

## Compte rendu de – TP05 - Domaines d' horloge

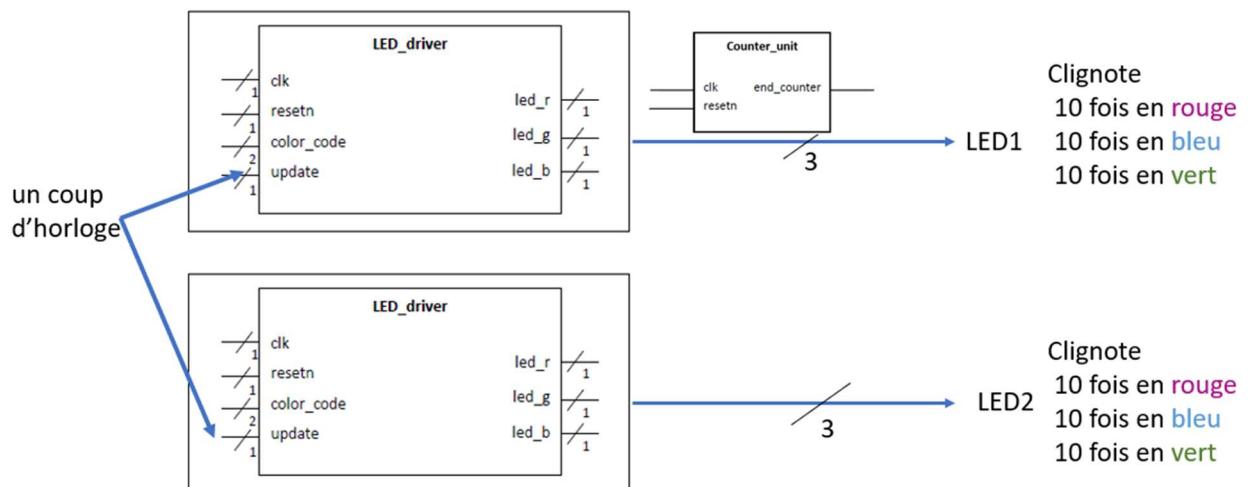
### 1.1 Objectif de ce TP

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.

**Question 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.

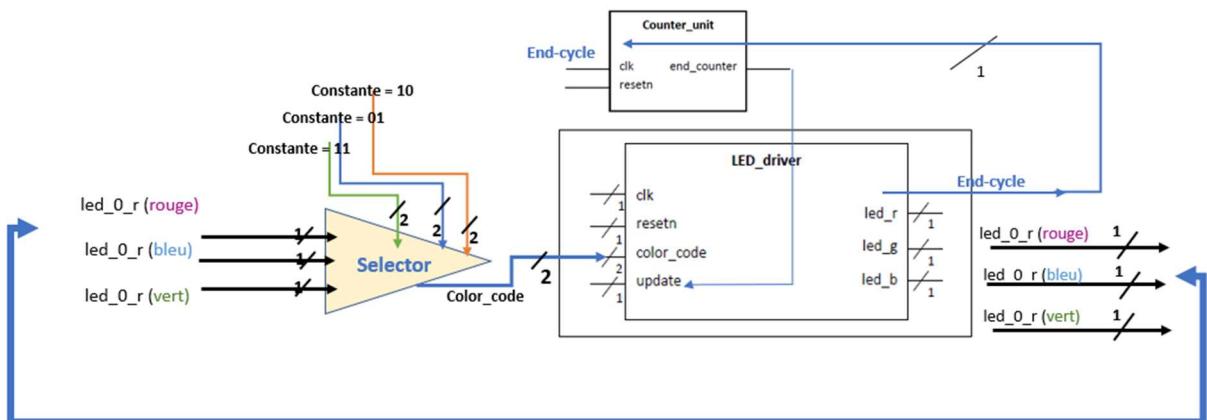
**Question 2 & Question 3 :** (Rédigez le code VHDL correspondant à votre architecture+ 3. Rédigez le testbench et simulez votre design. Vérifiez que les modules réceptionnent correctement le signal update.

Ma compréhension avant de commencer le coder dans le but de bien tracer mes RTL et mon architecture :



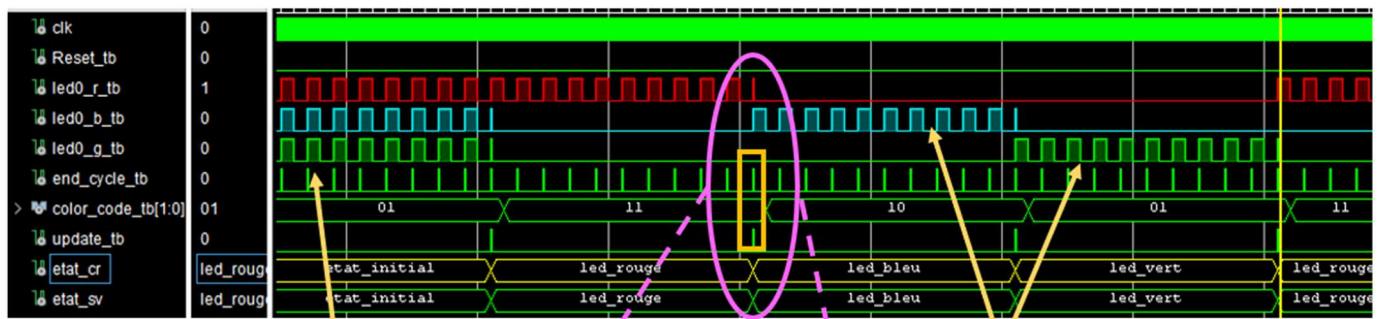
Le compteur de temporisation devra compter 100 000 000 coups d'horloge  
( $N= 100\ 000\ 000$ )

Les LEDs RGB devront clignoter 10 fois ( $M= 10$ )

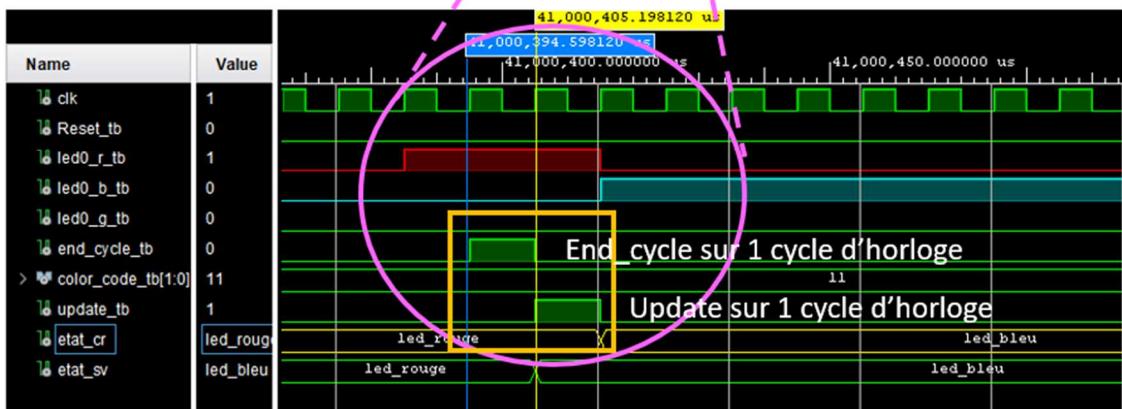


Un seul led driver

Simulation fonctionnelle de mon premier led\_driver avec le compteur et le selector :

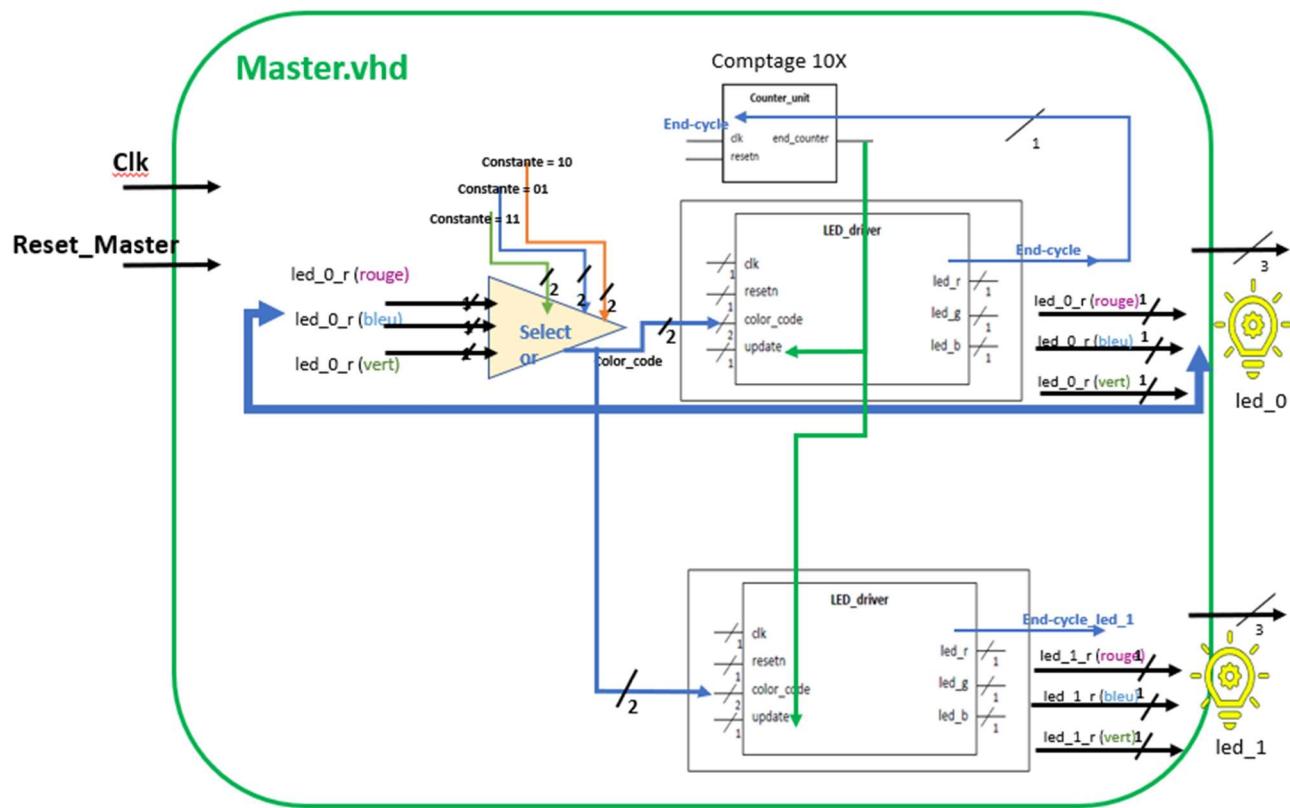


Mon état initial est blanc Les 3 leds allumées

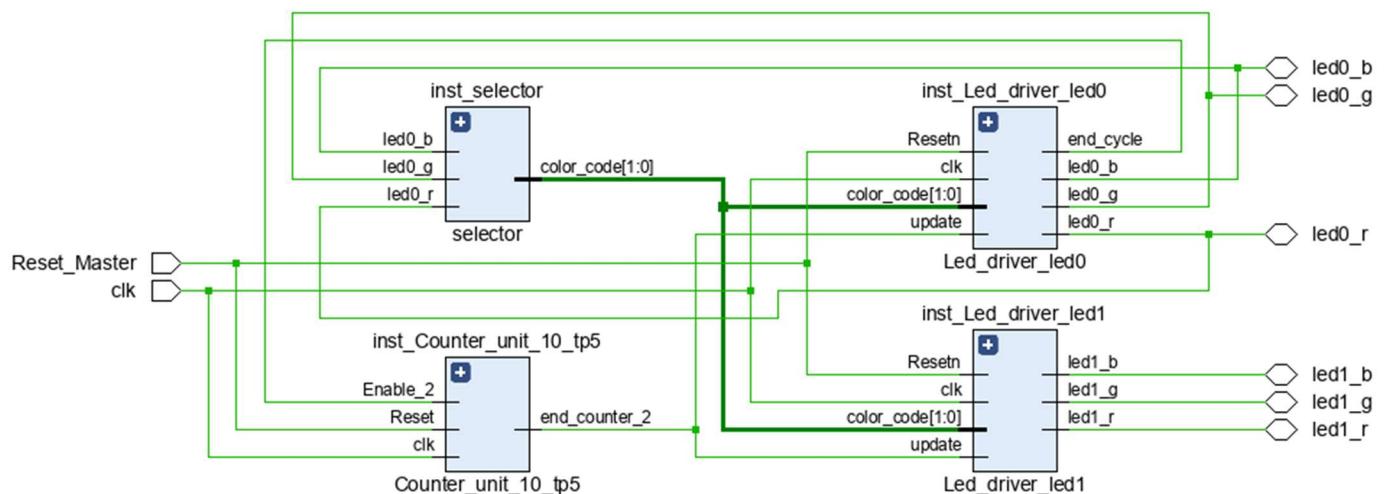


Maintenant je crée un MASTER.VHD qui regroupe mes composants avec une même horloge :

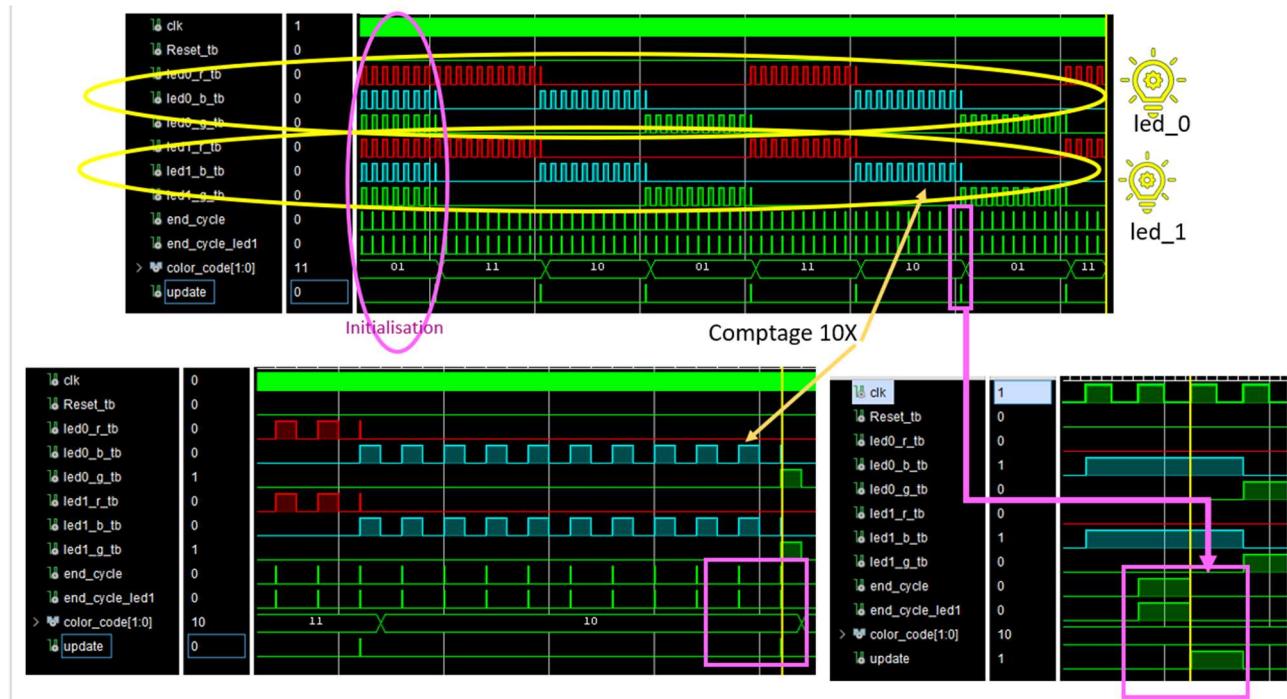
→ Le schéma de l'ensemble de gestion des deux leds (led\_0 et led\_1)



Le schéma RTL équivalent de mon MASTER :



Simulation fonctionnelle du MASTER.vhd qui regroupe mes composants :



→ on a bien les modules led0 et led1 qui réceptionnent correctement le signal update.

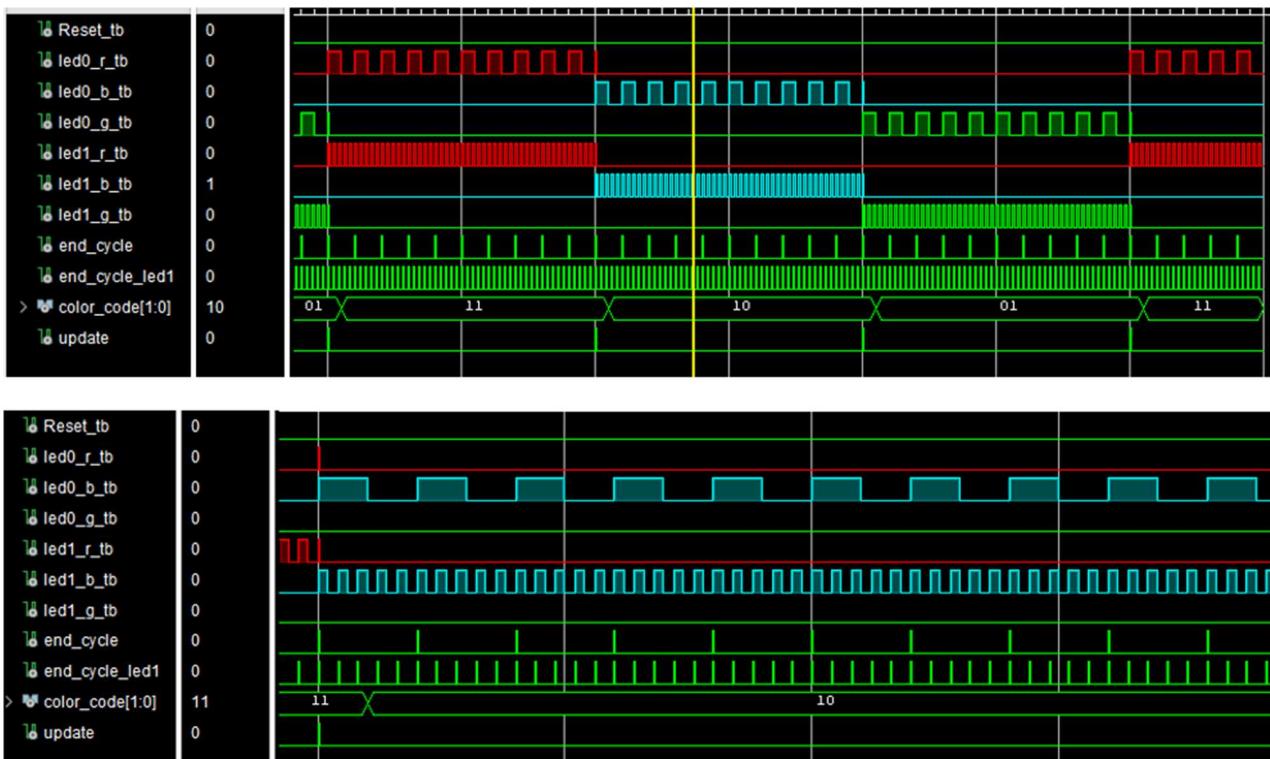
**Question 4.** Modifiez votre design pour gérer deux signaux d'horloge différents. La première horloge, `clkA`, est associé à la logique en dehors des modules `LED_driver` et au module `LED_driver` de la `LED0`. La deuxième horloge, `clkB`, est associé au module `LED_driver` de la `LED1`. → Le changement de couleur des deux LEDs à lieu lorsque la `LED0` à clignoté 10 fois. + **Question 5**

→ je commence à mettre à jour mes clock dans le master :

**1 cas :**

- Attribuer `clkA` à `Led_driver_led0` = 50Mz
- Attribuer `clkB` à `Led_driver_led1` = 250Mz

Après simulation on obtient :



Pour le cas led0 avec clkA = 50MHz (TA = 20ns)

Pour le cas led1 avec clkB = 250MHz (TB = 4ns)

La led0 clignote 10 coup à une couleur donnée tant que la led1 clignote 50x plus à une mm couleur

### 2eme cas :

- Attribuer clkA à Led\_driver\_led0 = 250Mhz
- Attribuer clkB à Led\_driver\_led1 = 50Mhz
- Comme j'ai un seul compteur, ce dernier est lié à l'horloge clkA (Counter\_unit\_10\_tp5)

Calcul des périodes pour simulation TB :

- Pour une fréquence de 50MHz nous avons une période = 20ns ( $1/F = TA$ )
- et pour une fréquence = 250MHz nous avons une période = 4ns. ( $1/F = TB$ )

```

begin
) inst_Led_driver_led0 : entity Led_driver_led0 port map (
    clk => clkA,
    Resetn => Reset_Master,
    color_code => color_code,
    update => update,
    end_cycle => end_cycle ,
    led0_r => led0_r,
    led0_b => led0_b,
    led0_g => led0_g);

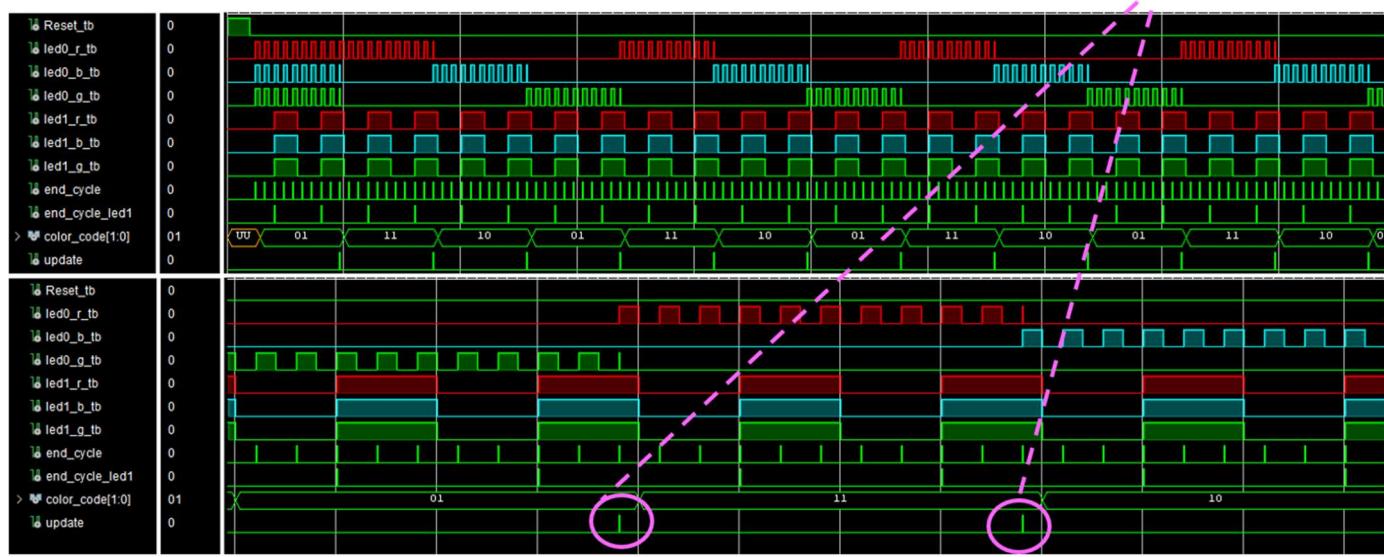
) inst_Led_driver_led1 : entity Led_driver_led1 port map (
    clk => clkB,
    Resetn => Reset_Master,
    color_code => color_code,
    update => update,
    end_cycle_led1 => end_cycle_led1,
    led1_r => led1_r,
    led1_b => led1_b,
    led1_g => led1_g);
) inst_Counter_unit_10_tp5 : entity Counter_unit_10_tp5 port map (
    clk => clkA,
    Reset => Reset_Master,
    Enable_2 => end_cycle,
    end_counter_2 => update);
) inst_selector : entity selector port map (
    led0_r => led0_r,
    led0_b => led0_b,
    led0_g => led0_g,
    color_code => color_code);
)

```

Pour le cas led0 avec clkA = 250MHz (TA = 4ns)

La led1 avec clk B reste à l'état initiale et ne voie pas le signal update

Pour le cas led1 avec clkB = 50MHz (TB = 20ns)



**Question 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 ?



D'après ces différentes captures de ma simulation nous constatons que la clkB [ à 50Mz et à une période de 20ns ] ne voie pas le signal « update » quand il se pointe.

Rappelons que le signale « update » est celui de « end\_counter » à la sortie du compteur 10X.

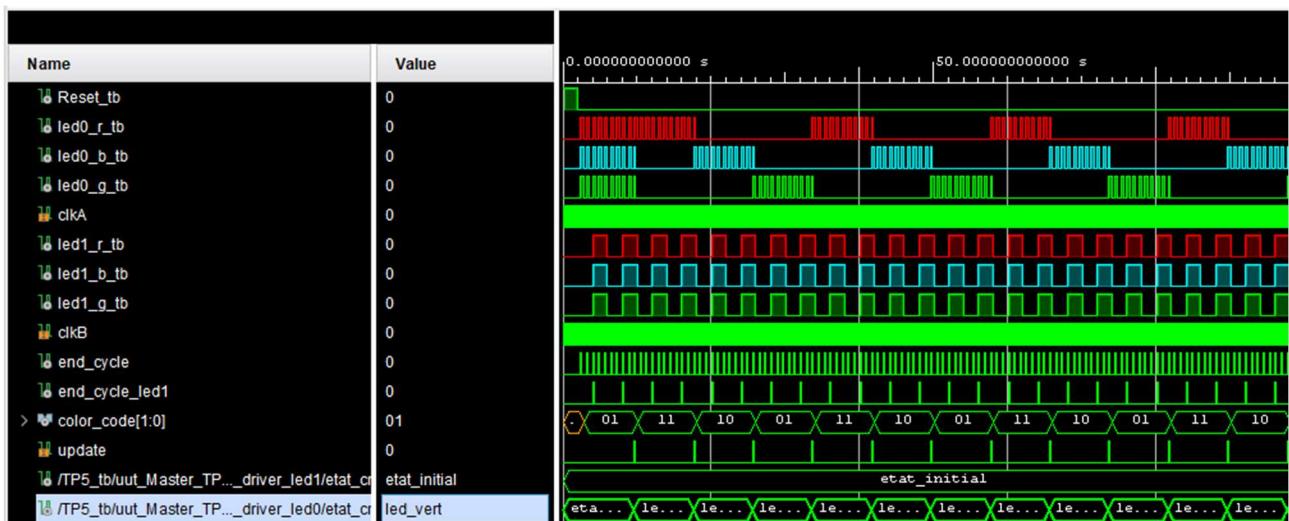
→ mon compteur 10X est relié avec la clkA avec la freq 250Mz

→ le signal « update » est en phase avec les fronts montants de la clk A qui est 5 fois plus rapide que la clkB

→ on remarque que ma première led0 clignote à une fréquence clk A donc chaque 10X de comptage nous aurons un changement de couleur de led (rouge puis bleu puis vert)

→ ce même signal « update » qui ordonne le changement de couleur de led0 est reçu par la led1. Sauf que à chaque passage, ce signal « update » n'est pas vu par ni un front montant de la clkB ni un front descendant → ce qui explique le **non changement de l'état initiale de ma led1**

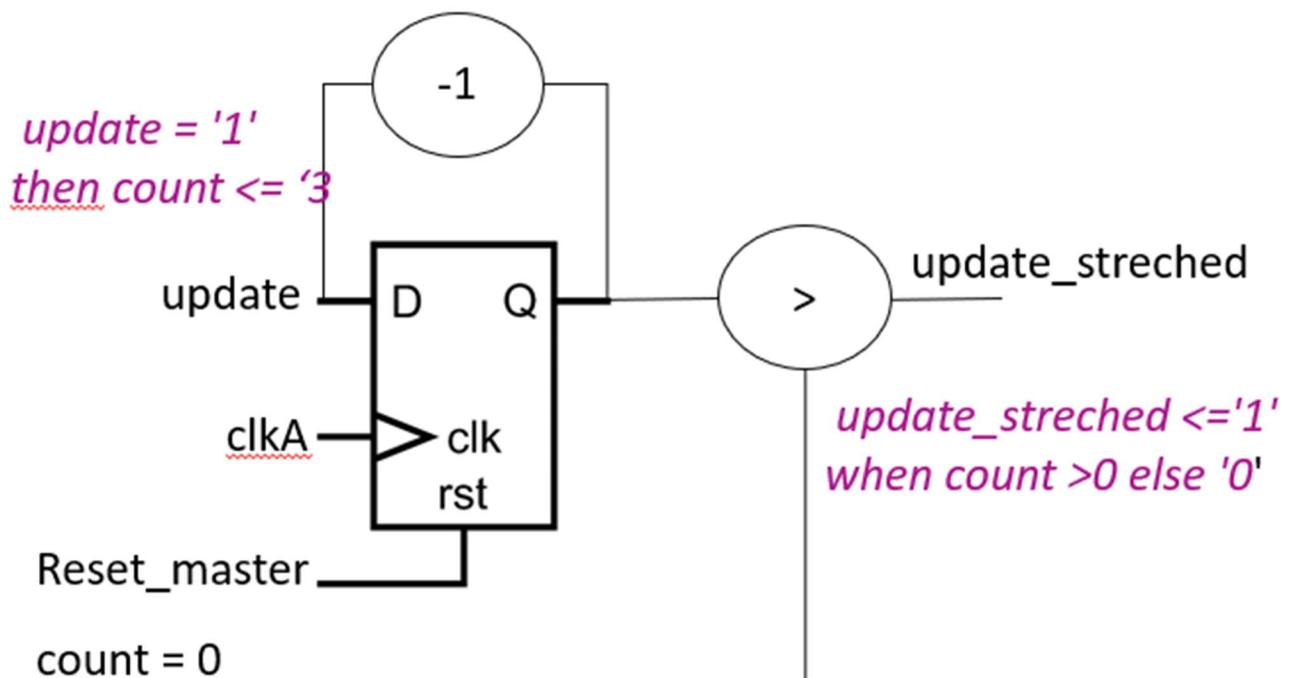
\*Autrement expliqué :après un certain nombres de fronts montant de ma clkA on a 1 front montant de clkB. Le signal « update » lui arrive , quand la clkB =0 et ne dure que, seulement, pendant un coup d'horloge de fréquence clkA → donc clkB passe de 0 à 1 et de 1 à 0 sans croiser le signal « update ».



**Question 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/>. Et

Pour corriger le problème de non changement d'état de la led1 : je crée une impulsion 'Update' plus longue qui dure (5 puis 4 puis 3) cycles d'horloges au lieu d'1 seul cycle => cette nouvelle pulse on va la nommer "update\_stretched". Pour développer cette solution RTL on aura besoin de rassembler la logique combinatoire et logique synchrone . Nous aurons besoin

1. d'un registre D-flip-flop
2. et un compteur qui décrémente
3. on va utiliser un comparateur à la sortie. Si les deux valeurs d'entrées sont égales, alors la sortie est à 1 sinon elle est à 0



Le code associé :

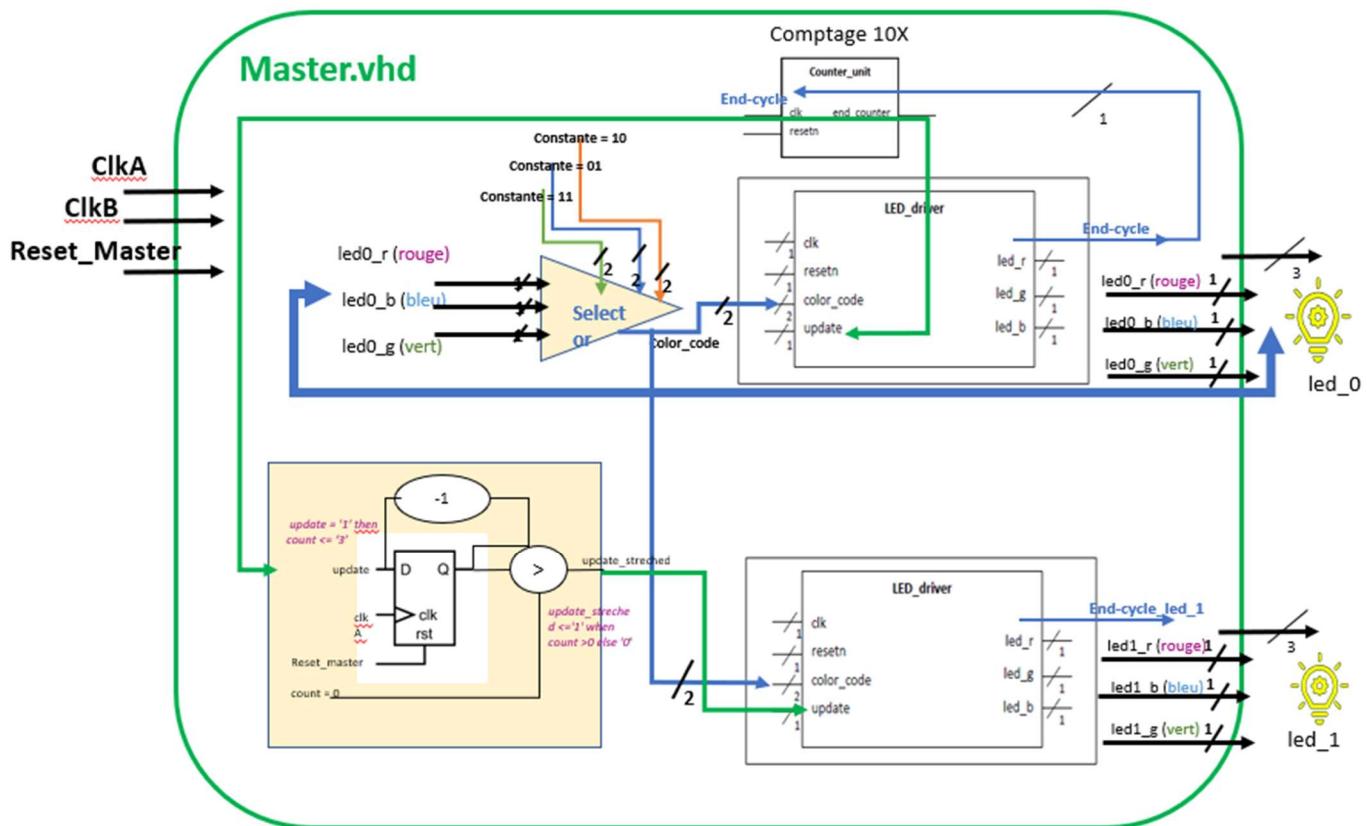
```
-- pour des besoins de sync_clkA_to_clkB (rapide à lente) :
-- je crée une impulsion 'Update' plus longue qui dure x cycles d'horloges au lieu d'un seul cycle
-- cette nouvelle pulse on va la nommer "update_stretched"
process (clkA,Reset_master)
begin --ici j'ai besoin d'un compteur "Count" qui compte jusqu'au 4
  if rising_edge(clkA) then
    --if Reset_master ='1' then count <= 4;
    if update = '1' then count <= 3;
    elsif count > 0 then count <= count - 1; --else count <= 4;
    end if;
    end if;
  --end if;
end process ;

-- je crée une impulsion longue qui s'appelle "update_stretched":
update_stretched <='1' when count >0 else '0';
```

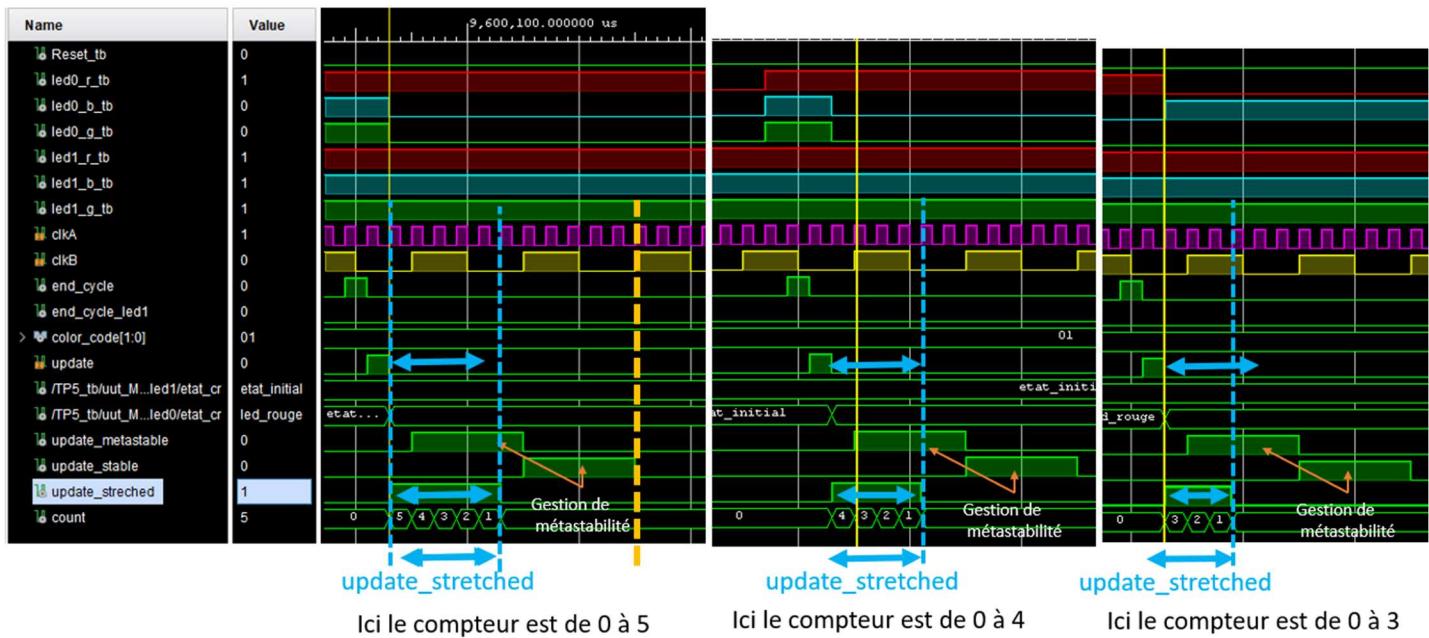
D'après le lien donnée dans la question 7 il y a une deuxième partie sur **la métastabilité** de la clkB avec la freq la plus lente à régler en plus de ce qu'on vient d'implémenter précédemment :

```
--maintenant je vais enlever la métastabilité de la fréquence lente CLKB = 50Mz:
process (clkB,Reset_master )
begin
  if rising_edge(clkB) then
    if Reset_master ='1' then
      update_metastable <= '0';
      update_stable <= '0';
    else
      update_metastable <= update_streched ;
      update_stable <= update_metastable;
    end if;
  end if;
end process ;
```

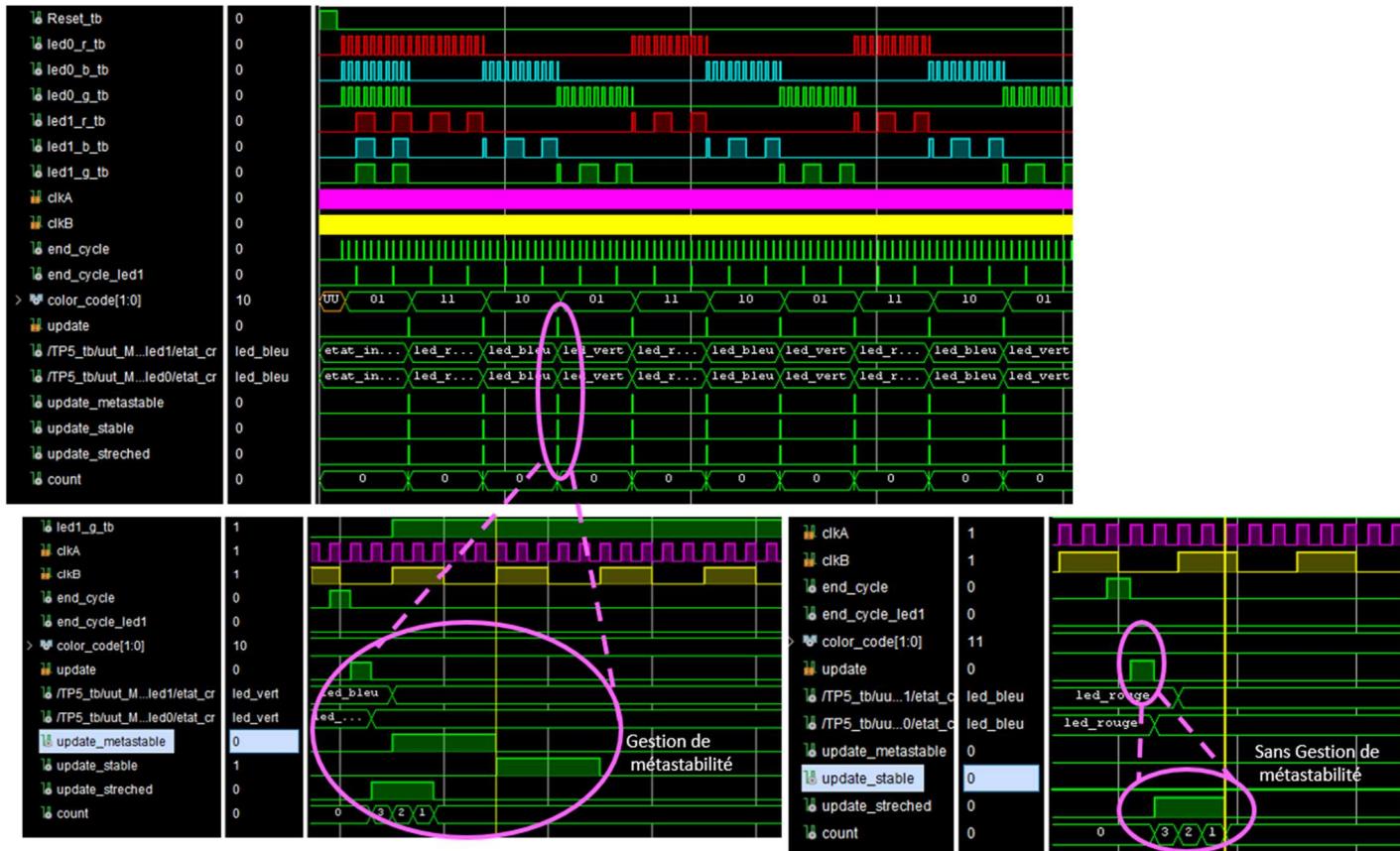
Cette partie de gestion de métastabilité des deux freq CLKA CLKB, ici dans notre cas, elle n'est pas très mise en valeur car les deux clkA et clkB sont en phase seulement une est 5 fois plus rapide que la deuxième. Voir ci-après simulation : dans question 8.



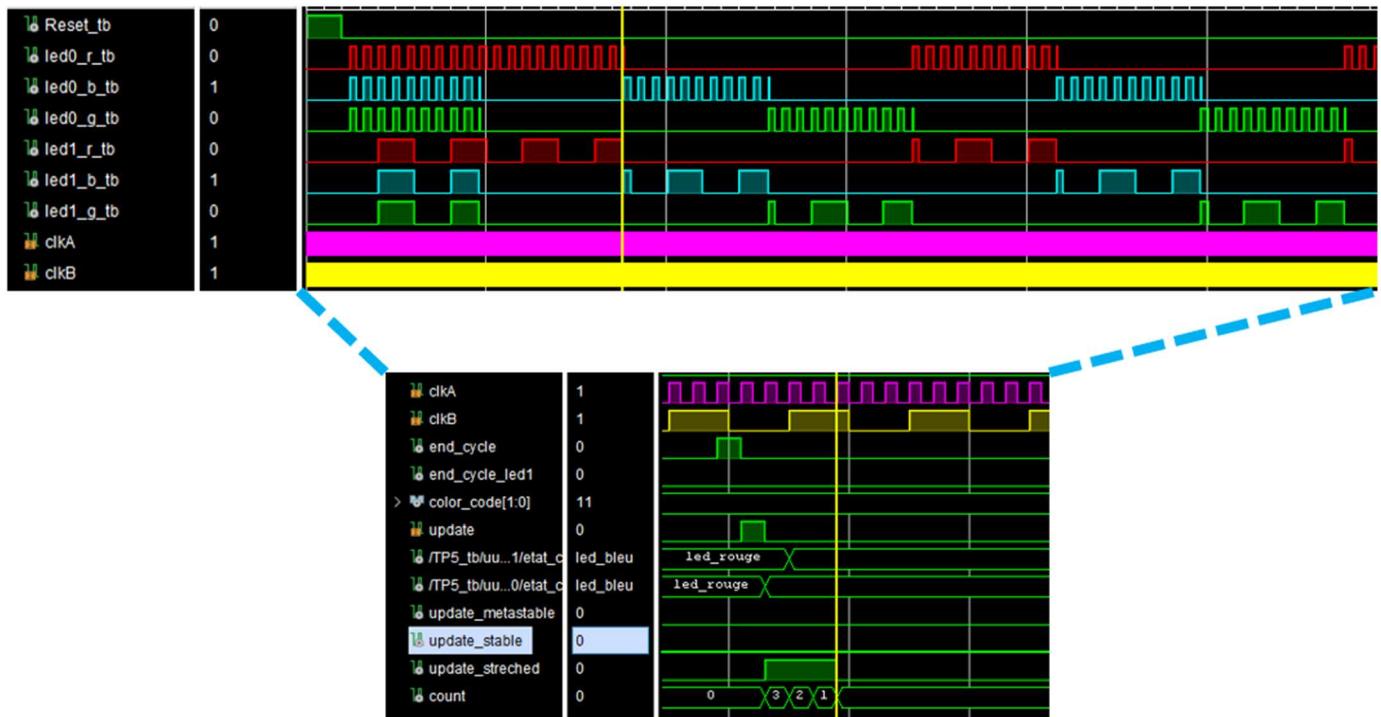
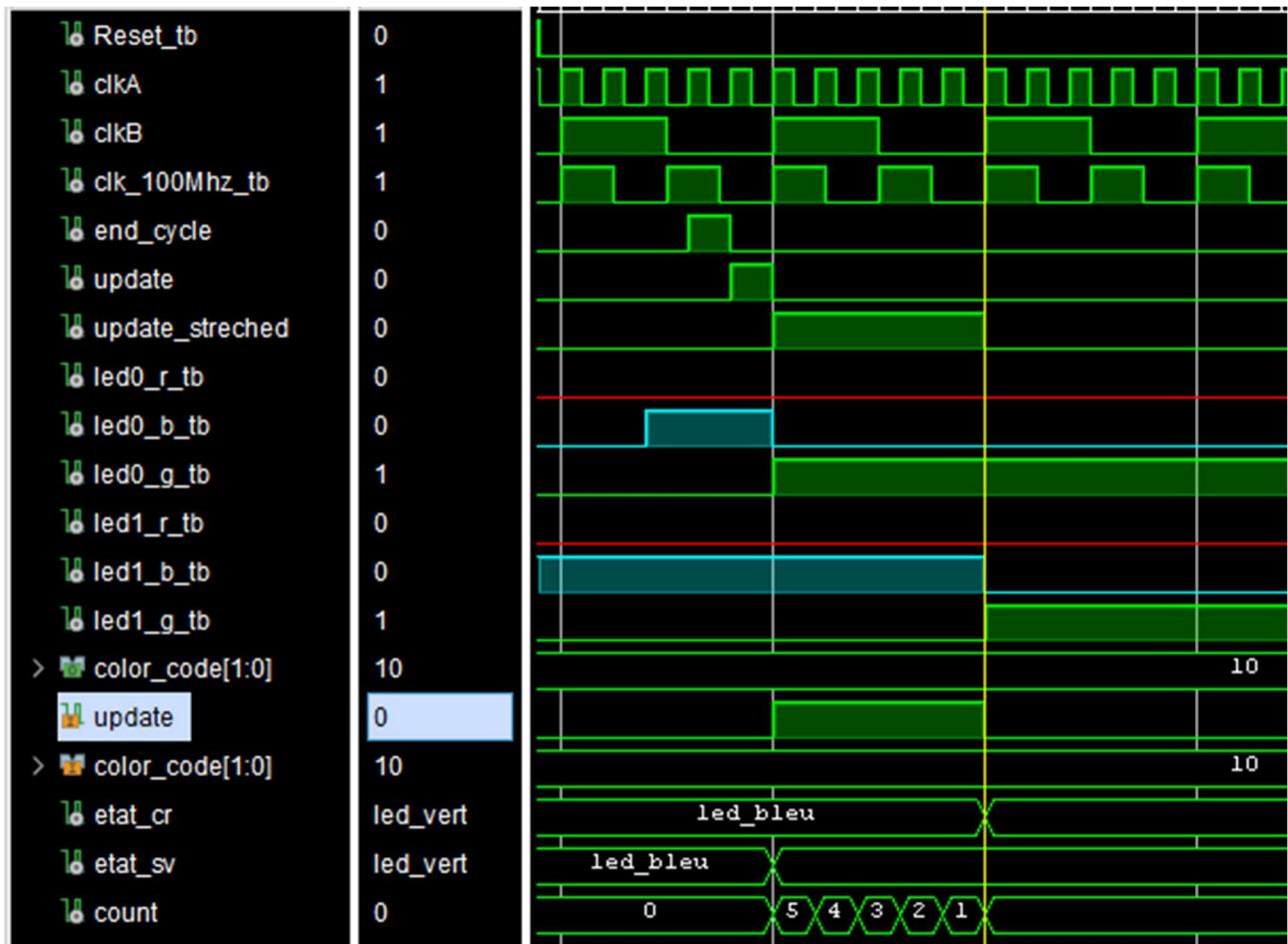
**Question 8.** Mettez en place votre solution et testez-la en simulation. Si votre résultat de simulation n'est toujours pas valide, proposez une autre solution.



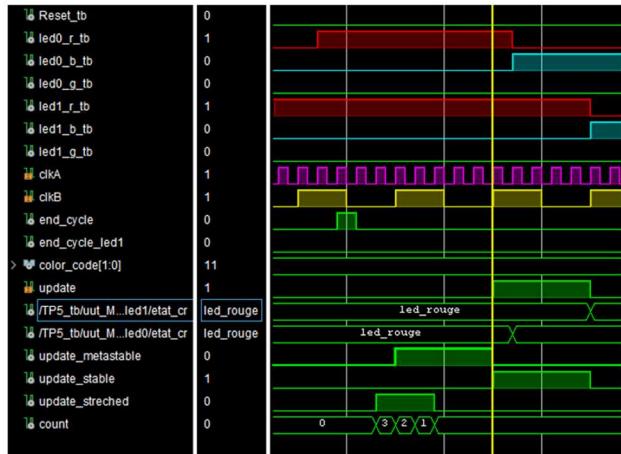
Avec et sans gestion de métastabilité :



Une autre remarque pour que les deux leds clignotent bien, il faut un stretch de signale update jusqu'au front montant de mon impulsion du clk B → sinon on aura, seulement, le premier état de la machine à état et la led 1 ne clignotera pas : restera figée .



Avec la gestion de métastabilité :

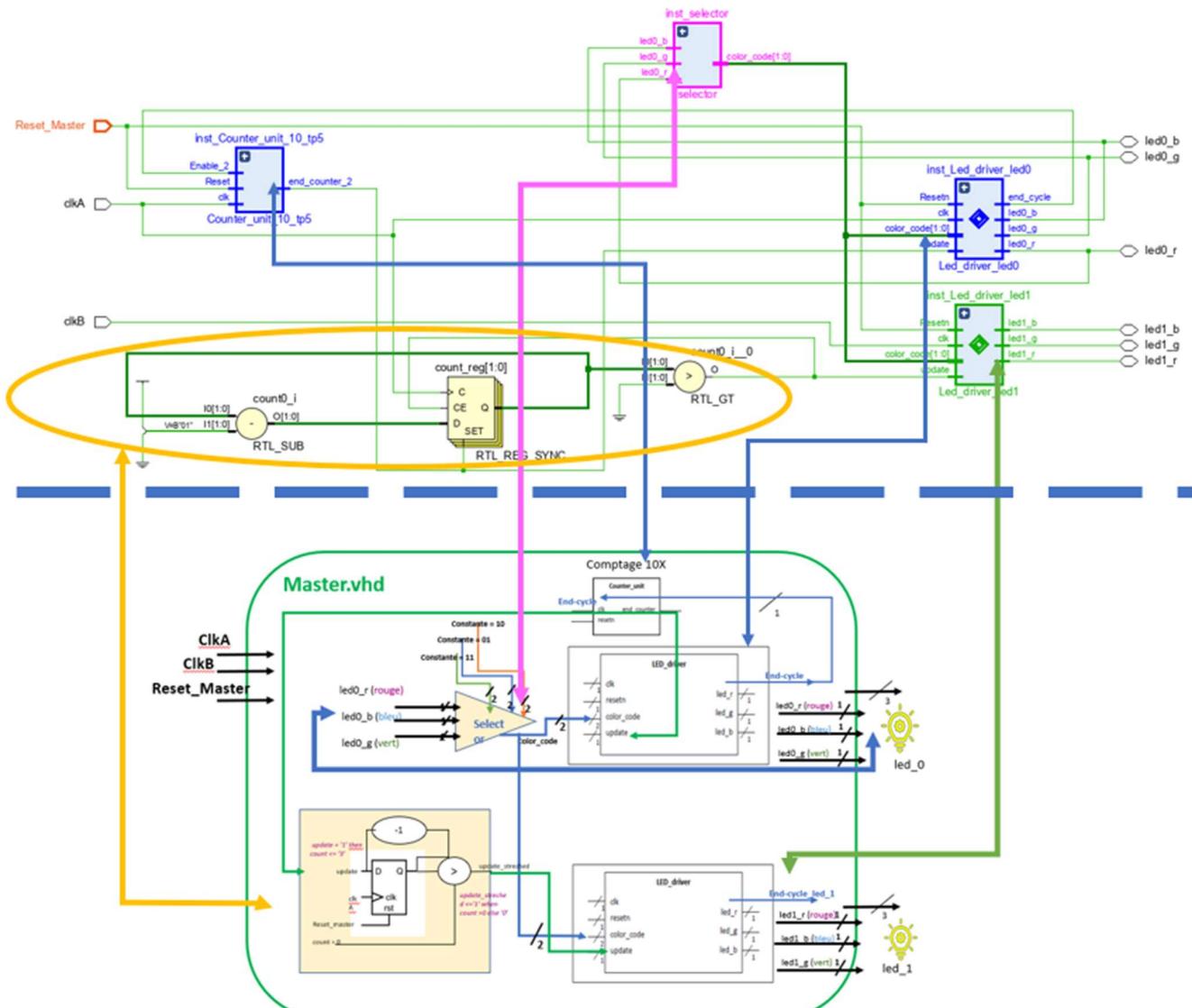


```

begin
inst_Led_driver_led0 : entity Led_driver_led0 port map (
    clk => clkA,
    Resetn => Reset_Master,
    color_code => color_code,
    --update => update,
    update => update_stable,
    end_cycle => end_cycle ,
    led0_r => led0_r,
    led0_b => led0_b,
    led0_g => led0_g);
inst_Led_driver_led1 : entity Led_driver_led1 port map (
    clk => clkB,
    Resetn => Reset_Master,
    color_code => color_code,
    --update => update_streched,
    update => update_stable,
    end_cycle_led1 => end_cycle_led1,
    led1_r => led1_r,
    led1_b => led1_b,
    led1_g => led1_g);
inst_Counter_unit_10_tp5 : entity Counter_unit_10_tp5 port map (
    clk => clkA,
    Reset => Reset_Master,
    Enable_2 => end_cycle,
    end_counter_2 => update);
inst_selector : entity selector port map (
    led0_r => led0_r,
    led0_b => led0_b,
    led0_g => led0_g,
    color_code => color_code);

```

Le schéma RTL équivalent du nouveau MASTER avec la synchronisation :



**Question 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.

Component Name: clk\_wiz\_0

Clocking Options   Output Clocks   Port Renaming   MMClock Settings   Summary

**Primary Input Clock Attributes**

Input Clock Frequency (MHz)	100.000
Clock Source	Single-ended_clock_capable_pin
Jitter	0.010

**Clocking Primitive Attributes**

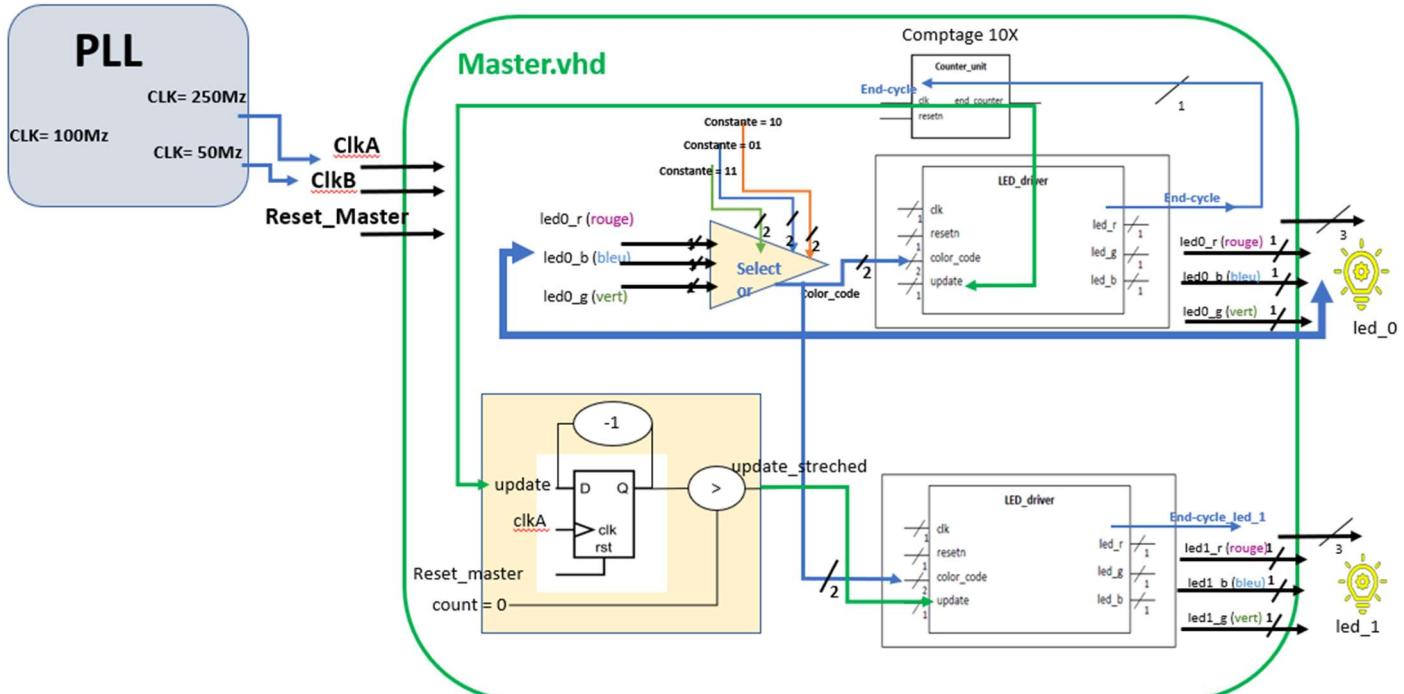
Primitive Instantiated : MMClock

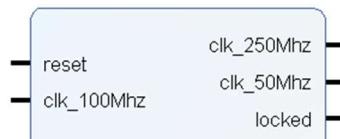
Divide Counter : 1

Mult Counter : 10.000

Clock Phase Shift : None

Clock Wiz O/p Pins	Source	Divider Value	Tspread (ps)	Pk-to-Pk Jitter (ps)	Phase Error (ps)
clk_250Mhz	MMCM CLKOUT0	4.000	OFF	110.209	98.575
clk_50Mhz	MMCM CLKOUT1	20	OFF	151.636	98.575
clk_out3	OFF	OFF	OFF	OFF	OFF
clk_out4	OFF	OFF	OFF	OFF	OFF
clk_out5	OFF	OFF	OFF	OFF	OFF
clk_out6	OFF	OFF	OFF	OFF	OFF
clk_out7	OFF	OFF	OFF	OFF	OFF





IP PLL : clk\_wiz\_0

J'appelle mon IP :

```

component clk_wiz_0  is
port ( clk_100Mhz :in STD_LOGIC;
      reset : in STD_LOGIC ;
      locked : out STD_LOGIC ;
      clk_250Mhz: out STD_LOGIC ;
      clk_50Mhz : out STD_LOGIC );
end component;

```

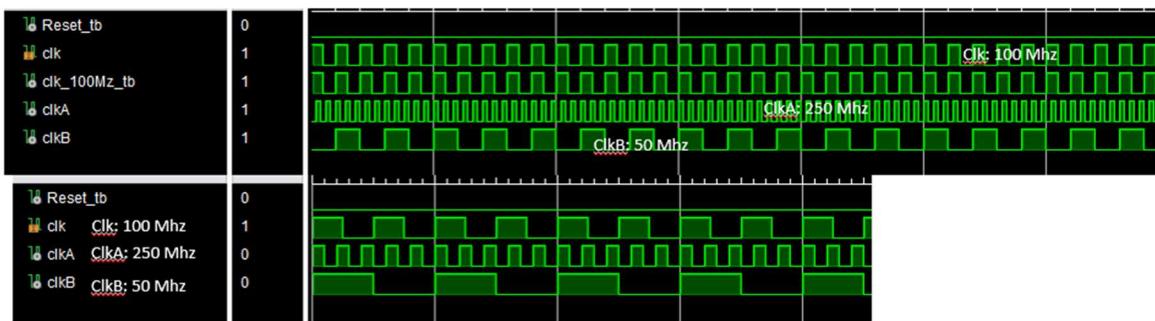
Ensuite j'instancie mon composant avec le MASTER\_TP5 :

```

inst_clk_wiz_0 : clk_wiz_0 port map (
    clk_100Mhz => clk,
    reset => Reset_Master,
    clk_250Mhz => clkA,
    clk_50Mhz => clkB);

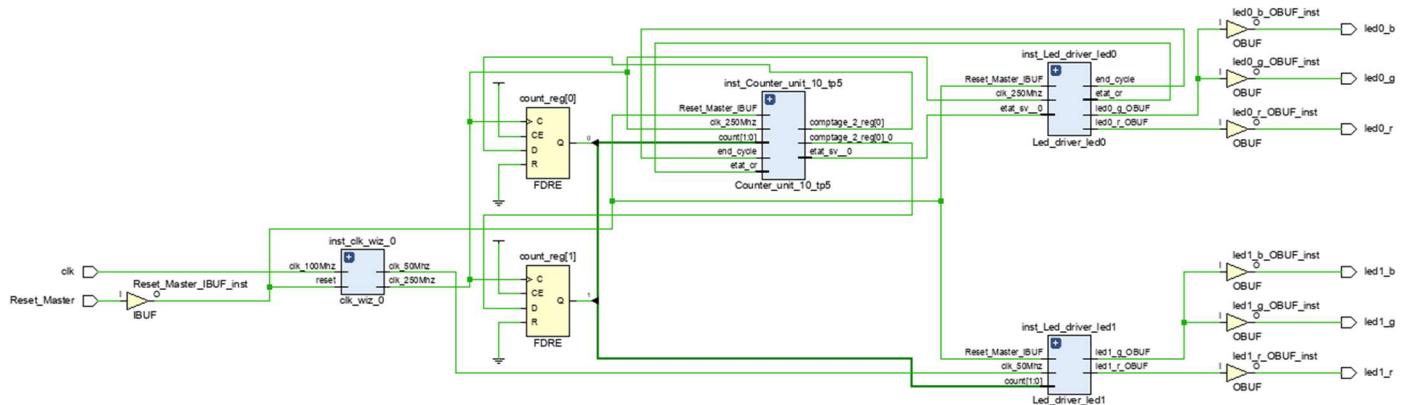
```

Simulation avec ce IP on aura :

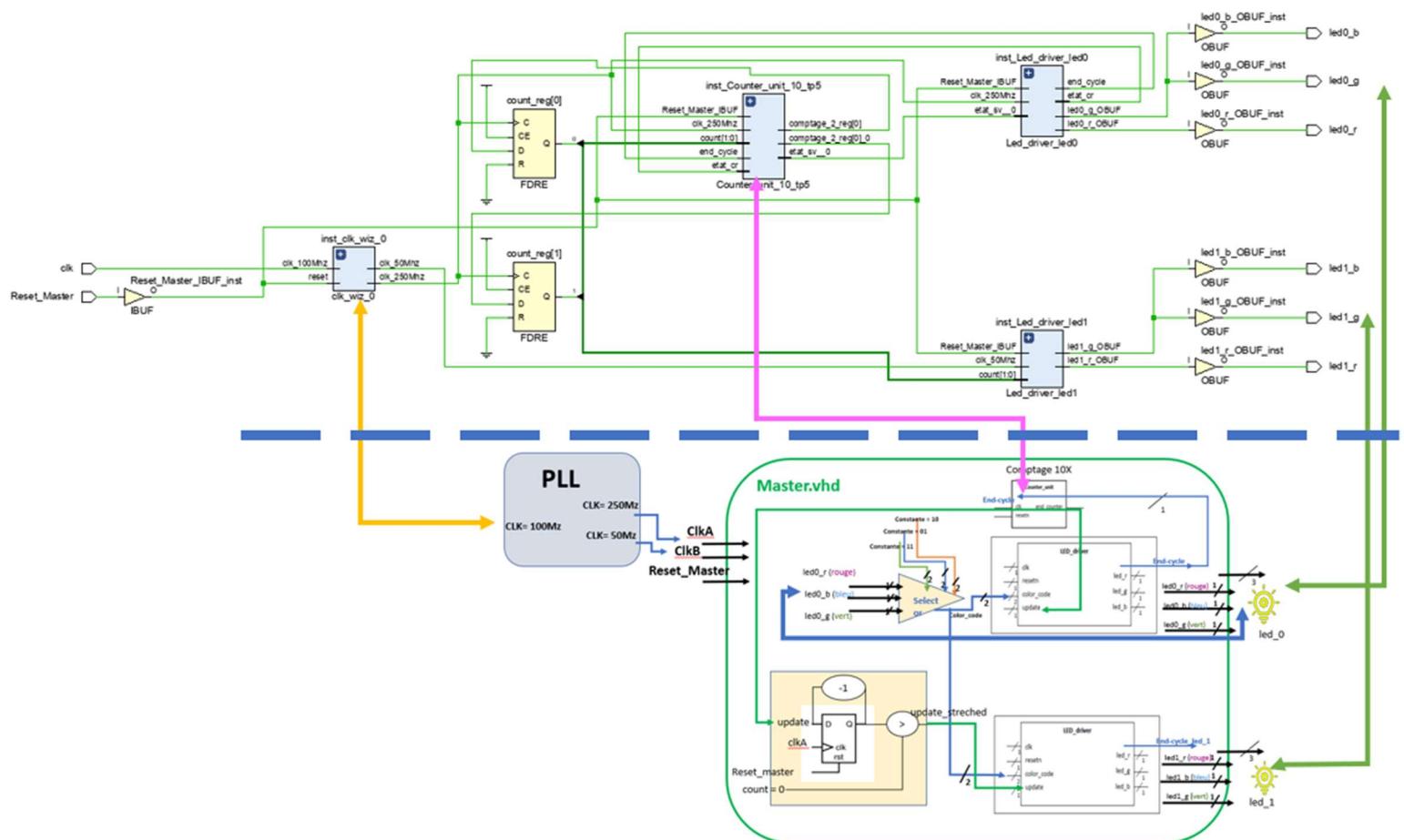


**Question 10.** Effectuez la synthèse et placer des sondes (ILA) sur les signaux pertinents pour vérifier que le changement de domaine d'horloge s'est correctement passé.

## 1.2 Schématique :



Ma schématique après implémentation en comparaison avec mon architecture :



**Question11.** Etudiez le rapport de synthèse.

### 1.3 Rapport de synthèse

Start RTL Component Statistics			Report Cell Usage:					
<hr/>								
Detailed RTL Component Info :								
+---Adders :			1	clk_wiz_0_bbox	1			
2 Input	2 Bit	Adders := 1	2	CARRY4	16			
+---Registers :			3	LUT1	3			
3 Bit	Registers := 2		4	LUT2	3			
2 Bit	Registers := 2		5	LUT3	1			
1 Bit	Registers := 6		6	LUT4	4			
+---Muxes :			7	LUT6	15			
6 Input	3 Bit	Muxes := 2	8	FDCE	6			
5 Input	3 Bit	Muxes := 2	9	FDRE	66			
<hr/>								
Finished RTL Component Statistics								
			10	IBUF	1			
			11	OBUF	6			
			+-----+	+-----+	+-----+			

### 1.4 Rapport de timing

Design Timing Summary							
WNS(ns)	TNS(ns)	TNS Failing Endpoints	TNS Total Endpoints	WHS(ns)	THS(ns)	THS Failing Endpoints	
0.512	0.000	0	144	0.186	0.000	0	
<hr/>							
From Clock: clk_250Mhz_clk_wiz_0							
To Clock: clk_250Mhz_clk_wiz_0							
<hr/>							
Setup :	0	Failing Endpoints, Worst Slack		0.512ns,	Total Violation	0.000ns	
Hold :	0	Failing Endpoints, Worst Slack		0.186ns,	Total Violation	0.000ns	
PW :	0	Failing Endpoints, Worst Slack		1.500ns,	Total Violation	0.000ns	
<hr/>							

Et le chemin critique :

```

Max Delay Paths

-----
Slack (MET) : 0.512ns (required time - arrival time)
  Source: inst_Led_driver_led0/inst_Counter_unit/inst_compteur/comptage_1_reg[6]/C
            (rising edge-triggered cell FDRE clocked by clk_250Mhz_clk_wiz_0 {rise@0}.
  Destination: inst_Led_driver_led0/inst_Counter_unit/inst_compteur/comptage_1_reg[12]/R
                (rising edge-triggered cell FDRE clocked by clk_250Mhz_clk_wiz_0 {rise@0}.

  Path Group: clk_250Mhz_clk_wiz_0
  Path Type: Setup (Max at Slow Process Corner)
  Requirement: 4.000ns (clk_250Mhz_clk_wiz_0 rise@4.000ns - clk_250Mhz_clk_wiz_0 rise@0.00)
  Data Path Delay: 2.873ns (logic 0.704ns (24.507%) route 2.169ns (75.493%))
  Logic Levels: 2 (LUT6=2)
  Clock Path Skew: -0.026ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD): -1.778ns = ( 2.222 - 4.000 )
    Source Clock Delay (SCD): -1.063ns
    Clock Pessimism Removal (CPR): 0.689ns
  Clock Uncertainty: 0.065ns ((TSJ^2 + DJ^2)^1/2) / 2 + PE
    Total System Jitter (TSJ): 0.071ns
    Discrete Jitter (DJ): 0.110ns
    Phase Error (PE): 0.000ns

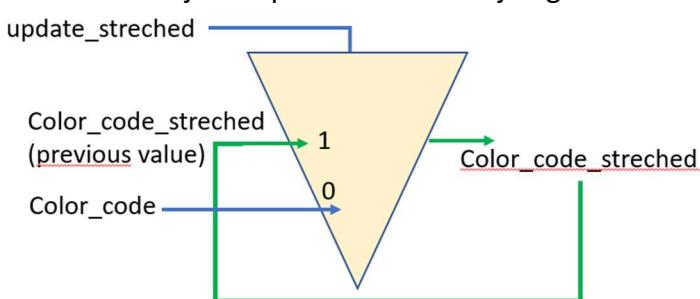
```

### Question12 : Générez le bitstream et vérifiez que vous avez le comportement attendu sur carte.

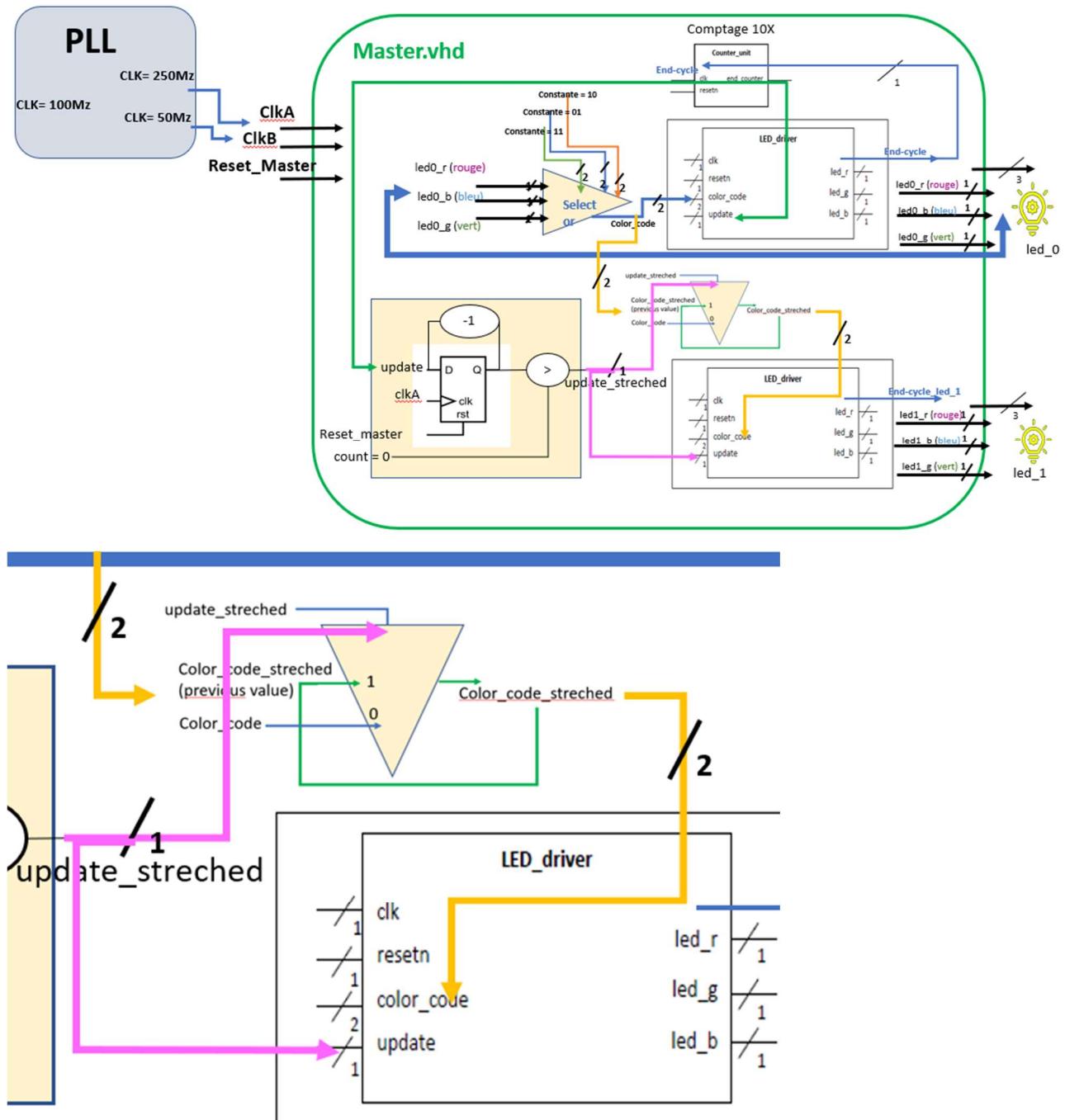
J'ai remarqué un décalage dans la couleur des leds

( il faut led0 soit en rouge et la led 1 soit aussi en rouge ainsi de suite)

- ⇒ Le changement de couleur des leds doit être fait au même moment que mon update\_stretched passe (question précédente du Tp : on s'est assuré que « le update » de changement de couleur soit vue par les deux led0 et led1 (en le stretchant))
- ⇒ C'est l'importance de stretch qu'on a mis en place
- ⇒ Après implémentation sur carte je remarque que le changement des couleurs se fait (rouge/ bleu/ vert) dans les deux led sans aucun souci. Sauf que les couleurs ne sont pas les mêmes au même moment : quand j'ai rouge dans led0 je n'ai pas rouge dans la led1 ce qui m'emmène à rajouter un registre qui maintient mon color code du led 1 tant que mon update\_stretched est active, comme ça je m'assure que ma couleur est maintenue dans ma led1 jusqu'au prochain update\_stretched=> et ça fonctionne très bien (voir vidéo).
- ⇒ Quand j'ai « update\_stretched » je fige la valeur de color code LED1



Inséré dans l'architecture on a :



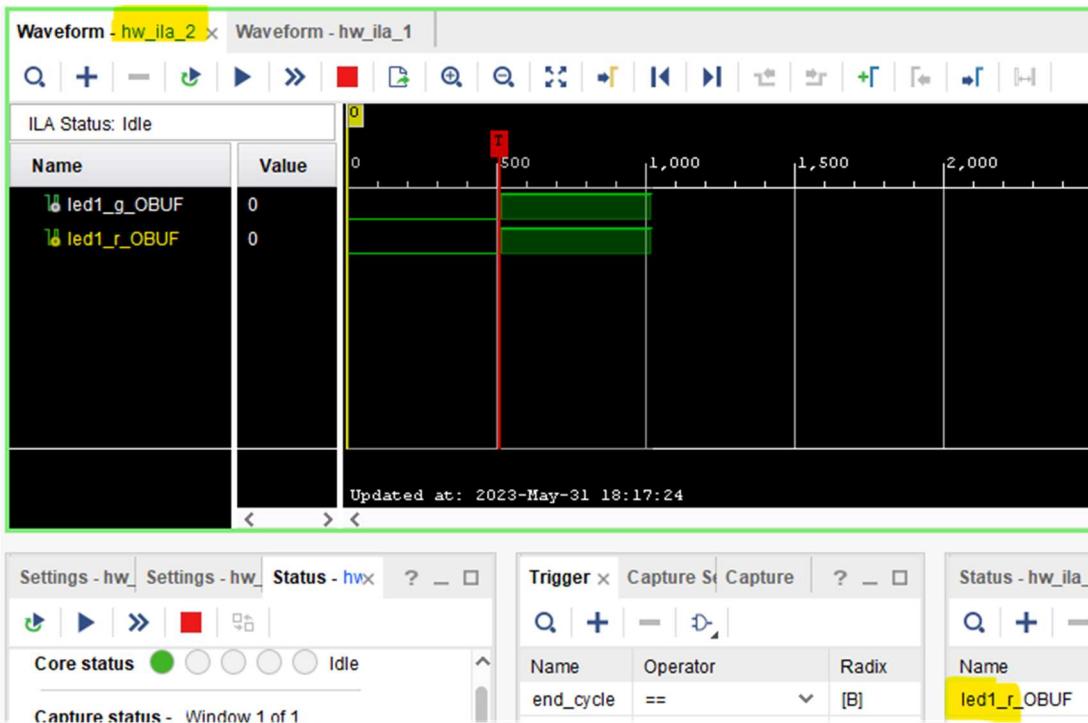
→ J'ai rajouté cette ligne à mon code :

```
process (update_stretched)
begin
if update_stretched = '0' then color_code_stretched <= color_code;
end if;
```

**Simulation ILA :** nous avons 2 fréquences 250MHZ et 50MHZ donc nous aurons 2 ilas :

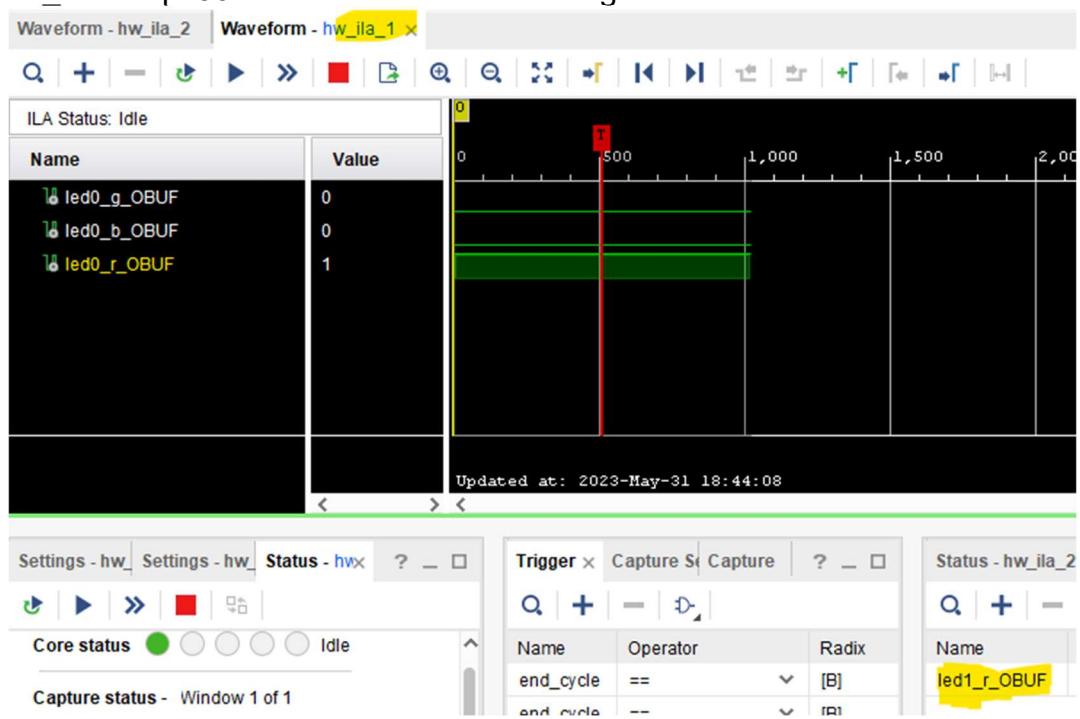
Ilia N°1 :

Ilia\_2 : freq 50MHZ : observation led rouge :



Ilia N°2 :

Ilia\_1 : freq 250MHZ : observation led rouge :



## 1.5 observation sur carte CORAZ7 : voir vidéo aussi

j'ai utilisé bouton 0 pour initialiser : donc mon Reset\_master

## Annexe

