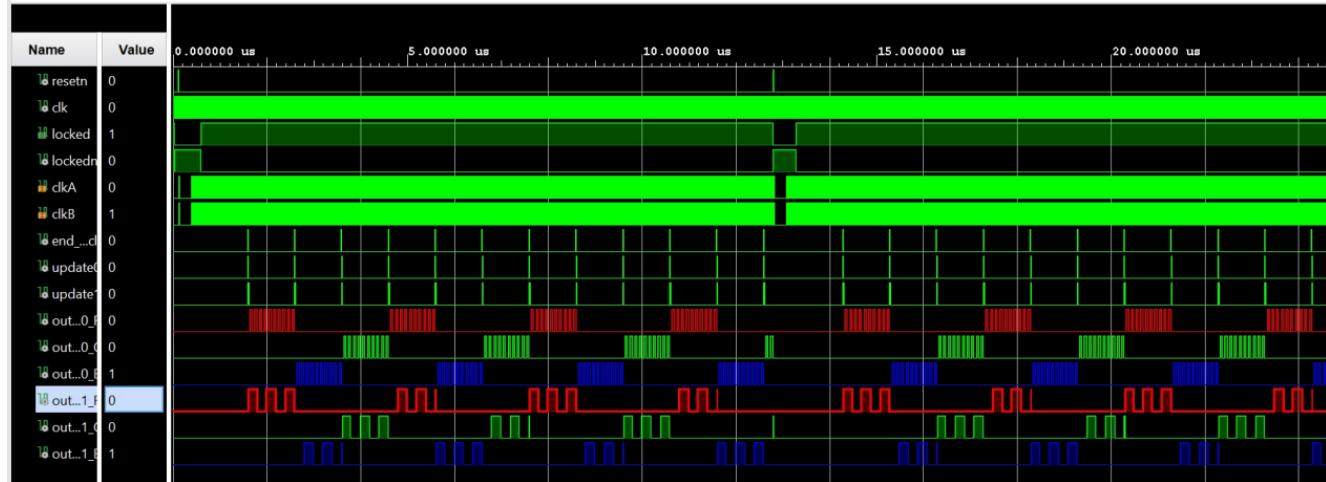
Architecture



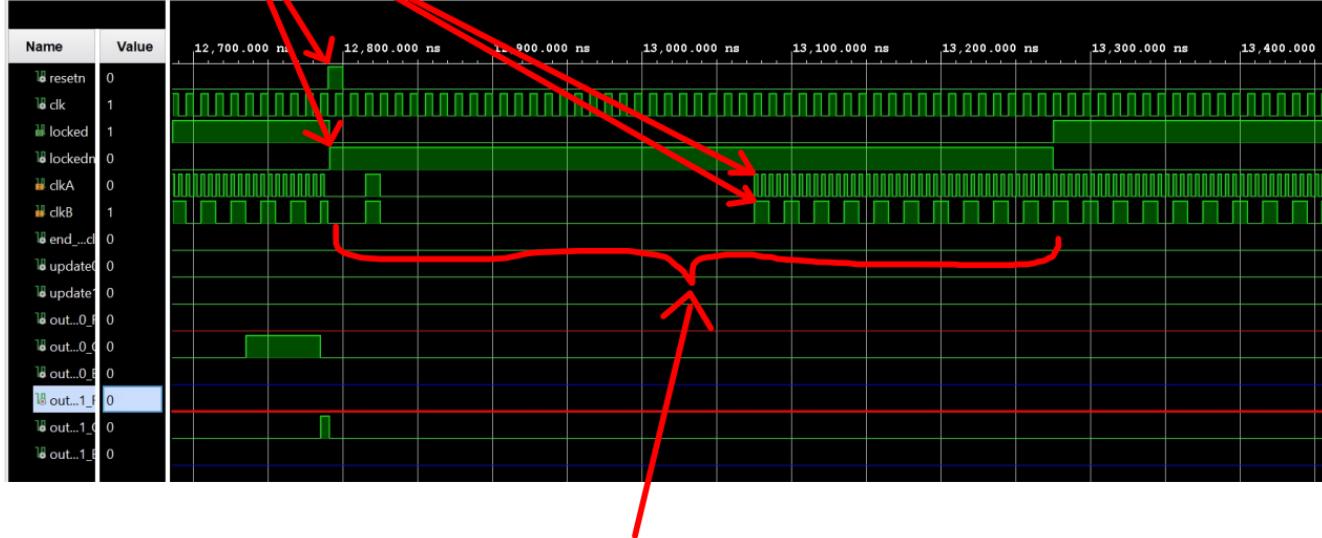
Reprise des schémas RTL des questions 7 et 8



Après modification du code, on obtient les résultats de simulation ci-dessous:



Le resetn redémarre bien la PLL comme attendu.



On a fait le choix d'utiliser la sortie locked du module PLL pour piloter le resetn du composant LEDs_2clocks.

Ce signal est à 1 tant que les horloges générées par la PLL ne sont pas stables.

10. Effectuez la synthèse et placer des sondes (ILA) sur les signaux pertinents pour vérifier que le changement de domaine d'horloge s'est correctement passé.

Set Up Debug

Nets to Debug

The nets below will be debugged with ILA cores. To add nets click "Find Nets to Add". You can also select nets in the Netlist or other windows, then drag them to the list or click "Add Selected Nets".

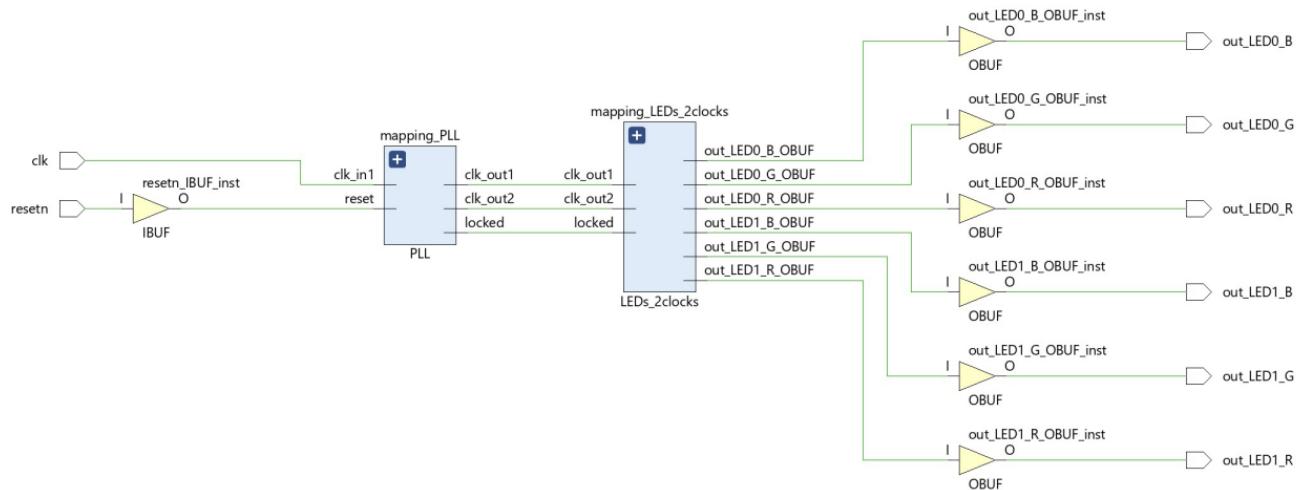
Name	Clock Domain	Driver Cell	Probe Type
> mapping_LEDs_2clocks/mapping_LED1_driver/cnt_stretch	mapping_PLL/inst/clk_out2	FDRE	Data and Trigger
out_LED0_B_OBUF	mapping_PLL/inst/clk_out2	LUT3	Data and Trigger
out_LED0_G_OBUF	mapping_PLL/inst/clk_out2	LUT3	Data and Trigger
out_LED0_R_OBUF	mapping_PLL/inst/clk_out2	LUT3	Data and Trigger
out_LED1_B_OBUF	mapping_PLL/inst/clk_out1	LUT3	Data and Trigger
out_LED1_G_OBUF	mapping_PLL/inst/clk_out1	LUT3	Data and Trigger
out_LED1_R_OBUF	mapping_PLL/inst/clk_out1	LUT3	Data and Trigger
resetn_IBUF	mapping_PLL/inst/clkbout_buf_PLL	IBUF	Data and Trigger
mapping_LEDs_2clocks/mapping_LED0_driver/update	mapping_PLL/inst/clk_out2	FDRE	Data and Trigger
mapping_LEDs_2clocks/mapping_LED0_driver/update0_reg	mapping_PLL/inst/clk_out2	LUT6	Data and Trigger

Find Nets to Add... Nets to debug: 12

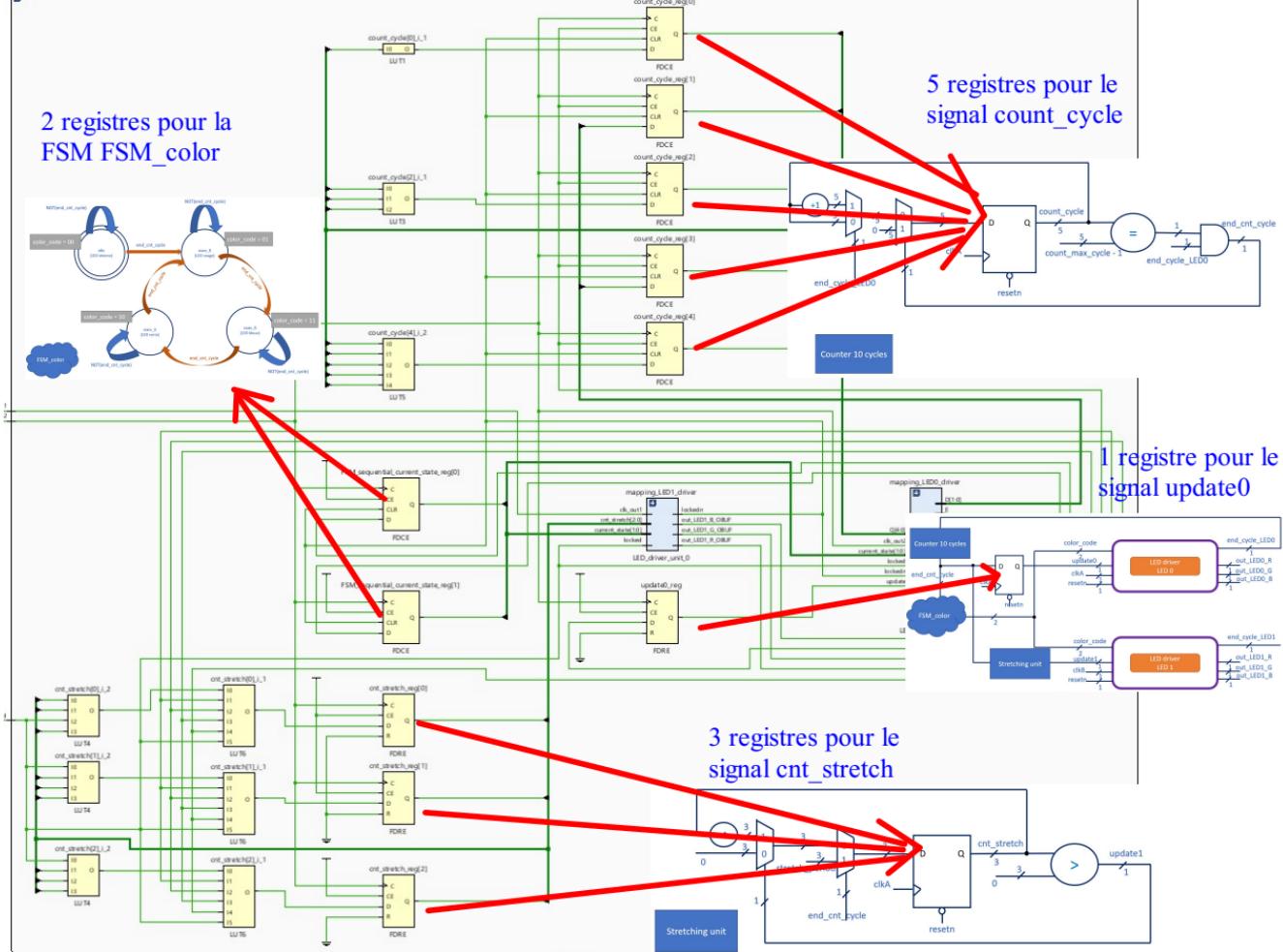
?

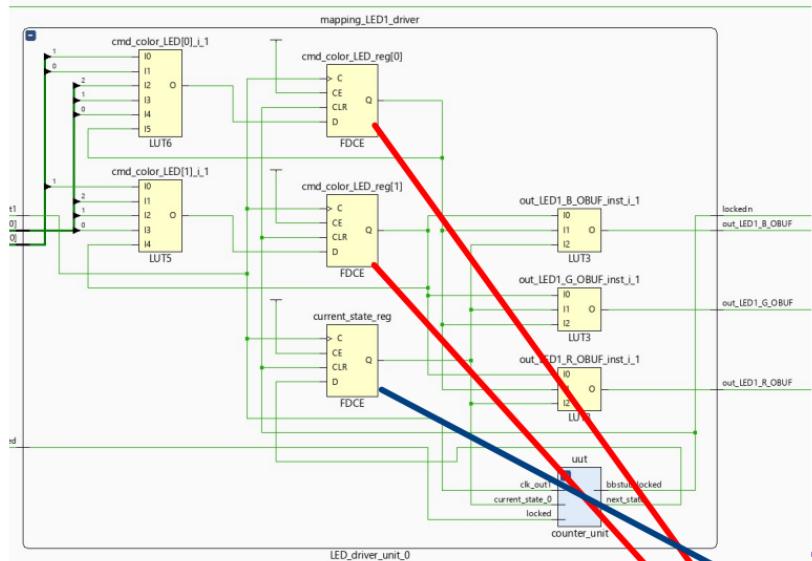
< Back Next > Finish Cancel

11. Etudiez le rapport de synthèse.



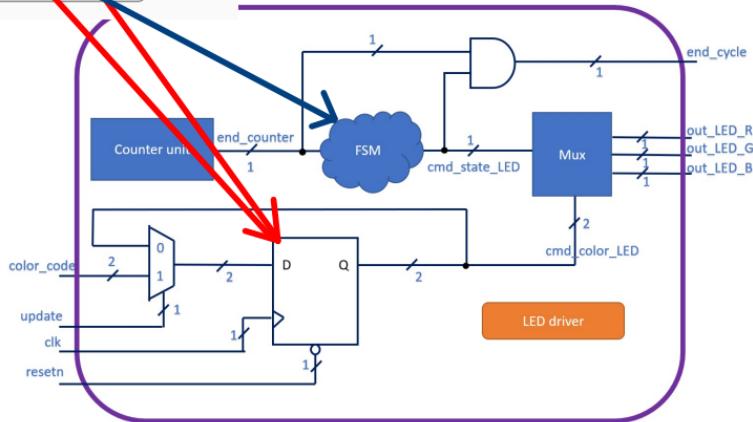
2 registres pour la
FSM FSM_color

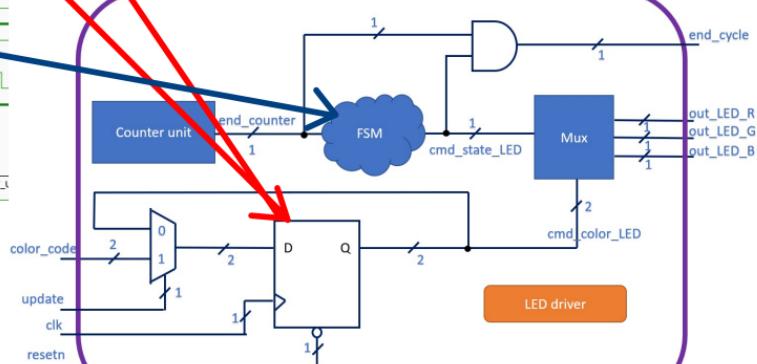
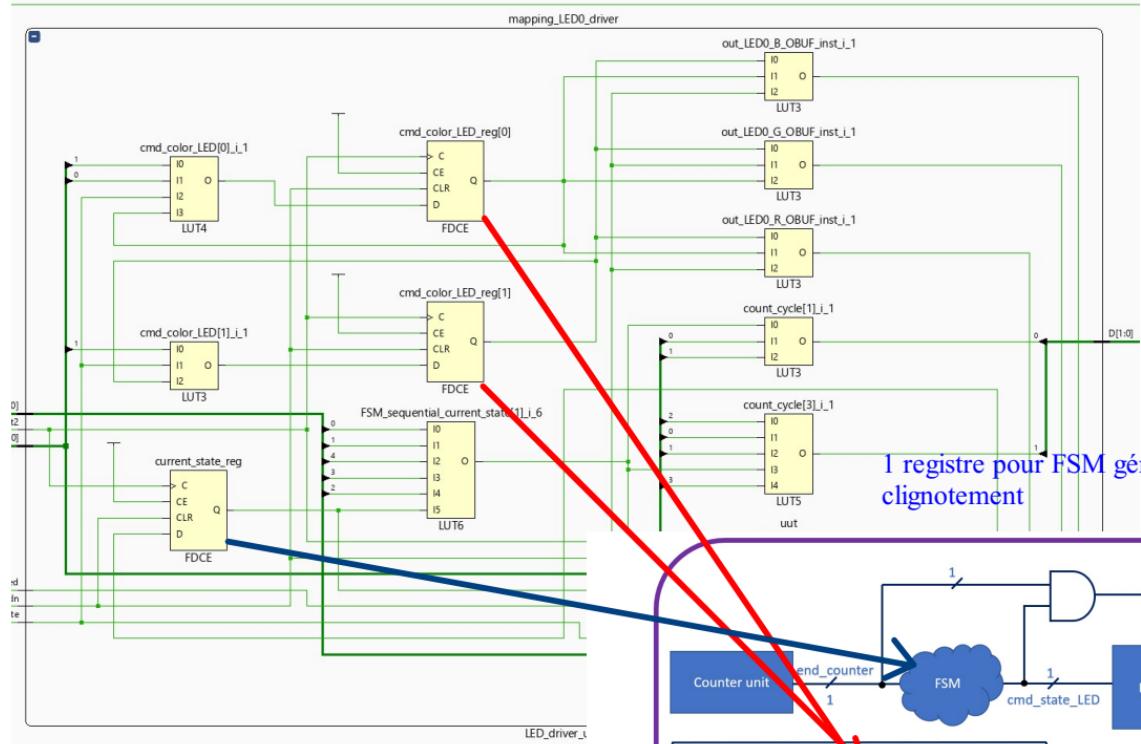




1 registre pour FSM gérant le clignotement

2 registres pour le signal cmd_color_LED





Codage des états dans la FSM du module LEDs_2clocks

	State	New Encoding	Previous Encoding
92			
93	idle	00	00
94			
95	state_r	01	01
96			
97	state_b	10	10
98			
99	state_g	11	11

La FSM_color du module LEDs_2 clocks est codée sur 2 bits.

1 adders à 5 bits dans le module "Counter 10 cycles"

1 adders à 3 bits dans le module "Stretching unit"

```
104 -----
105 Start RTL Component Statistics
106 -----
107 Detailed RTL Component Info :
108 +---Adders :
109     2 Input      5 Bit      Adders := 1
110     2 Input      3 Bit      Adders := 1
111 +---Registers :
112         5 Bit      Registers := 1
113         3 Bit      Registers := 1
114         2 Bit      Registers := 2
115         1 Bit      Registers := 3
116 +---Muxes :
117     2 Input      5 Bit      Muxes := 1
118     4 Input      2 Bit      Muxes := 2
119     2 Input      2 Bit      Muxes := 1
120     2 Input      1 Bit      Muxes := 3
121 -----
122 Finished RTL Component Statistics
```

1 registre à 5 bits pour le signal count_cycle

1 registre à 3 bits pour le signal cnt_stretch

1 registre à 2 bits pour la machine à états FSM_color

2 registres à 2 bits pour les signaux
cmd_color_LED des modules LED_driver

2 registres à 1 bit pour les FSM gérant le
clignotement des modules LED_driver

1 registre à 1 bit pour le signal update0

```

209 Report BlackBoxes:
210 +-----+-----+
211 |      |BlackBox name |Instances |
212 +-----+-----+
213 |1     |PLL           |      1|
214 +-----+-----+
215
216 Report Cell Usage:
217 +-----+-----+
218 |      |Cell       |Count  |
219 +-----+-----+
220 |1     |PLL         |      1|
221 |2     |CARRY4     |    14|
222 |3     |LUT1        |      3|
223 |4     |LUT3        |      9|
224 |5     |LUT4        |    52|
225 |6     |LUT5        |      9|
226 |7     |LUT6        |    11|
227 |8     |FDCE        |    67|
228 |9     |FDRE        |      4|
229 |10    |IBUF        |      1|
230 |11    |OBUF        |      6|
231 +-----+-----+
232
233 Finished Writing Synthesis Report :

```

Composants permettant l'optimisation des fonctions de comptage

LUT:
Look Up Tables pour la logique combinatoire (portes et multiplexeurs)

IBUF - 1 entrée
- resetn

FDCE et FDRE ($67+4=71$):

- $27 \times 2 = 54$ utilisés pour les modules counter unit
- 5 utilisés pour le signal count_cycle
- 3 utilisés pour le signal cnt_stretch
- 2 utilisés pour la machine à états FSM_color

- 4 utilisés pour les signaux cmd_color_LED des modules LED_driver
- 2 utilisés pour les FSM gérant le clignotement des modules LED_driver
- 1 utilisé pour le signal update0

TOTAL: 71 => Le compte est bon !

OBUF - 6 sorties:
- 3 sorties LED0
- 3 sorties LED1

12. Effectuez le placement routage et étudiez les rapports générés.

C:/FORMATION_SAFRAN/TP/TP5_Domaines_Horloge/Cora-Z7-10-Master.xdc

Q | F | ← | → | X | E | X | // | ■ | ? |

```
1 ## This file is a general .xdc for the Cora Z7-07S 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]
8 create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add [get_ports clk]
9
10 # RGB LEDs
11 set_property -dict {PACKAGE_PIN L15 IOSTANDARD LVCMOS33} [get_ports out_LED0_B]
12 set_property -dict {PACKAGE_PIN G17 IOSTANDARD LVCMOS33} [get_ports out_LED0_G]
13 set_property -dict {PACKAGE_PIN N15 IOSTANDARD LVCMOS33} [get_ports out_LED0_R]
14 set_property -dict {PACKAGE_PIN G14 IOSTANDARD LVCMOS33} [get_ports out_LED1_B]
15 set_property -dict { PACKAGE_PIN L14   IOSTANDARD LVCMOS33 } [get_ports { out_LED1_G }]; #IO_L22P_T3_AD7P_35 Sch=led1_g
16 set_property -dict { PACKAGE_PIN M15   IOSTANDARD LVCMOS33 } [get_ports { out_LED1_R }]; #IO_L23N_T3_35 Sch=led1_r
17
18 # Buttons
19 set_property -dict {PACKAGE_PIN D20 IOSTANDARD LVCMOS33} [get_ports resetn]
20 #set_property -dict {PACKAGE_PIN D19 IOSTANDARD LVCMOS33} [get_ports bouton_1]
..
```

Design Timing Summary										
	WNS(ns)	TNS(ns)	NS Failing Endpoints	TNS Total Endpoints		WHS(ns)	THS(ns)	HS Failing Endpoints	THS Total Endpoints	WPWS(ns)
125	0.586	0.000	0	76		0.109	0.000	0	76	2.000
126										
127										
128										
129										
130										
131										
132										

Total Negative Slack = 0
 => Pas de violation de setup

Total Hold Slack = 0
 => Pas de violation du hold

Chemin critique

```
240 Max Delay Paths
241 -----
242 Slack (MET) : 14.712ns (required time - arrival time)
243   Source: mapping_LEDs_2clocks/mapping_LED1_driver/uut/Q_out_cnt_reg[22]/c
244           (rising edge-triggered cell FDCE clocked by clk_out1_PLL (rise@0.000ns fall@10.000ns period=20.000ns))
245 Destination: mapping_LEDs_2clocks/mapping_LED1_driver/uut/Q_out_cnt_reg[25]/d
246           (rising edge-triggered cell FDCE clocked by clk_out1_PLL (rise@0.000ns fall@10.000ns period=20.000ns))
```

13. Générez le bitstream, programmez la carte et vérifiez les signaux du chipscope (ILA).