## Segment 2

**TP4 : Led driver**

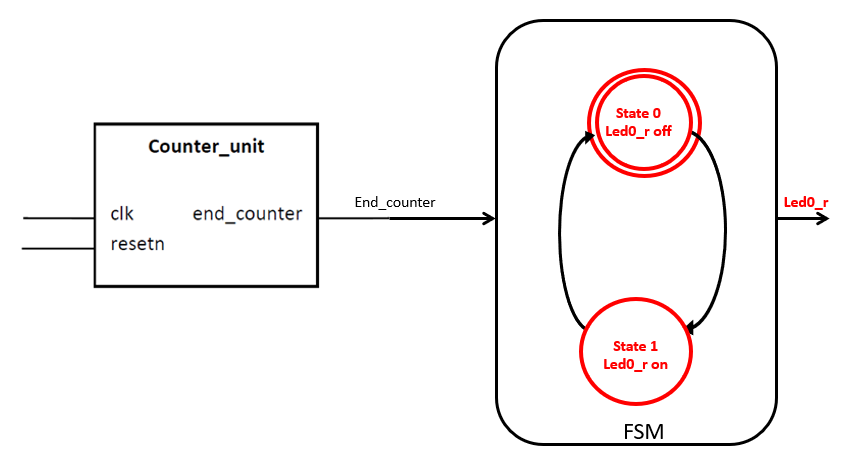
**Objectif**

L’objectif de cet TP est de réaliser un design permettant de faire clignoter une LED RGB avec une séquence de couleurs entrées à l’aide des boutons. Dans cette partie, nous définirons un module LED\_driver .

**Questions**

1. Créez une architecture RTL permettant de faire clignoter une LED (par exemple la led0\_r) en utilisant le module *Counter\_unit* du TP2 et une machine à état.

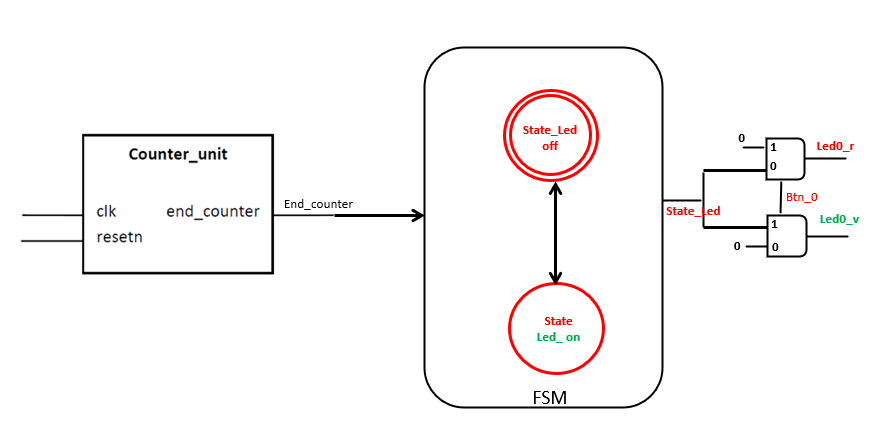
Pour répondre à cette question, nous allons utilisez le même schéma RTL du module connter\_unit qui permet de faire clignoter une led. Pour ce faire, nous utilisons le bloc counter\_unit et un bloc FSM qui gère les états de la led rouge allumée et éteinte. Notre état initial est la led éteinte. L’état suivant est la led allumée. On a idle=OFF, et State1=ON.



1. Modifiez votre architecture pour piloter une LED rouge et une LED verte. Lorsque le bouton\_0 est appuyé, la LED verte est allumée, sinon la LED rouge est allumée.

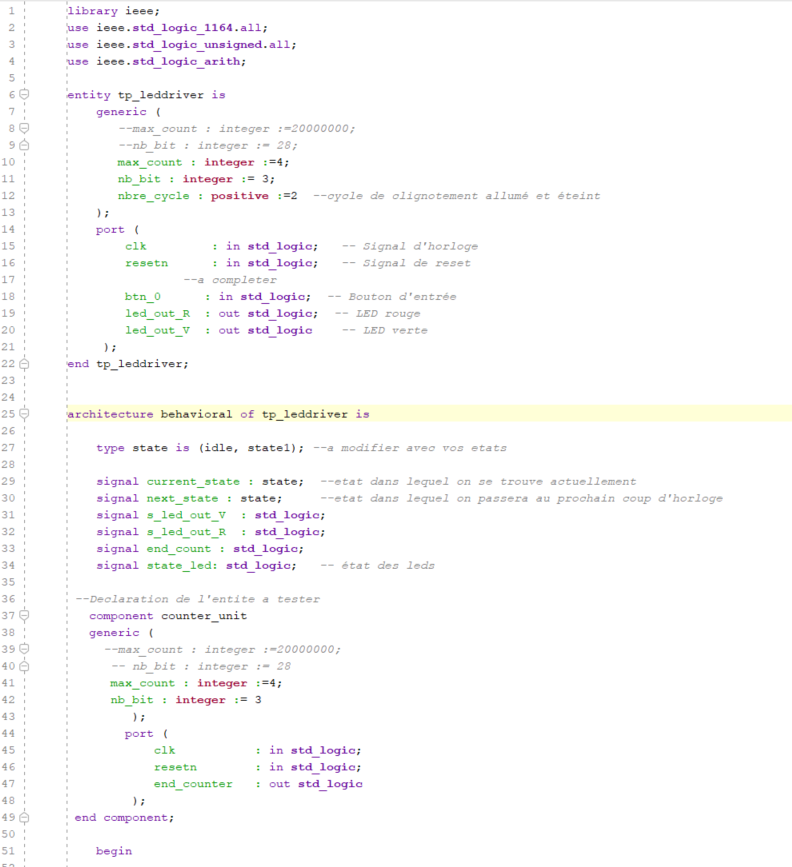
Pour ce faire nous rajoutons un bouton\_0(btn\_0) qui permet de piloter la led verte et la led rouge. Lorsque le btn\_0=1, la led verte est allumée et sinon la led rouge est allumée. Nous utilisons deux multiplexeurs qui nous permette d’envoyer la commande du signal btn\_0.

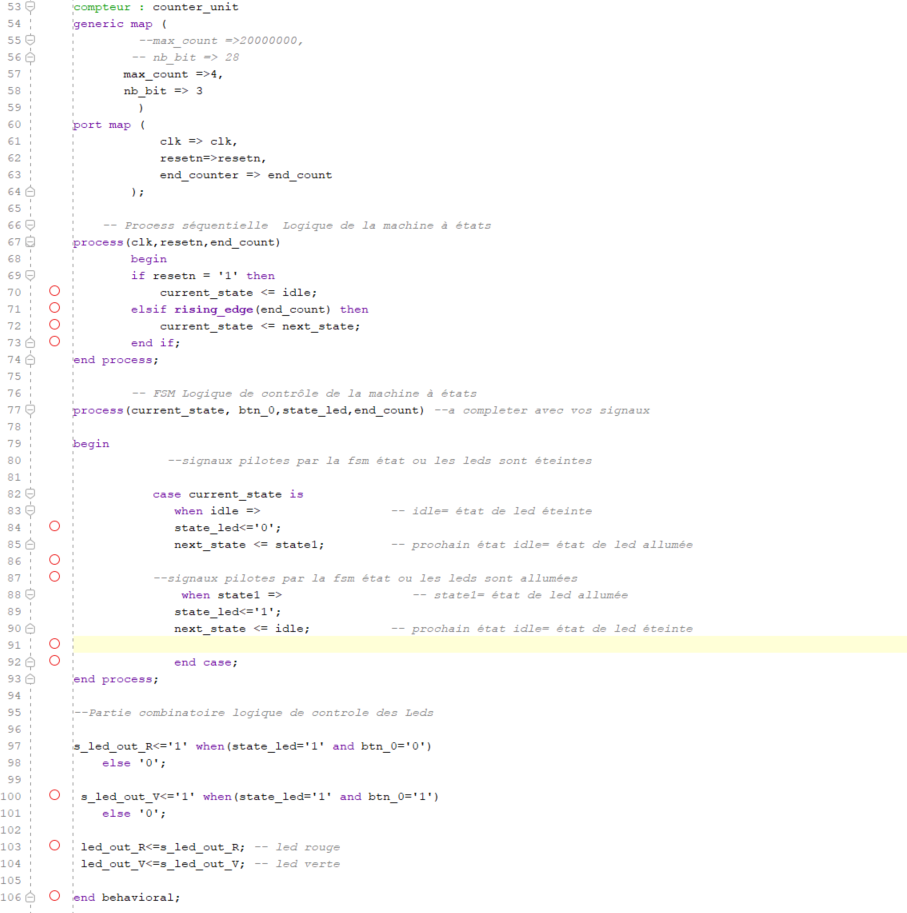
Pour allumer la led verte, il faut que state\_led='1' et btn\_0='1'. Pour allumer la led rougee, il faut que state\_led='1' et btn\_0='0'. Et lorsque state\_led='0', les deux leds sont étéintes.



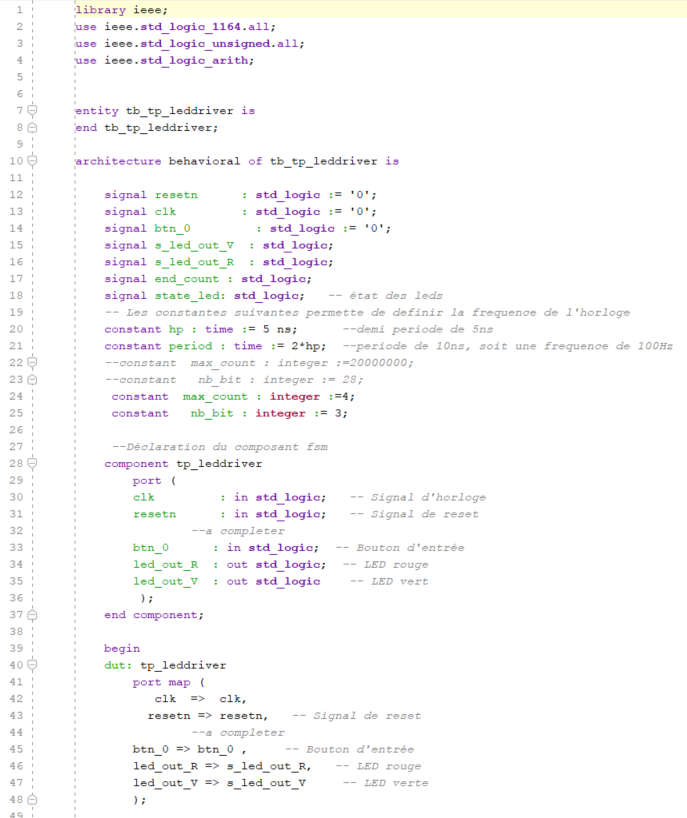
1. Rédigez le code VHDL de votre architecture.

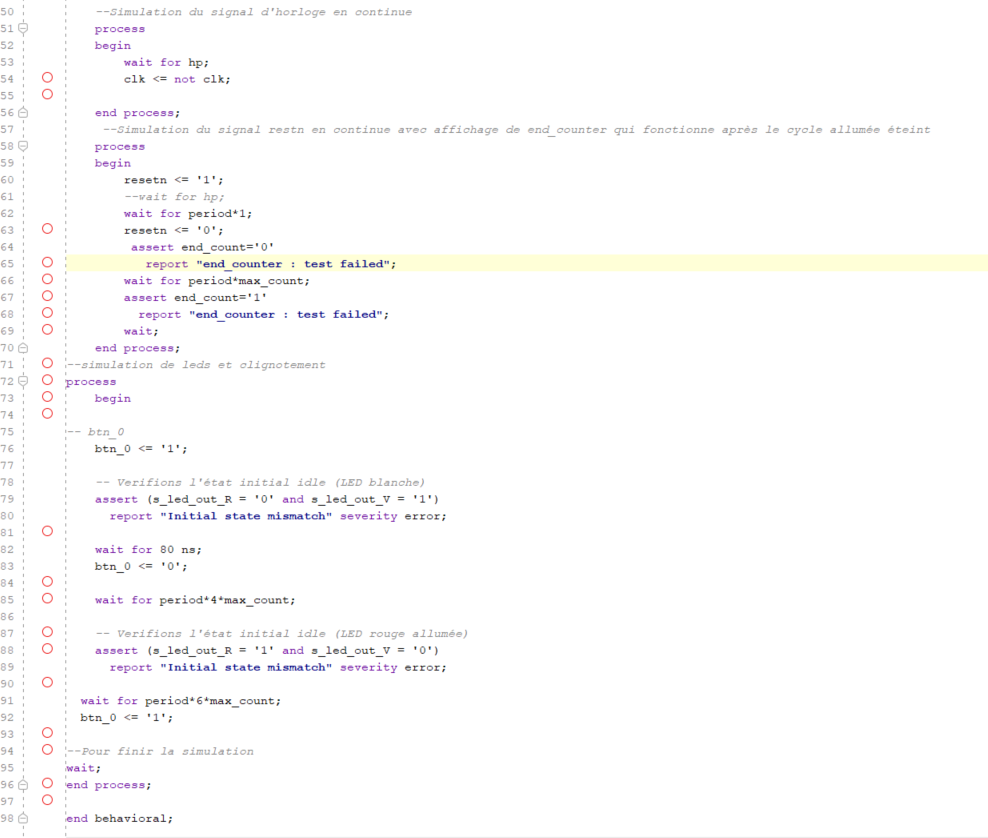
Pour cette partie, je conserve toujours le module counter\_unit et je crée un code FSM pour l’état de la led. Et un multiplexeur pour gérer les états allumés/éteints des leds.

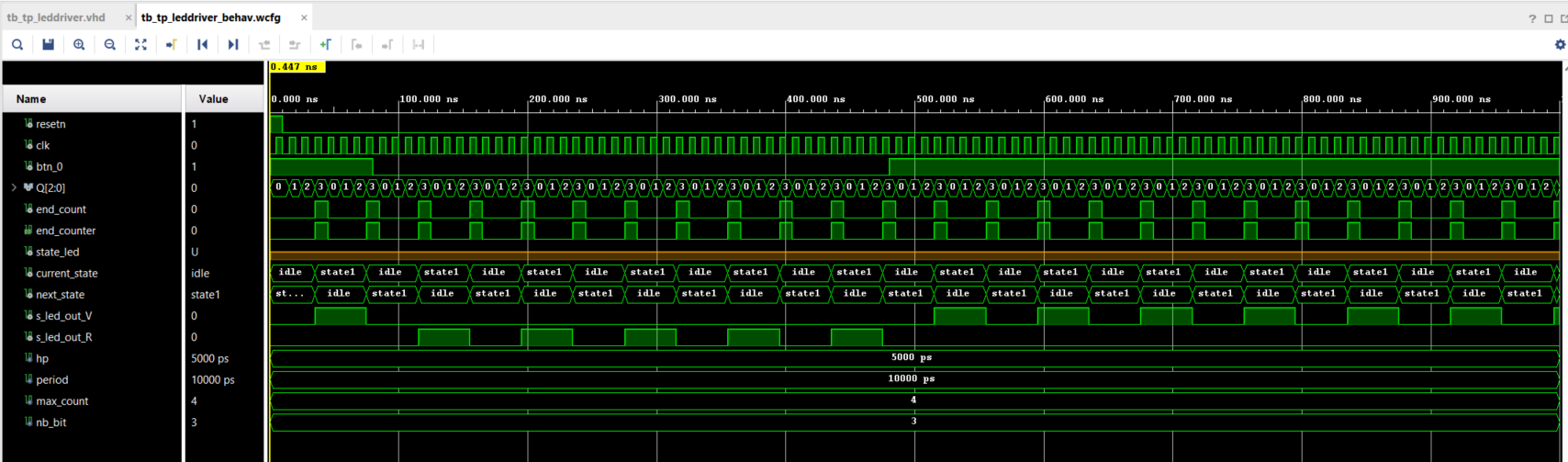




1. Réalisez une simulation en rédigeant un testbench. Que se passe-t-il si le bouton est pressé pendant plus d’un cycle d’horloge ?





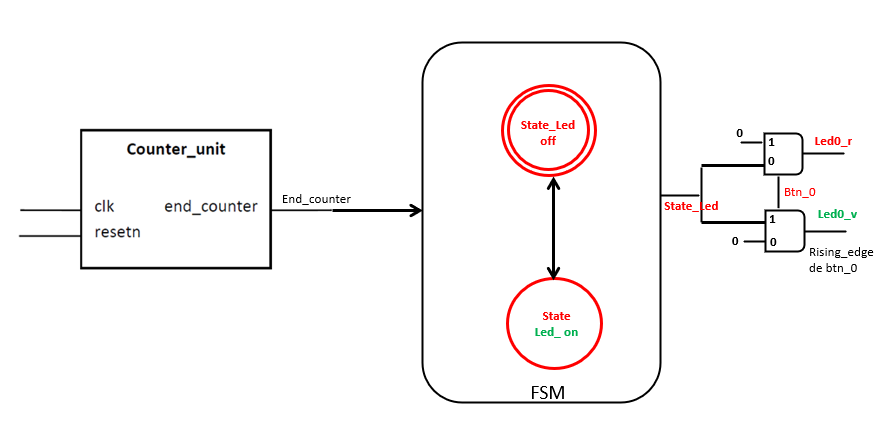


Lorsque le bouton est pressé pendant plus d’un cycle d’horloge, on observe que la led verte est allumée et s’éteint en fonction d’End\_count. Lorsque le bouton passe à zéro, n’est plus pressée, la led rouge s’allume et s’éteint en fonction d’End\_count.

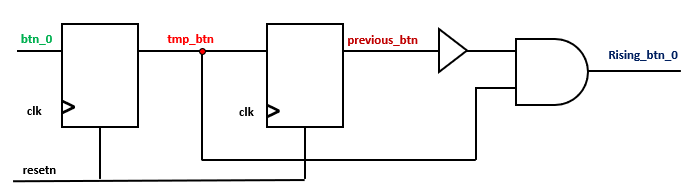
Si le bouton est longtemps pressé (plusieurs cycles d’horloges) alors la led verte clignote durant tout le cycle.

1. Que faudrait-il faire pour que la LED ne clignote en vert qu’une seule fois même si le bouton est maintenu ?

Pour que la led verte clignote une seule fois même si le bouton est maintenu, il faut que lors d’un front montant du bouton, la led verte clignote et sinon la led rouge clignote.

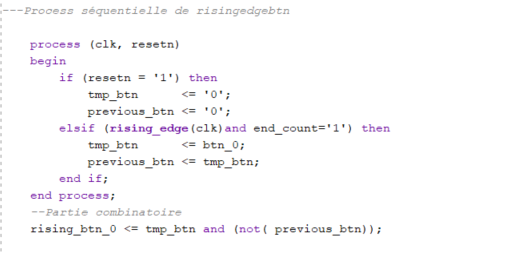


Le schéma RTL de rising\_btn\_0 est donné ci-dessous. Avec deux registres et une partie combinatoire, on sort le signal qui servira à piloter les leds rouge et verte.

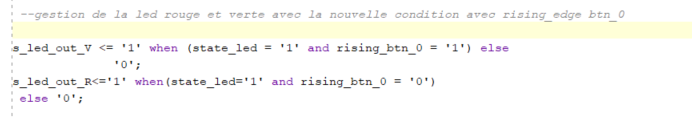


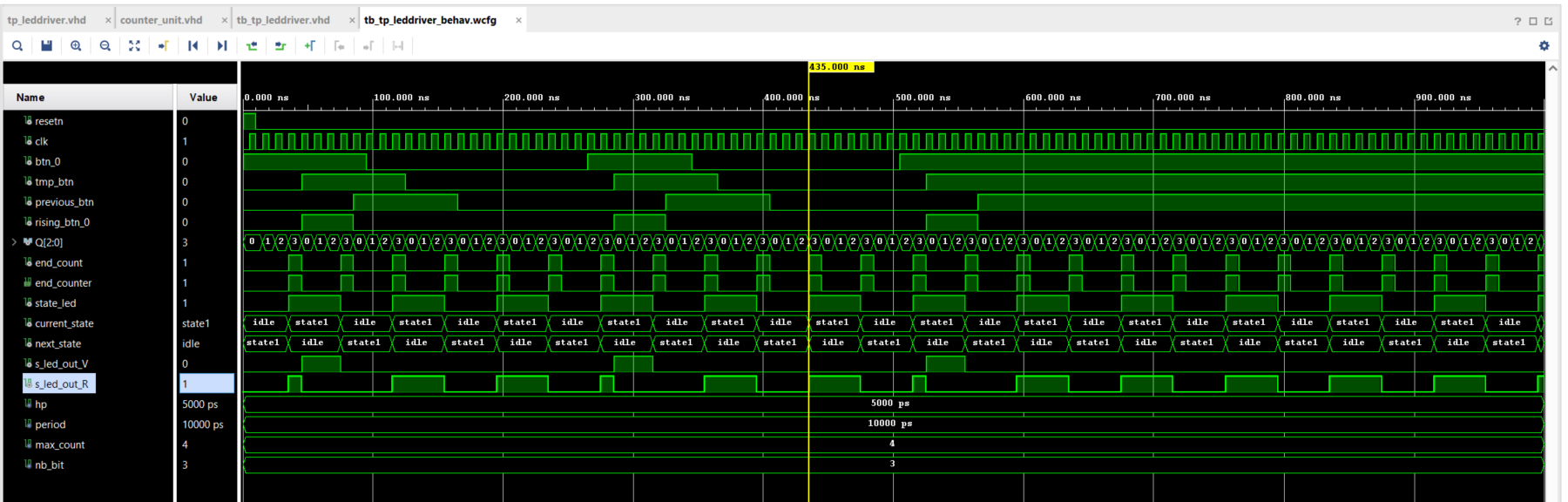
1. Ajoutez cette solution à votre architecture RTL.

Dans la partie combinatoire, nous récrivons la condition sur du front montant de btn\_0 qui est rising\_btn\_0



1. Mettez à jour votre code VHDL et vérifiez votre résultat à la simulation.



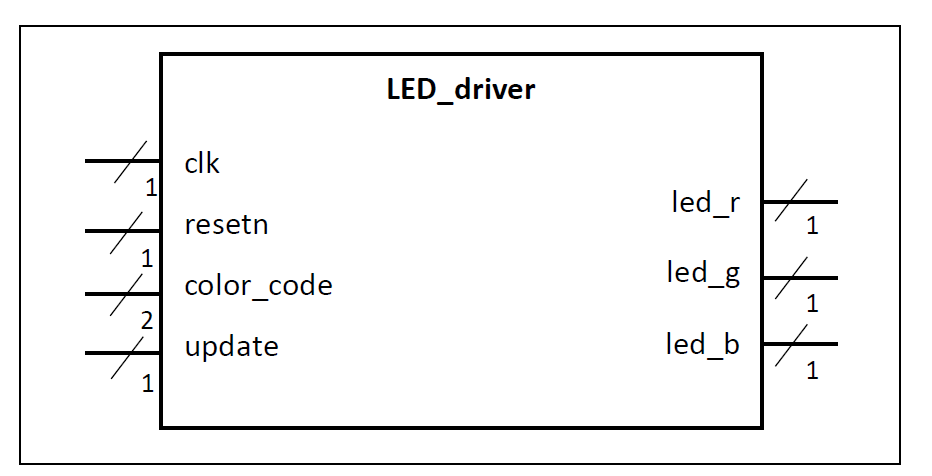


Dans cette logique combinatoire, lorsque state\_led est '1'. Si rising\_btn\_0 est '1', cela signifie que le front montant du signal btn\_0 a été détecté, donc la LED verte est allumée (s\_led\_out\_V est '1') et la LED rouge est éteinte (s\_led\_out\_R est '0').

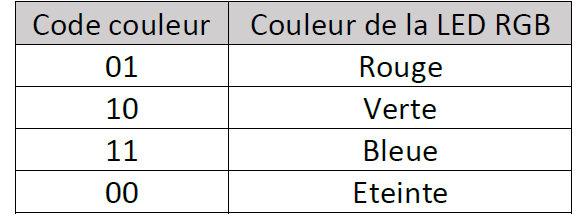
Si rising\_btn\_0 n'est pas actif (rising\_btn\_0='0'), mais que btn\_0 est '1', cela signifie que le bouton est maintenu enfoncé, donc la LED rouge est allumée (s\_led\_out\_R est '1') et la LED verte est éteinte (s\_led\_out\_V est '0').Enfin, si rising\_btn\_0 n'est pas actif et que btn\_0 est '0', cela signifie que le bouton est relâché, donc la LED rouge est allumée (s\_led\_out\_R est '1') et la LED verte est éteinte (s\_led\_out\_V est '0').

Lorsque state\_led est '0', les deux LEDs sont éteintes (s\_led\_out\_R et s\_led\_out\_V sont '0').

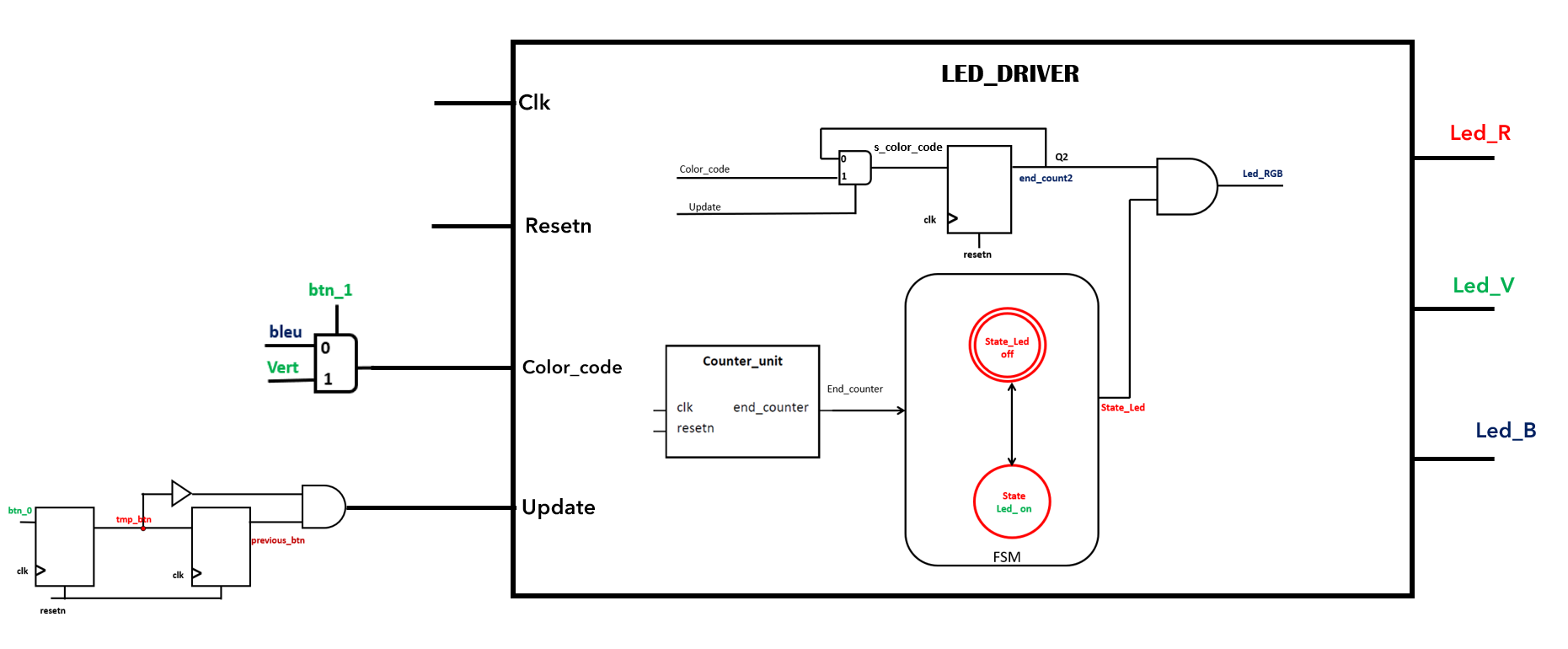
1. Créez un module de pilotage d’une LED RGB en RTL. Ce dernier doit permettre de faire clignoter une LED RGB connectée en sortie d’une couleur définie par un code couleur donné en entrée. Le changement de couleur de la LED RGB n’a lieu que si un signal *update* est reçu.



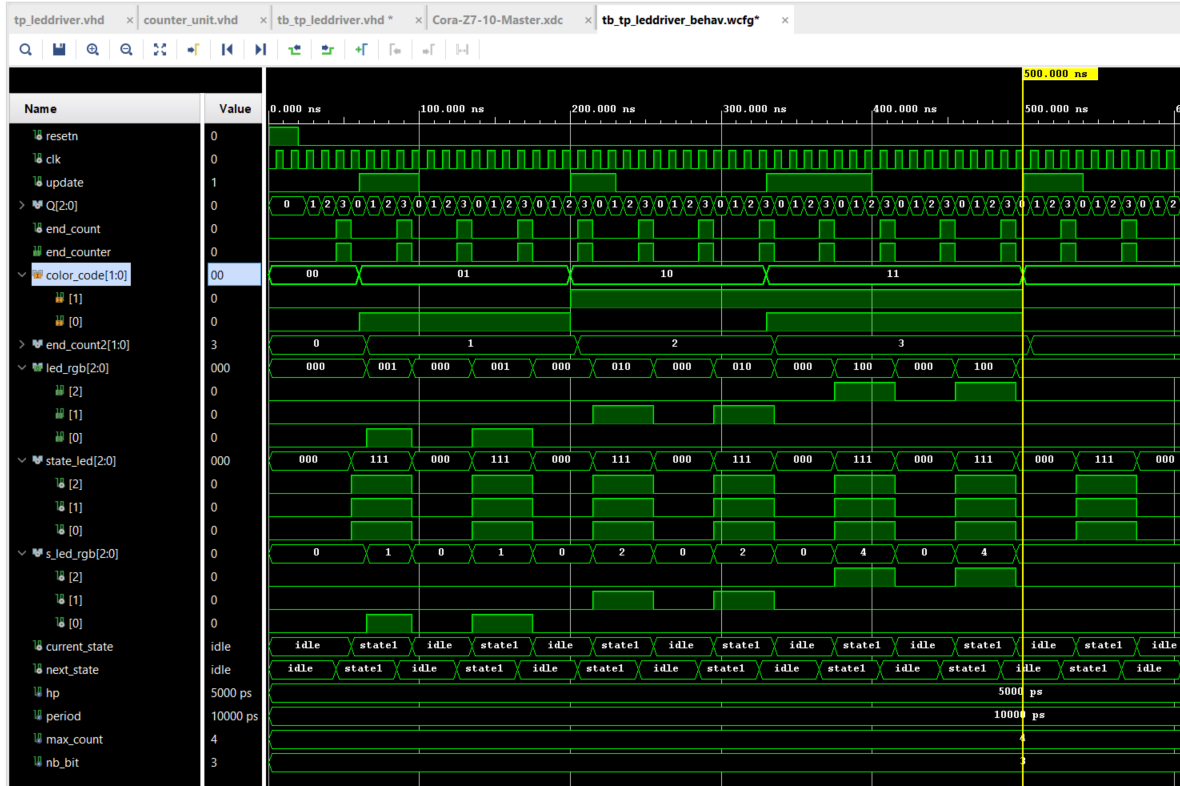
A titre d’exemple voici une table mettant en relation un code couleur avec une couleur de la LED RGB :



Pour créer le module de pilotage d’une LED RGB qui permet de clignoter une led RGB connectée en sortie d’une couleur définie par un code couleur, nous utiliserons un compteur pour contrôler le clignotement de la LED. Nous utiliserons le code couleur fourni : "00" pour éteint, "01" pour rouge, "10" pour verte, et "11" pour bleue. La partie de la logique combinatoire mappera les différents codes couleur sur les valeurs correspondantes de la LED RGB (rgb\_led). Le clignotement de la LED est ajouté en utilisant un compteur (counter2). Le compteur compte jusqu'à une valeur max\_count pour allumer la LED, puis l'éteint. Cette valeur peut être ajustée en fonction de la fréquence souhaitée du clignotement.



Son code vhld+testbench donne le résultat suivant :



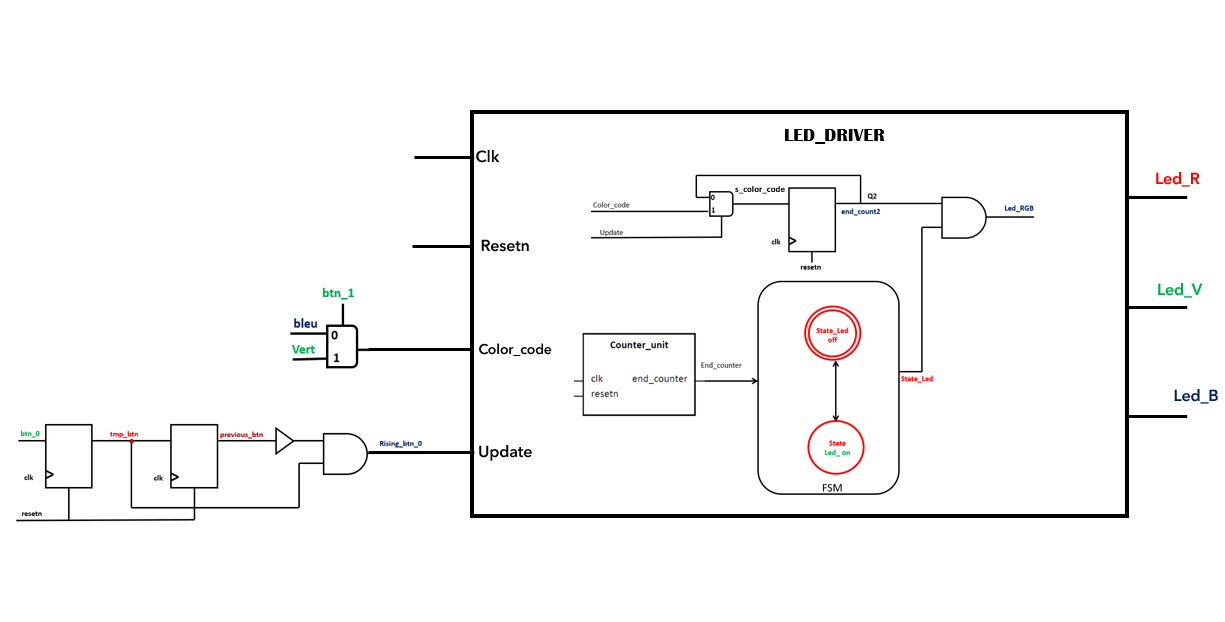
Sur le chronogramme, on voit bien le changement de color\_code qui évolue de 00,01, 10, 11. On remarque que la led rouge est allumé durant end\_count=01 et color\_code=01. La led verte est allumé durant end\_count=10 et color\_code=10. La led bleue est allumé durant end\_count=11 et color\_code=11. La led éteinte lorsque color\_code=00

1. Ajouter la logique nécessaire pour piloter les entrées/sorties de votre module.

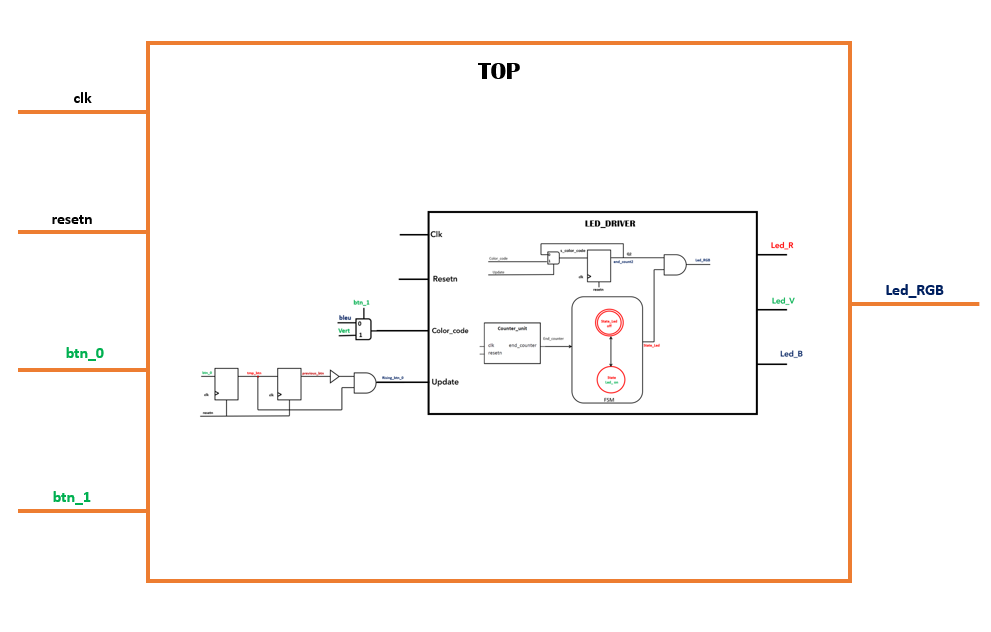
- Le signal *update* doit recevoir 1 uniquement lorsque le bouton \_0 vient d’être appuyé, maintenir le bouton enfoncé ne doit pas maintenir le signal *update* à 1

- Le signal *color\_code* doit recevoir soit le code couleur « vert » si le bouton\_1 est pressé, il doit recevoir le code couleur « bleu » sinon

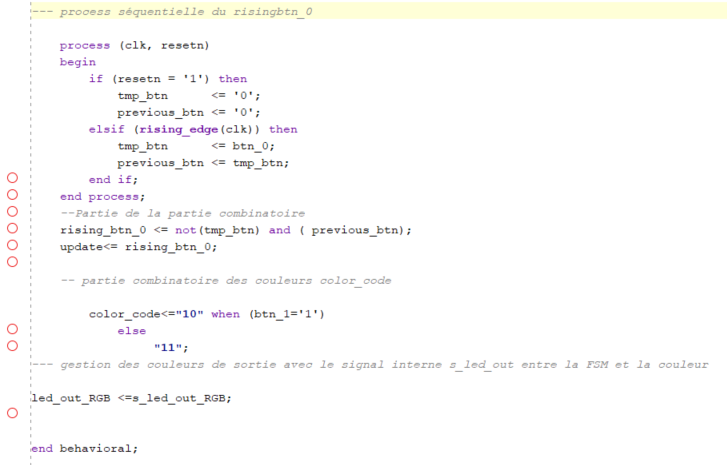
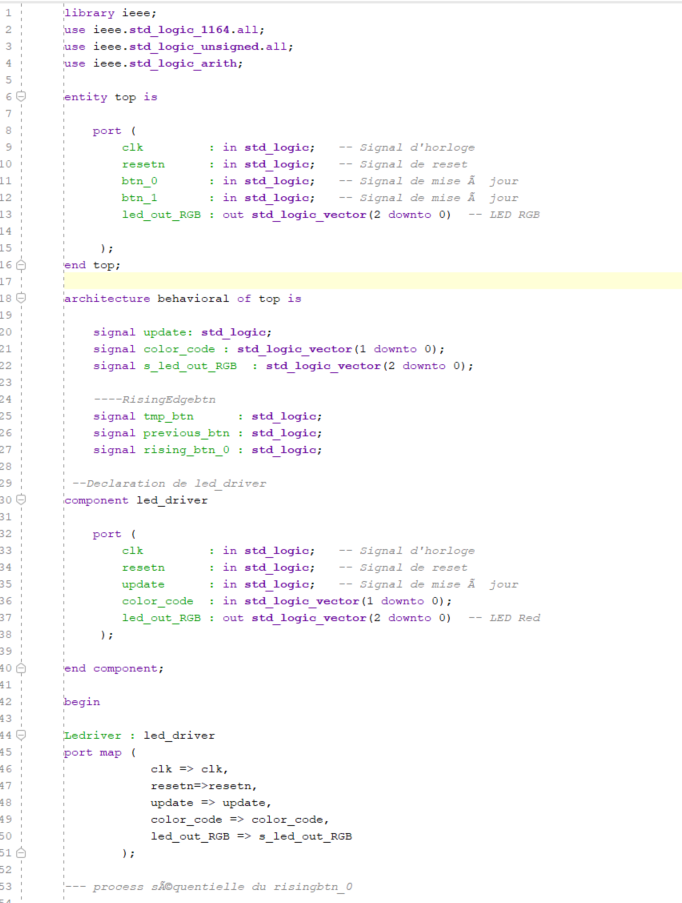
- Les LED R, G et B sont connectées avec les couleurs respectives de la LED\_0



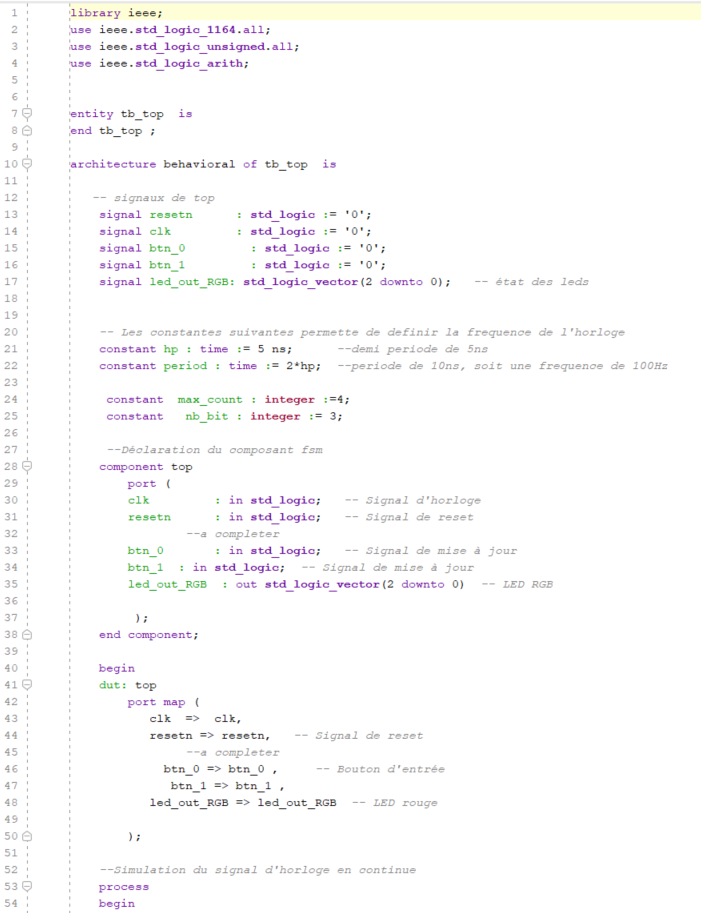
Pour cette partie, on ajoute un bloc pour le contrôle de l’update, ce bloc est relié au bouton btn\_0. Avec un multiplexeur, on gère le signal *color\_code qui* doit recevoir soit le code couleur « vert » si le bouton\_1 est pressé, et il doit recevoir le code couleur « bleu » sinon. Ce multiplexeur est relié au bouton btn\_1. Le module top représente l’ensemble avec le led\_driver. Ses signaux d’entrées sont : clk, resetn, btn\_0 et btn\_1. Le signal de sortie est la led RGB.

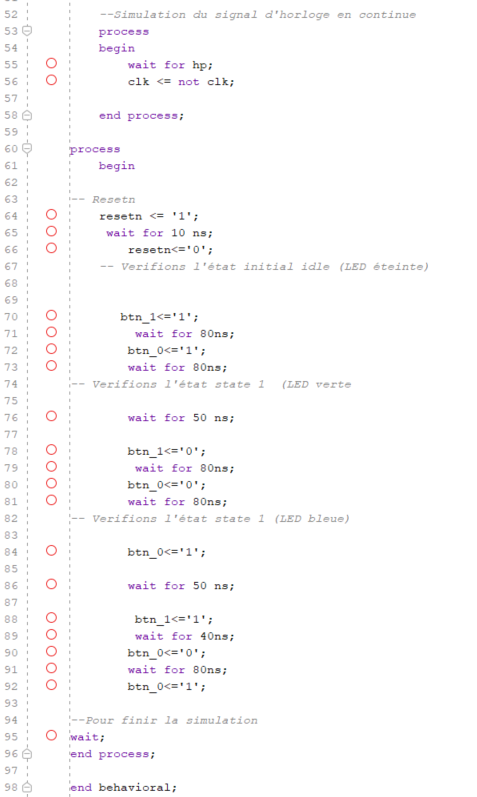


1. Ecrivez le code VHDL correspondant à votre architecture.



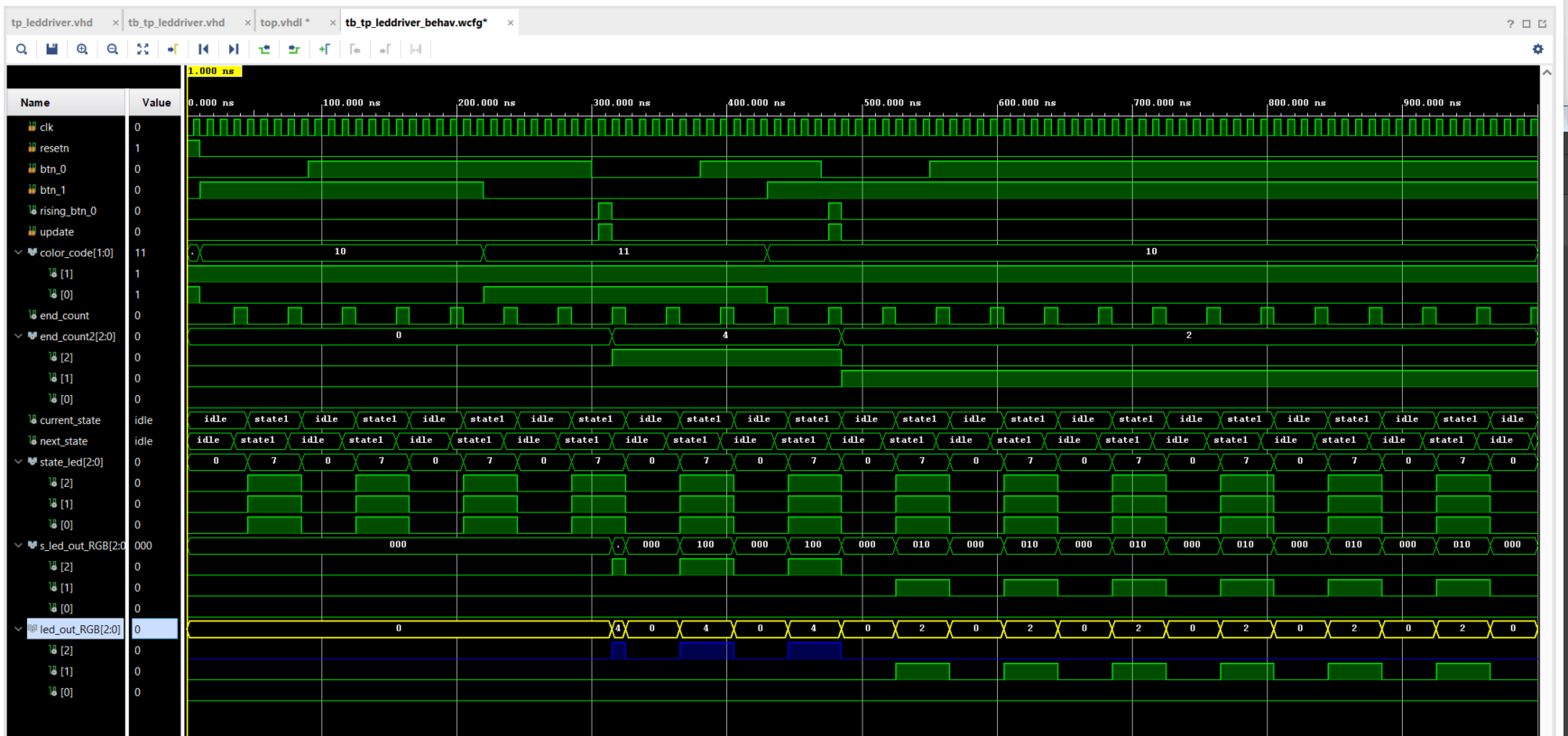
1. Ecrivez un testbench pour tester votre design.



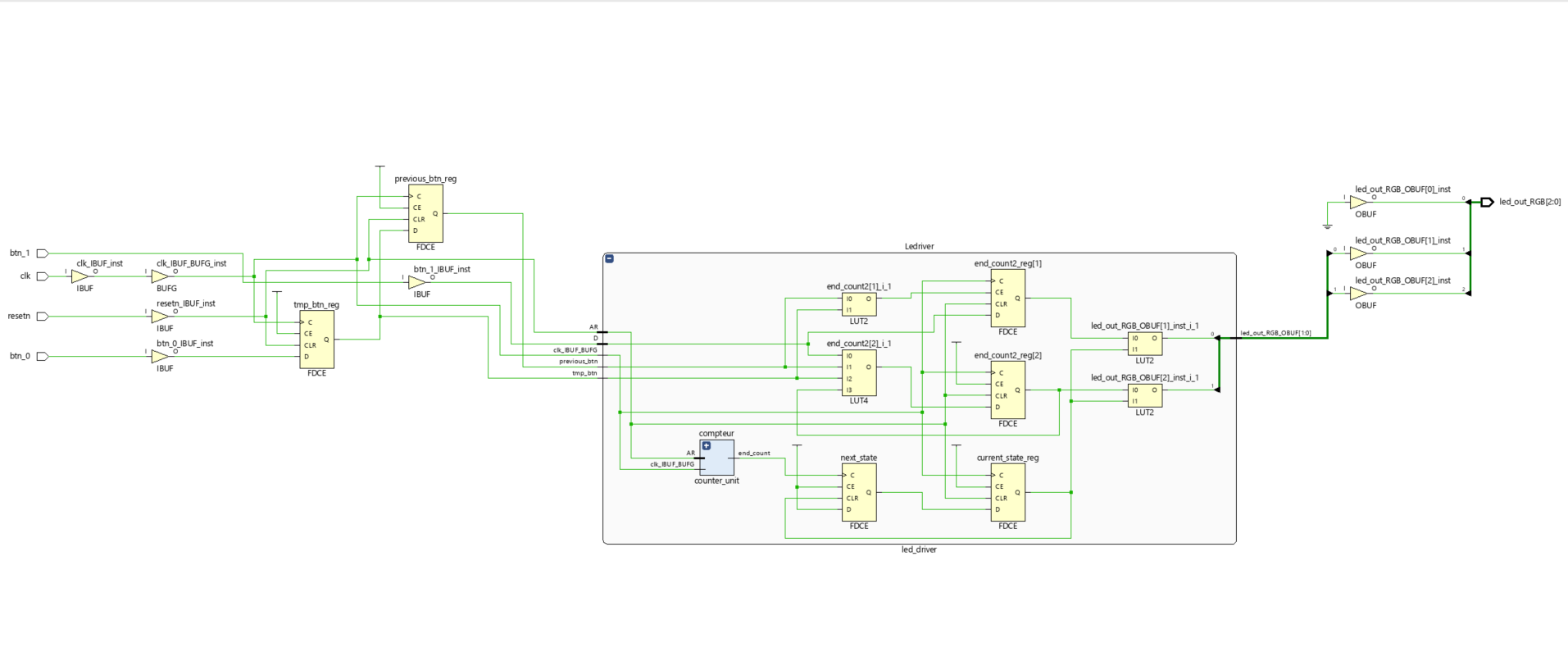


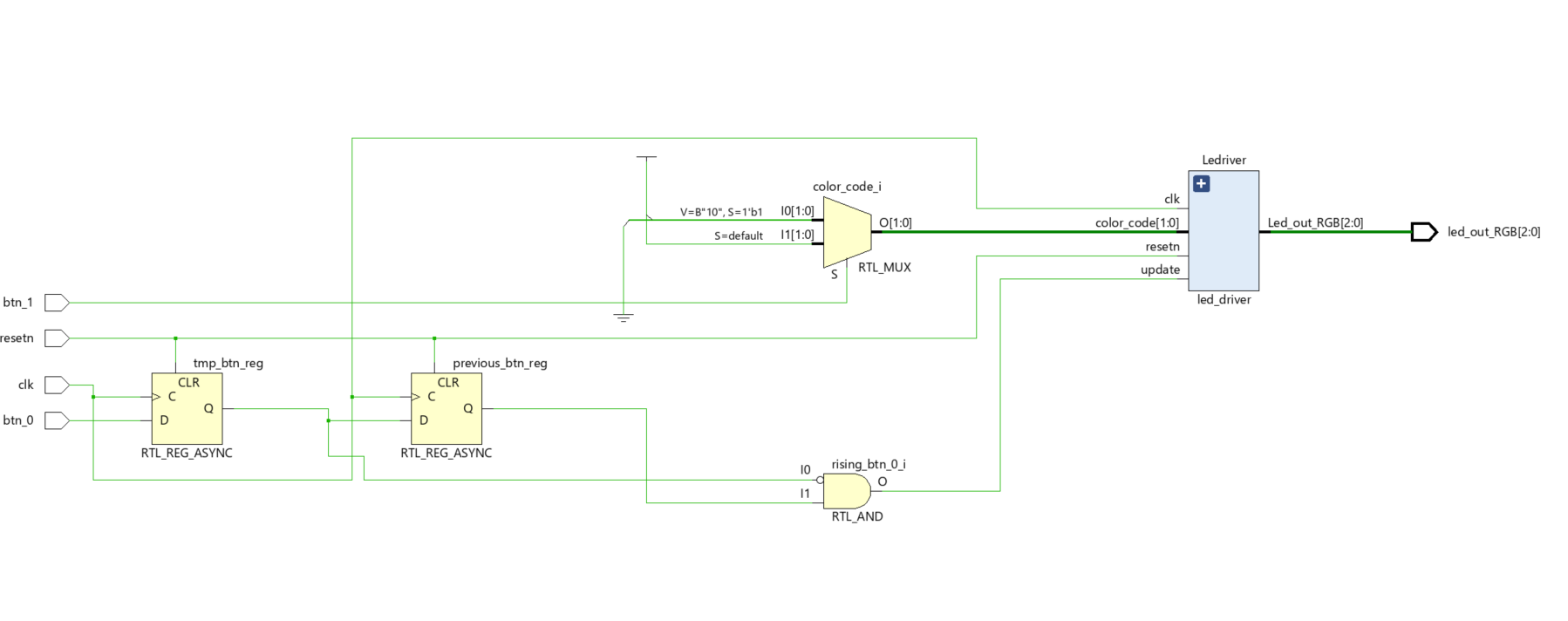
1. Vérifier votre résultat à la simulation.

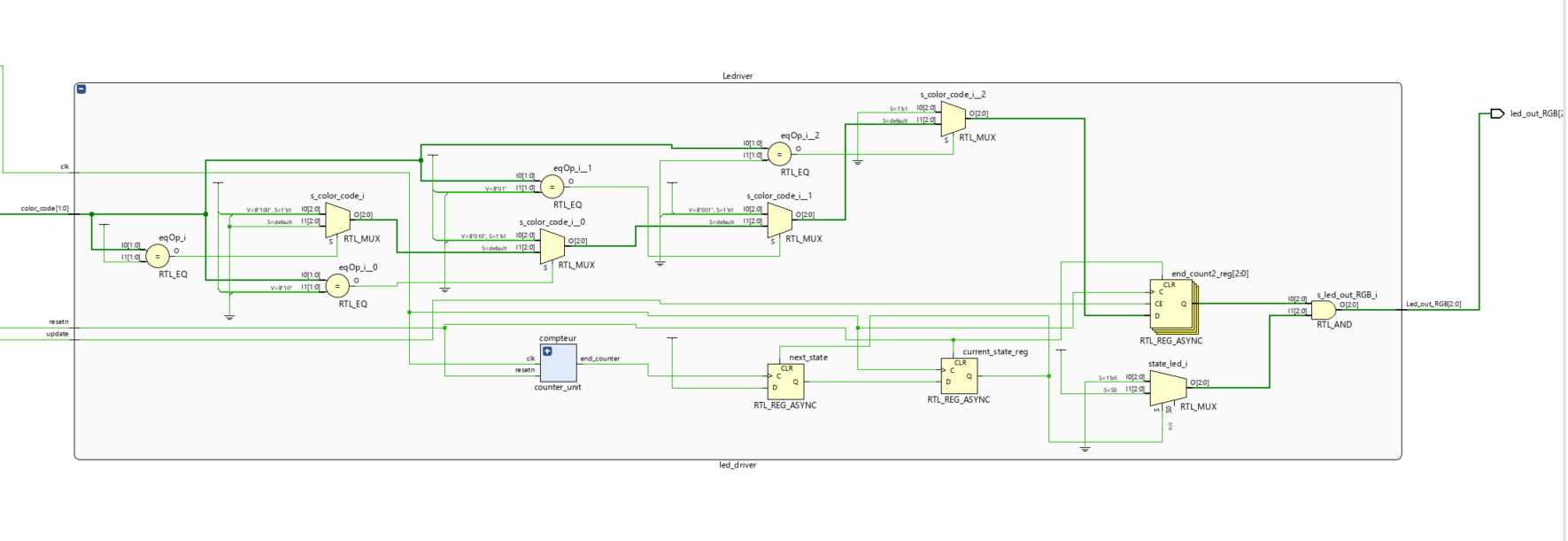
On observe bien le rising\_btn qui évolue lorsqu’on appui sur le btn\_1 et btn\_0. Le signal *update* doit recevoir 1 uniquement lorsque le rising\_btn vient d’être appuyé. Le signal *color\_code* doit reçoit bien le code couleur « vert » si le btn\_1 est pressé, et le code couleur « bleu » sinon. Le chronogramme ci-dessous prouve le bon fonctionnement de la simulation.



1. Réalisez une synthèse et étudiez le rapport de synthèse, les ressources utilisées doivent correspondre à votre schéma RTL.

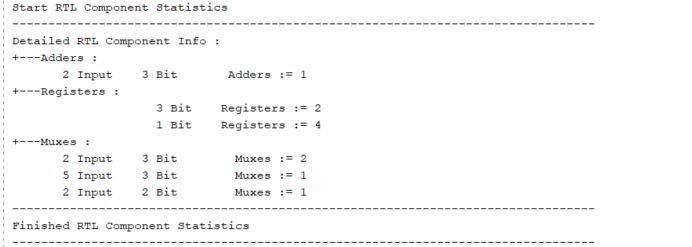
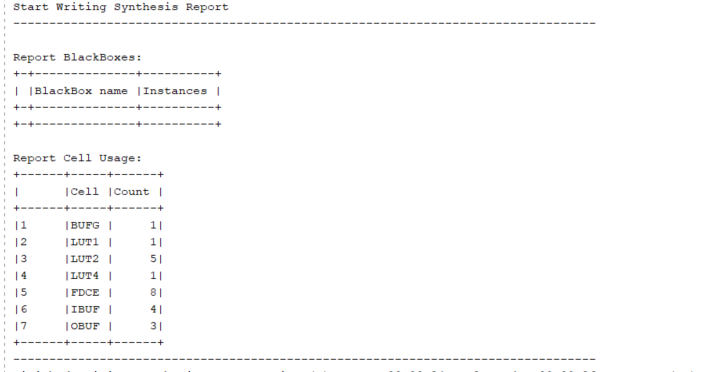






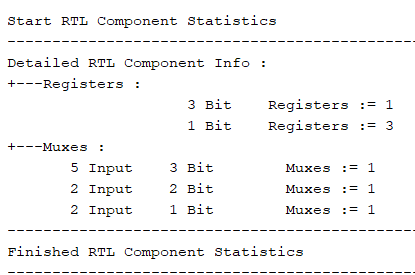
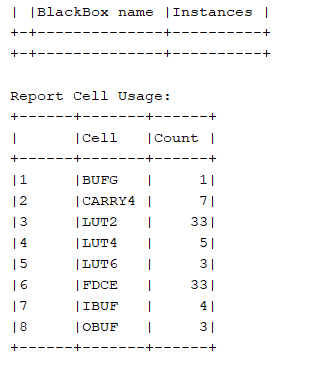
Le schéma est représenté ci-dessus. On retrouve bien notre module Led\_driver dans le quel se trouve notre counter\_unit et le compteur de la gestion du clignotement.

Les ressources correspondent bien à ce que nous avons utilisé pour notre schéma RT. Les éléments utilisés sont répertoriés ici.

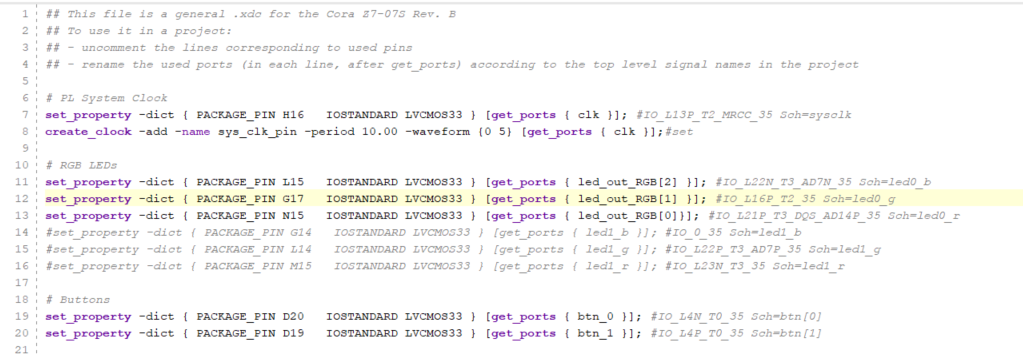
Tous les ressources sont répertoriées ci-dessus. Pour un générique de 4, on a 8 registres (2 pour la gestion des boutons, 1pour la machine d’état, 3 pour le counter2(pour 3 leds) et 2 pour le counter\_unit).

Lorsque le générique est de 200000000, on a donc 33 registres (2 pour la gestion des boutons, 1pour la machine d’état, 2 pour le counter2(2leds) et 28 pour le counter\_unit).

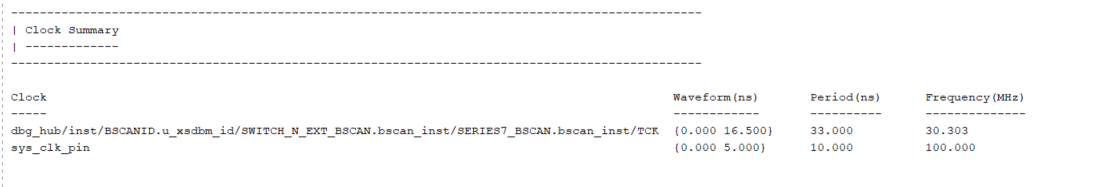
 

1. Effectuez le placement routage et étudiez les rapports.

Connexion des différents ports dans le fichier de contraintes.

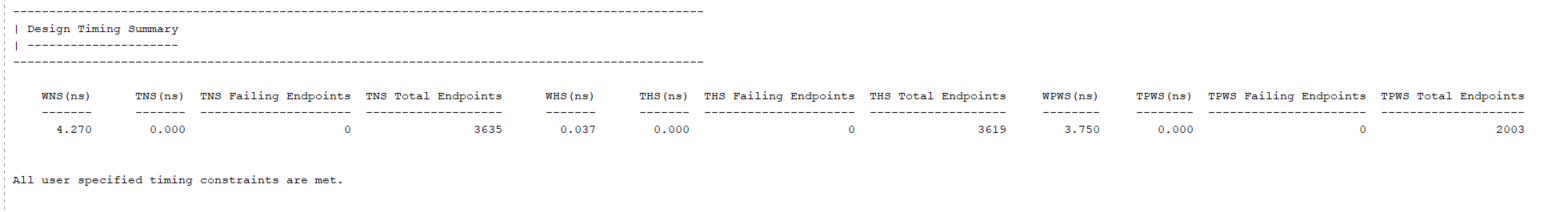


Le rapport de temps est le suivant :



On vérifie bien que la période est à 10ns et la fréquence est de 100MHz.

Les valeurs dans le THS et TNS sont à 0, il n’y a pas de violation du set up et du hold. Pas de métastabilité.

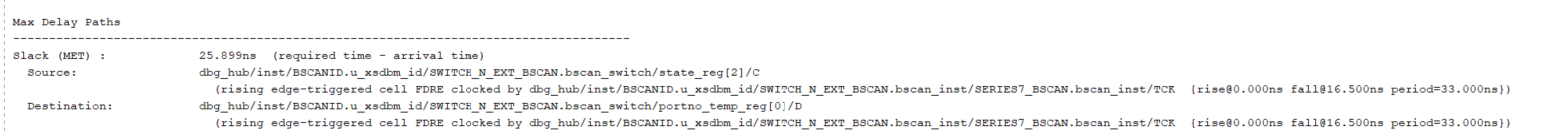


Le chemin critique est :

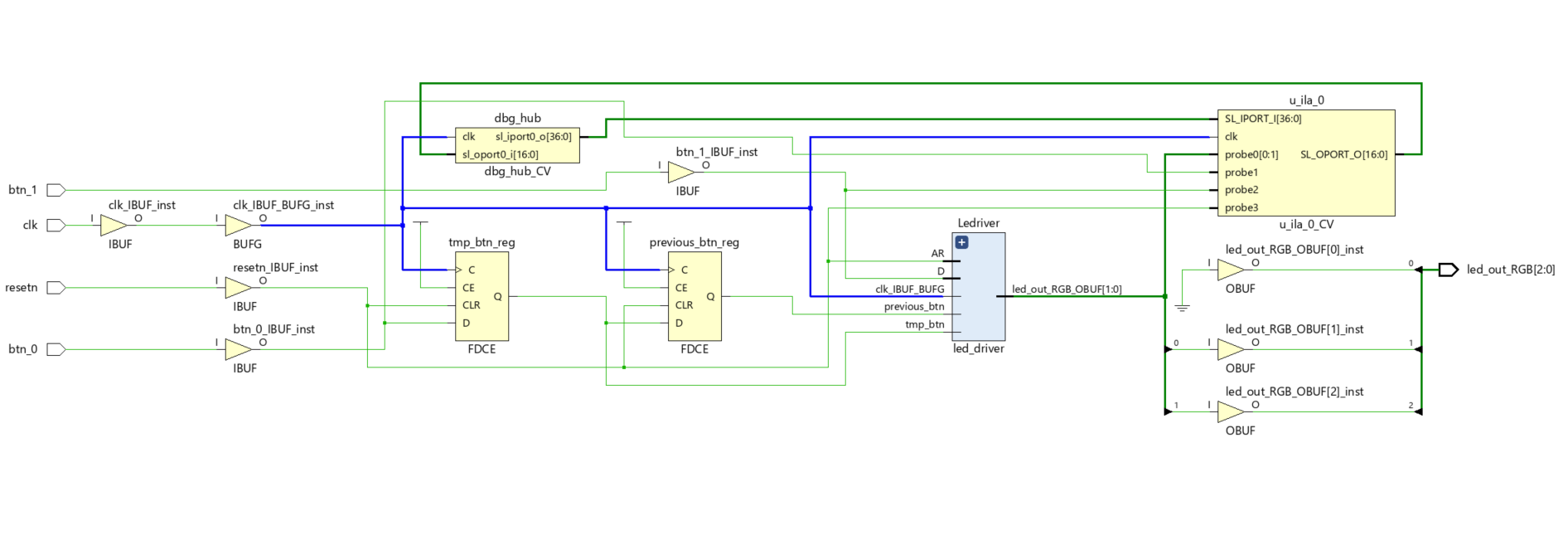
Slack (MET) : 25.899ns (required time - arrival time)

Source: dbg\_hub/inst/BSCANID.u\_xsdbm\_id/SWITCH\_N\_EXT\_BSCAN.bscan\_switch/state\_reg[2]/C(rising edge-triggered cell FDRE clocked by dbg\_hub/inst/BSCANID.u\_xsdbm\_id/SWITCH\_N\_EXT\_BSCAN.bscan\_inst/SERIES7\_BSCAN.bscan\_inst/TCK {rise@0.000ns fall@16.500ns period=33.000ns})

Destination: dbg\_hub/inst/BSCANID.u\_xsdbm\_id/SWITCH\_N\_EXT\_BSCAN.bscan\_switch/portno\_temp\_reg[0]/D

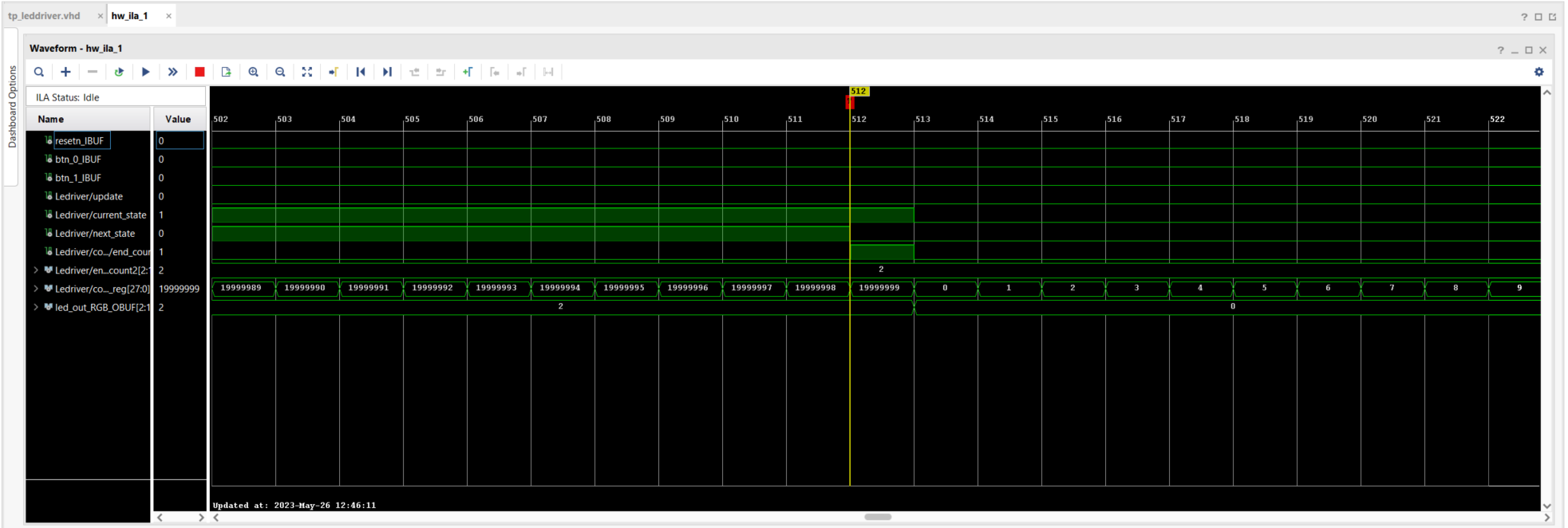


1. Générez le bitstream et vérifiez que vous avez le comportement attendu sur carte.

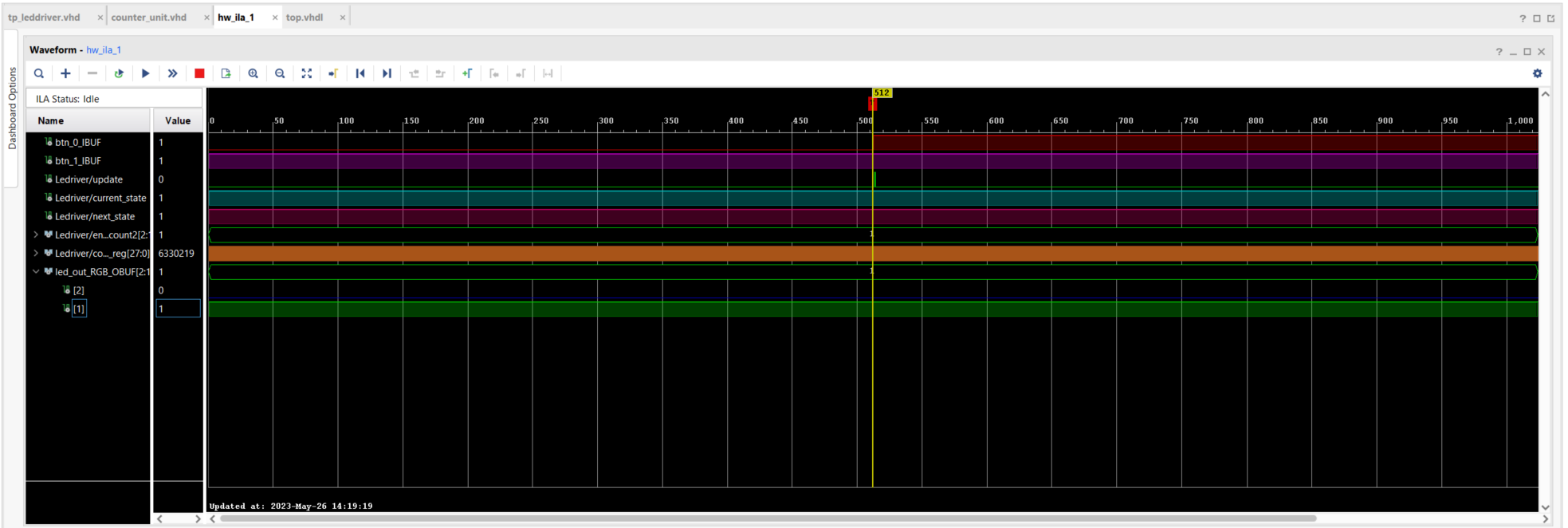


La simulation du bitstream donne les résultats suivants.

Aucun bouton appuyé.



Lorsque j’appuie sur le btn\_1 =1, la led verte clignote.



Lorsque j’appuie sur le btn\_0 =1, la led bleue clignote.



La vidéo de démonstration pour l’illustrer.