## Segment 2

**TP2 : Compteur de temporisation**

**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. L’horloge du système est fixée à 100MHz. Combien de période faut-il compter pour attendre 2 secondes ? Combien de bits faut-il au minimum pour représenter cette valeur ?

F=100MHz,

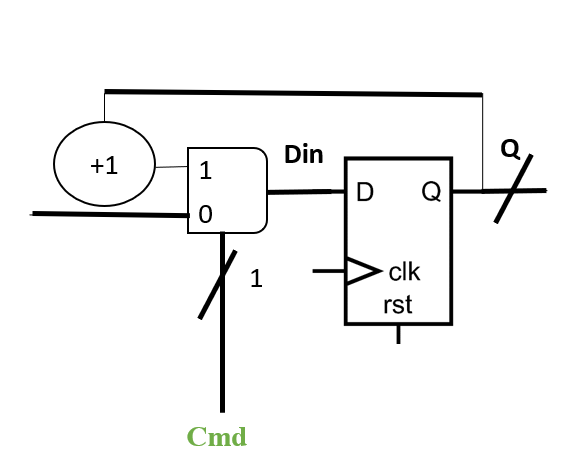
Soit T la période, T=1/F=10ns

Pour atteindre 2 secondes, il faut :

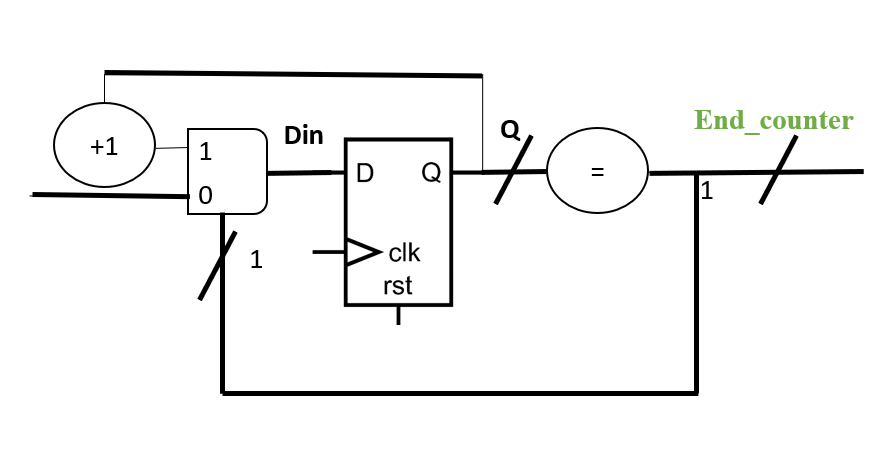
Nbp=2s/10ns= 2\*108

Le nombre de période est donc 2\*108 et le bits minimum pour représenter cette valeur est de 28.

1. Dessinez le schéma RTL de ce compteur. Si le compteur atteint la valeur calculée précédemment, un signal *end\_counter* passe à 1, sinon *end\_counter* vaut 0. N’oubliez pas de mettre sur chaque signal son nombre de bits. Commencez par réaliser une boucle d’incrémentation : +1 à chaque coup d’horloge.



1. Ajoutez une condition pour que le compteur soit remis à 0 lorsqu’il a atteint la valeur souhaitée



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

Entrée

clk : in std\_logic;

resetn

Sortie

end\_counter

1. Ecrivez à présent le compteur en VHDL en suivant le schéma RTL, faites attention de bien faire correspondre les noms des signaux de votre code VHDL avec ceux de votre schéma RTL.

*library ieee;*

*use ieee.std\_logic\_1164.all;*

*use ieee.std\_logic\_unsigned.all;*

*use ieee.numeric\_std.all;*

*entity counter\_unit is*

*port (*

*clk : in std\_logic;*

*resetn : in std\_logic;*

*-- restart : in std\_logic;*

*end\_counter : out std\_logic*

*);*

*end counter\_unit;*

*architecture behavioral of counter\_unit is*

*--Declaration des signaux internes*

*constant cte : positive := 20;*

*signal Q : std\_logic\_vector(27 downto 0);*

*signal end\_count : std\_logic;*

*begin*

*--Partie sequentielle*

*process(clk,resetn)*

*begin*

*if(resetn = '1') then*

*Q <=(others=>'0');*

*elsif(rising\_edge(clk)) then*

*if(end\_count= '1') then*

*-- if(end\_count= '1'or restart='1') then*

*Q <=(others=>'0');*

*else*

*Q <= Q+ std\_logic\_vector(to\_signed(1,28));*

*end if;*

*end if;*

*end process;*

*--Partie combinatoire*

*end\_count <= '1' when Q = std\_logic\_vector( to\_unsigned(cte-1, 28) )*

*else '0';*

*end\_counter <= end\_count;*

*end behavioral;*

1. Ecrivez un fichier de testbench pour tester votre design.

*library ieee;*

*use ieee.std\_logic\_1164.all;*

*use ieee.std\_logic\_unsigned.all;*

*use ieee.numeric\_std.all;*

*entity tb\_counter is*

*end tb\_counter;*

*architecture behavioral of tb\_counter is*

*signal resetn : std\_logic := '0';*

*signal clk : std\_logic := '0';*

*-- signal restart : std\_logic := '0';*

*signal end\_counter: std\_logic;*

*-- Les constantes suivantes permette de definir la frequence de l'horloge*

*constant hp : time := 5 ns; --demi periode de 5ns*

*constant period : time := 2\*hp; --periode de 10ns, soit une frequence de 100Hz*

*--Declaration de l'entite a tester*

*component counter\_unit*

*port (*

*clk : in std\_logic;*

*resetn : in std\_logic;*

*-- restart : in std\_logic;*

*end\_counter : out std\_logic*

*);*

*end component;*

*begin*

*--Affectation des signaux du testbench avec ceux de l'entite a tester*

*uut: counter\_unit*

*port map (*

*clk => clk,*

*resetn=>resetn,*

*-- restart=>restart,*

*end\_counter => end\_counter*

*);*

*--Simulation du signal d'horloge en continue*

*process*

*begin*

*wait for hp;*

*clk <= not clk;*

*end process;*

*process*

*begin*

*-- TESTS A EFFECTUER*

*resetn <= '1';*

*wait for 10ns;*

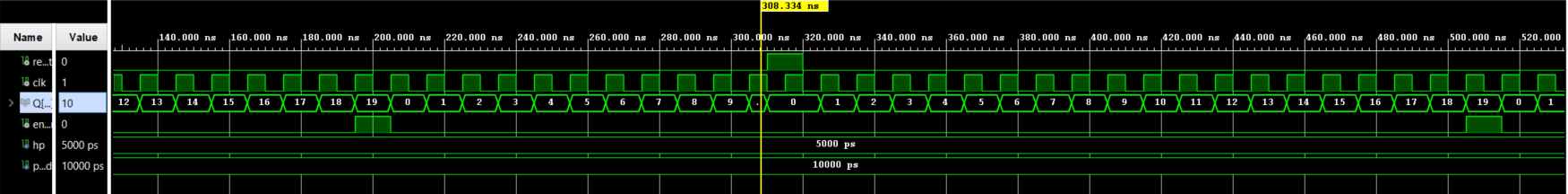
*resetn <= '0';*

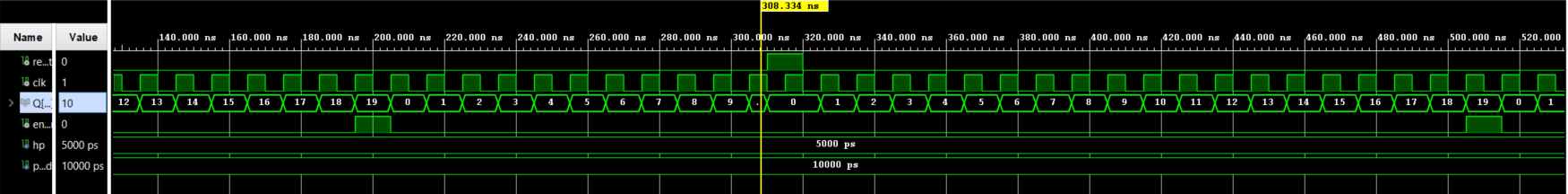
*--*

*wait for 300 ns;*

*end process;*

*end behavioral;*

1. Lancez une simulation. Que devez-vous observez sur votre chronogramme pour vérifier que votre design est valide ? 



1. Associez une LED avec le signal de teste d’arrêt du compteur. Pour cela, il faudra ajouter une sortie et la relier à une broche d’une LED dans le fichier de contrainte (.xdc). La LED sera alors allumée pendant seulement un coup d’horloge.
2. Modifiez le schéma RTL du compteur pour ajouter une remise à 0 lorsqu’un signal restart est à 1.

Ajoutez la logique nécessaire pour que la LED clignote telle que : allumée 2s, éteinte 2s.

*library ieee;*

*use ieee.std\_logic\_1164.all;*

*use ieee.std\_logic\_unsigned.all;*

*use ieee.numeric\_std.all;*

*entity tb\_counter is*

*end tb\_counter;*

*architecture behavioral of tb\_counter is*

*signal resetn : std\_logic := '0';*

*signal clk : std\_logic := '0';*

*signal restart : std\_logic := '0';*

*signal end\_counter: std\_logic;*

*-- Les constantes suivantes permette de definir la frequence de l'horloge*

*constant hp : time := 5 ns; --demi periode de 5ns*

*constant period : time := 2\*hp; --periode de 10ns, soit une frequence de 100Hz*

*--Declaration de l'entite a tester*

*component counter\_unit*

*port (*

*clk : in std\_logic;*

*resetn : in std\_logic;*

*restart : in std\_logic;*

*end\_counter : out std\_logic*

*);*

*end component;*

*begin*

*--Affectation des signaux du testbench avec ceux de l'entite à tester*

*uut: counter\_unit*

*port map (*

*clk => clk,*

*resetn=>resetn,*

*restart=>restart,*

*end\_counter => end\_counter*

*);*

*--Simulation du signal d'horloge en continue*

*process*

*begin*

*wait for hp;*

*clk <= not clk;*

*end process;*

*process*

*begin*

*-- TESTS A EFFECTUER*

*resetn <= '1';*

*wait for 10ns;*

*resetn <= '0';*

*--*

*wait for 300 ns;*

*end process;*

*end behavioral;*

1. Faites les mises à jour nécessaires sur le code VHDL pour correspondre au nouveau schéma. Le signal restart sera une entrée du design.

*library ieee;*

*use ieee.std\_logic\_1164.all;*

*use ieee.std\_logic\_unsigned.all;*

*use ieee.numeric\_std.all;*

*entity counter\_unit is*

*port (*

*clk : in std\_logic;*

*resetn : in std\_logic;*

*restart : in std\_logic;*

*end\_counter : out std\_logic*

*);*

*end counter\_unit;*

*architecture behavioral of counter\_unit is*

*--Declaration des signaux internes*

*constant cte : positive := 20;*

*signal Q : std\_logic\_vector(27 downto 0);*

*signal end\_count : std\_logic;*

*begin*

*--Partie sequentielle*

*process(clk,resetn)*

*begin*

*if(resetn = '1') then*

*Q <=(others=>'0');*

*elsif(rising\_edge(clk)) then*

*if(end\_count= '1'or restart='1') then*

*Q <=(others=>'0');*

*else*

*Q <= Q+ std\_logic\_vector(to\_signed(1,28));*

*end if;*

*end if;*

*end process;*

*--Partie combinatoire*

*end\_count <= '1' when Q = std\_logic\_vector( to\_unsigned(cte-1, 28) )*

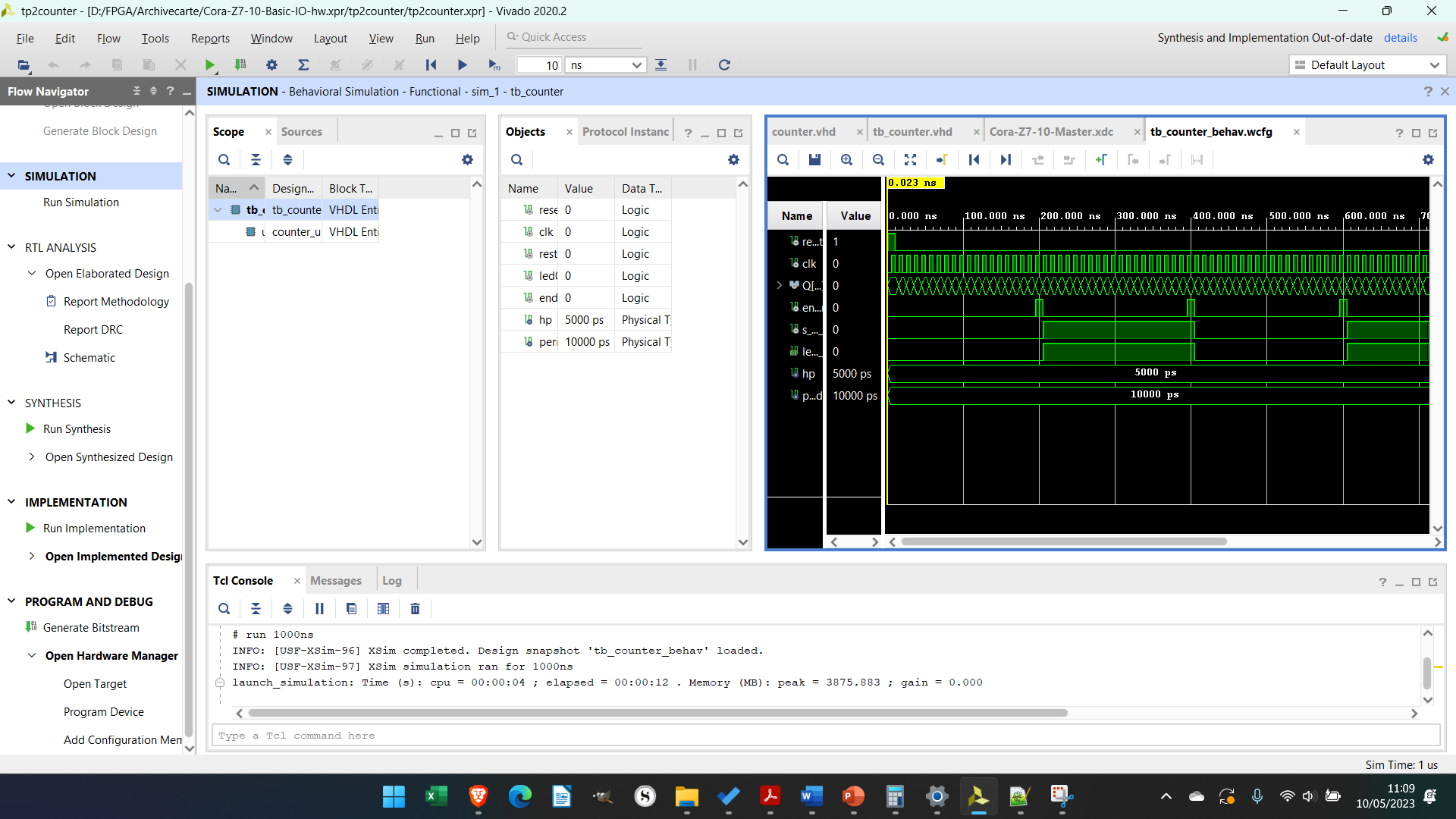
*else '0';*

*end\_counter <= end\_count;*

*end behavioral;*

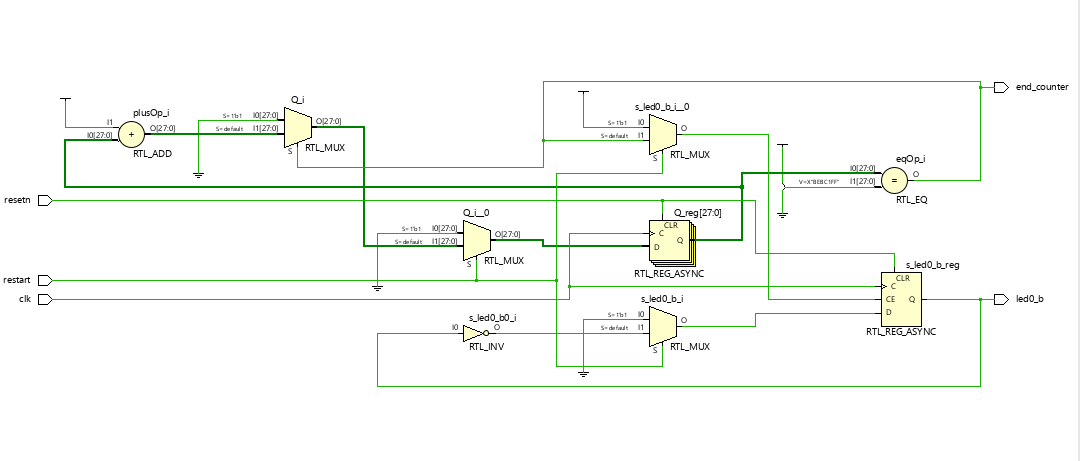
1. Associez la nouvelle entrée restart à un bouton.
2. Mettez à jour votre testbench puis vérifier votre design avec une simulation. Quels sont les signaux que vous devez observer ?

Clk, led, end\_couteur, restart,resetn



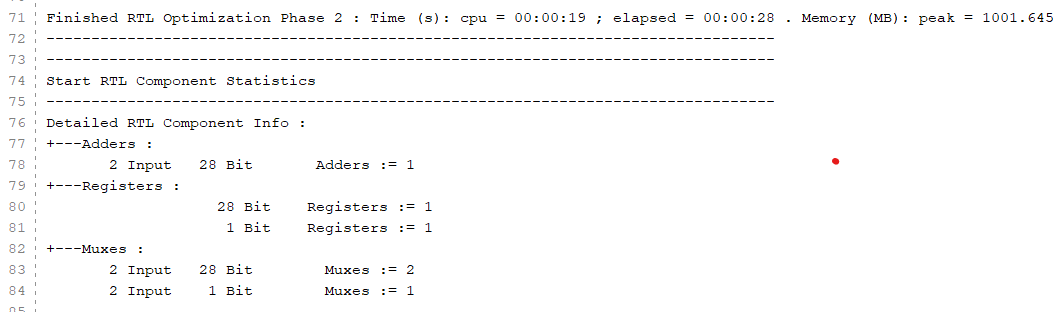
Les chronogrammes suivent bien le résultat attendu. On observe bien l’allumage de la led passe à un (allumé) lorsque end\_counter passe à 0 (à la fin du comptage). Et respecte bien les 2s avant de se réactiver.

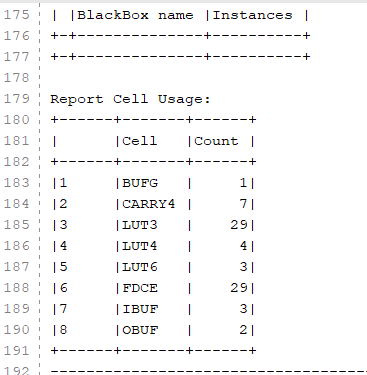
1. Exécutez la synthèse puis ouvrez la schématique. Identifiez sur la schématique les différents éléments de votre architecture RTL.



On remarque deux additionneurs, 4 multiplexeurs, un registre et un bloc de registres de 28 bits.

1. Ouvrez le rapport de synthèse et relevez les ressources utilisées. Comparez vos résultats avec les résultats attendu selon votre architecture RTL.

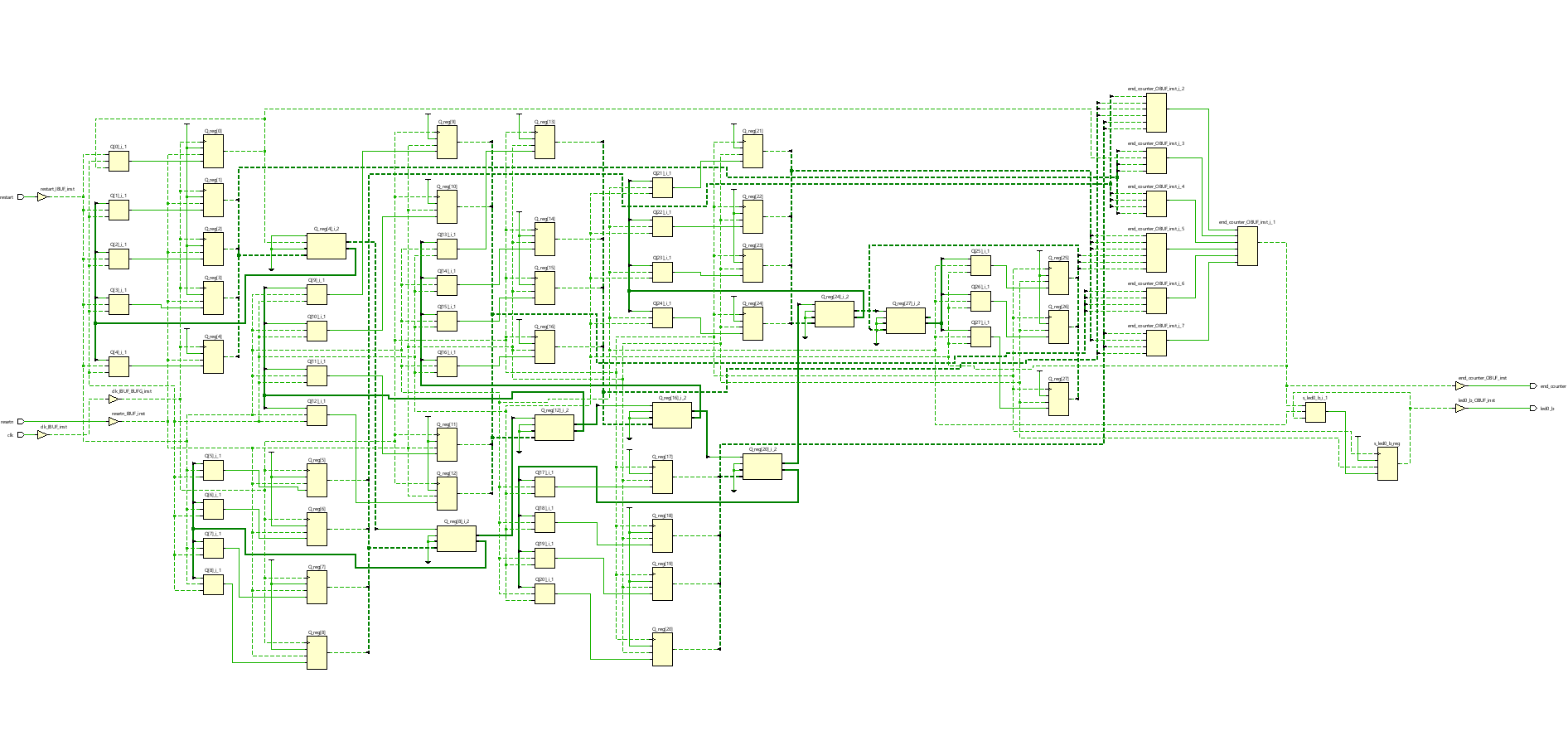




On Trouve les mêmes éléments sur le RTL que dans le rapport de synthèse.

29 registres à reset , 3 input buffer( resetn,clk,restart) et 2Out buffer(Led et end\_conter).

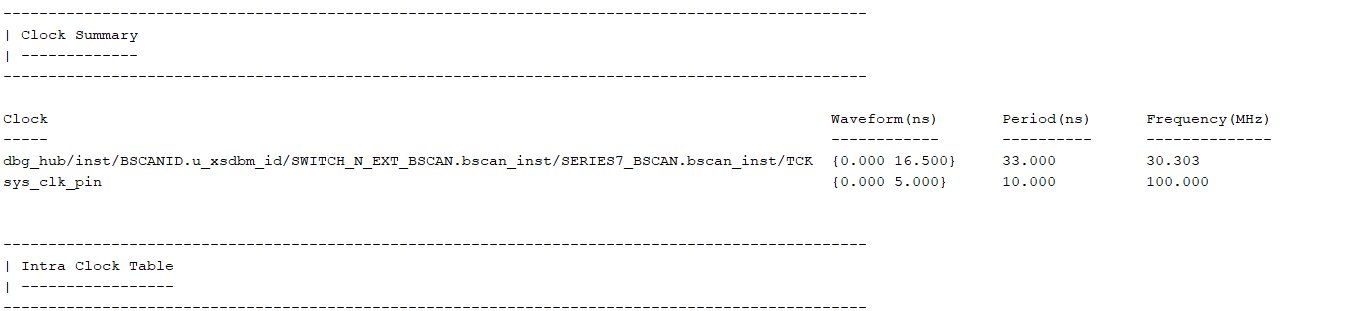
1. Ouvrez le Set Up Debug. Placez des sondes sur les signaux à observer que vous avez défini à la question 12.



On remarque le trajet des signaux sur le schéma.

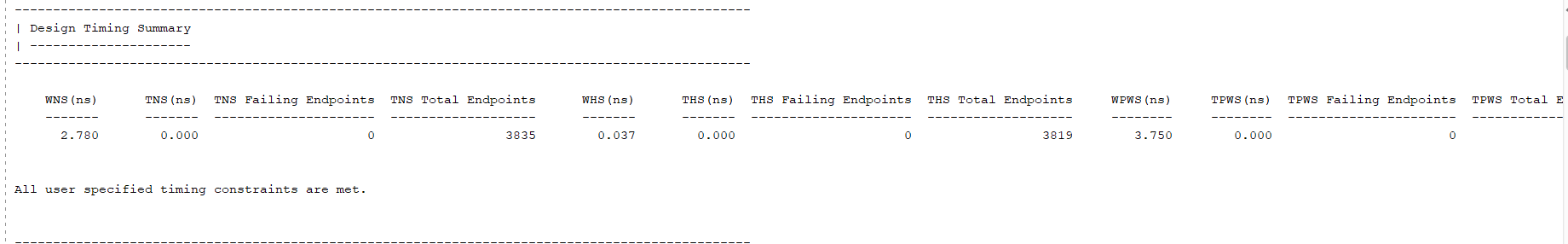
1. Lancez l’implémentation puis étudiez le rapport de timing (vérifiez les violations de set up et de hold et identifiez le chemin critique).

Clock



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

Valeur à 0, il n’y a pas de violation du set up et du hold.



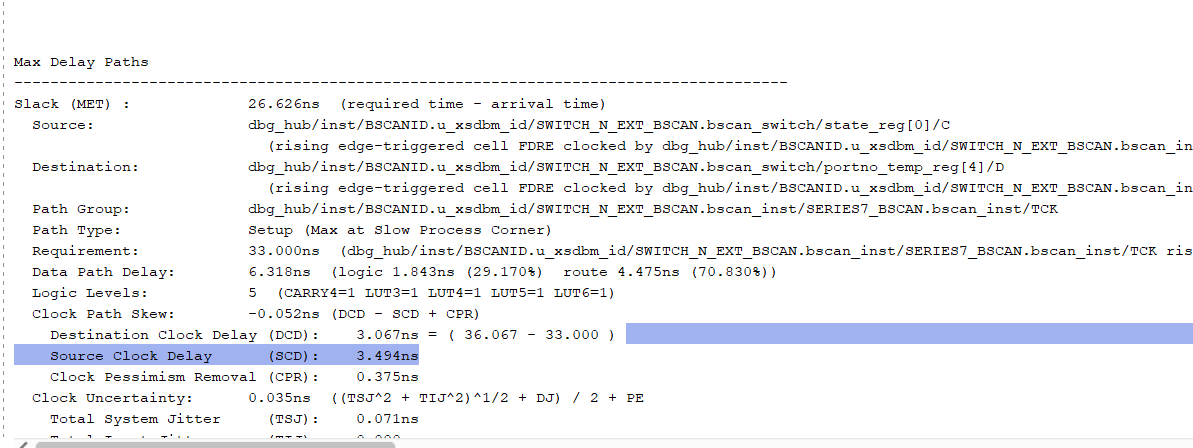
Chemin critique :

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[4]/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})

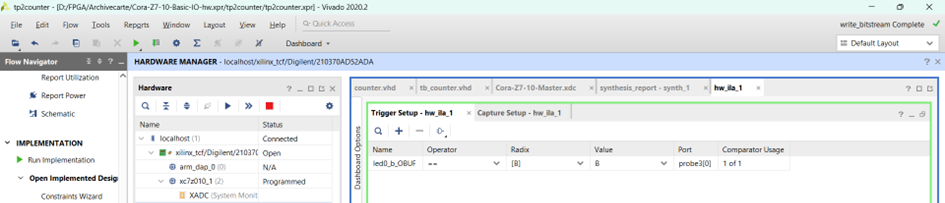


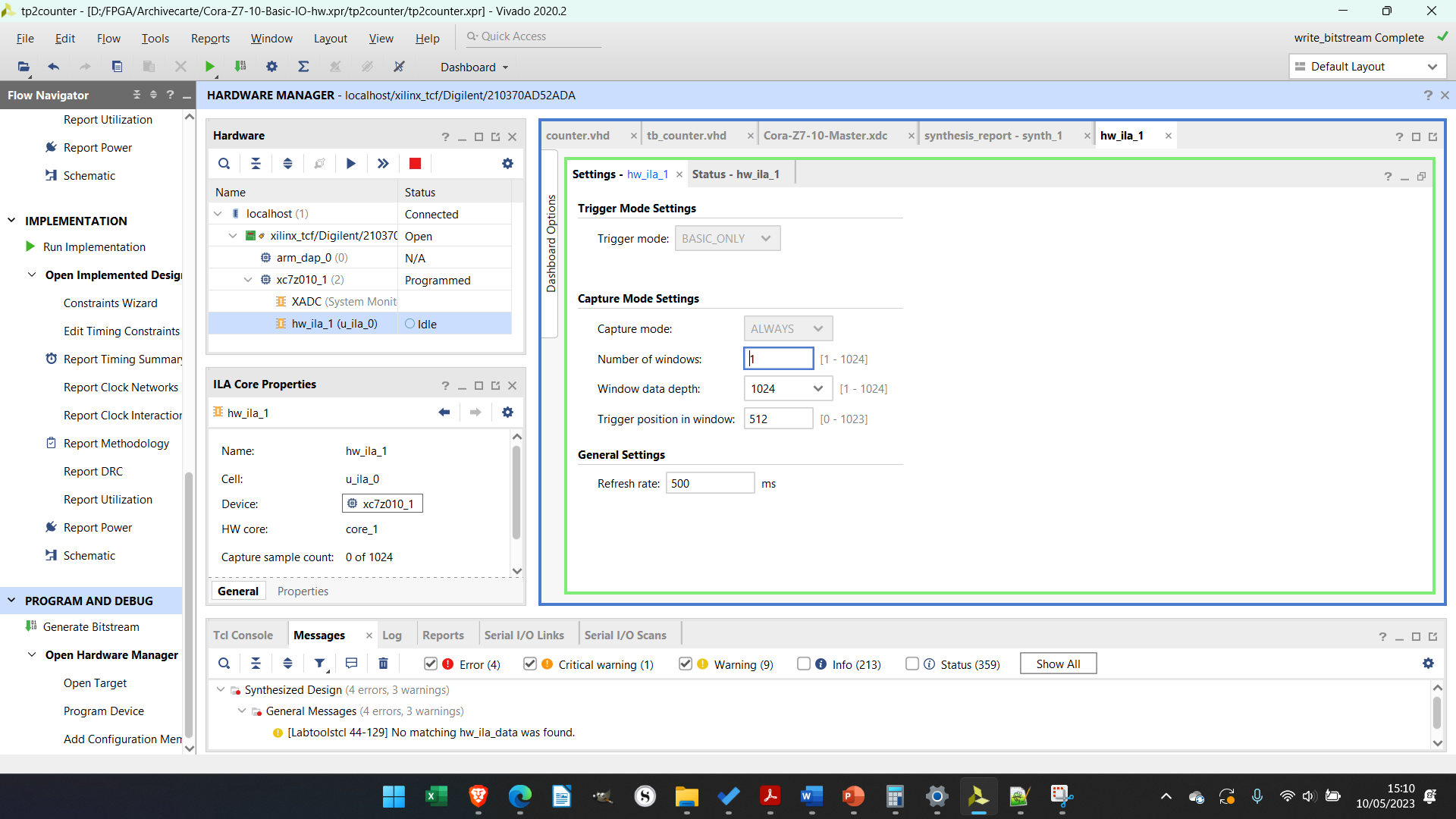
1. Générez le bitstream pour observer le système sur carte. Relevez les résultats de la ILA.

Les signaux led, et couteur,

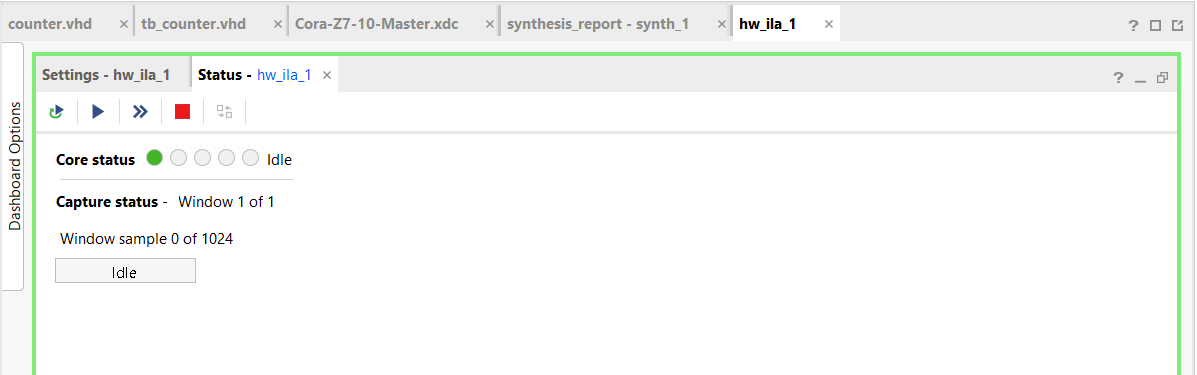
hw\_ila\_1

Nous demarrons l’enrégistrement de l’ILA sur le front montant et descendant du port de sortie de la led et du compteur. On procède à un changement de valeur sur le signal de déclenchement entraînera l'ILA pour commencer à enregistrer les signaux sondés (Led, et Q). Ceci est fait dans le déclencheur (trigger setup).





Cliquez sur le déclencheur Exécuter bouton (play). l'ILA se déclenche et enregistre les signaux .



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 le signal end\_counter et le compteur Q.

