TP05 – Domaine d'horloge

Cédrine Socquet & Samory Diaby

Rendu

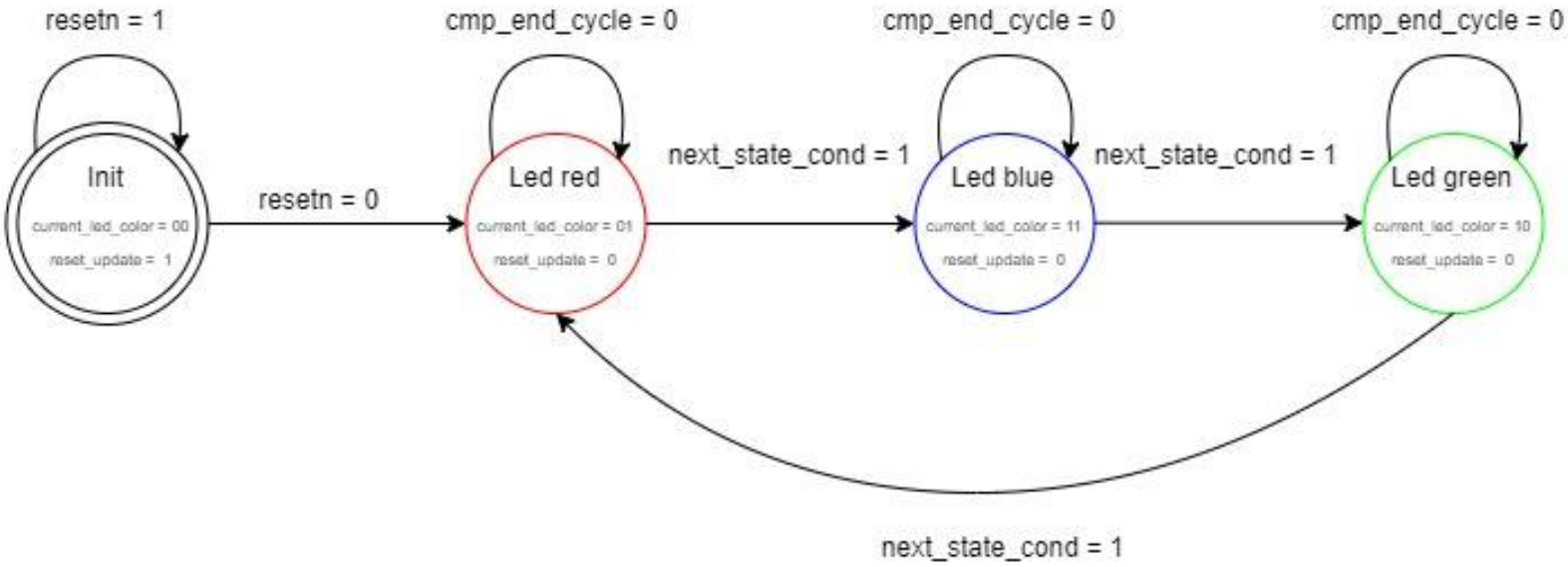
Votre rapport devra contenir :

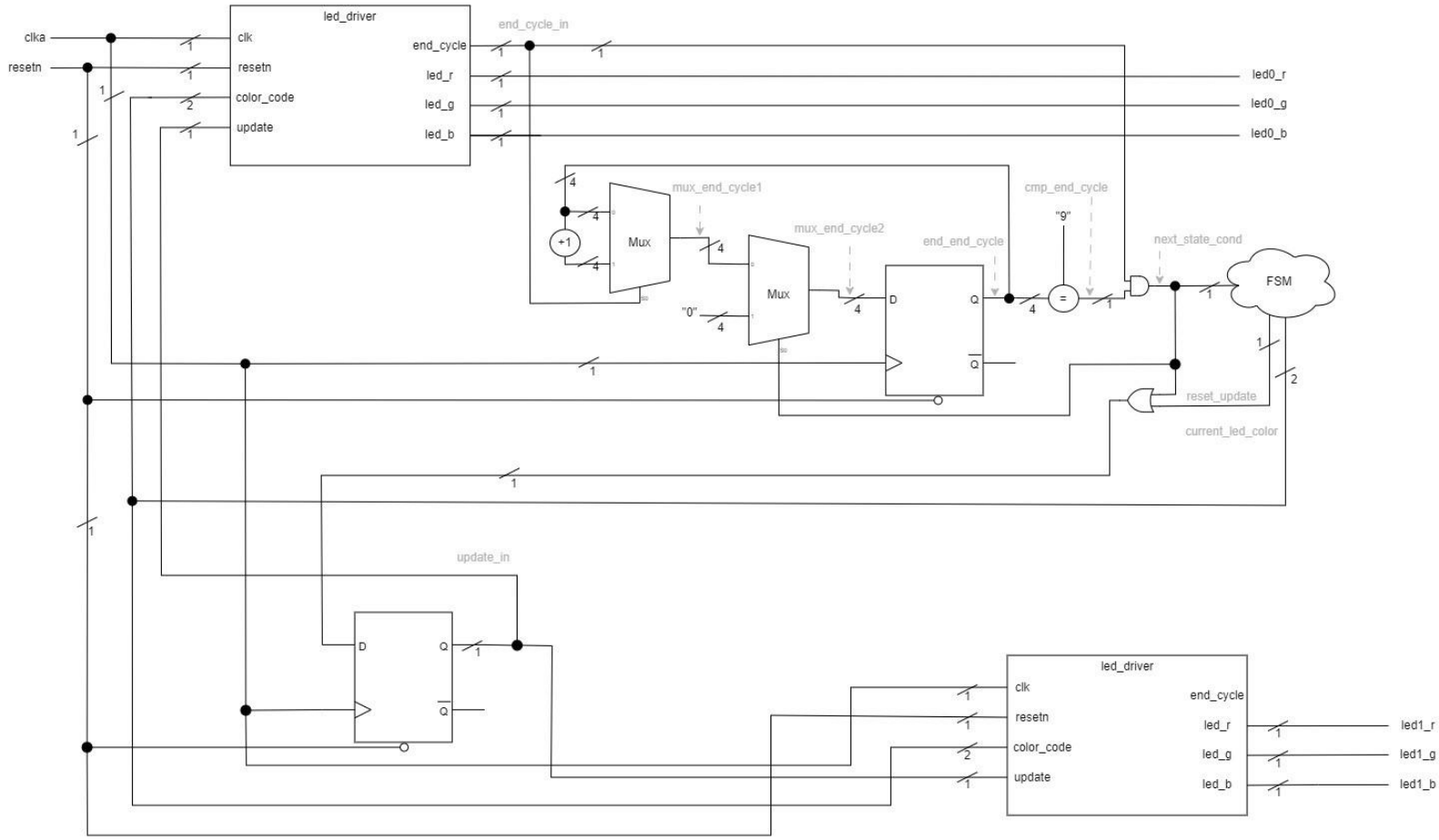
- Vos schéma RTL
- Vos résultats de simulation avec vos chronogrammes commentés
- Vos résultats de synthèse (analyse de vos ressources utilisées)
- Vos résultats de STA (analyse du rapport de timing)
- Vos résultats de mesure ILA
- Une démonstration de votre design

Vous fournirez également vos codes source commentés.

Réponses

1. Voici notre schéma FSM ainsi que notre schéma RTL.

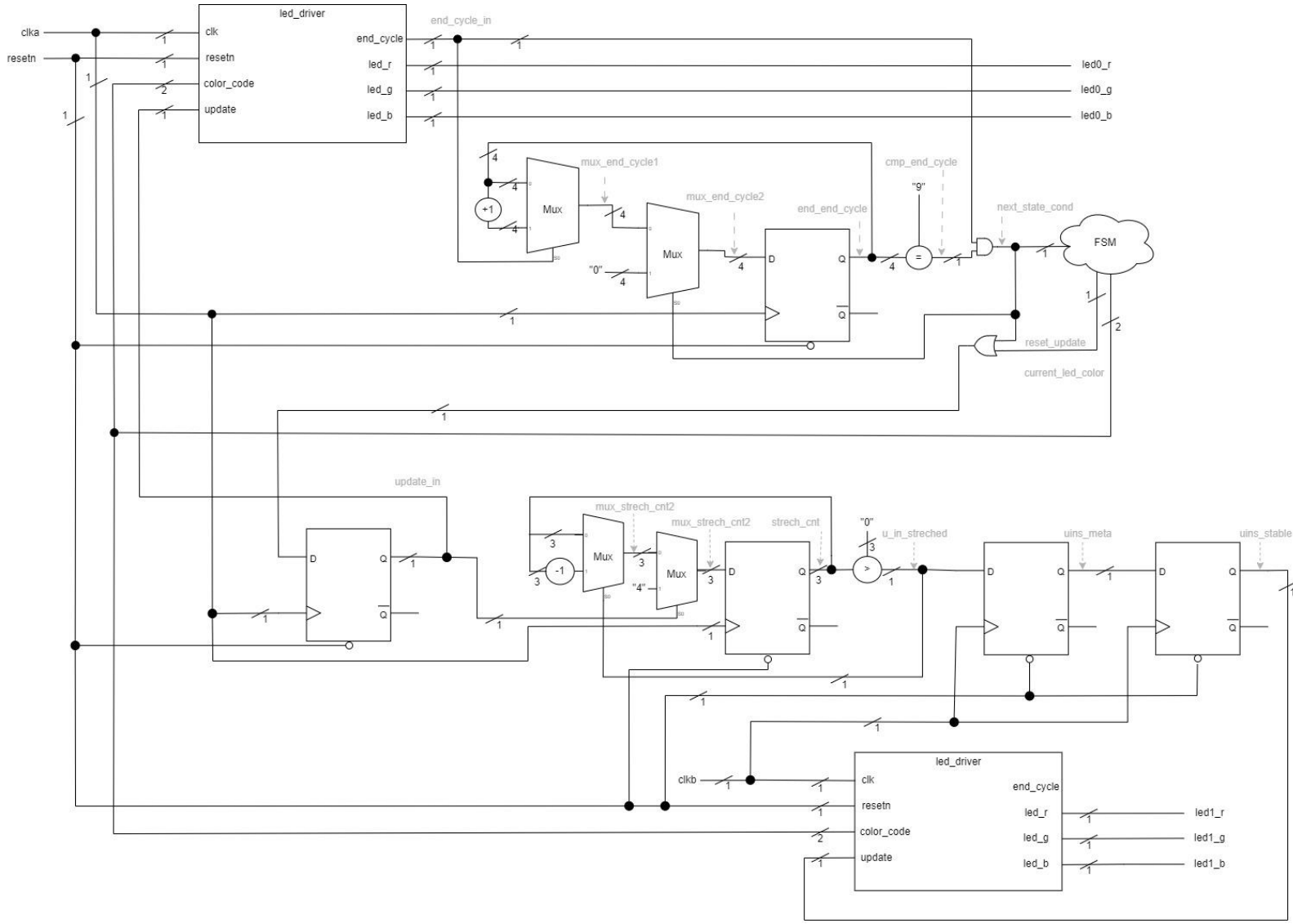




2. Voir le code q1.vhd.

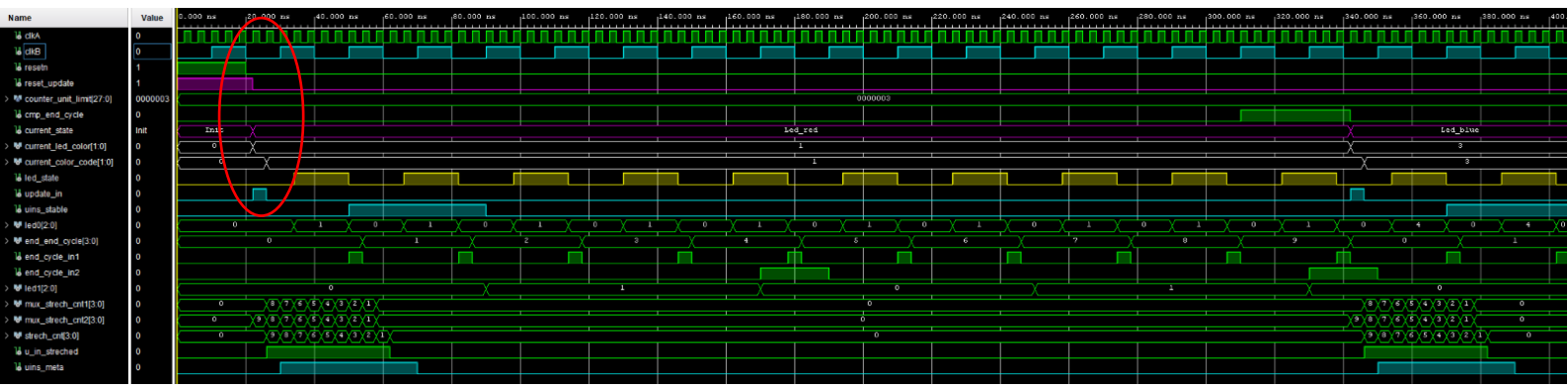
3. Voir le code tb_q1.vhd.

4. Voici ci-dessous notre nouveau design permettant de 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.



5. L'horloge clkA a une fréquence de 50Mz et clkB de 250MHz, voir le testbench tb_q1.vhd.

6. Lors du lancement de la simulation, au niveau des signaux « update » des modules LED_driver, on peut apercevoir que le signal est trop rapide par rapport à la clock clkB.



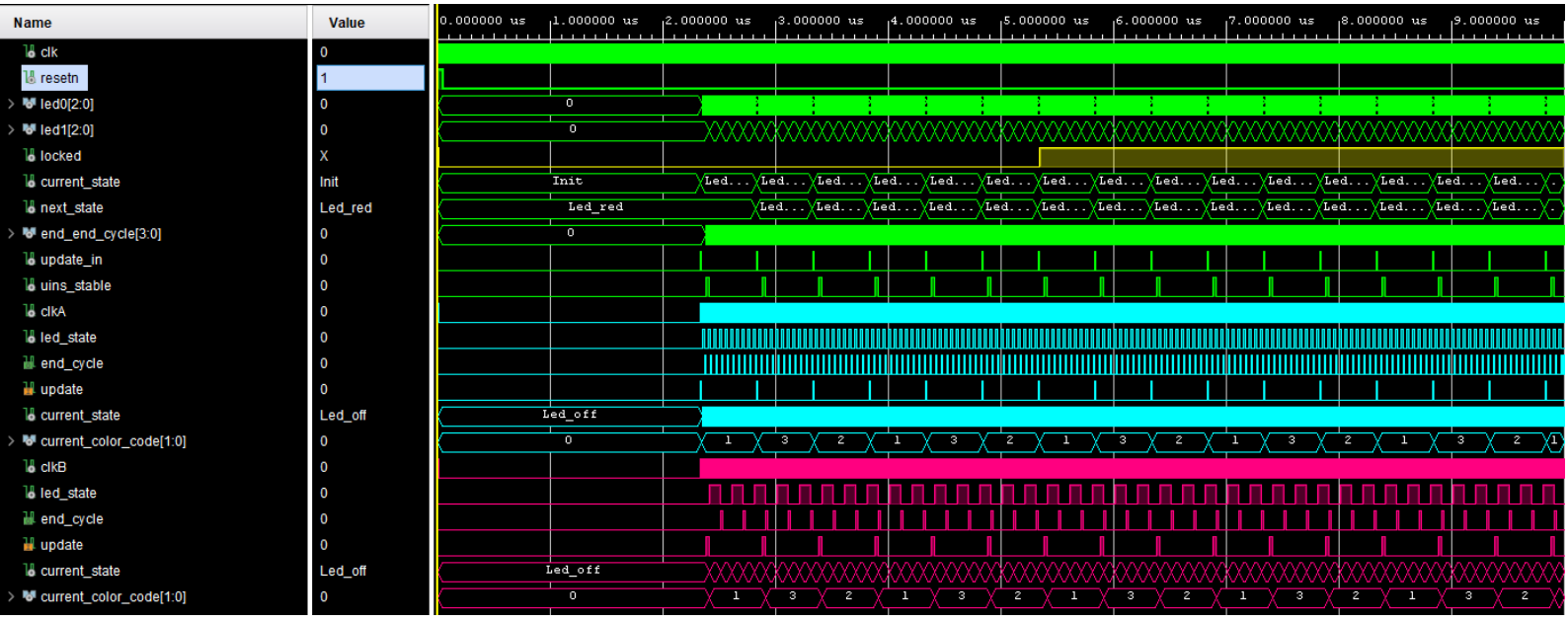
Le module “led_driver”1 associé à la clock clkB ne voit jamais le signal update passer à ‘1’ car le signal “update_in” qui lui est donné en entrée est calé sur 1 coup d’horloge de clkA, ce qui fait que “update_in” vaut dans notre cas toujours ‘0’ lors d’un front montant de clkB empêchant ainsi la mise à jour de “led_driver1”.
L’incidence que cela a sur les LEDs est que celles associées au module “led_driver0” se comportent normalement, tandis que celles associées au module "led_driver1" restent éteintes car le module n’est jamais mis à jour.

7. La solution que nous proposons pour corriger le problème lié au changement de domaine d’horloge est d'allonger le signal “update_in” pour qu'il puisse être lu par la clkB, ce qui nécessite l’ajout de 2 registres. Il est également possible d’utiliser une FIFO ; ce qui permet de choisir entre les deux solutions c'est la quantité de données. Lorsqu’il y a peu de signaux (1 ou 2) l’utilisation d’une FIFO n’est pas nécessaire, en revanche elle l’est lorsque les signaux de donnés sont plus volumineux.

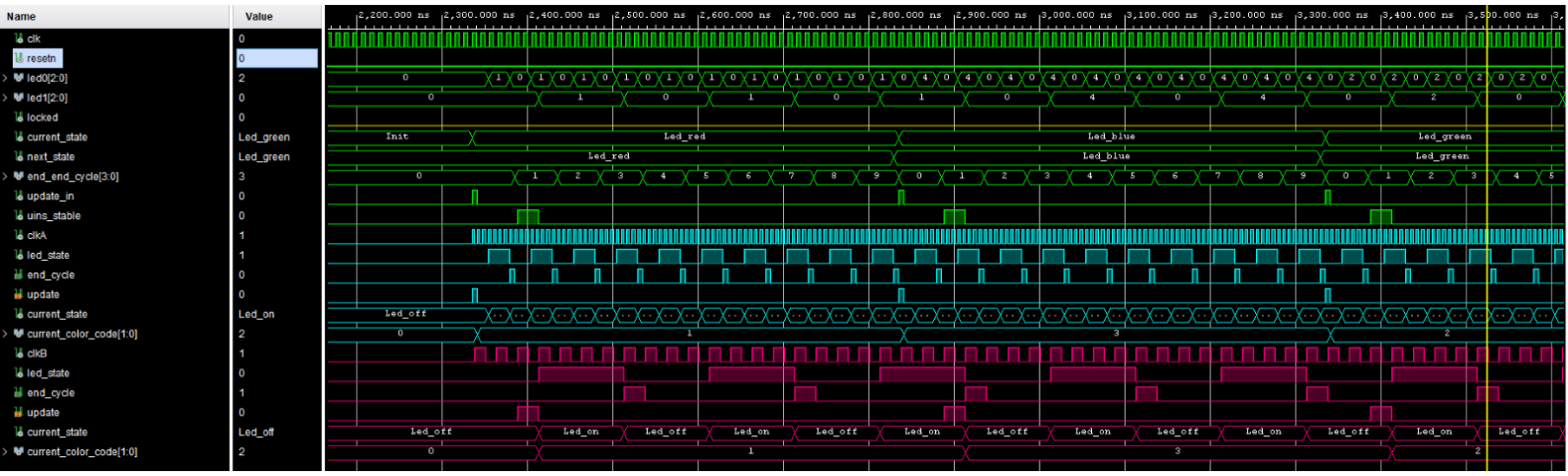
8. Voir les codes q1.vhd ainsi que tb_q1.vhd.

9. Voir les codes Top.vhd, tb_Top.vhd, q1.vhd ainsi que tb_q1.vhd et clk_wiz_0.v.

Voici les résultats de simulations obtenus pour le changement de domaine d’horloge en utilisant une PLL pour générer 2 signaux d’horloges. Sur le chronogramme ci-dessous on observe donc la mise en place de la PLL, qui commence à générer nos signaux d’horloge à partir de 2.4µs.



Sur le chronogramme ci-dessous, on se concentre sur la partie de changement d'état de la FSM, que l'on observe à l'aide du signal "current_state". On remarque que "current_state" change correctement la couleur de sortie des LEDs en fonction de l'état en cours. On remarque également que les 2 LEDs (led0 et led1) clignotent, ce qui signifie que le problème de "Cross Clock Domain" a bien été résolu par l'utilisation du signal "u_in_strech" permettant d'étendre le signal "update_in" jusqu'à ce que le module "led_driver1" puisse être mis à jour lors d'un front montant de "clkB" via le signal "uins_stable".



10. Nous avons effectué la synthèse et placer des sondes (ILA) sur les signaux qui nous semblait pertinents (voir ci-dessous) pour vérifier que le changement de domaine d'horloge s'est correctement passé.
A savoir :
- end_end_cycle
- current_state
- led0
- led1
- update_in
-uins_stable
- u_in_streched

11. Voici l'étude du rapport de synthèse :

On retrouve notre machine à états avec nos 4 états ainsi que l'encodage associé dans cette section du rapport

State	New Encoding	Previous Encoding
init	00	00
led_red	01	01
led_blue	10	10
led_green	11	11

En analysant la section "Detailed RTL Component Info" on retrouve :

- 2 registres 4bits qui correspondent au compteur de "end_cycles" et au compteur "strech_cnt" permettant d'étendre le signal "update_in" pour palier au problème de "Cross Clock Domain".

- 4 registres 2 bits qui sont utilisés par les modules “led_driver” (1 pour stocker le code couleur et 1 pour compter 1 cycle on/off)
- 5 registres 1 bit qui correspondent à “update_in”, “uins_meta”, “uins_sttable”, “reset_update” et “current_led_color”

```
Report Cell Usage:
+------+--------+------+
|      |Cell    |Count |
+------+--------+------+
|1     |clk_wiz |     1|
|2     |CARRY4  |    14|
|3     |LUT1    |     2|
|4     |LUT2    |     2|
|5     |LUT3    |    14|
|6     |LUT4    |    15|
|7     |LUT5    |    60|
|8     |LUT6    |     6|
|9     |FDCE    |    71|
|10    |FDRE    |     5|
|11    |FDSE    |     1|
|12    |IBUF    |     1|
|13    |OBUF    |     6|
+------+--------+------+
```

```
Detailed RTL Component Info :
+---Adders :
      2 Input    4 Bit       Adders := 2
      2 Input    2 Bit       Adders := 2
+---Registers :
                   4 Bit    Registers := 2
                   2 Bit    Registers := 4
                   1 Bit    Registers := 5
+---Muxes :
      2 Input    4 Bit        Muxes := 1
      2 Input    2 Bit        Muxes := 3
      4 Input    2 Bit        Muxes := 2
      2 Input    1 Bit        Muxes := 5
      4 Input    1 Bit        Muxes := 2
```

Dans la section « Report Cell Usage » :
On observe qu’il y a 77 registres au total, de type FDCE, FDRE et FDSE.
On retrouve notamment 71 registres FDCE, dont les 66 utilisés pour les modules Led_driver ((28 du compteur + 5 internes) x2 pour chaque module). On retrouve également 1 bouton resetn en entrée (IBUF) ainsi que les 2 LEDs (led0 et led1) sur 3 bits chacune, soit 6 (OBUF). On retrouve aussi « clk_wiz » le module d’instanciation des 2 clocks (clka et clkb).

12. Voici le placement de routage et l’étude des rapports générés :

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

WNS(ns)	TNS(ns)	TNS Failing Endpoints	TNS Total Endpoints	WHS(ns)	THS(ns)
0.236	0.000	0	89	0.078	0.000

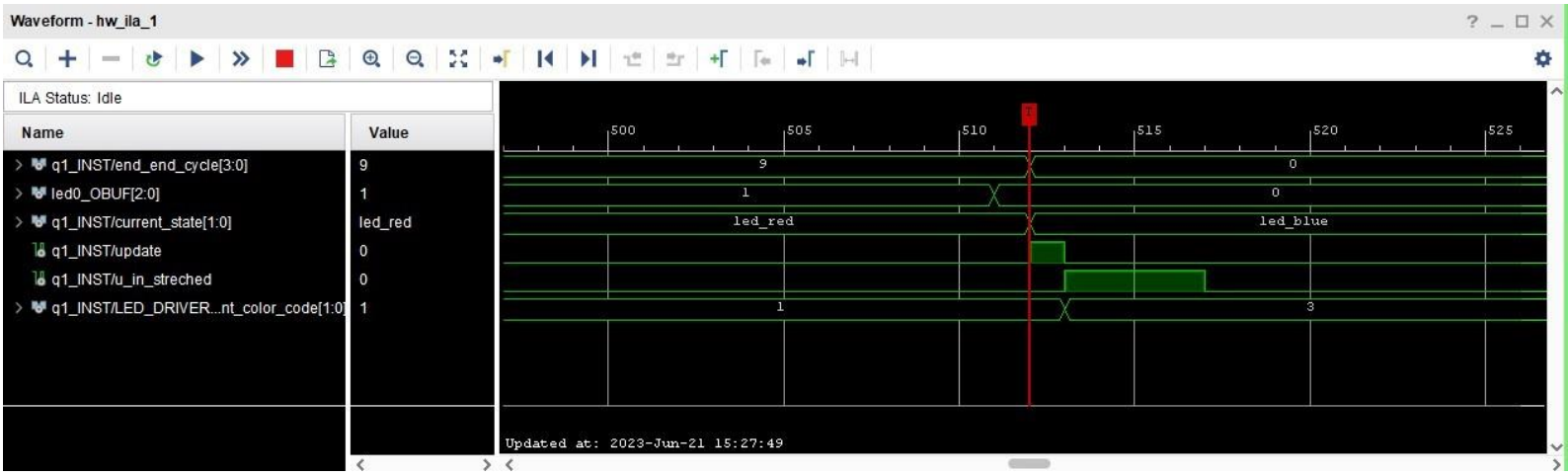
```
Max Delay Paths
--------------------------------------------------------------------------------
Slack (MET) :             0.236ns  (required time - arrival time)
  Source:                 q1_INST/LED_DRIVER_INST1/LED_COUNTER/cnt_reg[16]/C
                            (rising edge-triggered cell FDCE clocked by clk_a_clk_wiz_0  {rise@0.000ns fall@2.500ns period=5.000ns})
  Destination:            q1_INST/LED_DRIVER_INST1/LED_COUNTER/cnt_reg[25]/D
                            (rising edge-triggered cell FDCE clocked by clk_a_clk_wiz_0  {rise@0.000ns fall@2.500ns period=5.000ns})
```

On voit dans le rapport de timing qu’il n’y a aucune violation de setup ou de hold (TNS et WHS = 0), le Worth Negative Slack (WNS) est de 0.236 ns et le chemin critique (Max Delay Paths) prend sa source ici :
« q1_INST/LED_DRIVER_INST1/LED_COUNTER/cnt_reg[16]/C » et sa destination ici :

« q1_INST/LED_DRIVER_INST1/LED_COUNTER/cnt_reg[25]/D »
C'est le chemin qui boucle dans le compteur "LED_COUNTER" du module "LED_DRIVER_INST1".

13. Le bitstream a été généré et le comportement observé sur carte est celui attendu.
En effet, en utilisant l'ILA, on peut observer le comportement de notre design sur la carte cora Z7. On remarque tout d'abord que nous avons 2 fenêtres ILA, 1 pour chaque domaine d'horloge. Nous ne pouvons donc pas observer tous les signaux du design dans une même fenêtre.

Commençons par l'ILA lié au domaine d'horloge "clkA". Nous pouvons observer ci-dessous, que l'état de la FSM est correctement changé lorsque le compteur "end_end_cycle" passe de '9' à '0' et cela entraine également un passage à '1' du signal "update" qui met à jour le module "led_driver0" associé à l'horloge "clkA". On remarque aussi que le signal "u_in_streched" est étendu pendant 4 périodes de "clkA", ce qui correspond à 1 période de "clkB", afin de permettre au module "led_driver1" d'observer un changement d'état du signal "update_in" et ainsi mettre à jour "led_driver1".



Sur l'image ci-dessous, on peut observer que la deuxième ILA est liée au domaine d'horloge "clkB". On peut voir un changement d'état du signal "uins_stable" pendant 1 coup d'horloge "clkB", ce qui entraine le changement de couleur de la LED en sortie, ce que l'on remarque avec le passage de "led1_OBUF" de '1' à '4'. Ce changement de couleur spontané est dû au fait que de soit le module "led_driver0" qui génère les signaux update, or "led_driver0" suit la cadence de l'horloge "clkA" et va donc clignoter plus vite que le module "led_driver1" ce qui fait que le changement de couleur s'effectue normalement du coté de "led_driver0" mais en plein milieu d'un clignotement pour "led_driver1".