Rapport TP 04

Samory DIABY

Table des matières

1 Objectifs

L'objectif de ce TP est de réaliser un module permettant de piloter une led RGB.

2 Notes

Pour ce tp, j'ai assumé que les compteurs étaient codés sur 28 bits.

3 Réalisations

3.1 Clignotement d'une LED à partir du compteur du TP02

Pour réaliser un circuit permettant de faire clignoter une LED à partir du compteur du TP02, on dessine le schéma suivant :

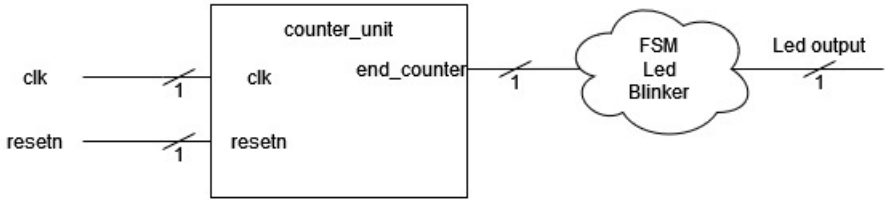

Figure 1 – Schéma RTL du circuit permettant de faire clignoter une LED (Q1)

Le bloc "counter_unit" représente le compteur du TP02 et "FSM_LED_Blinker" représente la FSM ci-dessous :

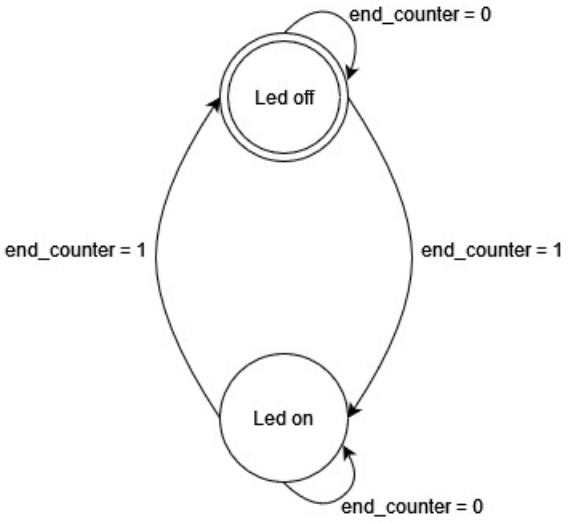

Figure 2 – Schéma RTL de la FSM permettant de faire clignoter une LED (Q1)

3.1.1 Ajout d'une LED verte au circuit

On souhaite maintenant ajouter une LED verte au circuit qui doit clignoter uniquement lorsqu'un bouton (ici le bouton 0) est pressé. Pour ce faire on ajoute un démultiplexeur qui va s'occuper de rediriger le signal "end_counter", qui sert à faire clignoter une LED, dans la FSM correspondante. Si le bouton est pressé on passe dans la FSM de la LED verte, sinon celle de la LED rouge.

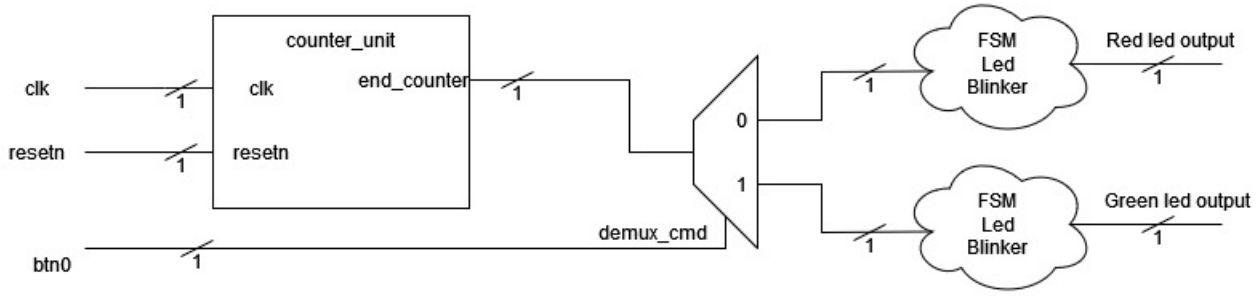

Figure 3 – Ajout d'une LED verte + FSM sur le schéma (Q2)

Après écriture et simulation du code VHDL associé au schéma ci-dessus, on obtient le chronogramme suivant :

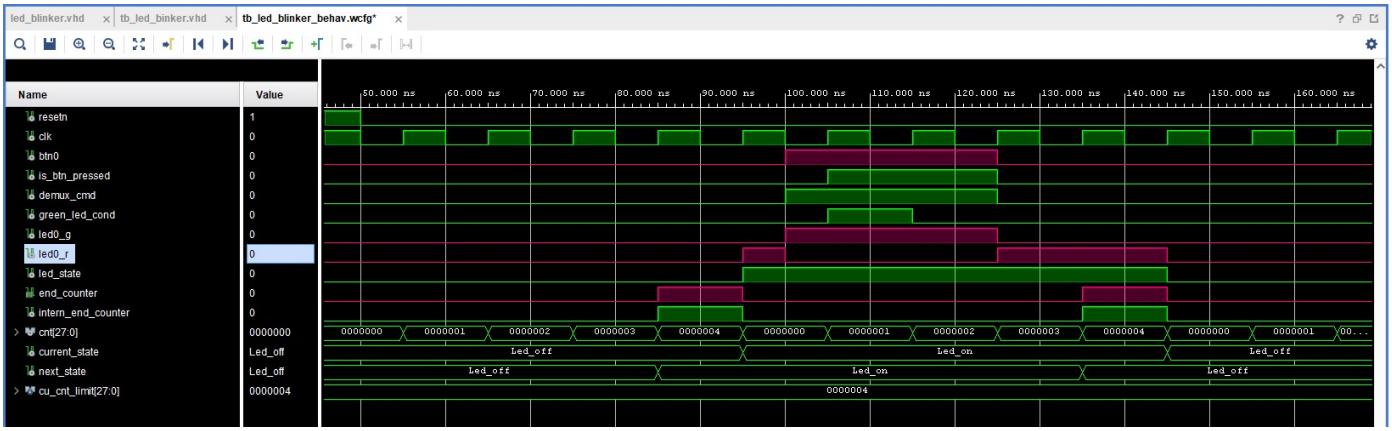

Figure 4 – Chronogramme testbench Q4

On peut observer que la LED verte remplace la LED rouge tant que le bouton 0 est pressé, et la LED rouge 'reprend la main' dès que le bouton est relâché. La LED verte peut donc clignoter pendant plus d'un cycle d'horloge tant que le bouton est maintenu.

3.1.2 Un seul clignotement pour la LED verte

Pour que la LED verte ne clignote qu'une fois, il faut prendre en compte l'état précédant du bouton (en terme de cycles d'horloge) et le comparer à l'état actuel. On ne fait alors "clignoter" la LED verte que si le bouton est actuellement pressé et ne l'était pas à l'état précédant. On peut observer ce comportement sur le chronogramme ci-dessous :

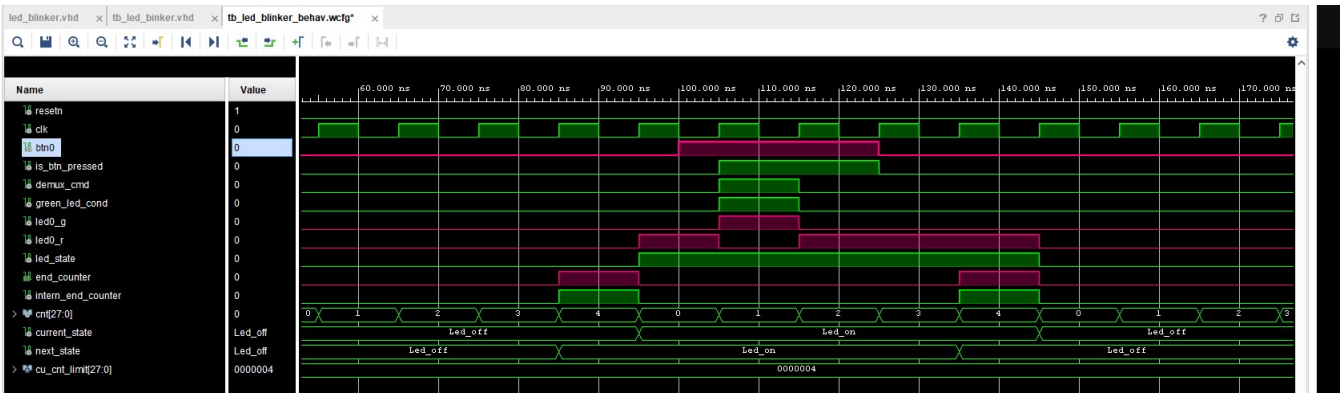

Figure 5 – Chronogramme testbench Q7

Ci-dessous le schéma RTL du circuit :

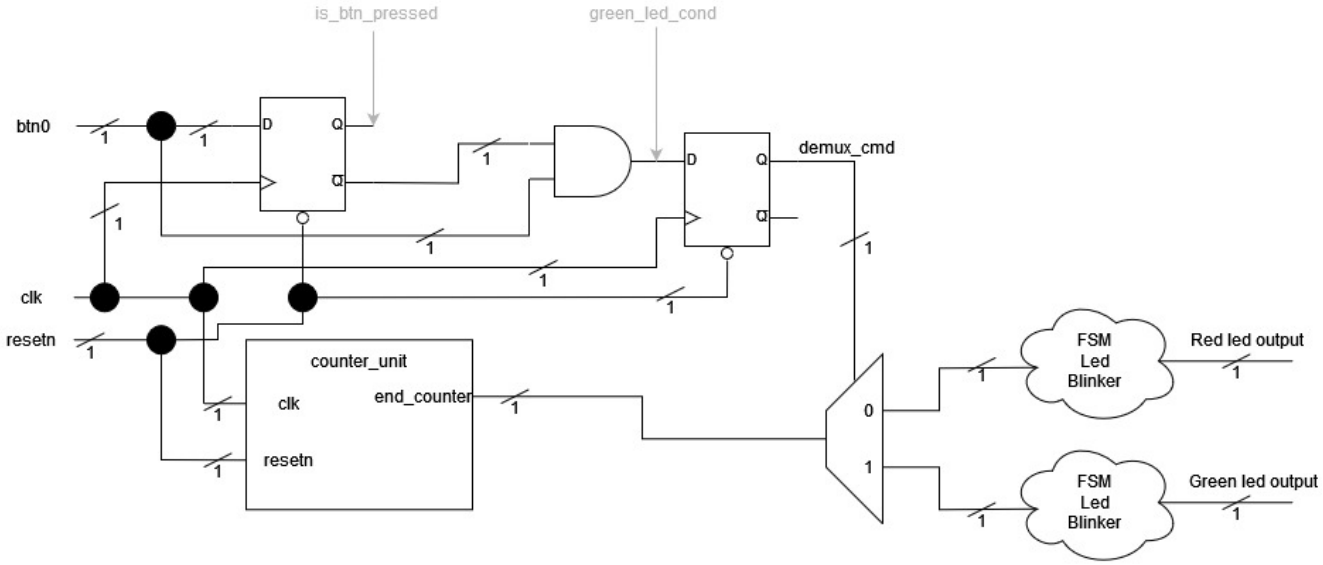

Figure 6 – Schéma RTL (Q6)

On peut voir l'utilisation des 2 registres : **is_btn_pressed** pour garder l'état précédant du bouton et **demux_cmd** pour garder l'état de la commande du démultiplexeur (pour faire en sorte que la LED ne clignote qu'une fois même si le bouton reste appuyé).

3.2 Module LED_driver

Pour créer le module de pilotage de la LED RGB, on dessine le schéma ci-dessous :

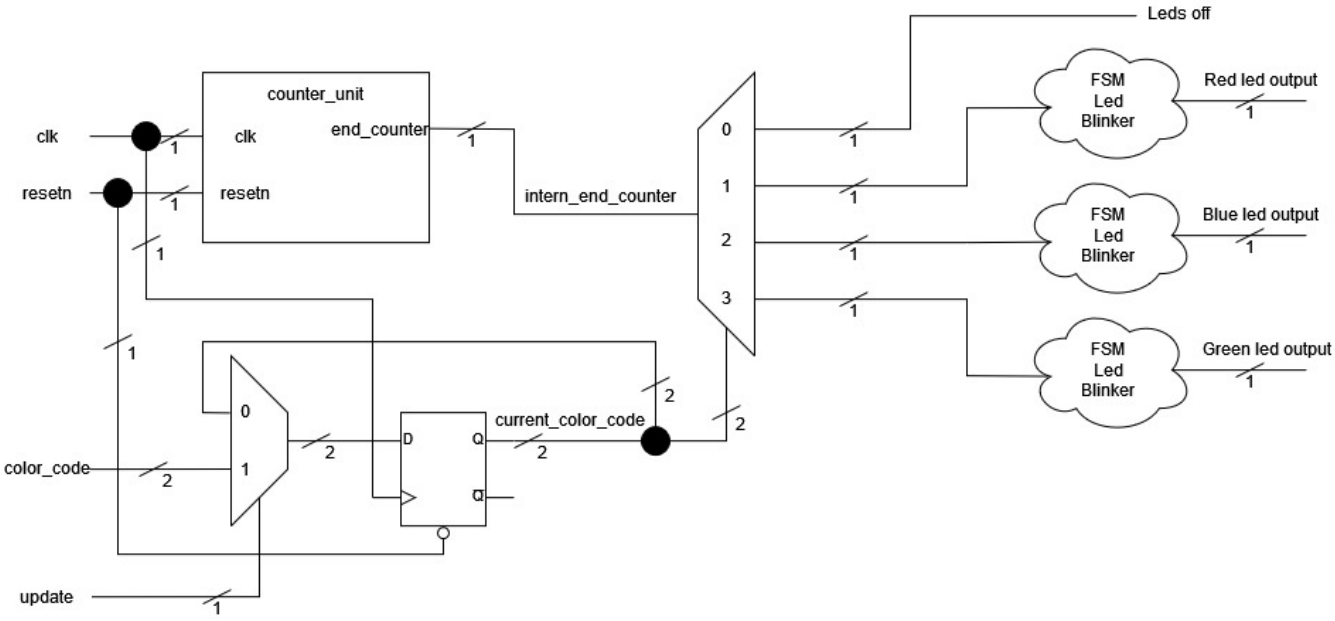

Figure 7 – Schéma RTL du module LED_driver (Q8)

Le code couleur utilisé sera celui de l'exemple :

Code couleur	Couleur de la LED RGB
00	Eteinte
01	Rouge
10	Verte
11	Bleue

3.2.1 Logique supplémentaire (Q9)

Dans le schéma ci-dessous, on rajoute la logique demandé pour la Q9 (avec le bloc led_driver représentant le schéma RTL de la Q8) :

Après écriture en VHDL et simulation du schéma de la Q9, on obtient le chronogramme ci-dessous :

On peut voir que les LEDs sont éteintes jusqu'au premier appui sur le bouton 0 qui sert à l'update. Après le premier appui sur le bouton 0, le signal update est mis à jour au front montant suivant ce qui va mettre à jour la sélection de couleur encore 1 front montant après. On a donc **cuurent_color_code** qui vaut la valeur de l'entrée **color_code** (slected_color sur le chronogramme) qui est ici "3" soit la couleur bleue.

A partir de cette instant, seule la LED bleue clignotera jusqu'à ce qu'on appui sur le

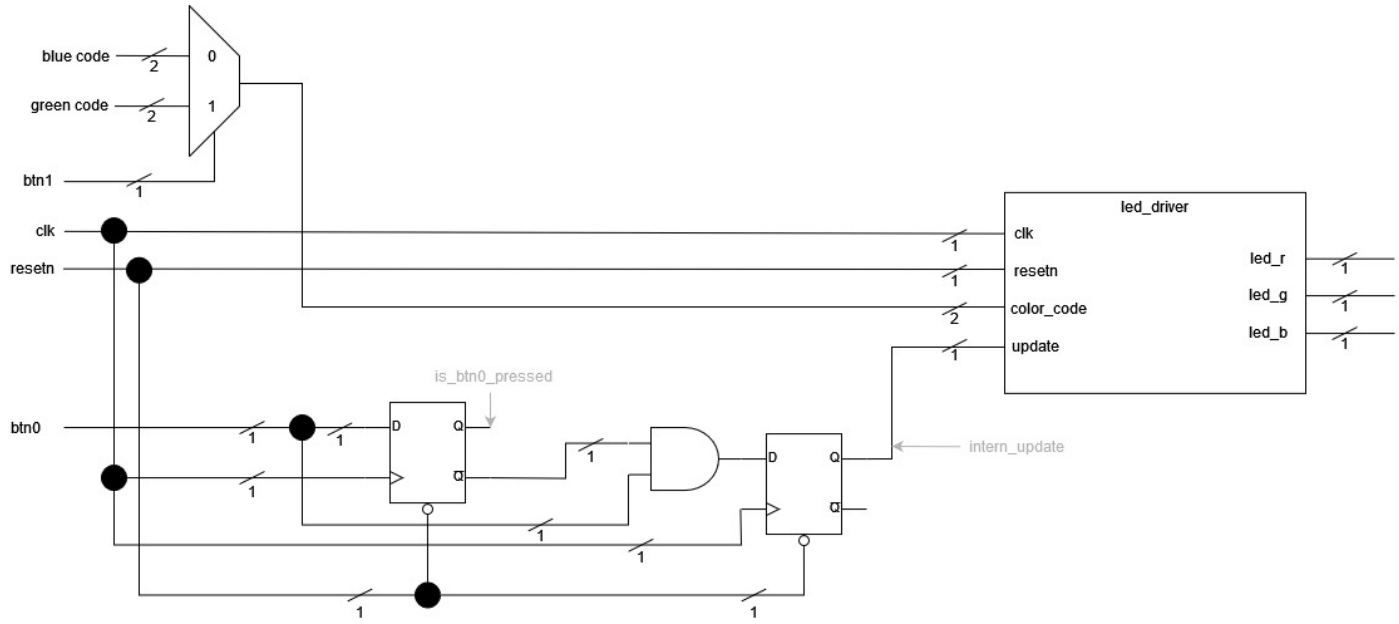

Figure 8 – Ajout de logique au module LED_driver (Q9)

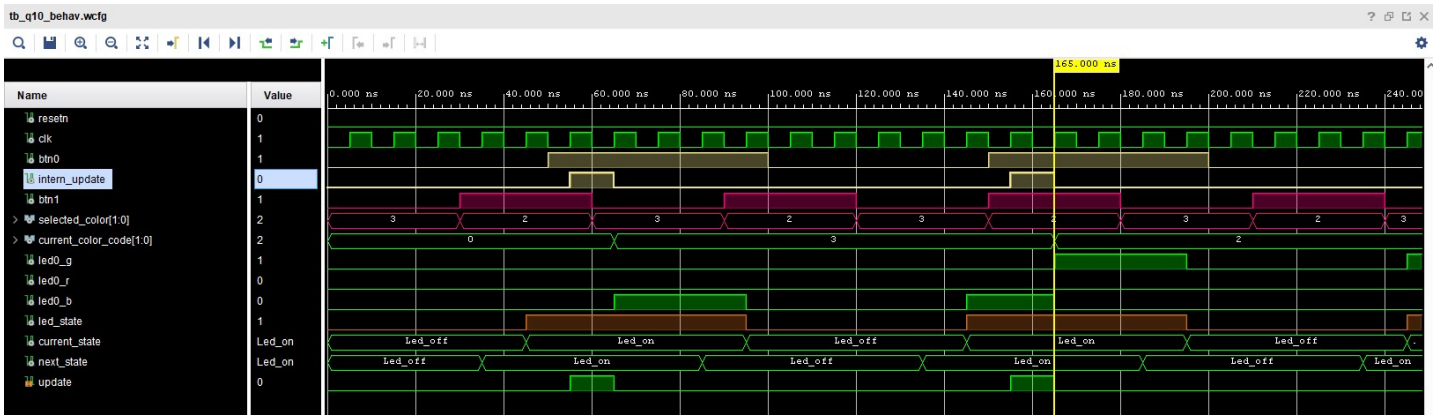

Figure 9 – Chronogramme testbench Q11

bouton 1 et le bouton 0 en même temps(btn1 pour changer la valeur de sortie du multiplexeur de choix de couleur, et btn0 pour mettre à jour **current_color_code**). Si les 2 boutons sont appuyés en même temps la LED verte clignote jusqu'à un autre appui simultané sur les 2 boutons (pour repasser sur la LED bleue).

3.3 Etude du rapport de synthèse

Le rapport de synthèse reprend les éléments des schémas Q8 et Q9. On retrouve dans la partie **RTL Component Statistics** 4 registres : 1 de 2 bits pour **current_color_code**, 1 de 1 bit pour pour l'état de la FSM et 2 de 1 bit pour gérer l'appui sur le bouton 0.
On retrouve également 2 multiplexeurs, mais ils en manquent comparé au schéma RTL (celui de 1 bit semble lié au **next_state de la FSM**).

```
84  ---------------------------------------------------------------------------------
85  Start RTL Component Statistics
86  ---------------------------------------------------------------------------------
87  Detailed RTL Component Info :
88  +---Registers :
89                     2 Bit    Registers := 1
90                     1 Bit    Registers := 3
91  +---Muxes :
92         2 Input    2 Bit        Muxes := 1
93         2 Input    1 Bit        Muxes := 1
94  ---------------------------------------------------------------------------------
95  Finished RTL Component Statistics
96  ---------------------------------------------------------------------------------
```

Figure 10 – RTL Component Statistics

Il manque la section concernant la FSM dans le rapport, mais la section **Report Cell Usage** nous indique que l'on a 33 registres (FDCE) au total, ce qui correspond à ce que l'on a sur nos schémas (28 FDCE pour le **compteur**, 2 FDCE pour **current_color_code**, 2 FDCE pour gérer le **bouton 0** et 1 pour la **FSM**).

3.4 Etude du rapport de timing

En étudiant le rapport de timing, on observe qu'il n'ya pas de violation de setup ou de hold (THS et TNS sont à 0).

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

    WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints      WHS(ns)      THS(ns)  THS Failing Endpoints  THS Total Endpoints     WPWS(ns)     TPWS(ns)
    -------      -------  ---------------------  -------------------      -------      -------  ---------------------  -------------------     --------     --------
      5.539        0.000                      0                   32        0.275        0.000                      0                   32        4.500        0.000


All user specified timing constraints are met.
```

Figure 11 – Design Timing Summary

On peut également voir que le chemin critique donné par vivado est le chemin qui boucle dans le compteur de **LED_COUNTER**.

```
Max Delay Paths
--------------------------------------------------------------------------------------
Slack (MET) :             5.539ns  (required time - arrival time)
  Source:                 COMP_LED_DRIVER/LED_COUNTER/cnt_reg[26]/C
                            (rising edge-triggered cell FDCE clocked by sys_clk_pin  {rise@0.000ns fall@5.000ns period=10.000ns})
  Destination:            COMP_LED_DRIVER/LED_COUNTER/cnt_reg[25]/D
                            (rising edge-triggered cell FDCE clocked by sys_clk_pin  {rise@0.000ns fall@5.000ns period=10.000ns})
```

Figure 12 – Chemin critique - rapport de timing

3.5 Résultats de l'ILA

Pour vérifier le fonctionnement du design, on utilise l'ILA. Le chronogramme ci-dessous montre le résultat de l'ILA lors du tout premier appui sur le bouton 0 (juste après l'envoi du code sur la carte). On peut voir que le signal **update** passe à "1" pendant seulement une période de l'horloge et va provoquer le changement du code couleur qui passe de "0" à "3" (**current_color_code**). On remarque également que le changement du code couleur entraîne bien un changement au niveau de la LED allumée sur la carte (ici on passe d'aucune LED allumée à la LED bleue allumée).

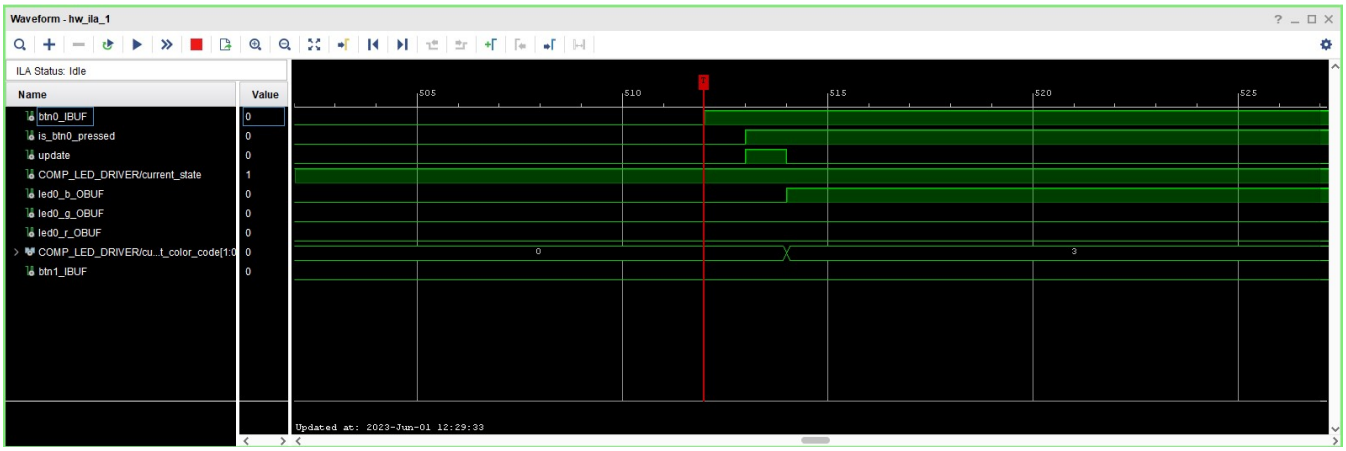

Figure 13 – ILA - appui sur bouton 0

On essaye maintenant de changer la couleur de la LED allumée en appuyant sur le bouton 1 et sur le bouton 0 pour mettre à jour le code couleur dans le module **led_driver**. On voit donc sur le chronnogramme que le signal **current_color_code** passe de "3" à "2" ce qui correspond au changement de la LED bleue vers la LED verte. La LED verte n'est pas allumée car on est dans l'état **LED off** de la FSM (le signal **current_state** est à "0").

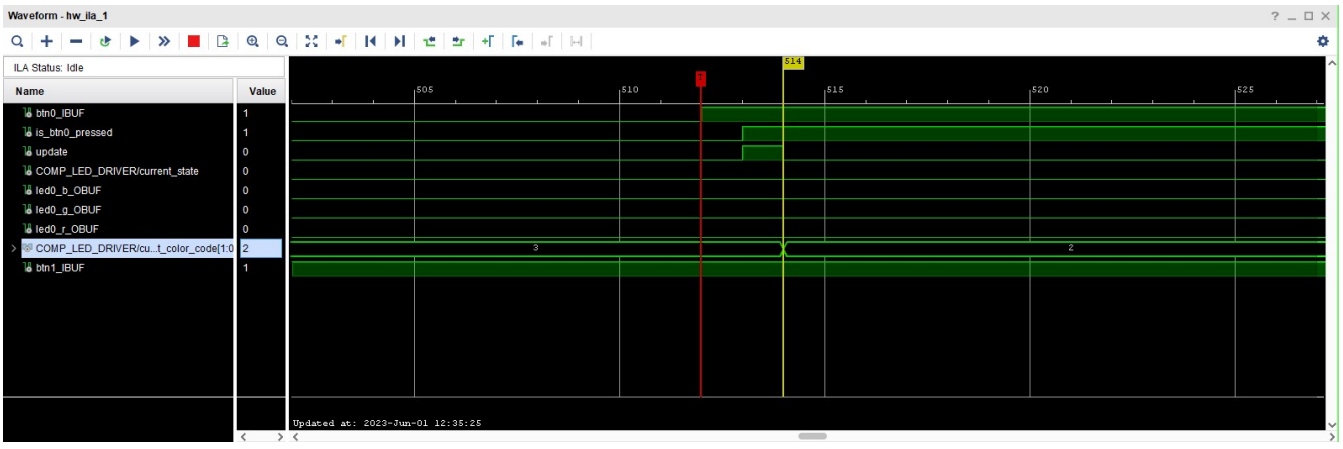

Figure 14 – ILA - appui sur bouton 1 + bouton 0 (LED éteinte)

Voici un autre chronogramme montrant la même action mais avec cette fois ci l'état

LED on. On peut maintenant observer le changement de LED allumée au moment du changement de code couleur.

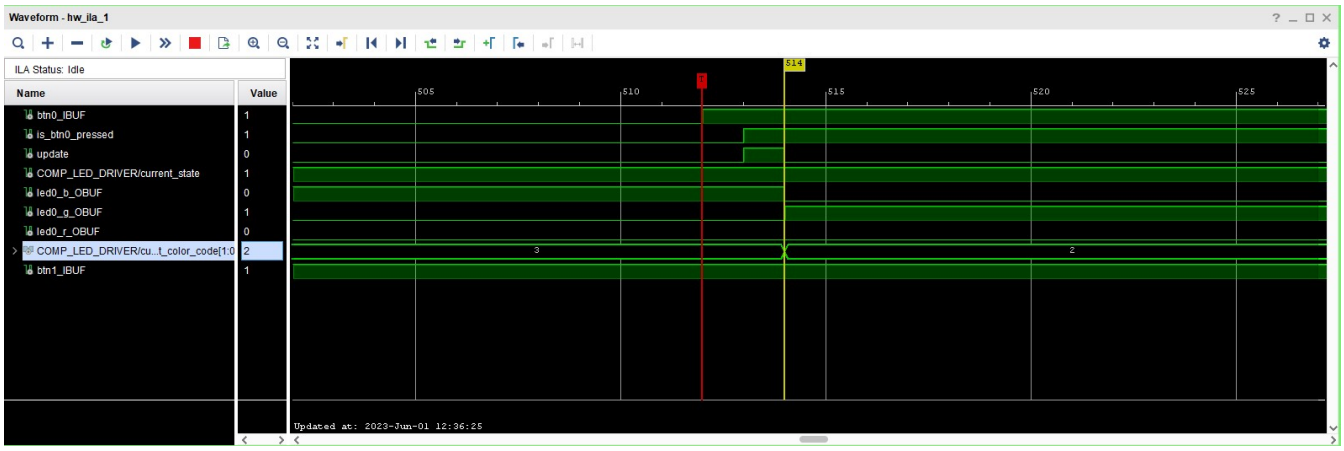

Figure 15 – ILA - appui sur bouton 1 + bouton 0 (LED allumée)