

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.

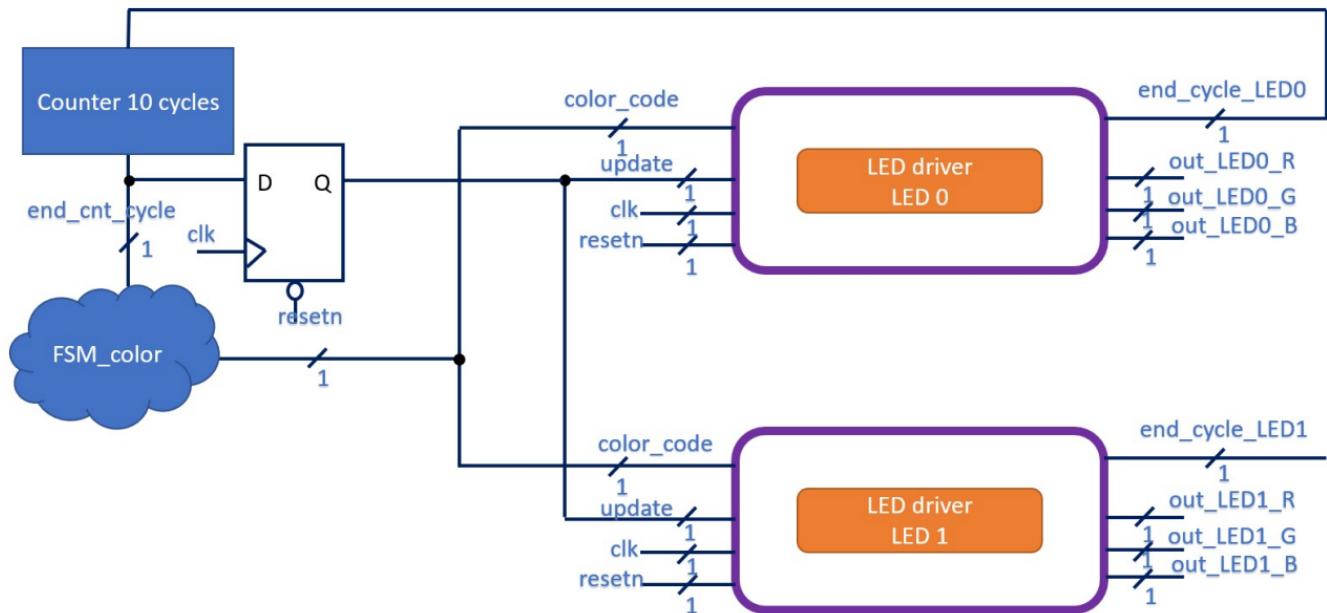
Le compteur de temporisation devra compter 100 000 000 coups d'horloge (pour une fréquence d'horloge à 100MHz, cela correspond à 1s). **Dans la suite de ce TP, la valeur du compteur de temporisation ne sera pas modifiée, même lorsque les fréquences d'horloges changeront.**

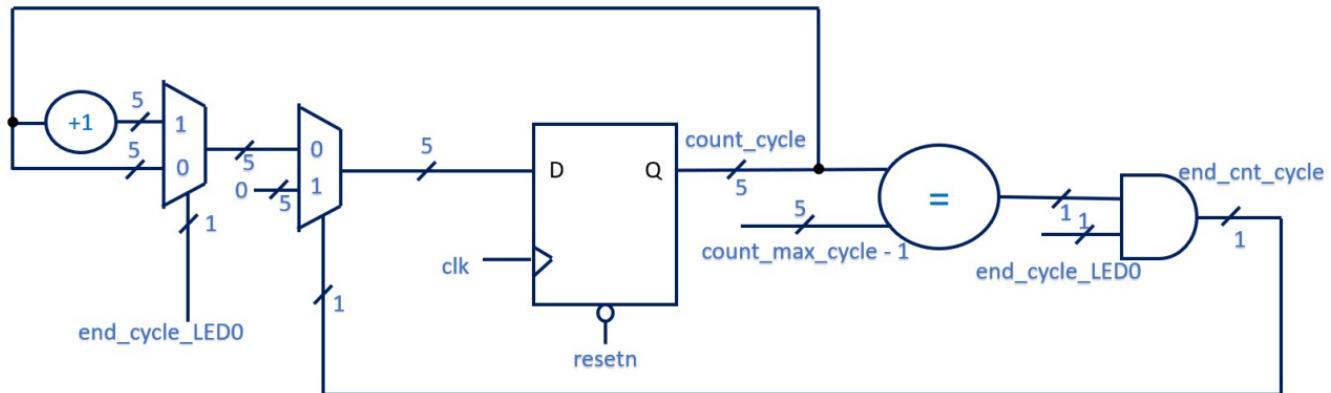


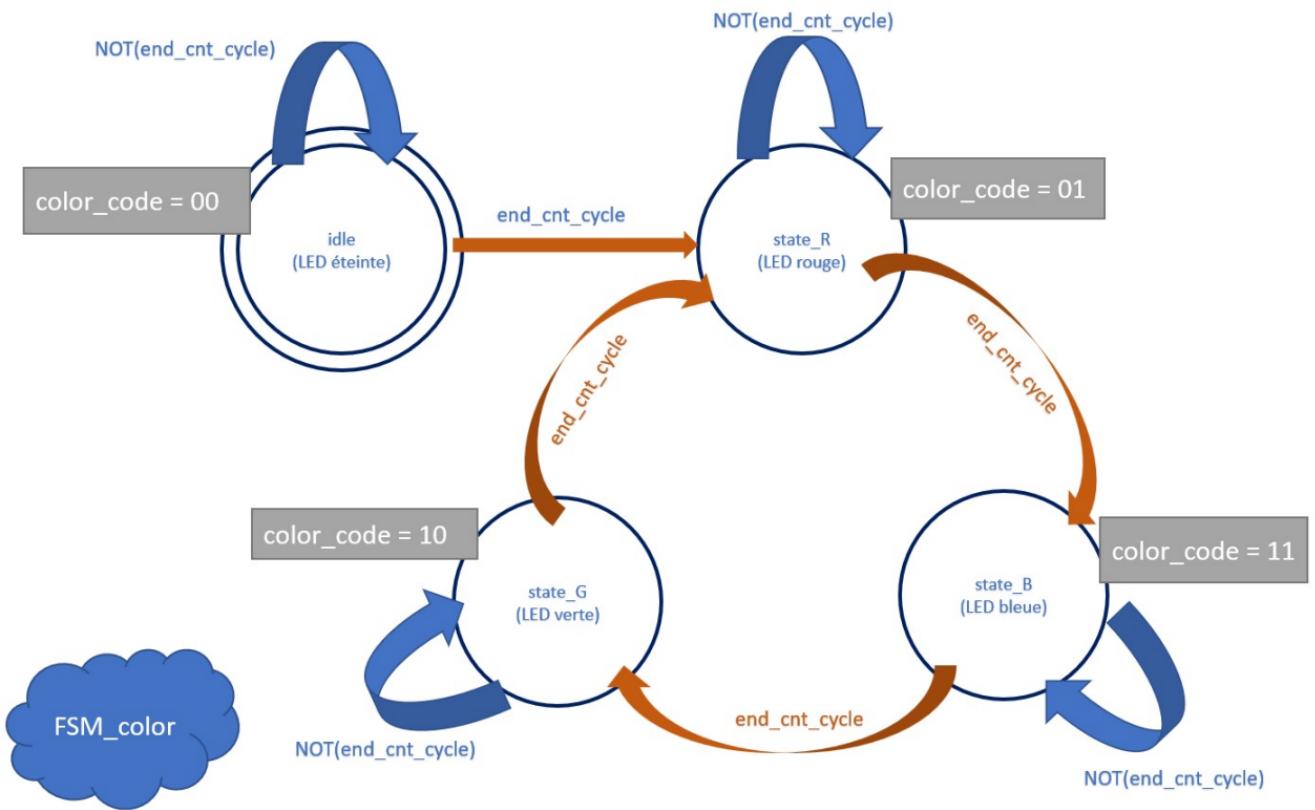
Le signal `end_cycle_LED0` est envoyé à chaque fin de cycle allumé/éteint.

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

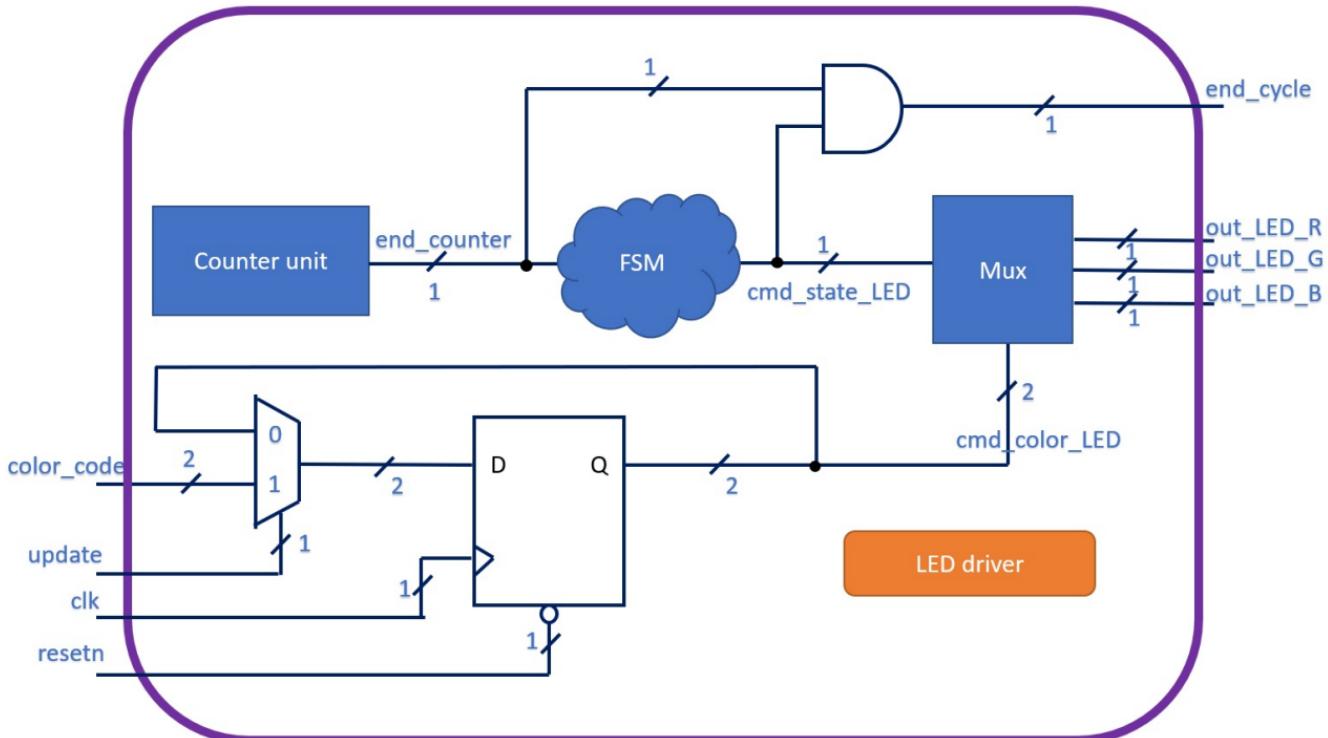
NB: Nous devons insérer un registre entre les signaux `end_cnt_cycle` et `update` afin de synchroniser les signaux `color_code` et `update`.



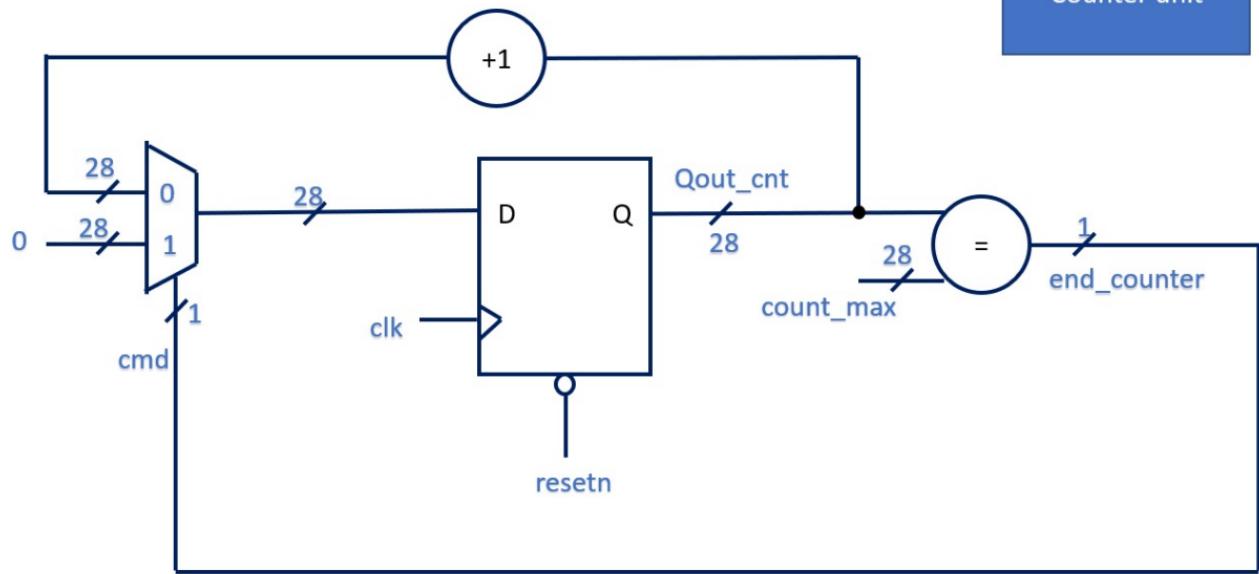




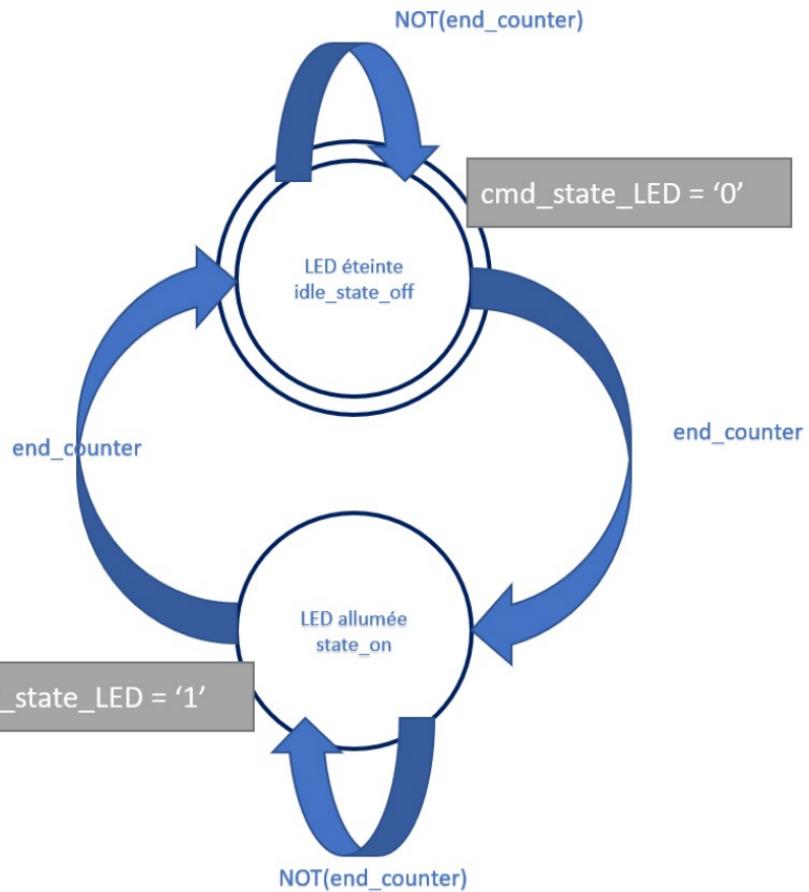
Module LED_driver (version 3)

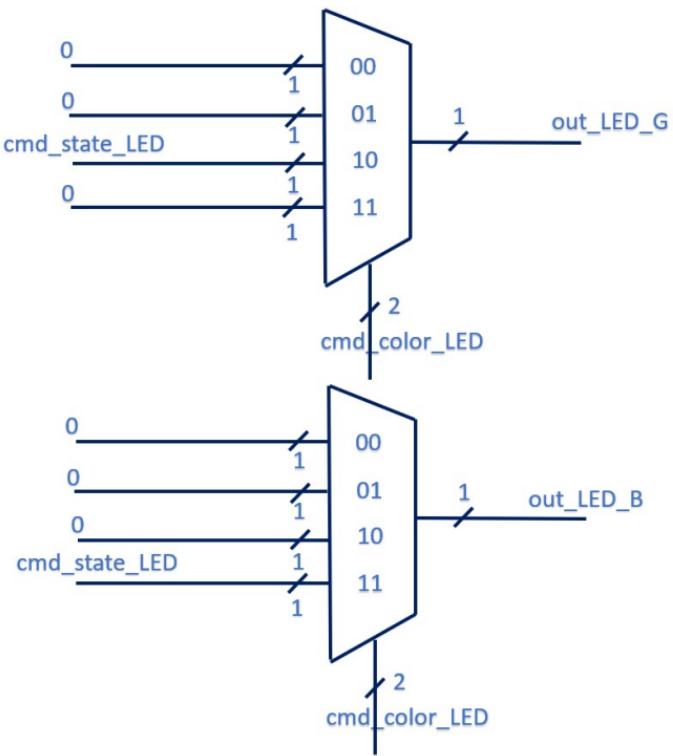
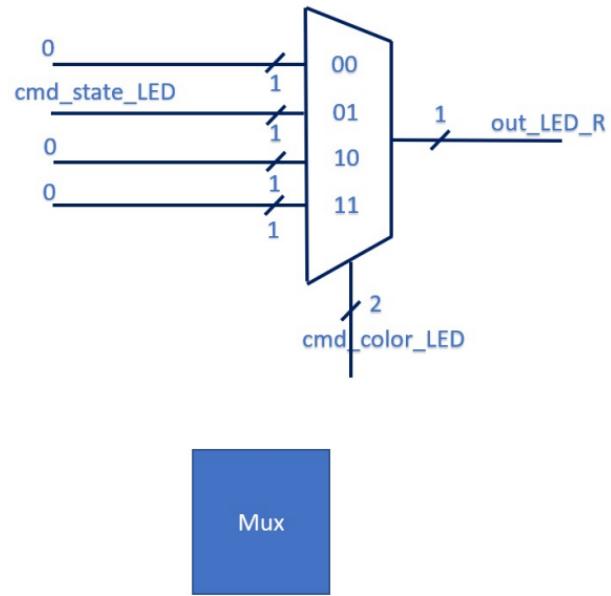


Counter unit



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 SIGNAUX
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
```

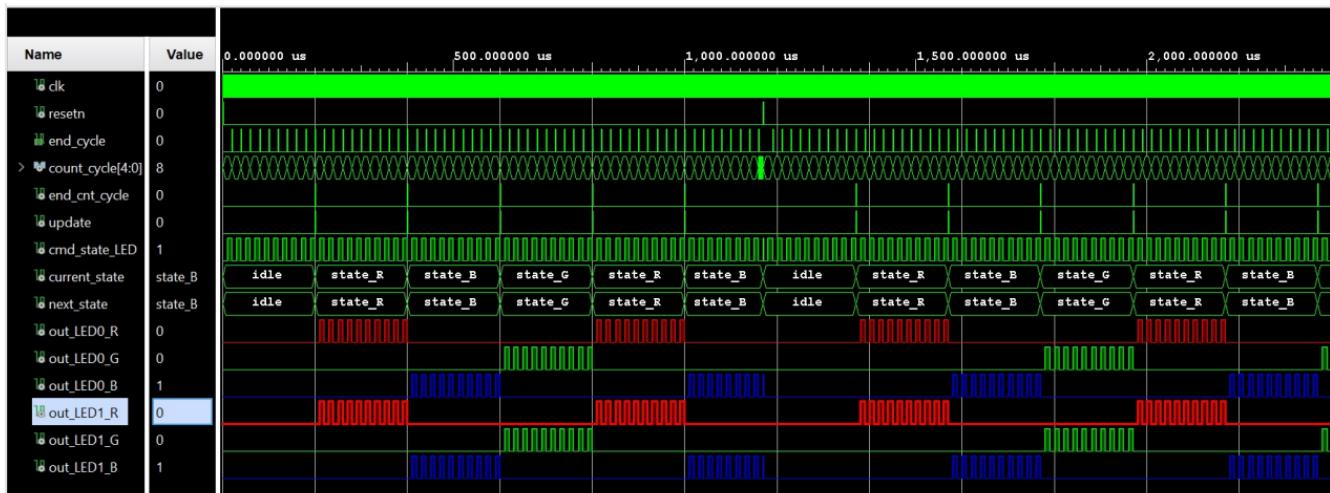
```
107 -----
108 -- PARTIE SEQUENTIELLE
109 -----
110 process(clk, resetn)
111 begin
112   if resetn = '1' then
113     current_state <= idle;           -- Retour de la FSM à l'état initial
114
115   count_cycle <= (others => '0'); -- Réinitialisation du compteur
116
117   elsif rising_edge(clk) then
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

--

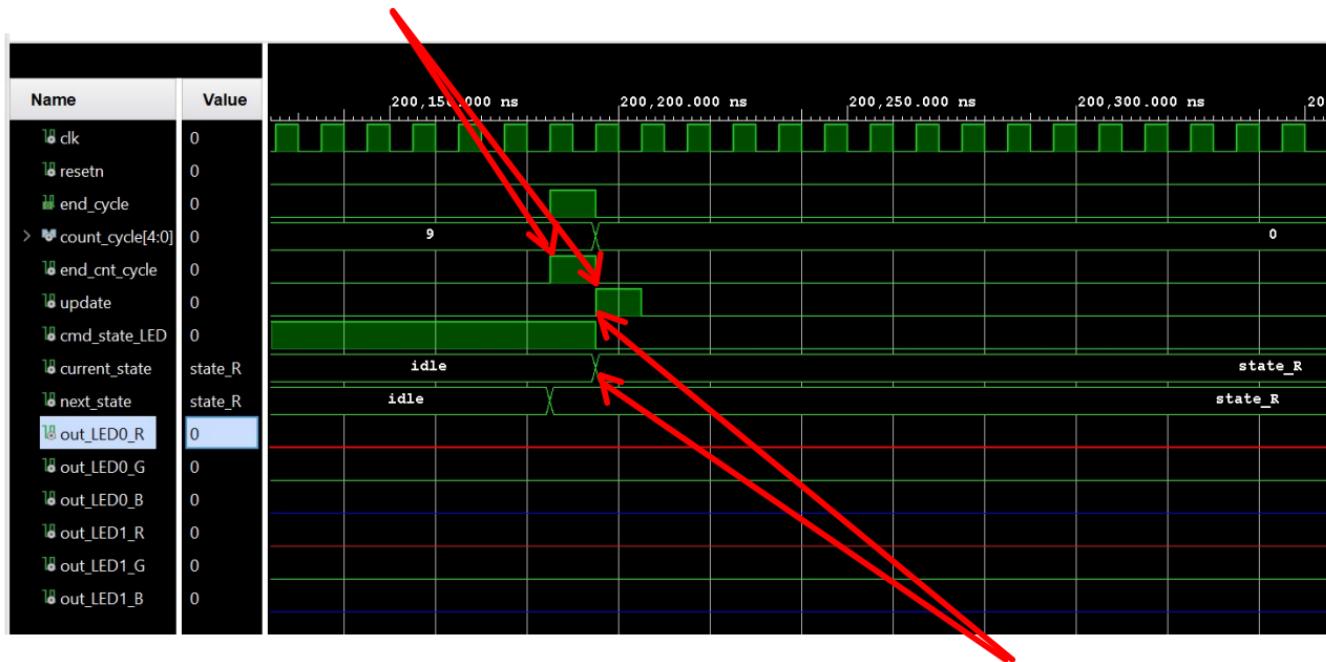
-- Test des 10 cycles "allume/eteint"

-- 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.



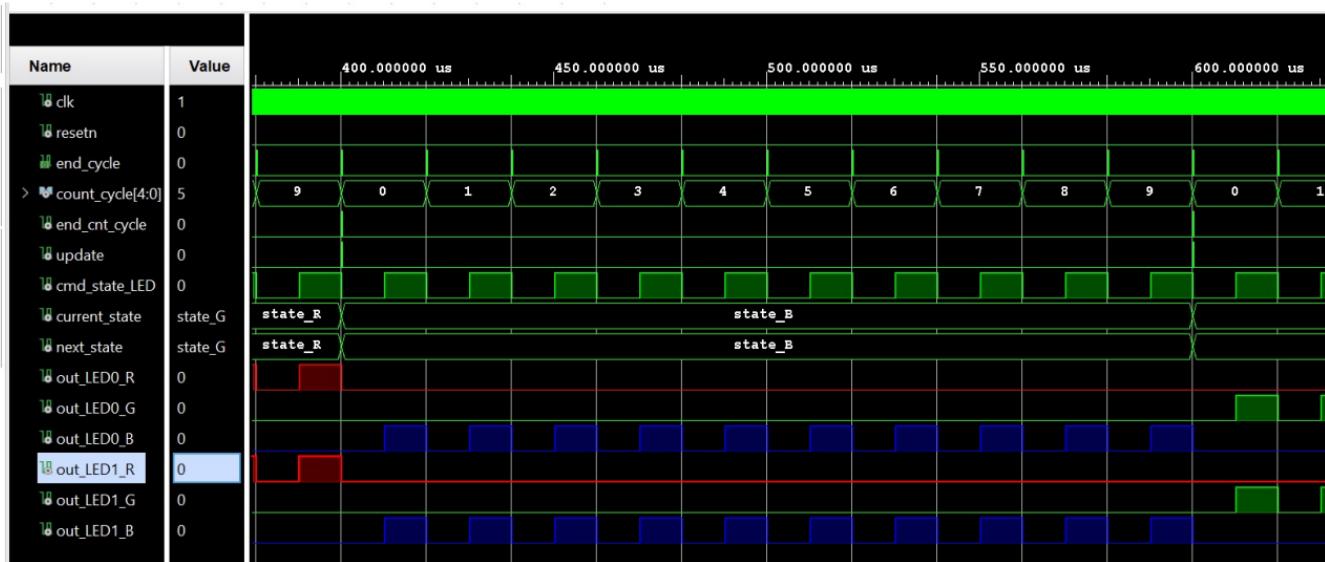
Le signal update est bien en phase avec le signal current_state.

-- 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.



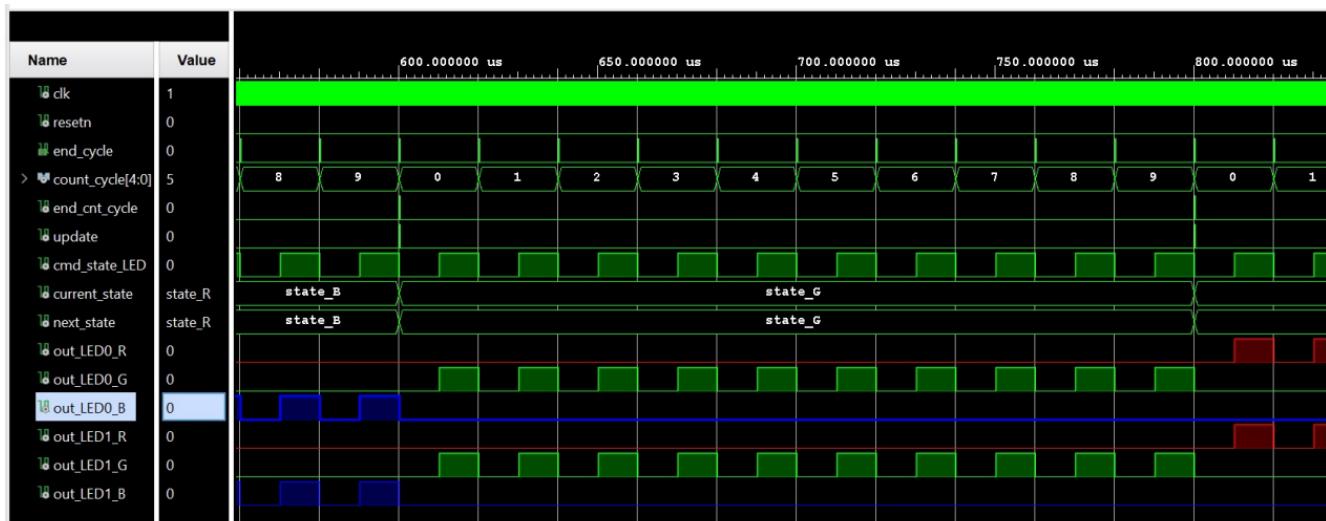
Le signal update est bien en phase avec le signal current_state.

-- QUATRIEME TEST - ETAT VERT

--

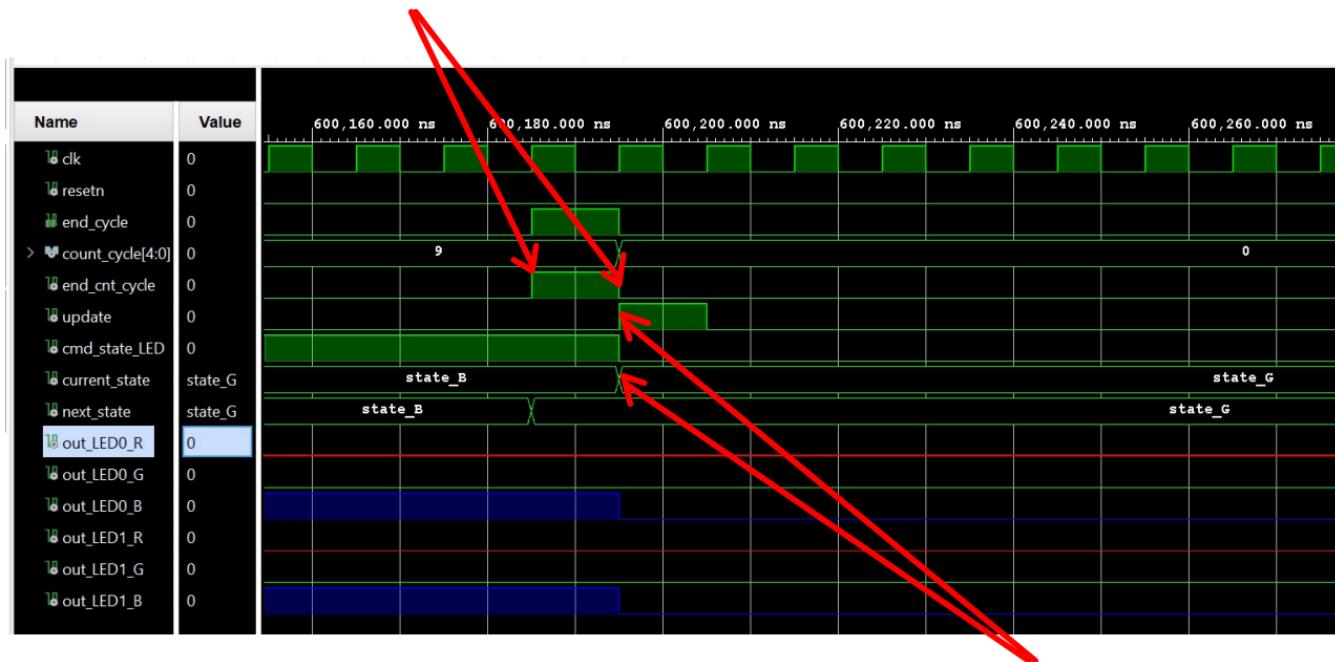
-- Test des 10 cycles "allume/eteint"

-- 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



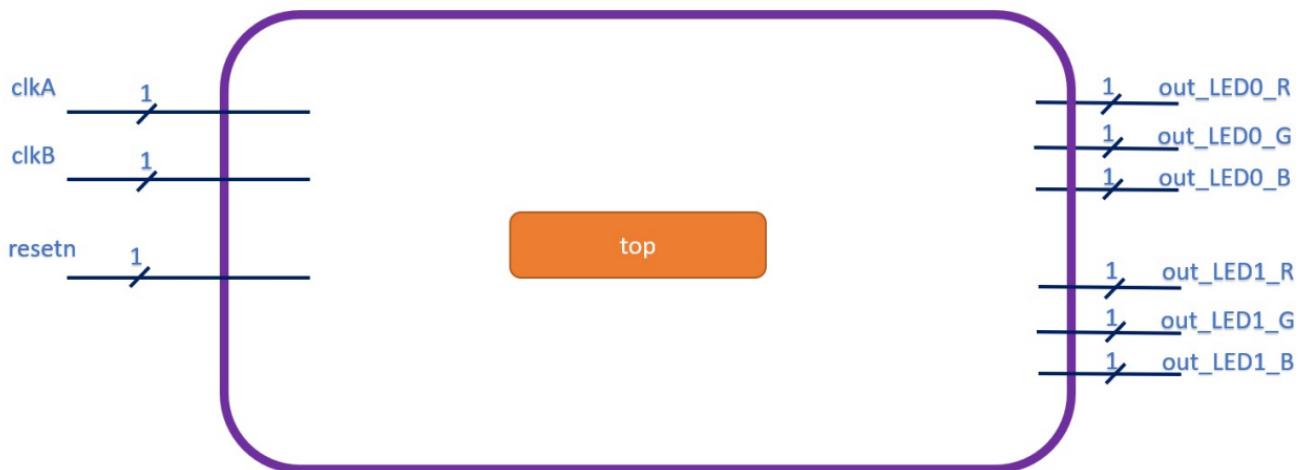
Si on zoom pour s'intéresser au comportement lors du resetn, les LEDs arrêtent de clignoter en bleu, la FSM passe bien à l'état "idle".



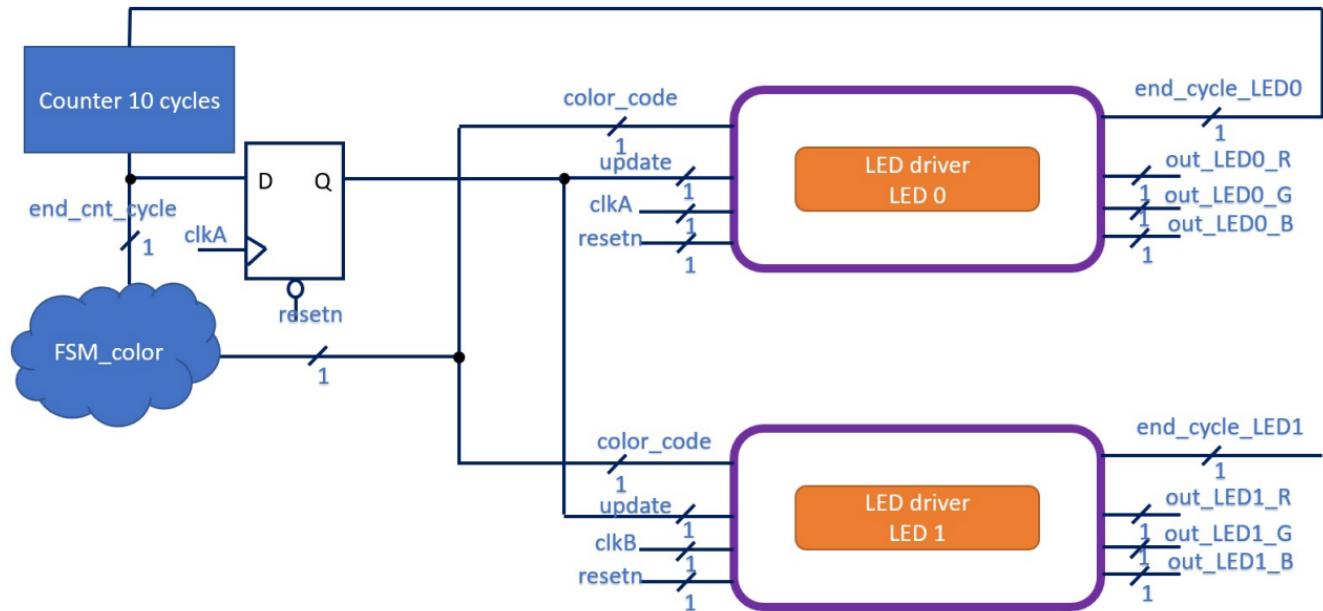
4. Modifiez votre design pour gérer deux signaux d'horloge différents. La première horloge, *clkA*, est associé à la logique en dehors des modules *LED_driver* et au module *LED_driver* de la LED0. La deuxième horloge, *clkB*, est associé au module *LED_driver* de la LED1.

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

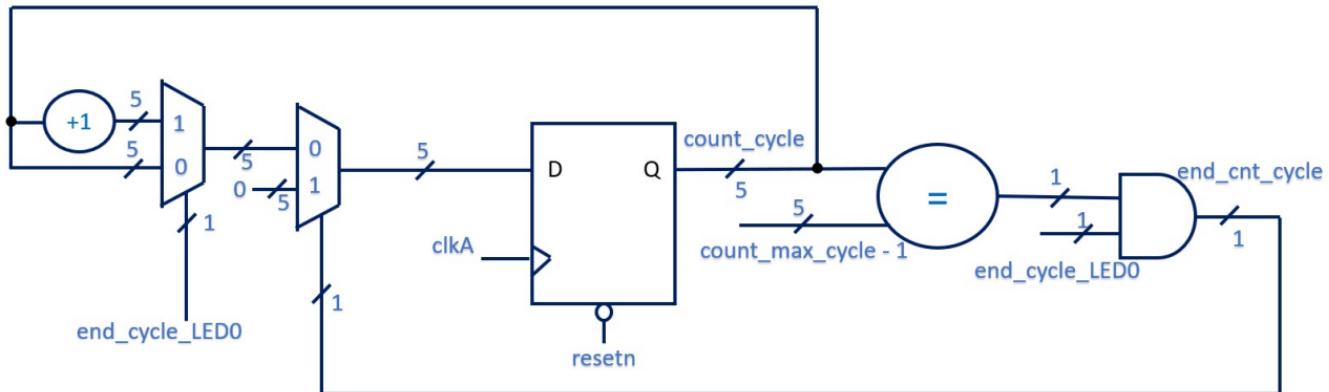
On conserve le design des questions 1 à 3 en séparant les horloges des composants *LED_driver*.



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

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

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 ?

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/>.

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.

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.

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. The top navigation bar includes tabs for Project Summary, tp_domaineHorloge_Q1-3.vhd, LED_driver_unit_v3.vhd, tb_domaineHorloge_Q1-3.vhd, and IP Catalog, with the IP Catalog tab currently selected. Below the tabs, there are two tabs: Cores and Interfaces, with Cores being the active tab. A toolbar with various icons follows, including a magnifying glass for search, a refresh icon, and other filtering options. A search bar labeled "Search:" is present. The main content area displays a table with columns for Name, Status, License, and VLVN. The table lists several IP cores under the "Clocking" category, with one entry highlighted: "Clocking Wizard" (Status: Production, License: Included, VLVN: xilinx.com:ip:clk_wiz:6.0).

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

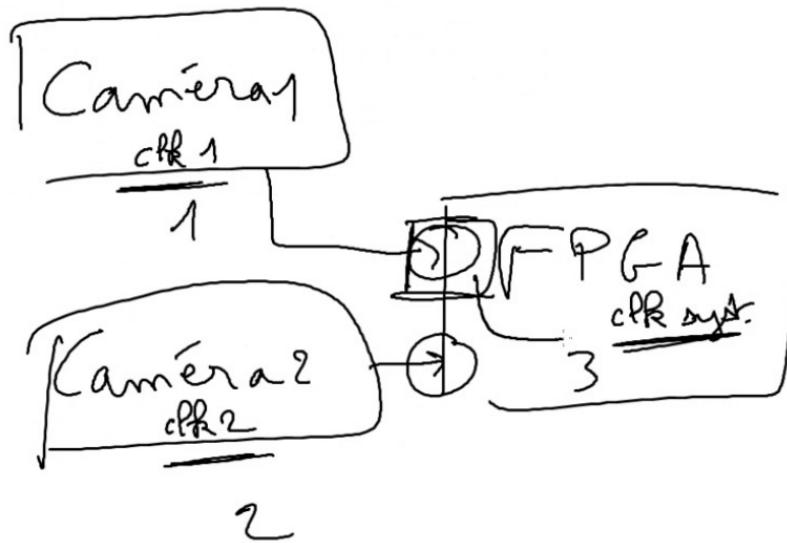
10. Effectuez la synthèse et placer des sondes (ILA) sur les signaux pertinents pour vérifier que le changement de domaine d'horloge s'est correctement passé.

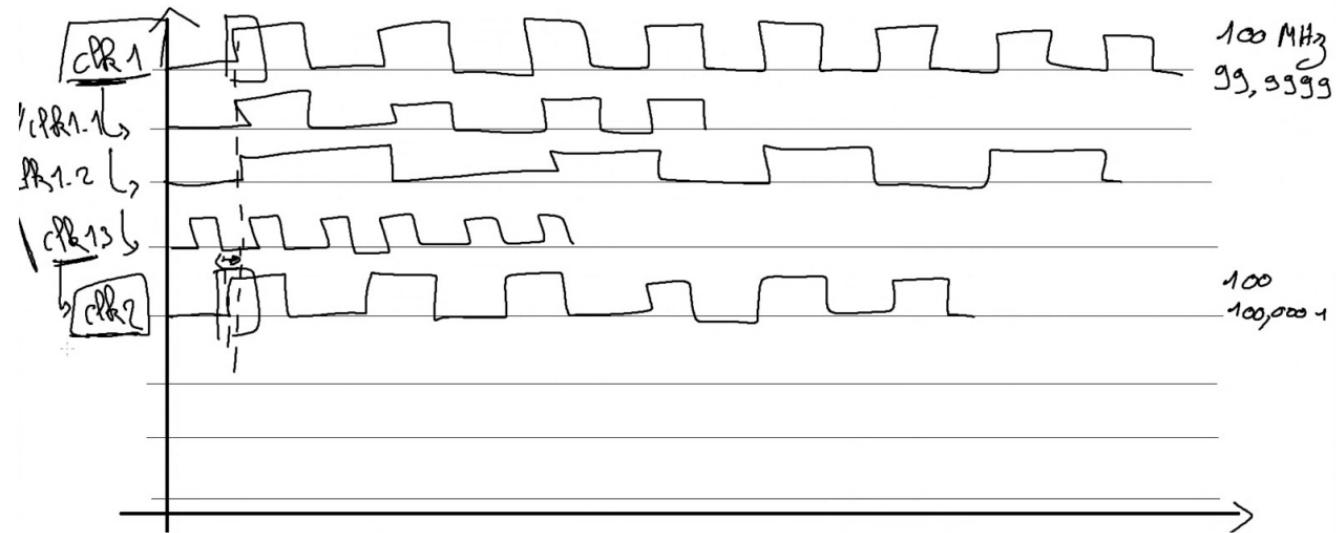
11. Etudiez le rapport de synthèse.

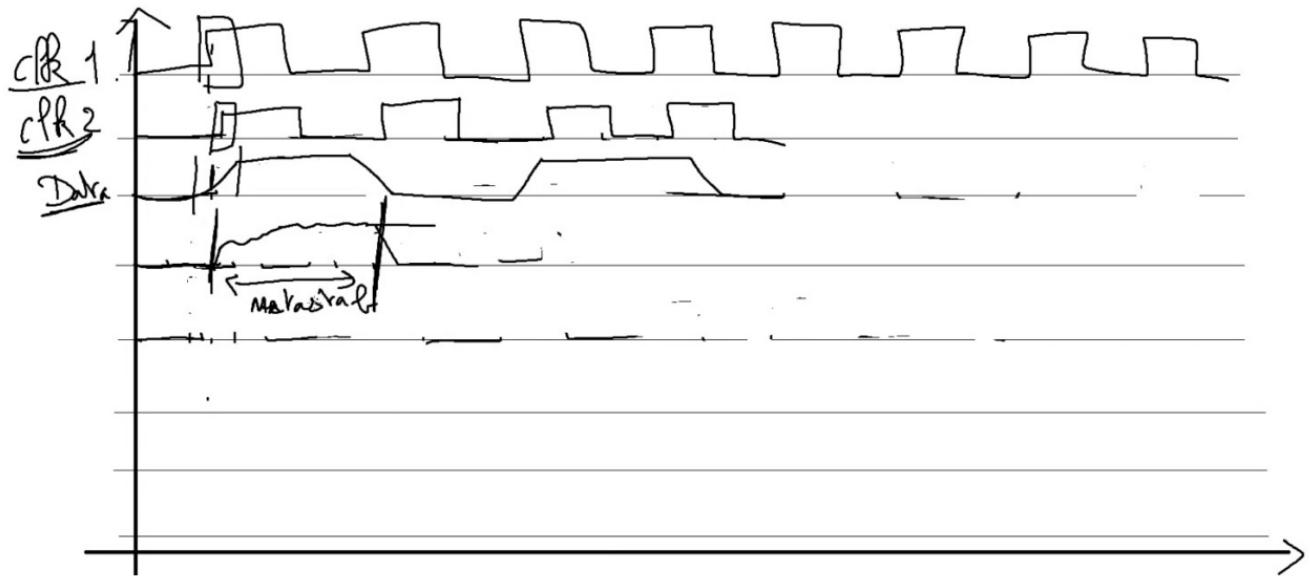
12. Effectuez le placement routage et étudiez les rapports générés.

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

3 domaines d'horloge

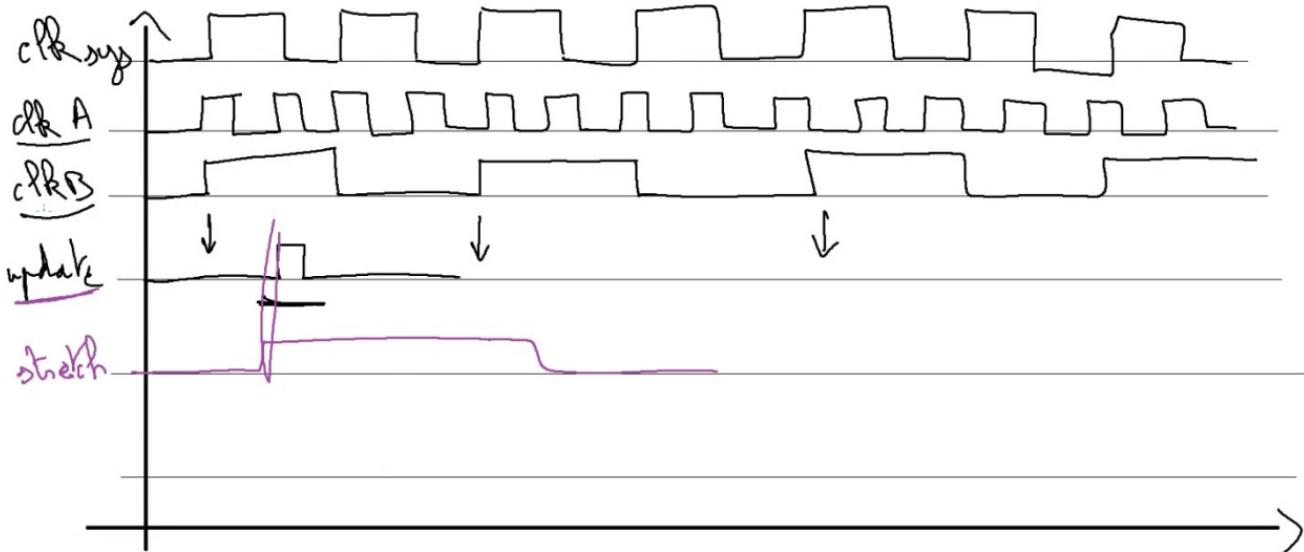




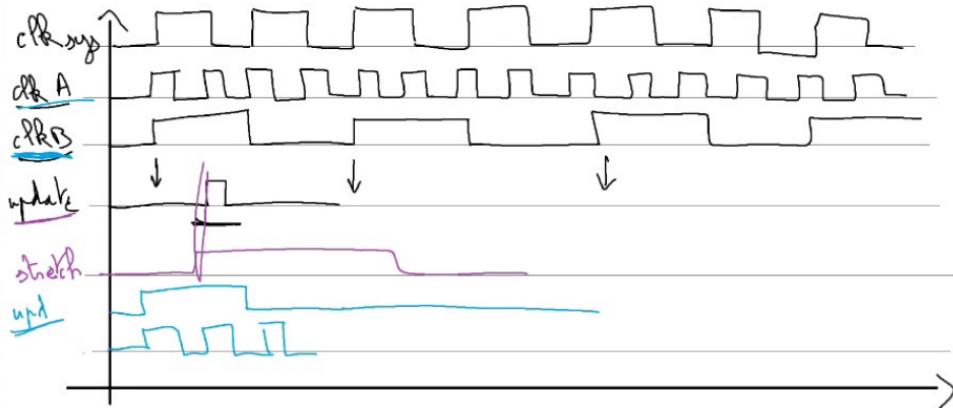


Solution du double rideau pour palier à la métastabilité

TP



Exercice



Si le signal update est généré par la clkB, il sera vu plusieurs fois par la clkA.
Pour palier à ce pb, on peut transférer le signal update bleu en un front montant.

[Retour au TP](#)

Pb du reset

=> les 2 horloges ne repartent pas forcément en même temps

=> Solution: strecth de l'update => 1 update pour chaque horloge

=> Pas de double rideau pour le TP

Question 4

Les horloges ne clignotent pas à la même cadence.

Architecture avec PLL

