TP04 – Pilotage de LED et mémoire

Partie 1

Cédrine Socquet

Rendu

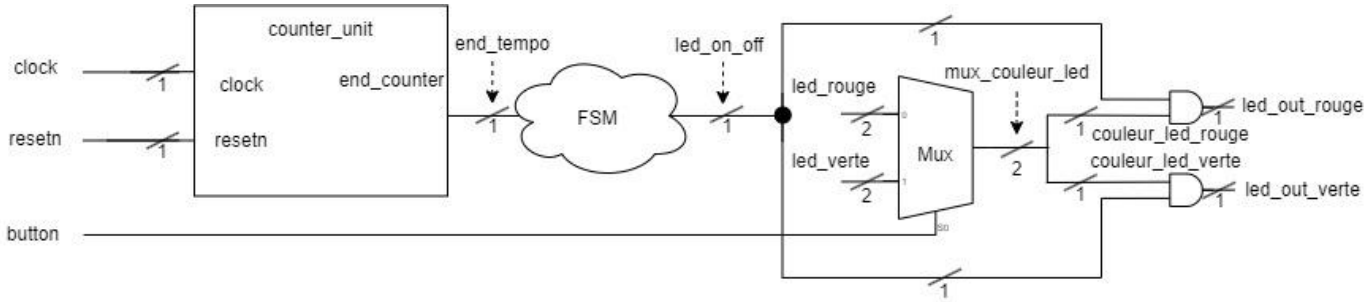
Votre rapport devra contenir :

- 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)
- Une démonstration de votre design

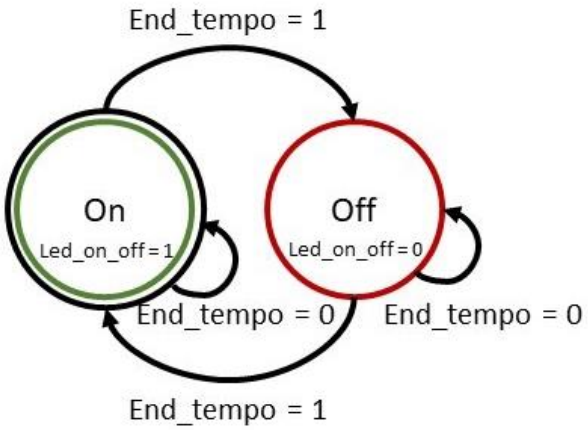
Vous fournirez également vos codes source commentés.

Réponses

1 - 2. Voici mon architecture RTL permettant de faire clignoter une LED et de la piloter en rouge ou en vert à l'aide d'un bouton, de sorte que la LED soit rouge puis devienne verte lorsque le bouton est appuyé.

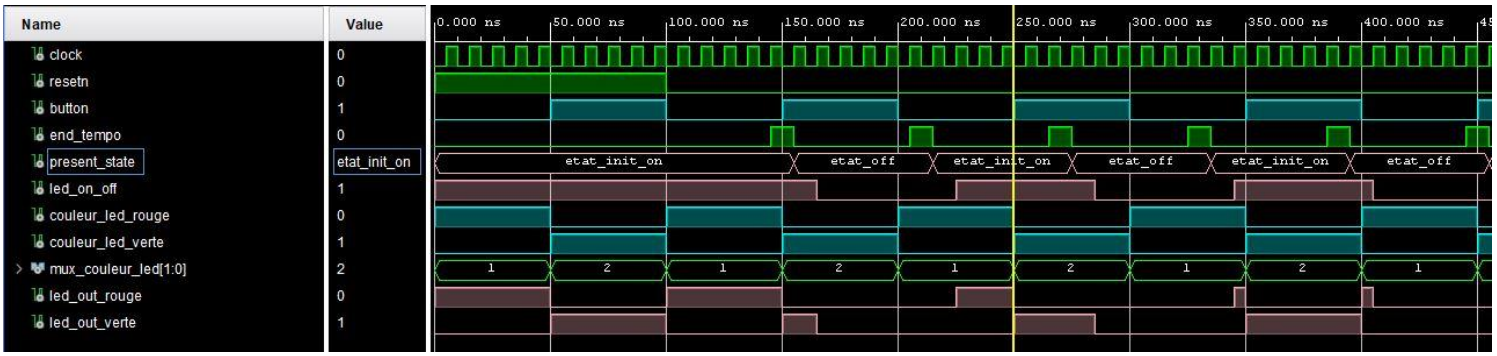


Voici ma FSM qui pilote le clignotement de la LED (RVB).



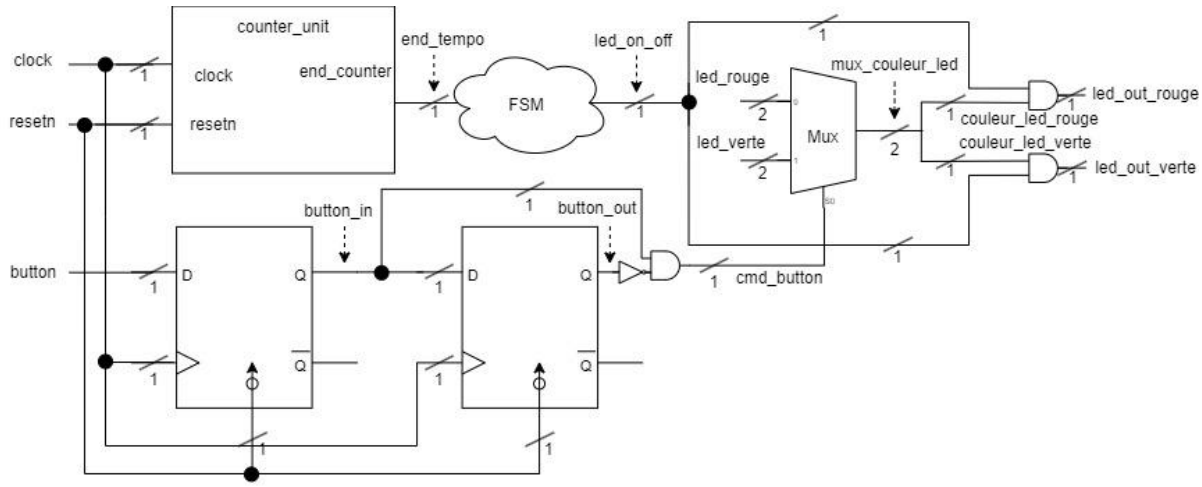
3. Voir les codes tp_led.vhd et tb_tp_led.vhd.

4. Lors de la simulation du testbench, si le bouton est pressé pendant plus d'un cycle d'horloge, on voit que la LED passe du rouge au vert

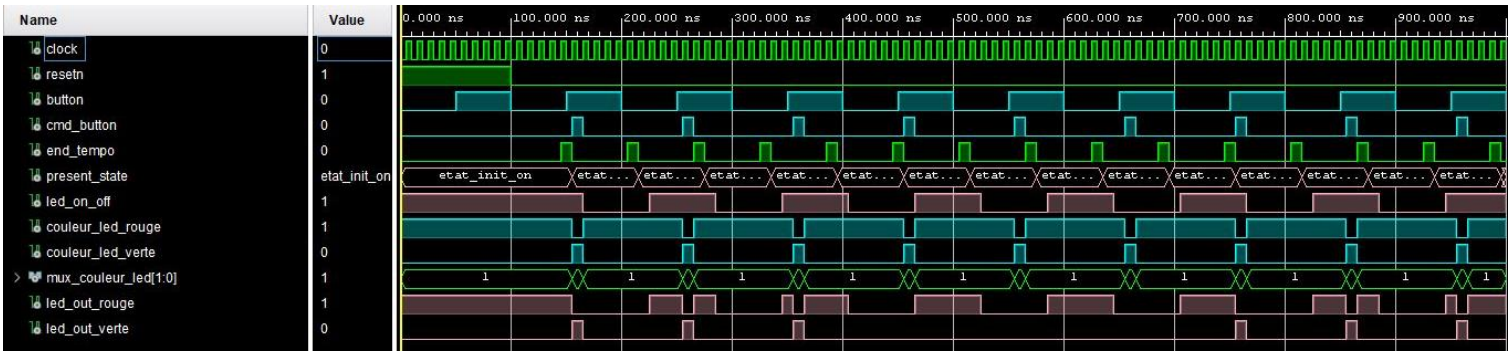


5. Pour que la LED ne clignote en vert qu'une seule fois même si le bouton est maintenu, il faudrait faire un détecteur de front montant uniquement pour le bouton, qui ne soit actif que pour une période de l'horloge.

6. Voici mon schéma RTL mis à jour.

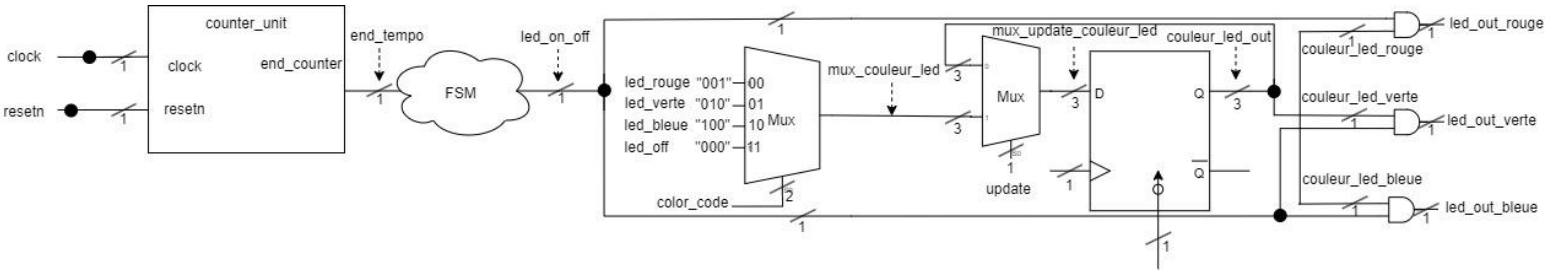


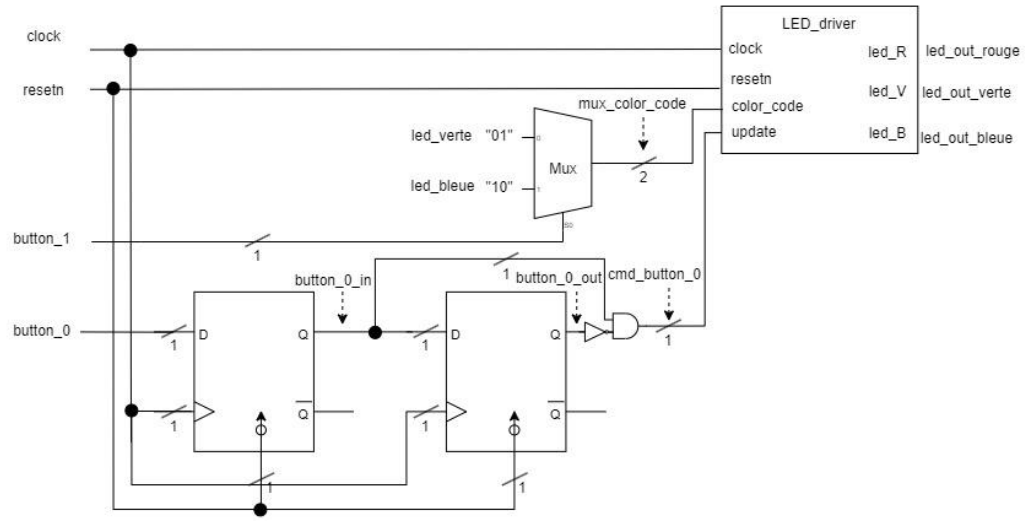
7. Voir les codes tp_led.vhd et tb_tp_led.vhd. Voici le chronogramme qui permet de vérifier que lorsque button = 1, la LED verte ne clignote que pendant une période de l'horloge.



8-9. Voici mon module de pilotage d'une LED RGB en RTL. Il permet 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. Il permet également de piloter les entrées/sorties du module :
- Le signal update reçoit 1 uniquement lorsque le bouton _0 vient d'être appuyé, maintenir le bouton enfoncé ne maintient pas le signal update à 1.
- Le signal color_code reçoit soit le code couleur « vert » si le bouton_1 est pressé, sinon il reçoit le code couleur « bleu »
- Les LED R, G et B sont connectées avec les couleurs respectives de la LED_0

Ci-dessous le schéma Led_driver, ci-après le schéma tp_with_Led_driver.





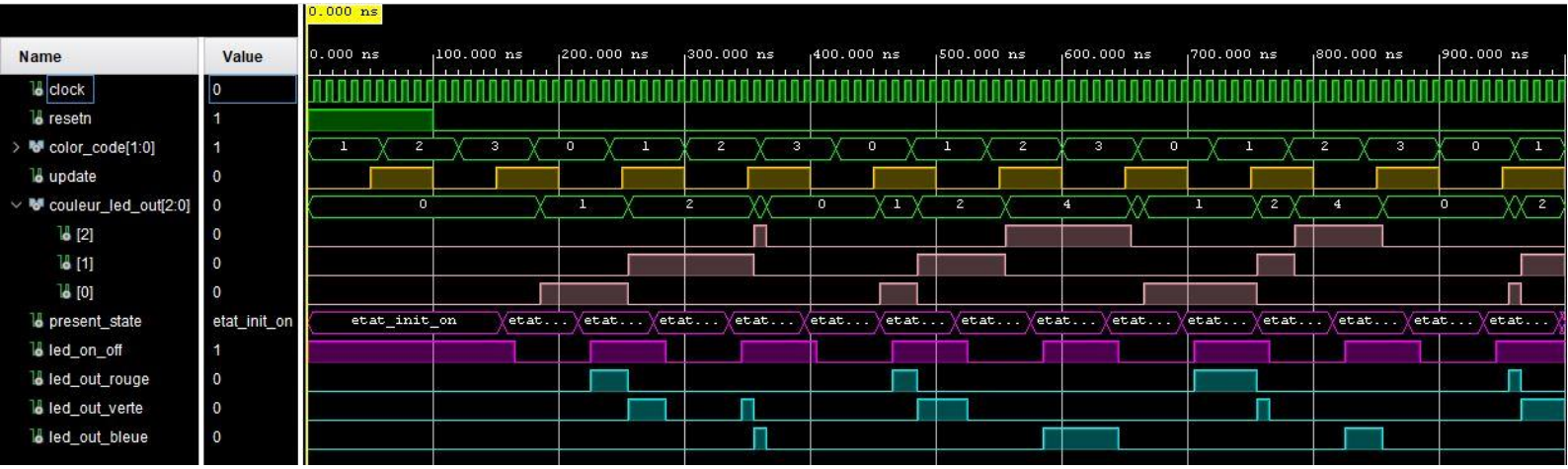
10. Voir les codes Led_driver.vhd et tp_with_Led_driver.vhd.

11. Voir les codes tb_Led_driver.vhd et tb_tp_with_Led_driver.vhd.

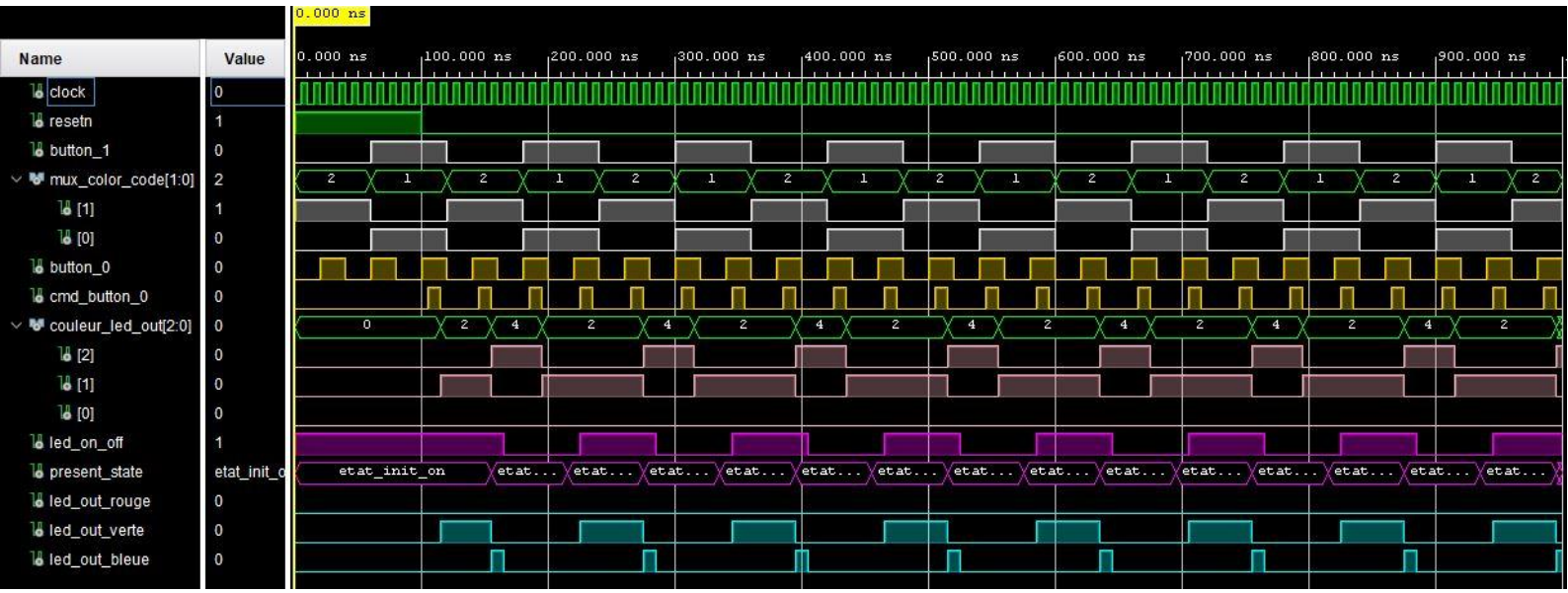
12. Nous pouvons voir que le résultat attendu à la simulation est le bon :

Chronogramme de LED_driver :
En effet, à 0 ns nous avons « resetn = 1 », après le resetn le prochain front montant est celui de « update » à 150 ns, qui prend comme valeur 3 (off) du color_code et fini sur 0 (rouge). Comme l'état de la FSM present_state est en « off » à ce moment-là, aucune led ne s'allume. Dès que la FSM change d'état et passe en « on », au prochain coup d'horloge le signal led_on_off va passer à 1 ce qui

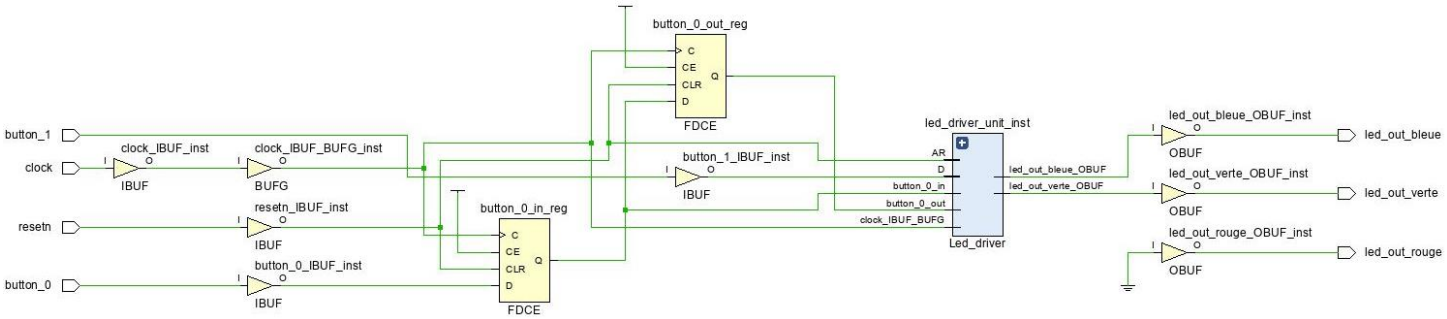
permettra également l'allumage des LEDs. La LED rouge passe donc à 1 puisque le dernier code couleur « color_code » en mémoire de « update » était 0. Update passe à nouveau à 1 et prend le code couleur 1 qui correspond à vert, la led verte s'allume donc tant que l'état de la FSM est allumé (on).
Et ainsi de suite, le motif se répète sur les différentes LEDs.



Chronogramme de tp_with_LED_driver :
Dans ce chronogramme, le mode de fonctionnement est identique, sauf que « update » est remplacé par « cmd_button_0 » et qu'il y a l'ajout du bouton « button_1 » qui vient piloter les couleurs verte et bleue. La couleur de la LED est donc verte mais passe à bleue lorsque button_1 est appuyé et que cmd_button_0 enregistre la valeur bleue au bon moment et que l'on est dans un état « on » de la FSM.



13. Voici le rapport de synthèse, les ressources utilisées correspondent bien au schéma RTL du tp_with_LED_driver.
En effet, on retrouve bien les entrées du schéma (clock, resetn, button_0 et button_1) et les sorties (led_out_bleue, led_out_verte et led_out_rouge) ainsi que les deux registres du button_0 et le composant LED_driver.




```
Report Cell Usage:
+------+-------+------+
|      |Cell   |Count |
+------+-------+------+
|1     |BUFG   |     1|
|2     |CARRY4 |     7|
|3     |LUT1   |     1|
|4     |LUT2   |    33|
|5     |LUT4   |     5|
|6     |LUT6   |     3|
|7     |FDCE   |    33|
|8     |FDPE   |     1|
|9     |IBUF   |     4|
|10    |OBUF   |     3|
+------+-------+------+
```

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

Dans le Report Cell Usage, on retrouve bien les 4 entrées du schéma (clock, resetn, button_0 et button_1) ainsi que les 3 sorties (led_out_bleue, led_out_verte et led_out_rouge). On retrouve également les 33 registres FDCE, les 28 du compteur, les 2 des boutons, 2 pour le couleur_led_out du Led_driver et le dernier pour le present_state de la FSM. On retrouve aussi 1 registre FDPE pour la led_on_off.
Dans le Detailed RTL Component Info, on retrouve les registres ainsi que leur nombre de bits mais aussi l'ensemble des multiplexeurs utilisés dans le schéma.

14. On voit dans le rapport de timing qu'il n'y a pas de violations (TNS et WHS = 0) et que le chemin critique (WNS) est de 4.950 ns.

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

    WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints      WHS(ns)      THS(ns)
    -------      -------  ---------------------  -------------------      -------      -------
      4.950        0.000                      0                   33        0.254        0.000
```

15. Le bitstream est généré et l'on obtient bien le résultat attendu sur carte.