## Segment 2

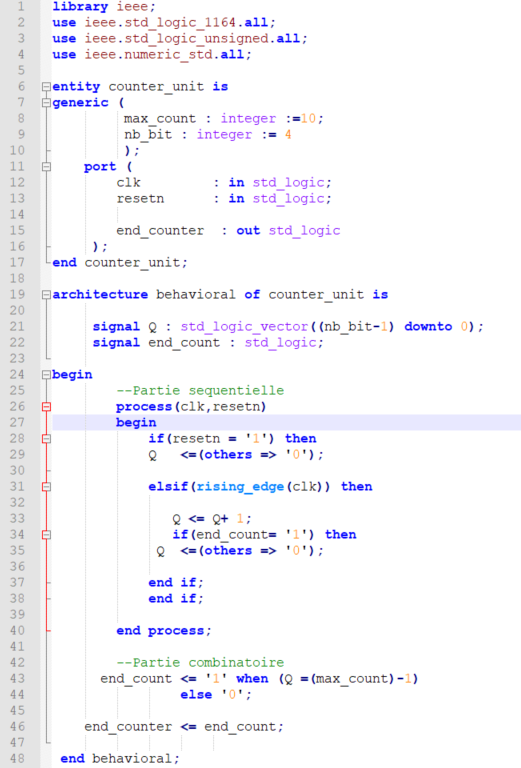
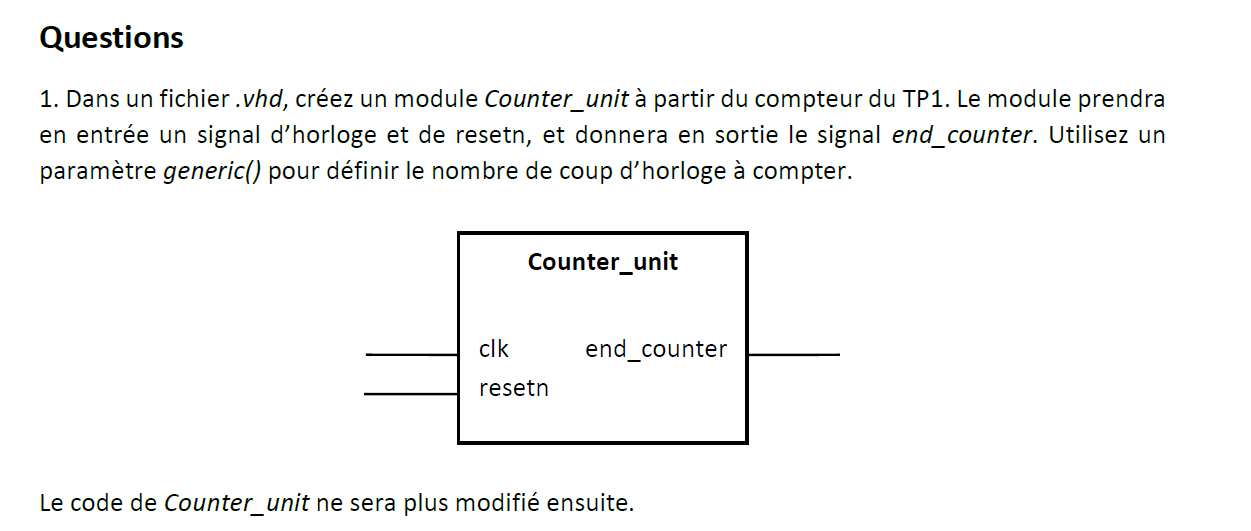
**TP3 : FSM**

**Objectif**

L’objectif de cet TP est faire clignoter une LED en utilisant un compteur de temporisation. Un compteur de temporisation permet de compter le nombre de coup d’horloge nécessaire pour attendre un temps voulu. En connaissant la fréquence de l’horloge il est possible de déterminer combien de périodes d’horloge il faut compter pour attendre 3 secondes par exemple.

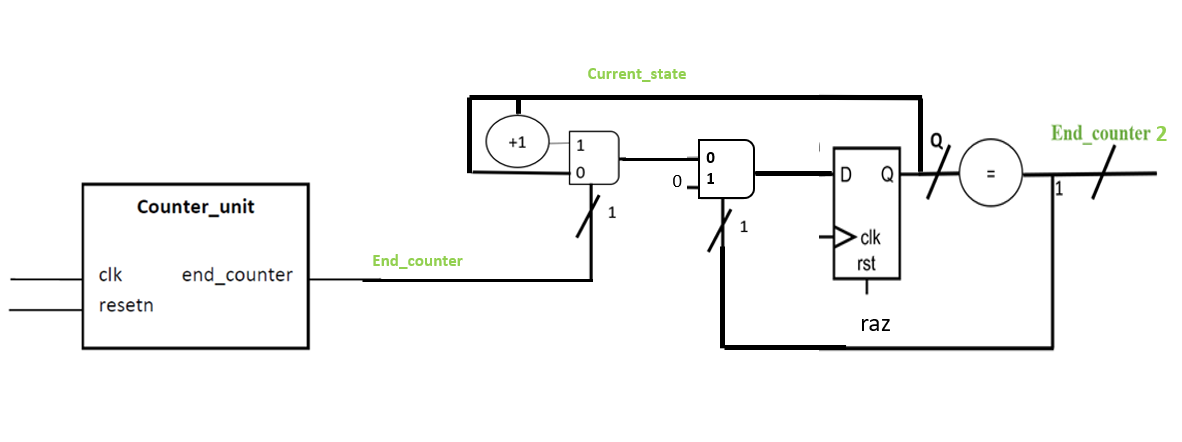
**Questions**

1. Dans un fichier *.vhd*, créez un module *Counter\_unit* à partir du compteur du TP1. Le module prendra en entrée un signal d’horloge et de resetn, et donnera en sortie le signal *end\_counter*. Utilisez un paramètre *generic()* pour définir le nombre de coup d’horloge à compter. Le code de Counter\_unit ne sera plus modifié ensuite.



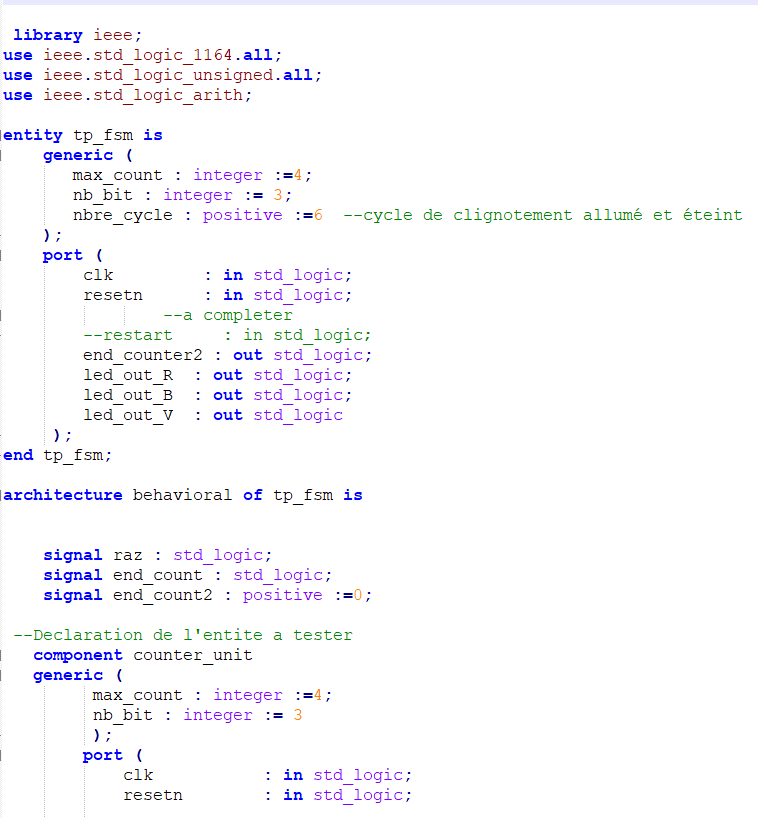
Dans ce module Counter\_unit, nous avons utilisé un paramètre générique max\_count pour définir le nombre de coups d'horloge à compter. Le compteur counter\_unit est une valeur non signée de 4 bits qui compte jusqu'à max\_count - 1. Lorsque le compteur atteint max\_count - 1, le signal end\_counter est mis à '1' pour indiquer que le comptage est terminé.

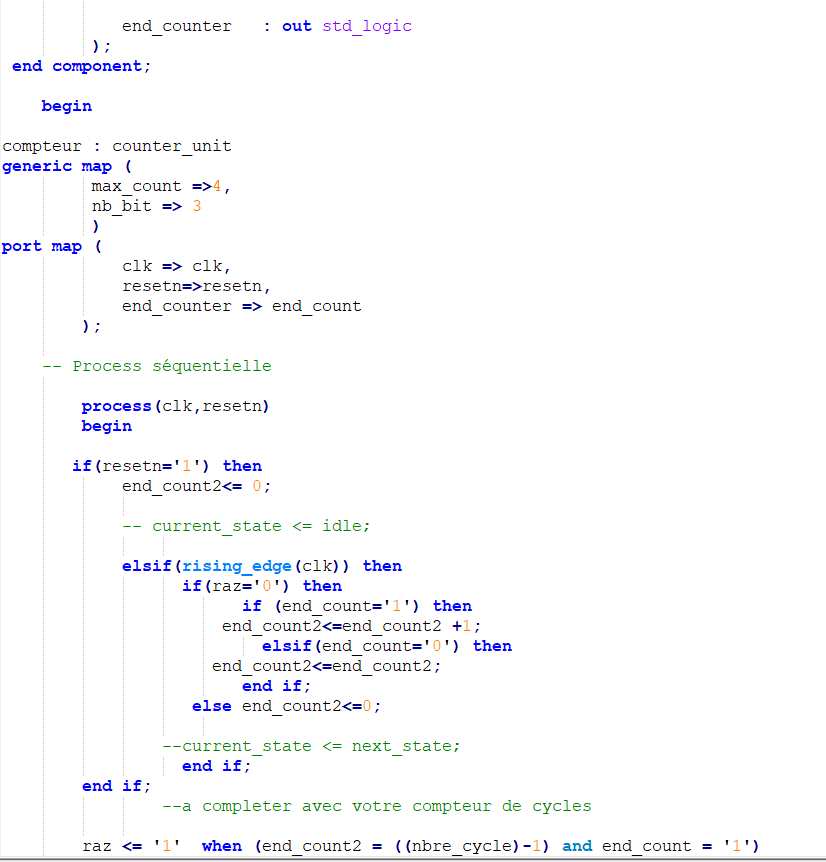
1. En schéma RTL, créez un compteur du signal *end\_counter*. Ce compteur doit permettre de déterminer le nombre de cycles allumé/éteint qui ont été effectués par la LED. Le compteur doit pouvoir être remis à 0, maintenir sa valeur actuelle ou s’incrémenter.

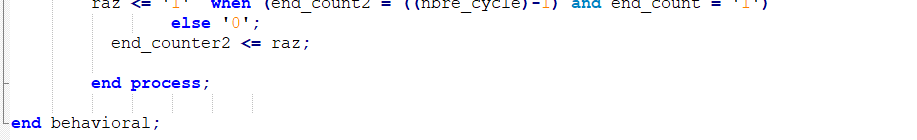


Le passage à 1 de end\_counter 2 avec le paramètre générique permet de définir le temps des états des leds(allumé/éteint). Le nombre de cycles allumé/éteint dans notre tp est de 6. On a trois étapes ou cycle d’allumer et trois d’états d’éteints.

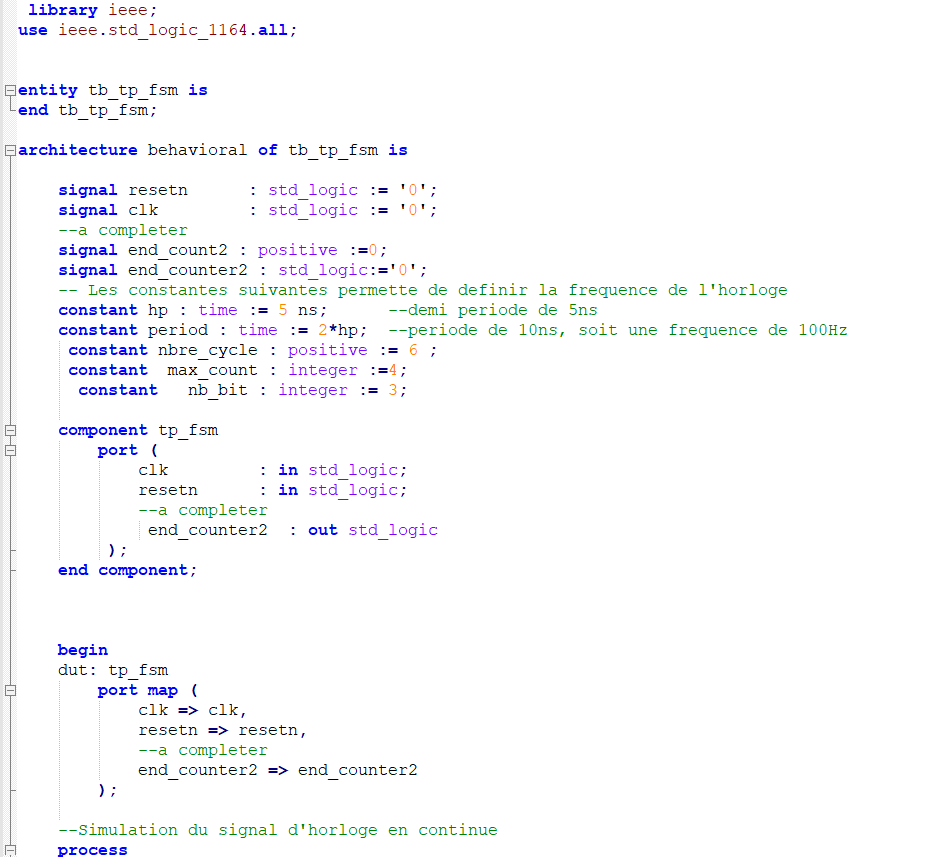
1. Ecrivez un code VHDL décrivant ce compteur de cycle, vous utiliserez le module Counter\_unit.

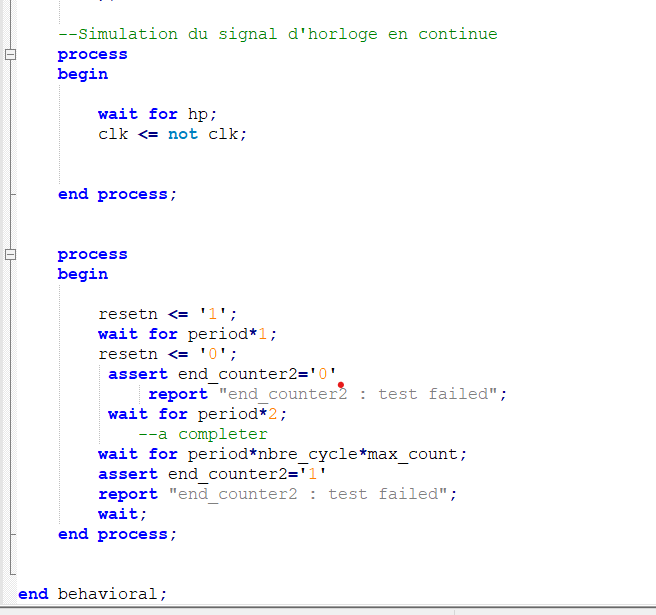


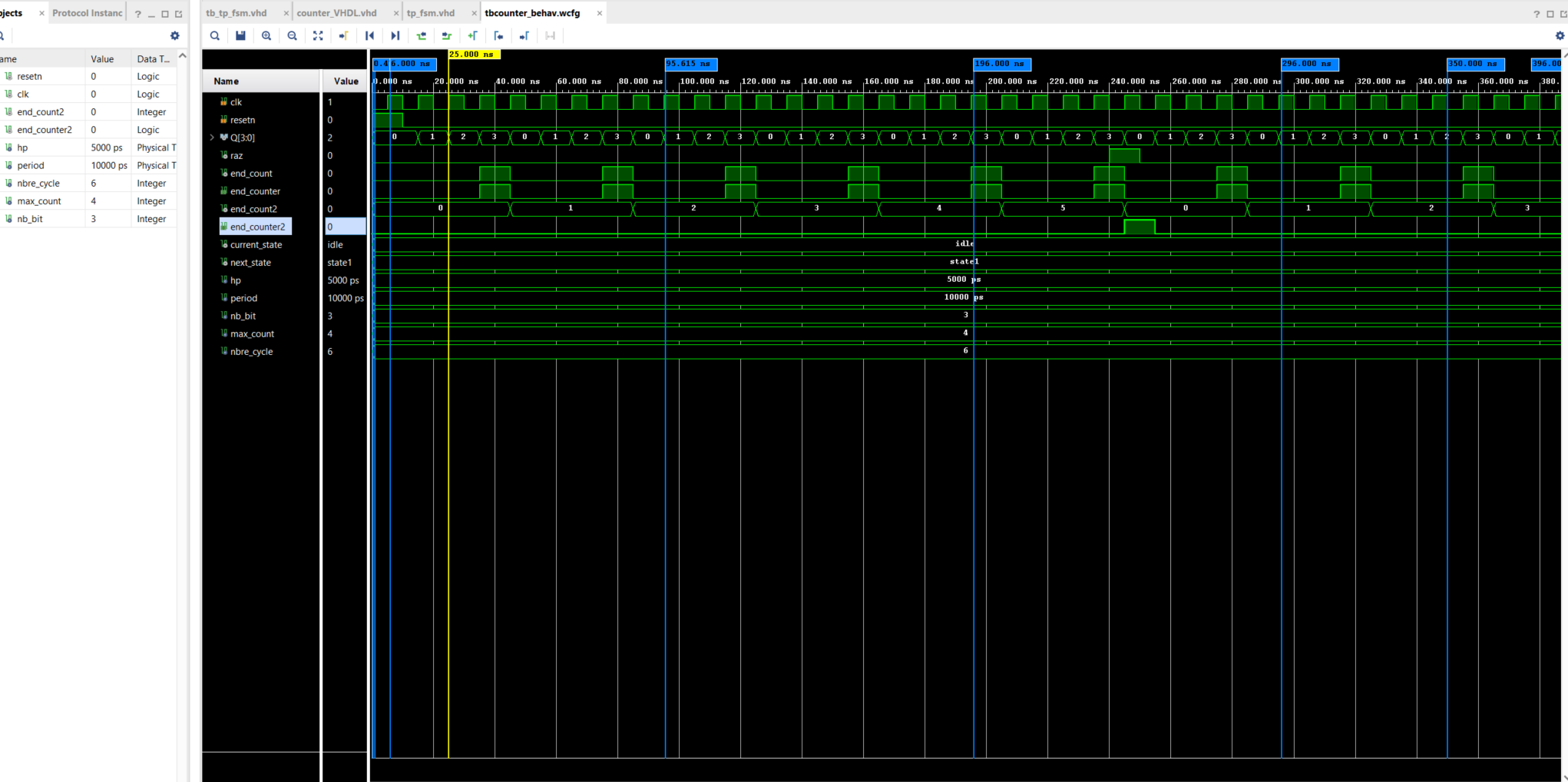




1. Tester votre architecture avec un testbench.

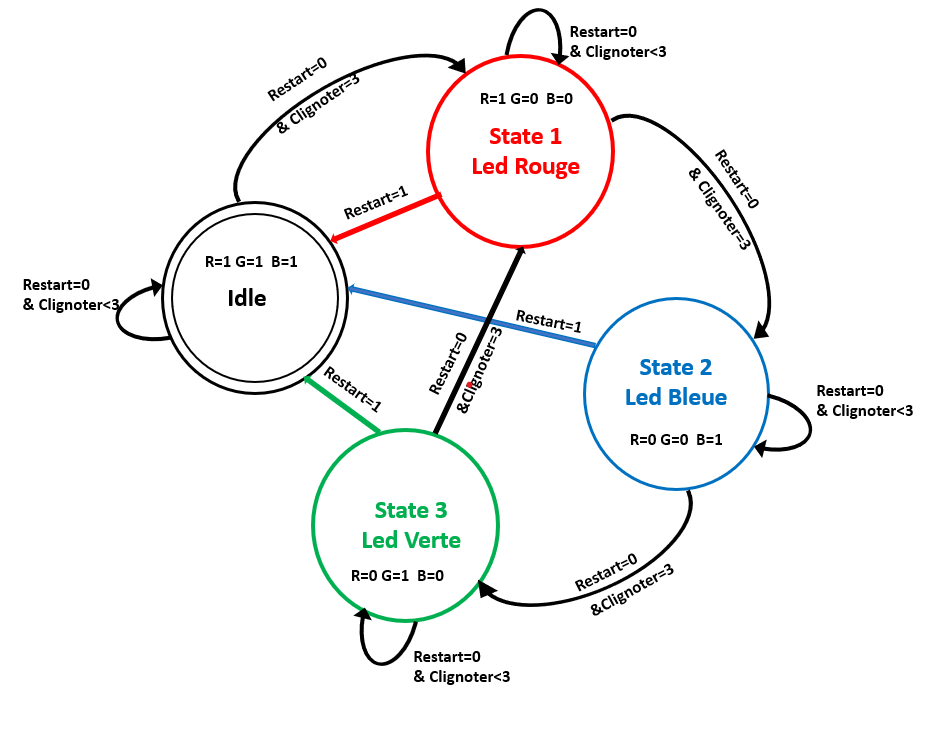






On remarque que le comportement du compteur permet de déterminer le nombre de cycles allumé/éteint qui ont été effectués par la LED. Le compteur est ensuite remis à 0, pour maintenir sa valeur actuelle ou s’incrémenter. On compte 6 fronts montant d’end\_counter et end\_count2. End\_couter2 est alors à 1, puis remise à 0 pour un nouveau cycle.

1. Créez en RTL une machine à états (FSM) permettant de faire clignoter une LED RGB en rouge puis bleu et enfin en vert avant de recommencer le cycle (rouge, bleu, vert, …). Dans chaque état la LED devra clignoter 3 fois. De plus, si le bouton restart est appuyé, on retourne dans l’état initial quel que soit l’état dans lequel on se situe. L’état initial est l’état dans lequel on se situe au démarrage, on passe à l’état rouge après 3 clignotements de la LED en blanc (rouge, vert et bleu actifs en même temps).



1. Listez les signaux d’entrée, de sortie et les signaux internes de votre architecture.

Les signaux d’entrées

-Clk : l’horloge

-Resetn : le reset

-Restart : la remise à l’état initial de notre FSM

Les signaux de sorties

-end\_counter2 : Sortie du compteur

-Led\_out : Leds RGB

les signaux internes

current\_state : état dans lequel se trouve la led actuellement

-next\_state : état dans lequel va se trouver la led au prochain coup d’horloge

-s\_led\_out : signal internpour led\_out RGB

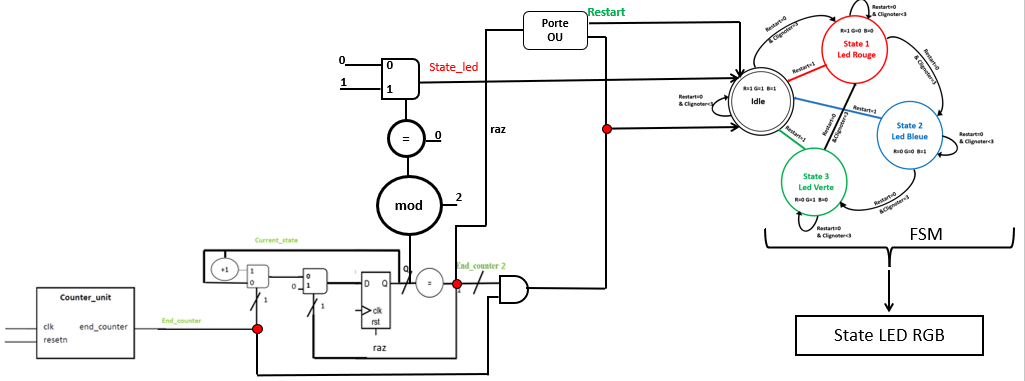
-count\_clig : compte nombre de clignotement

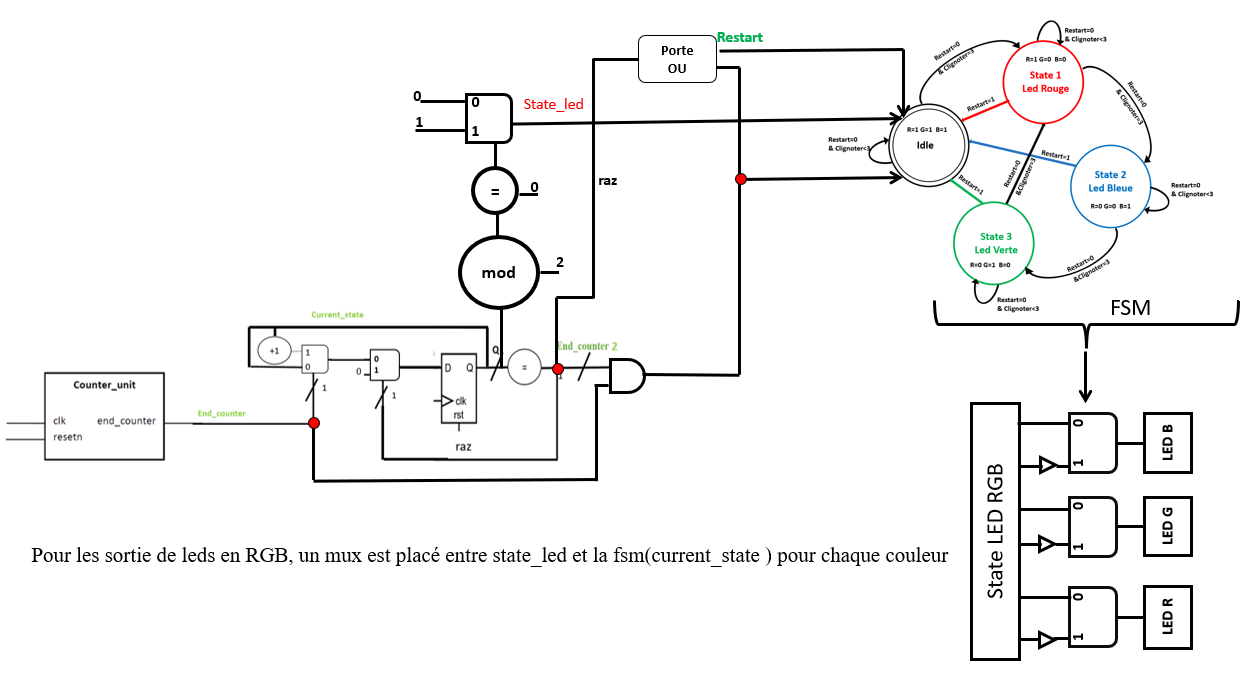
-val\_clig : permet la validation du current\_state à next\_state

-state\_led : état de la led.

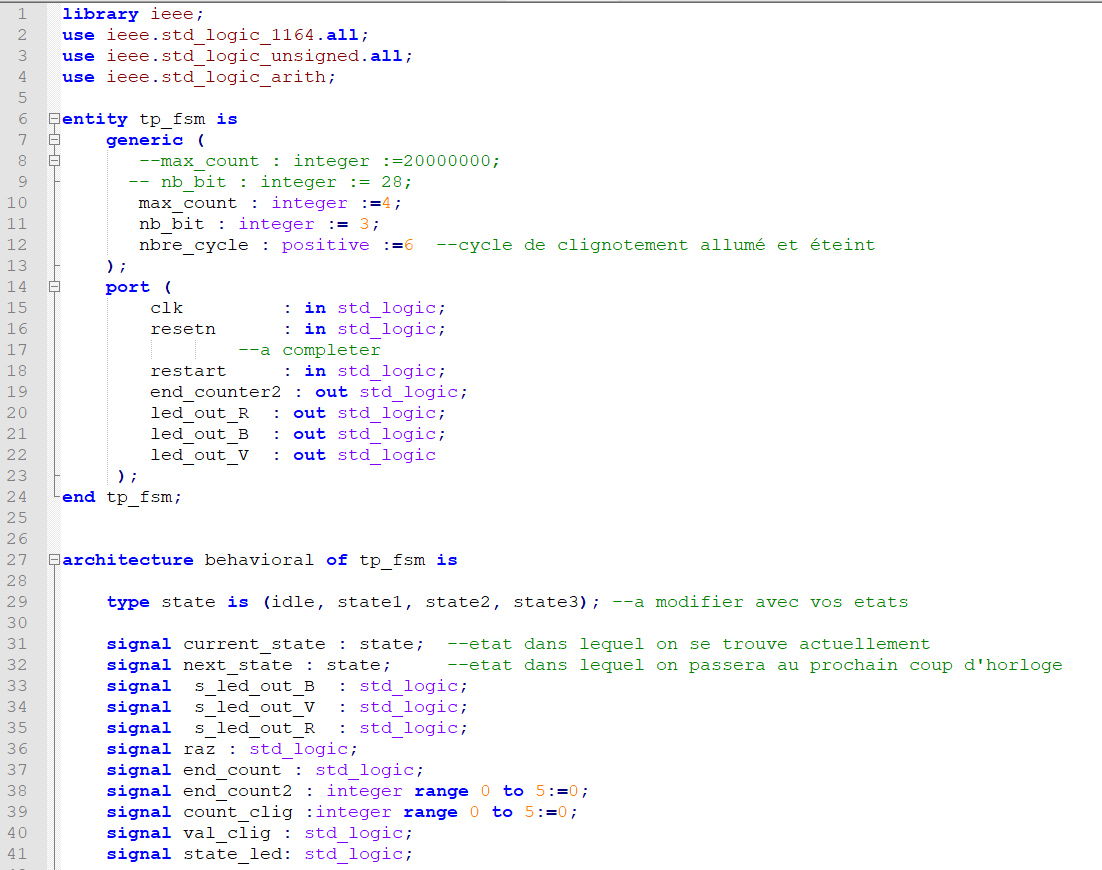
On garde tous les signaux internes déclarer pour le compteur unit et la remise à zéro.

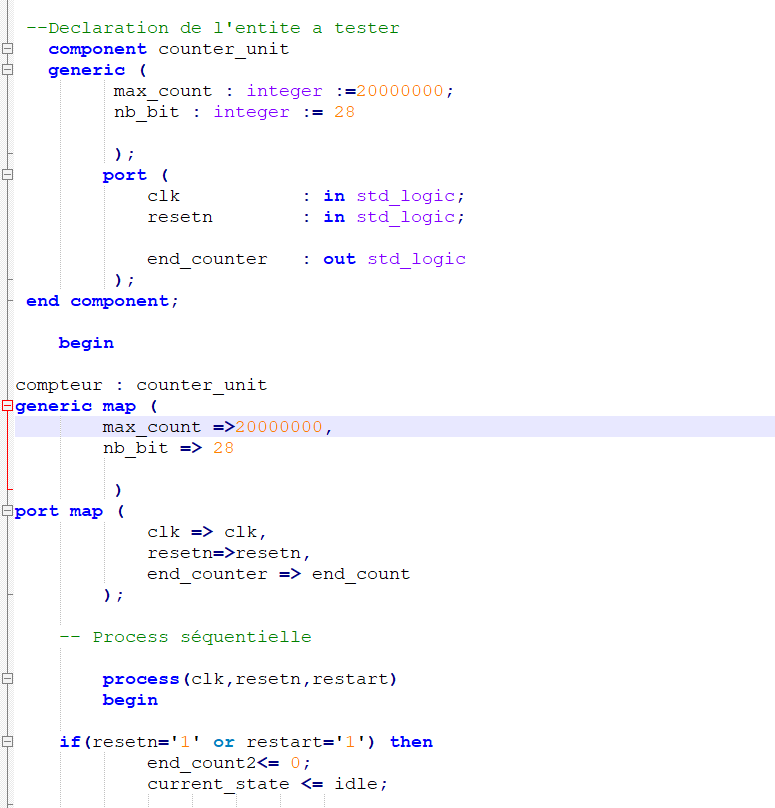
1. Ajoutez à votre code VHDL les éléments que vous venez de créer.

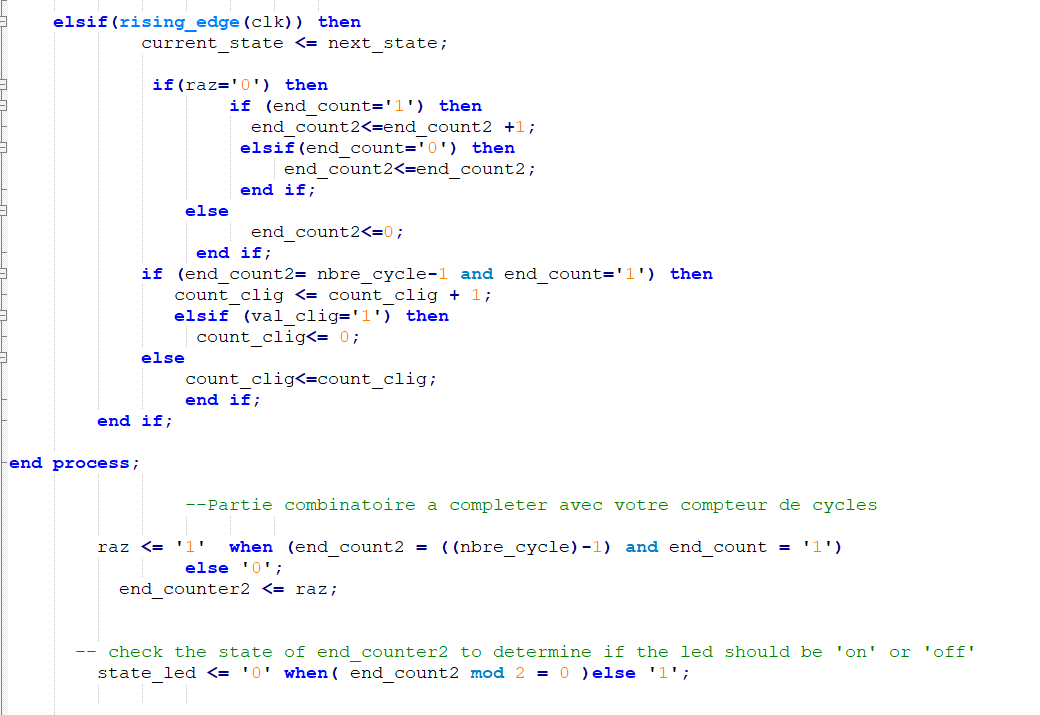


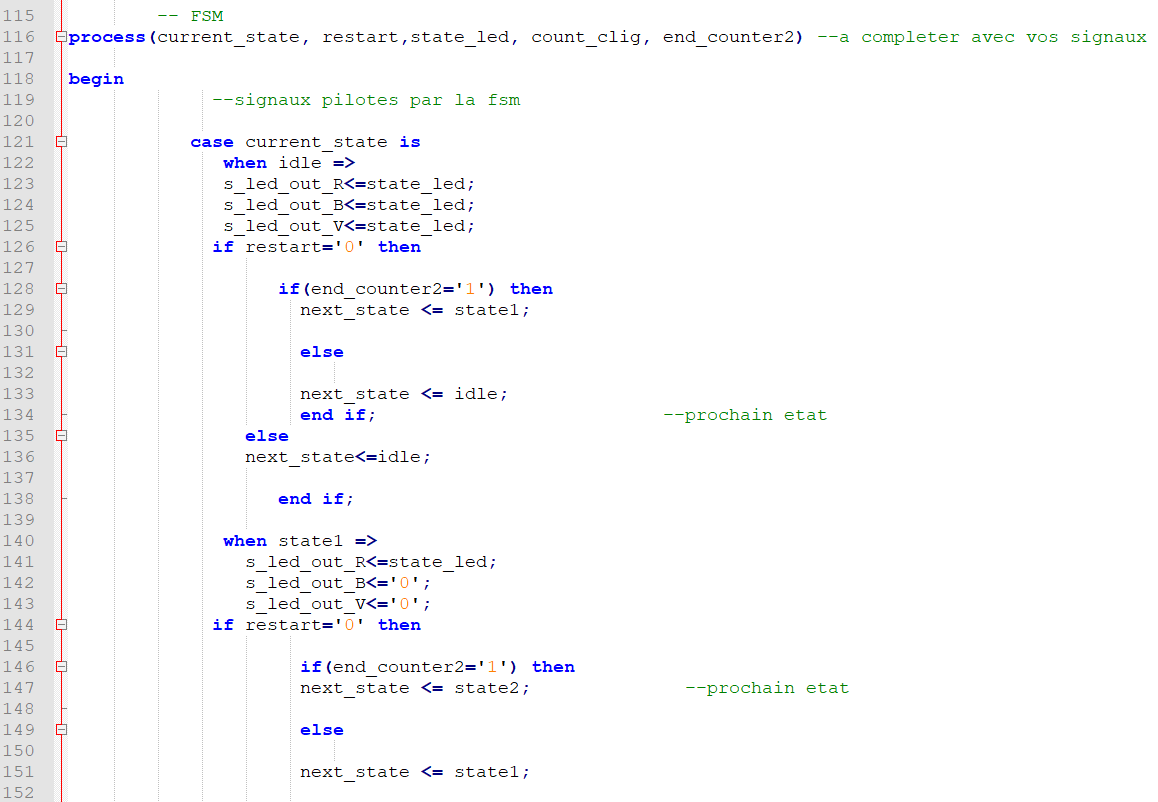


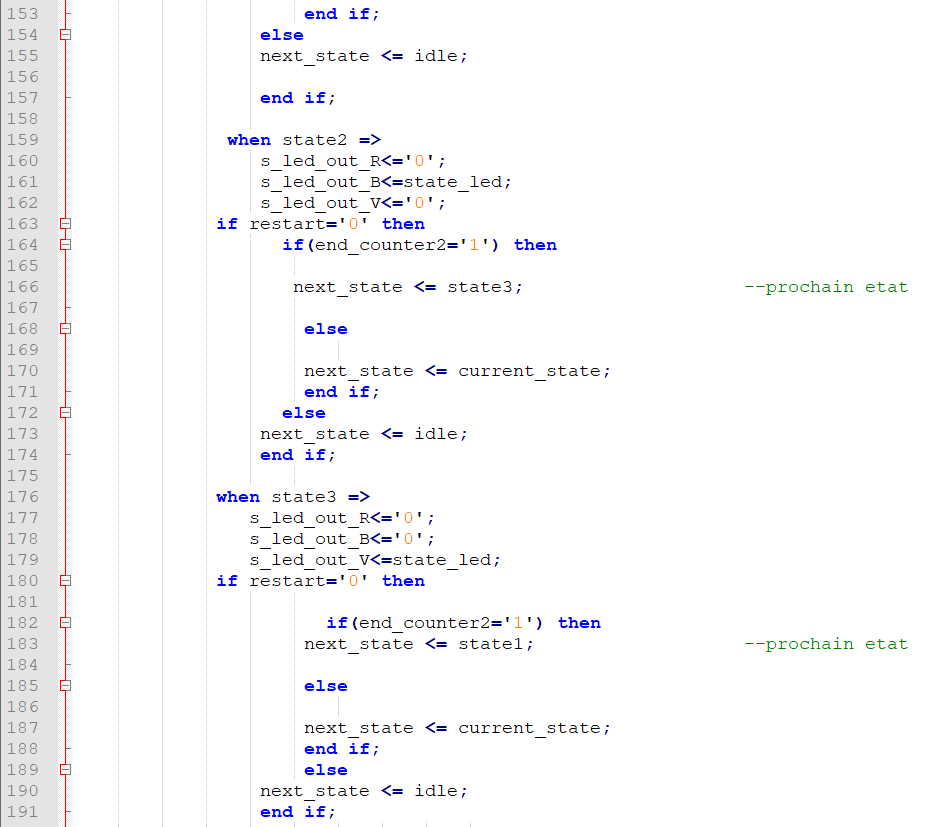
Le schéma de l’ensemble, les signaux clock et resten sont reliés entres eux. Les points rouges sont les points de connexions du même signal. Sur ce schéma RTL, nous rajoutons à la FSM qui gère les états de la led, le module du compteur, des opérateurs logiques et le restart pour un bon fonctionnement. Pour gérer le clignotement de la led(éteint/allumé), un opérateur modulo et un diviseur 2 est utilisé. En sortie d’end\_count2, nous avons 0 pour le pair ou 1 pour les impairs. Les leds sont allumées sur les impairs (state\_led). La FSM vient ensuite gérer les états : initial, led rouge, led bleue et led verte. Le compteur changerait d’état après 6 cycles au front montant.

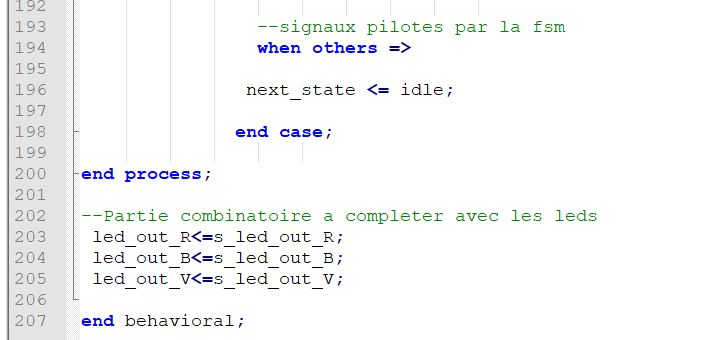




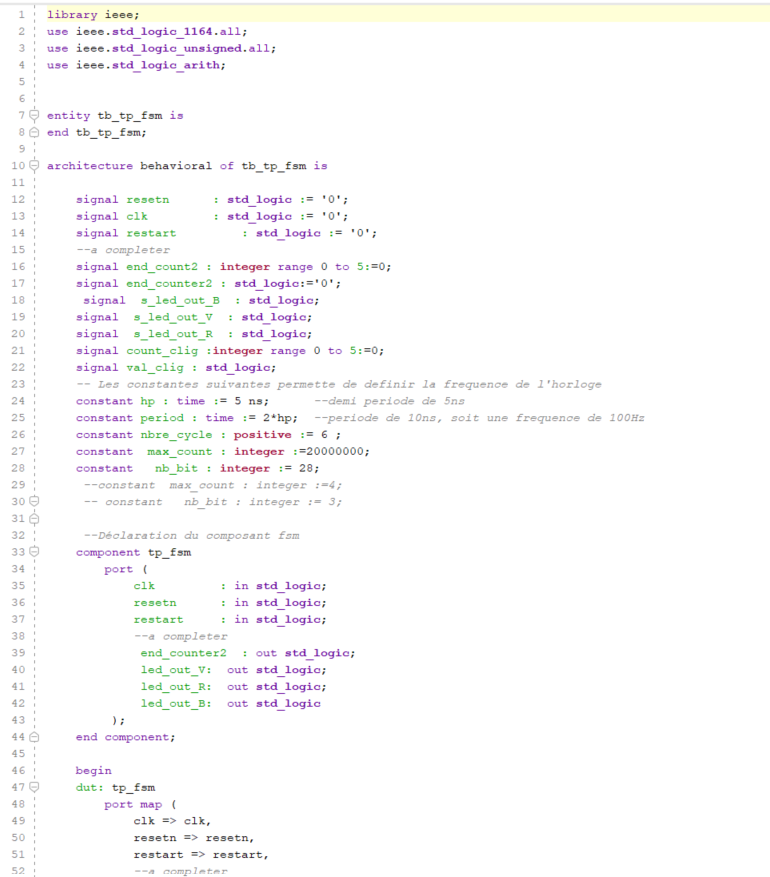


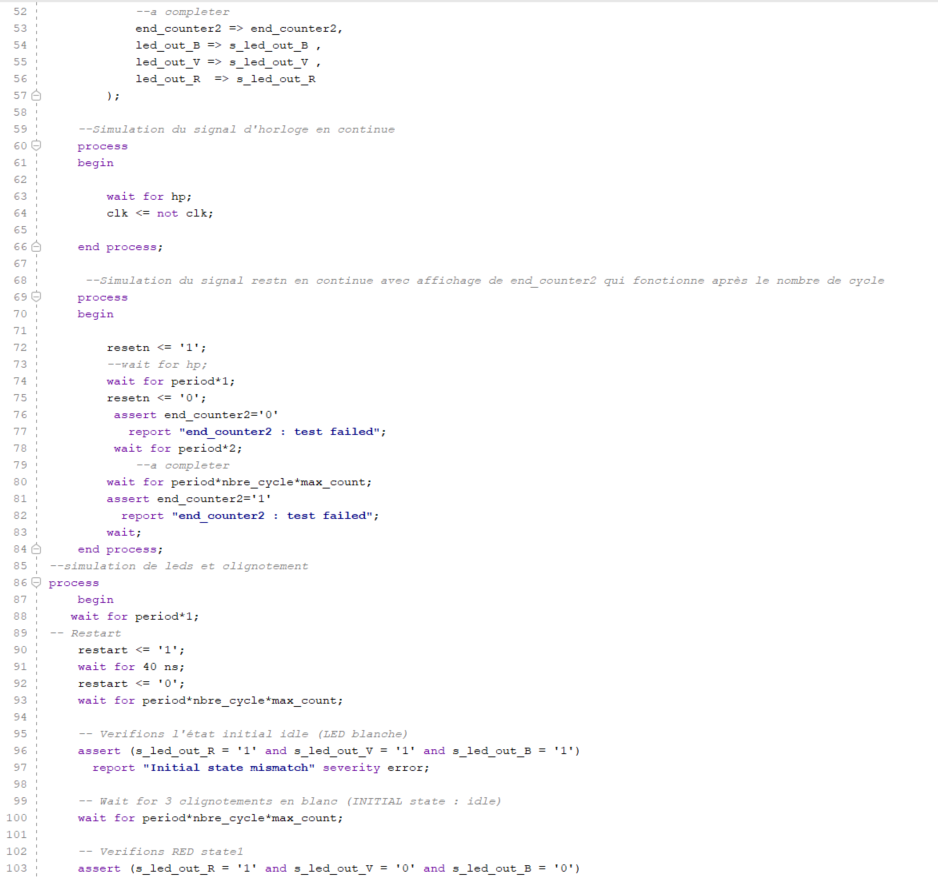


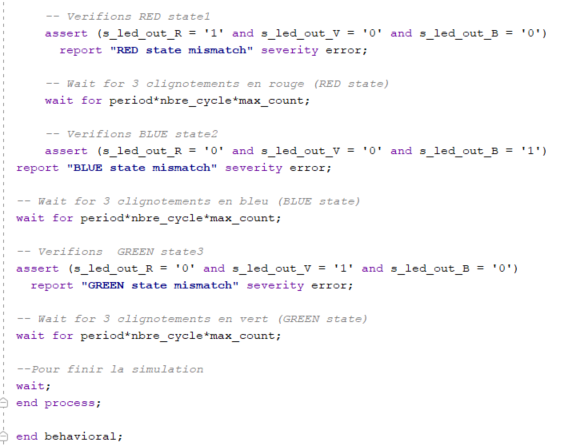


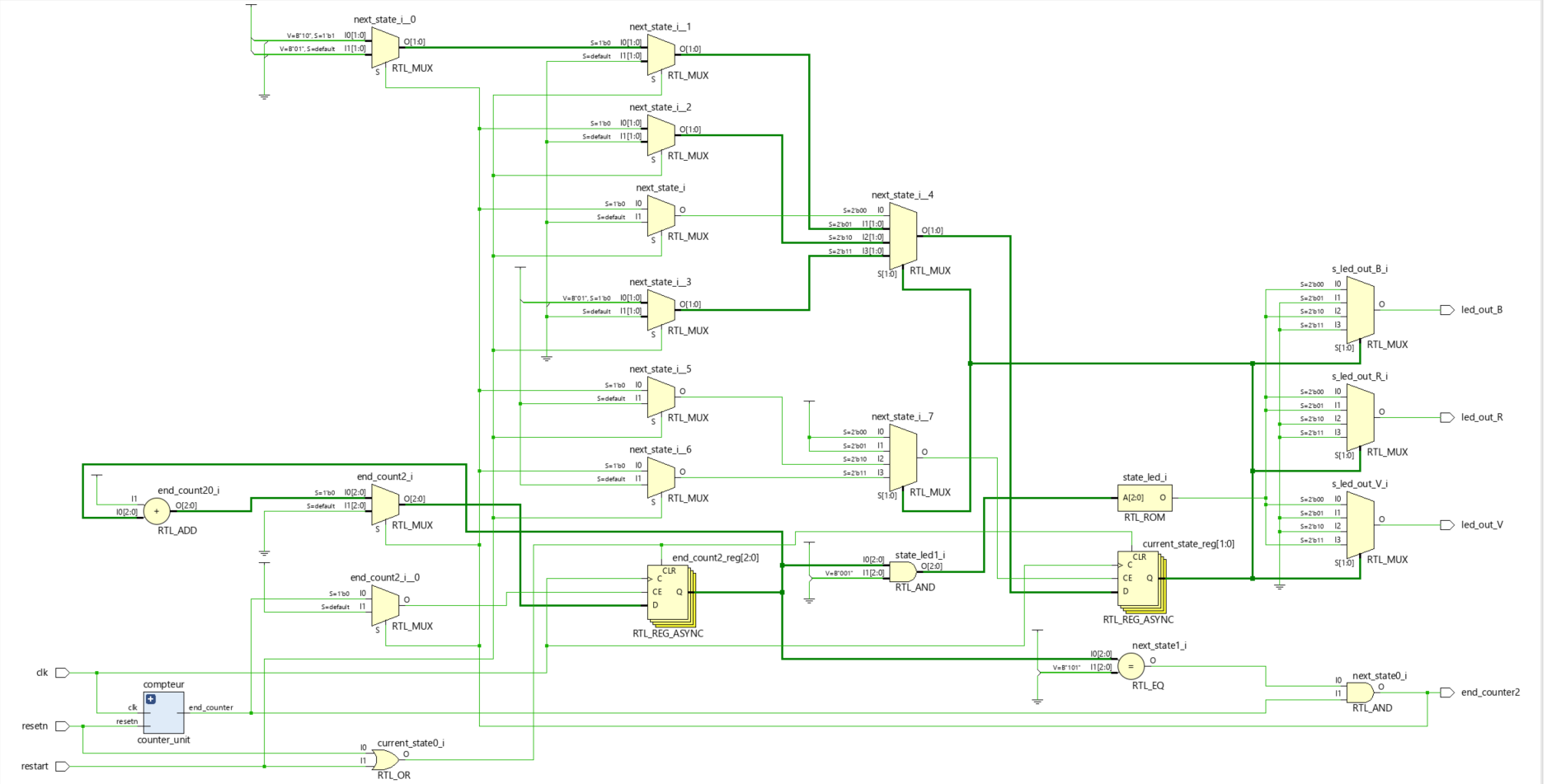


1. Ecrivez un testbench pour tester votre architecture. Vérifiez à la simulation que vous obtenez le résultat attendu.



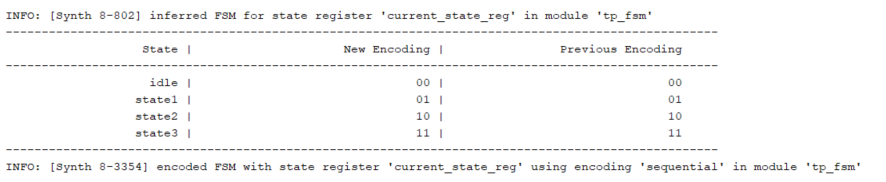




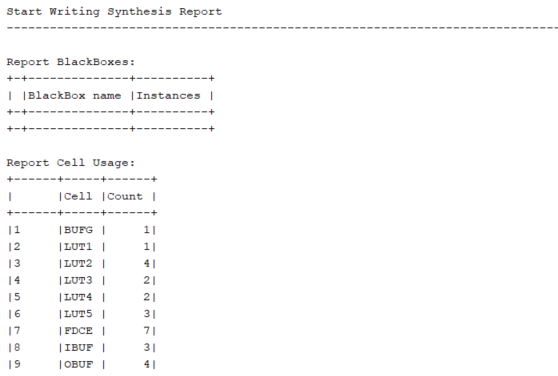
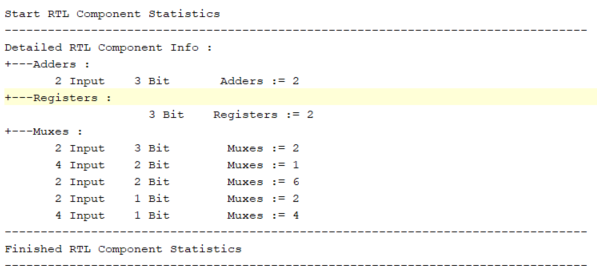


1. Exécutez la synthèse et relevez les ressources utilisées (y compris la FSM). Sur la schématique, identifiez où se situe votre compteur de cycle.

Nous retrouvons dans la synthèse les états de nos leds. Ils sont stockés dans un registre de 2 bits.

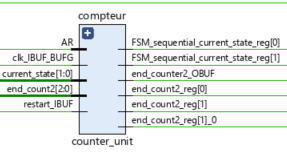


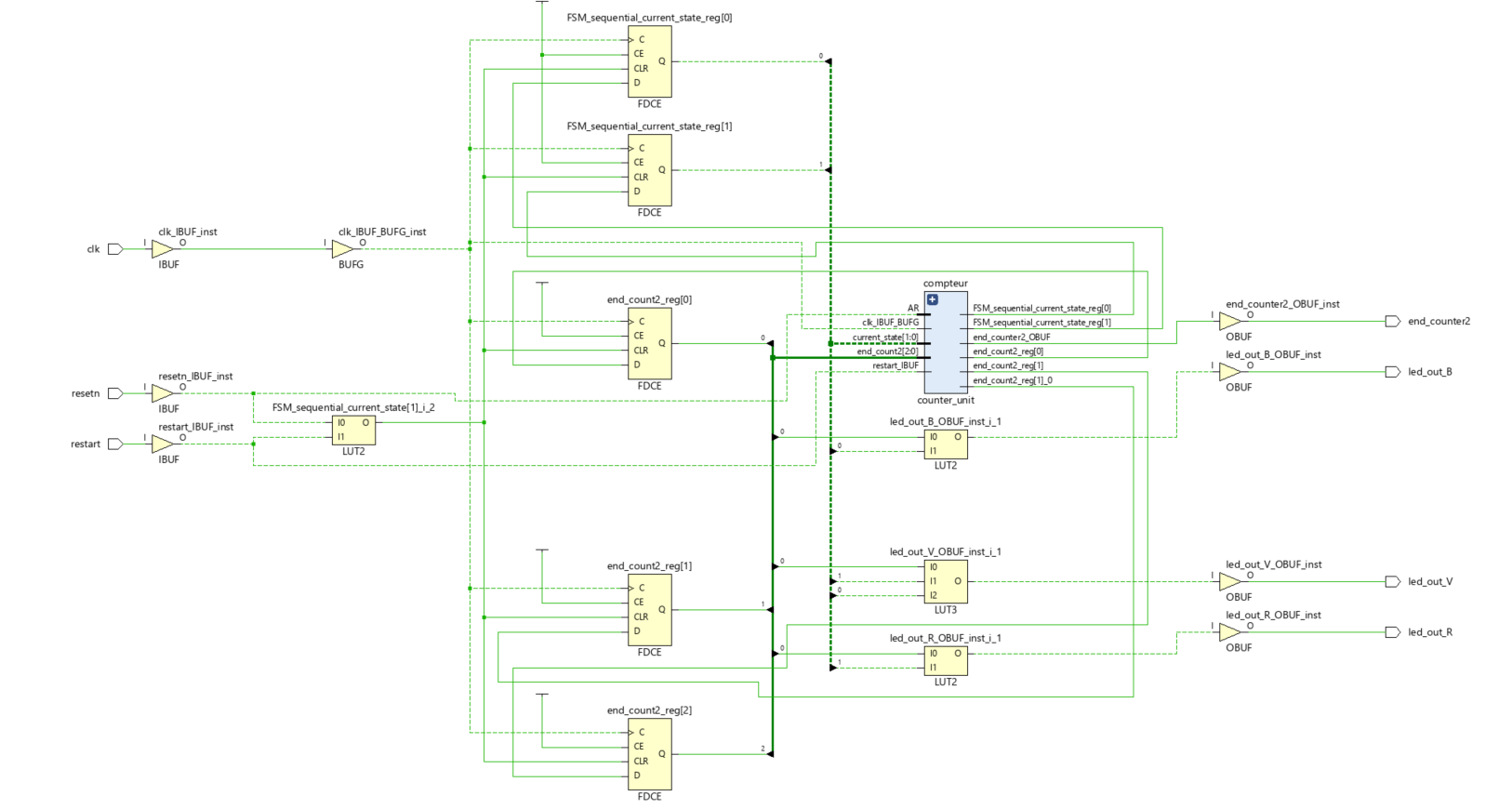
La description du RTL dans la synthèse.

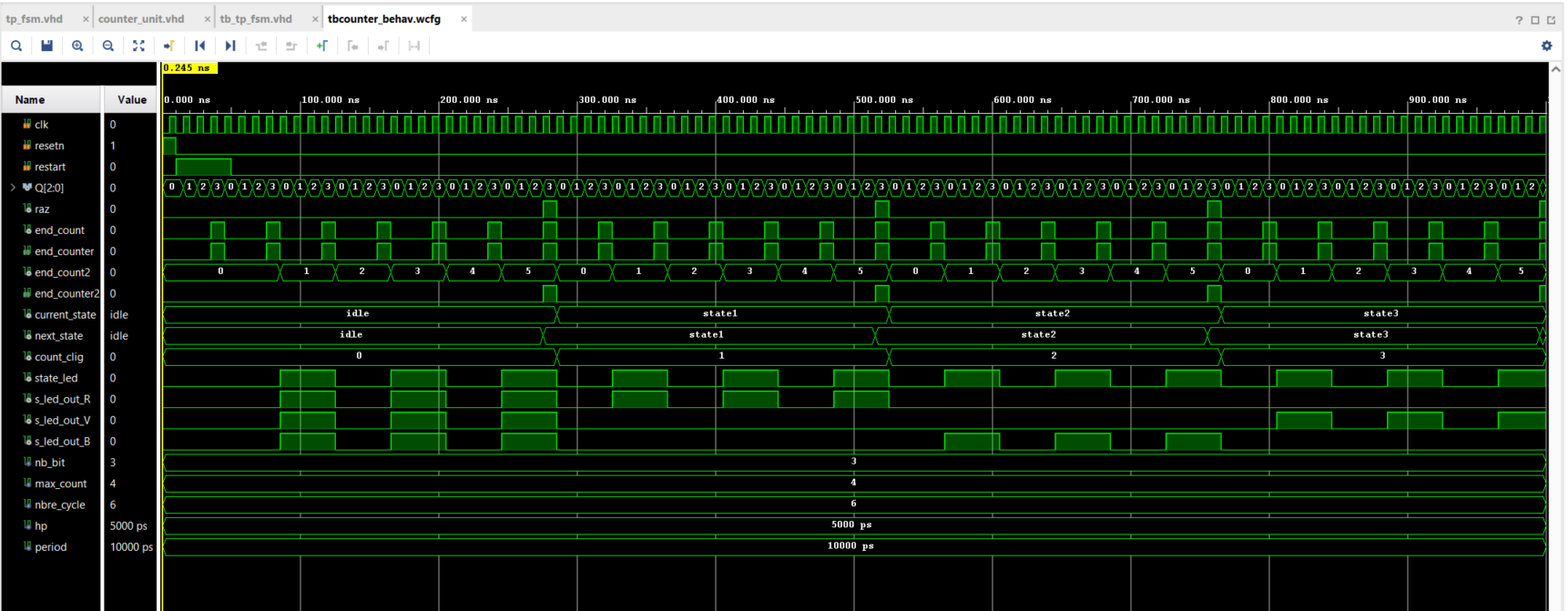


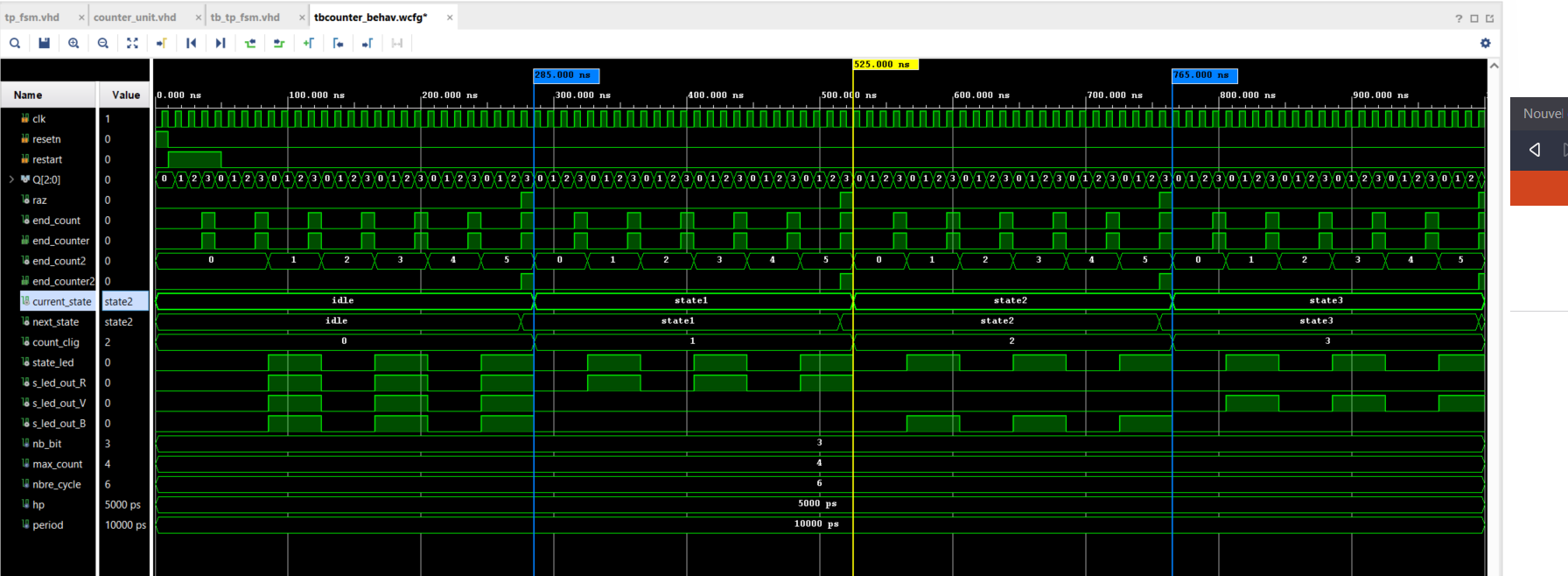
Tous les ressources sont répertoriées ci-dessus.

Dans la synthèse : voici le compteur de cycle. On retrouve autour de lui, les régistre qui gère

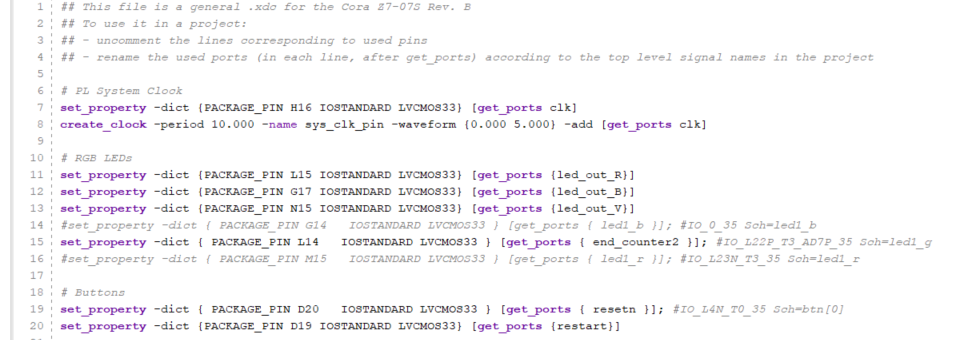








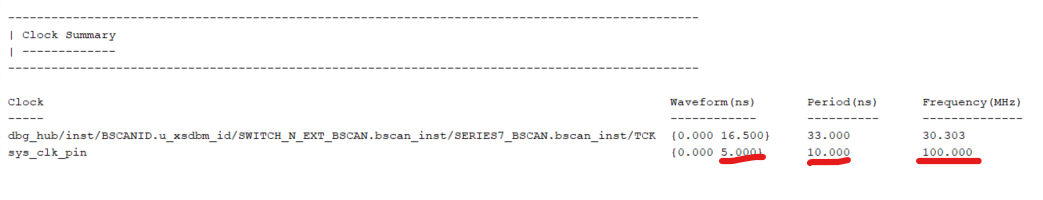
1. Modifiez le fichier de contraintes pour connecter vos entrées / sorties du système avec les broches de la carte. Réglez l’horloge pour que sa fréquence soit à 100MHz.



La fréquence est réglée à 100MHZ. La schématique est représentée ci-dessous.

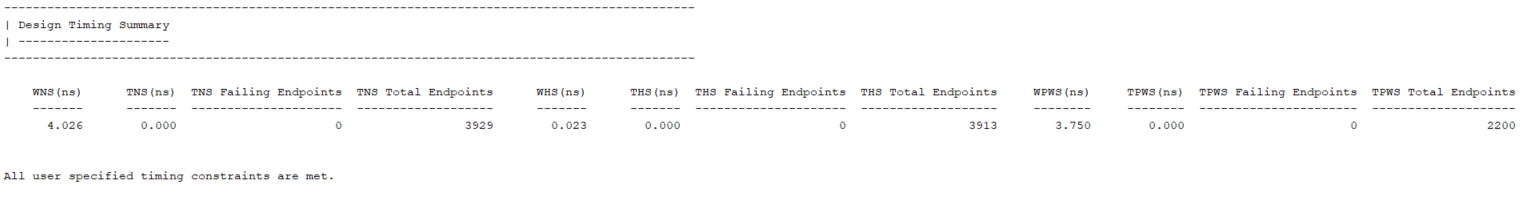
1. Lancez l’implémentation puis étudiez le rapport de timing (vérifiez les violations de set up et de hold et identifiez le chemin critique).

Le Clock



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 :

Max Delay Paths

--------------------------------------------------------------------------------------

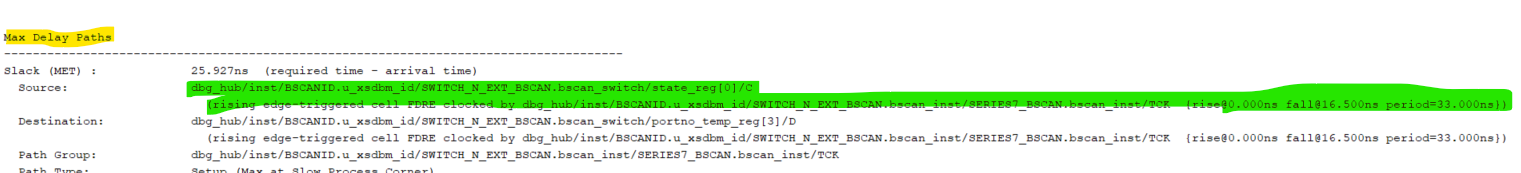
Slack (MET) : 25.927ns (required time - arrival time)

Source: dbg\_hub/inst/BSCANID.u\_xsdbm\_id/SWITCH\_N\_EXT\_BSCAN.bscan\_switch/state\_reg[0]/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[3]/D

(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})



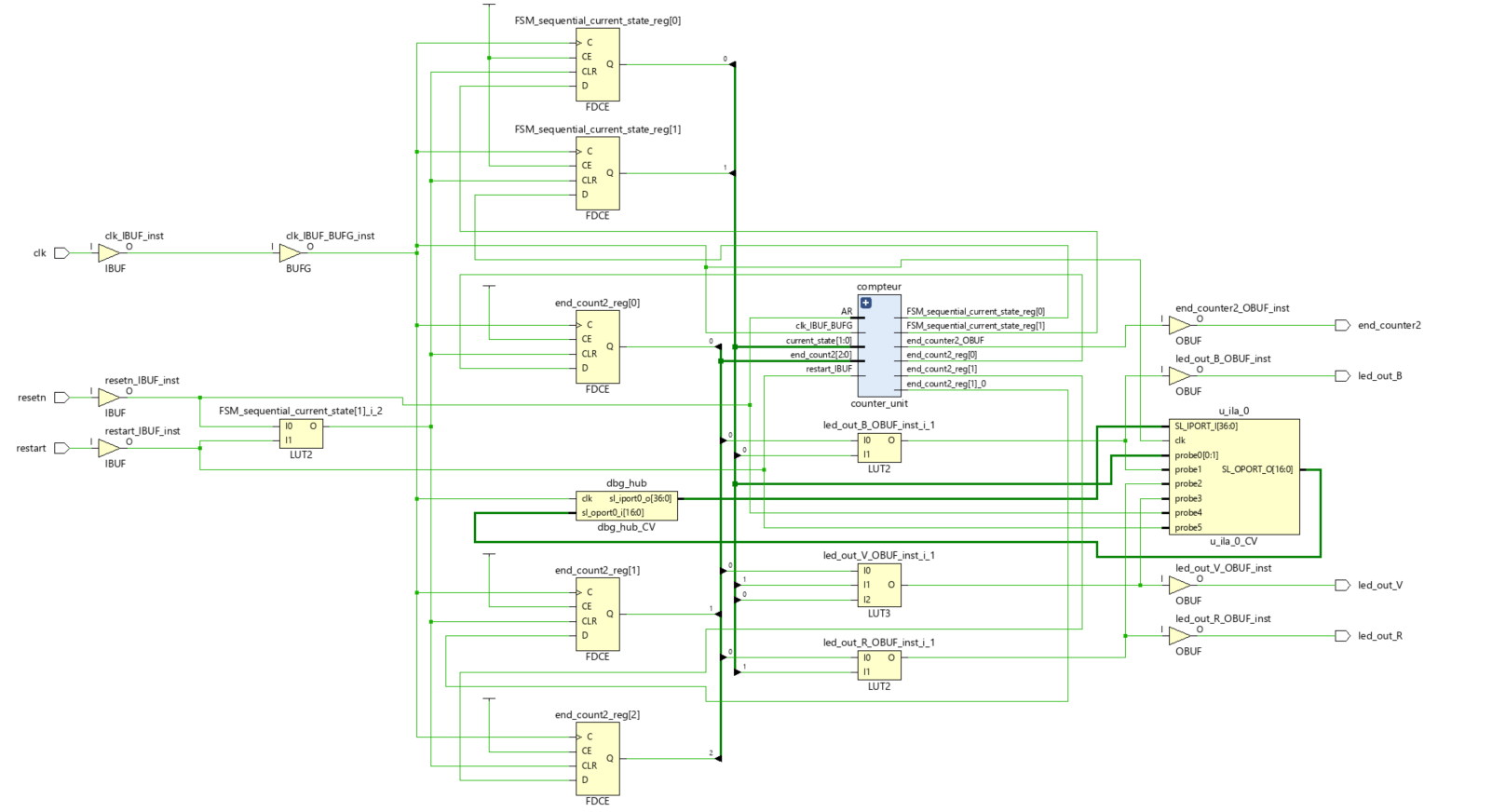
1. Générez le bitstream pour vérifier le système sur carte.

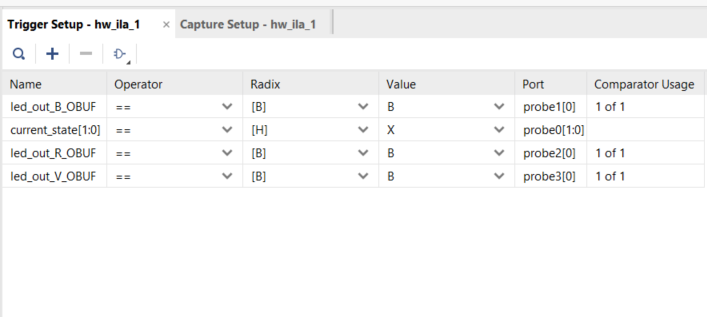
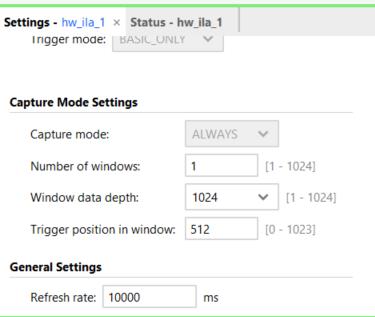
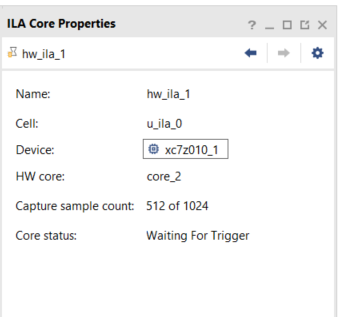
Les signaux leds out , et current\_state sont observés dans le trigger.

hw\_ila\_1

Pour rappel, la principale fonction d'une ILA est de surveiller et d'analyser les signaux numériques à l'intérieur d'un circuit intégré ou d'un FPGA (Field-Programmable Gate Array). Elle permet de détecter les erreurs, de valider le fonctionnement des circuits et de comprendre le comportement des signaux lors de l'exécution d'un programme ou d'une séquence d'opérations.

Nous démarrons l’enregistrement de l’ILA sur le front montant et descendant du port de sortie du current\_state. On procède à un changement de valeur sur le signal de déclenchement entraînera l'ILA pour commencer à enregistrer les signaux sondés (Led,out RGB). Ceci est fait dans le déclencheur (trigger setup).





Nous voyons la ligne verticale rouge (marqueur) sur le front montant de notre signal de déclenchement (port trigger de led), et il est en position 512. Nous pouvons également vérifier que le signal compte se comporte correctement et change de couleur suivant les changements d’état.

