TP2 compteurs
Jean Baptiste NARI

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.

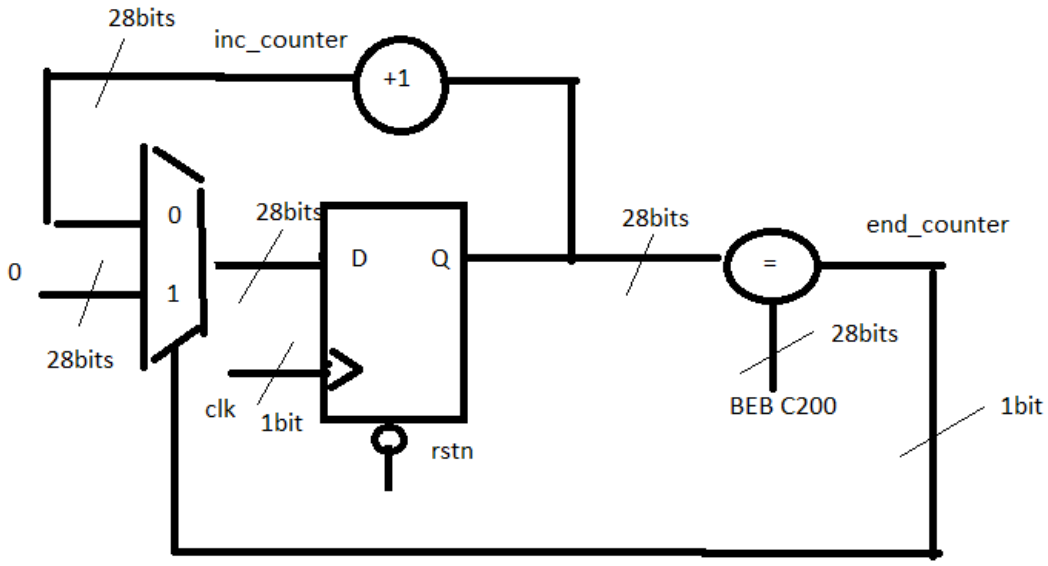
Question 1

Calcul du nombre de cycles d'horloges à compter pour une temporisation de 2 secondes.
Calcul du nombre de bits nécésssaires pour représenter cette valeur.

Horloge 100MHz, periode 10ns, pour 2 secondes il faut 200M cycles. Il faut 28 bits (log2(200M)) minimum pour représenter 200M.

Question 2 , 3

Schéma RTL de notre compteur.



A chaque front d''horloge Q est incrémenté de 1 ce qui permet de compter les fronts d'horloges.
Une fois que Q est égale à BEB C200 soit 200M le signal end_counter passe à 1.
Et ce passage à 1 à travers le multiplexuer vient remetre à zero le compteur.

Question 4

Liste des entrées sorties et signaux internes.

clk : entrée
rstn : entrée
end_counter : sortie

inc_counter, D ,Q, end_counter_int : interne

end_counter est aussi un signal interne, il permet la remise à zero du compteur.

Comme le signal end_counter est aussi un signal qui doit être lu, decription d'un signal end_counter_int qui sera affecté sur la sortie end_counter.
Il est nécéssaire de créer ce signal car le logiciel vivado empeche de lire une sortie à l'interrieur d'un processus.

Question 5

Description vhdl de notre compteur :

```
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;
                rstn : in std_logic;
                end_counter : out std_logic
    );
end counter_unit;

architecture behavioral of counter_unit is

        --Declaration des signaux internes
   --constant cte : positive := 200000000;
   constant cte : positive := 20;

        signal Q : std_logic_vector(27 downto 0);
        signal end_counter_int : std_logic;



begin
process(clk, rstn)
begin
```

```
        if (rstn = '1') then -- 1
                Q <= (others => '0');

        elsif (rising_edge(clk)) then

                if (end_counter_int='1') then
                        Q <= (others => '0');

                else
                        --Q <= Q + std_logic_vector(to_signed(1, 28));
                    Q <= Q + 1;
                end if;

        end if;
end process;


        --end_counter_int <= '1' when Q =  std_logic_vector( to_unsigned(cte, 28) )
        --                        else '0';

        end_counter_int <= '1' when Q =  cte - 1
                                else '0';

        end_counter <= end_counter_int;

end behavioral;
```

Un process avec la liste de sensibilité clk, rstn permet de créer notre registre avec reset asynchrone.
Un if else permet de faire le choix entre l'incrementation ou la remise à zero du compteur.

Le signal end_counter_int est affecté avec l'instruction when. Si Q est égale à la valeur de la constante moins 1, end_counter_int vaut 1 sinon 0.
Puis le signal end_counter_int est affecté à la sortie end_counter.

Question 6

Description du test bench de notre compteur:

```
library ieee;
use ieee.std_logic_1164.all;

entity tb_counter is
end tb_counter;

architecture behavioral of tb_counter is

        signal rstn      : std_logic := '0';
        signal clk       : 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
        --constant long_time : time := 2000 ms;

        --Declaration de l'entite a tester
        component counter_unit
                port (
                        clk                     : in std_logic;
                        rstn            : 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,
        rstn=>rstn,
        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
          rstn <= '1';
          wait for 10ns;
          rstn <= '0';

          wait for 300ns;
--        wait for long_time;
--        assert end_counter='1' report "test fail" severity failure;


        end process;

end behavioral;
```

Avant le debut de la simulation un reset du circuit est réalisé. Rstn est affecté à 1 pendant 10ns puis repasse à 0.

Ce test bench consiste à attendre le temps nécéssaire au comptage puis à vérifier que la sortie end_counter passe à 1 au bout de ce temps.

Question 7

Dans ce cas, la valeur maximal du compteur est fixé 20, le compteur compte jusqu'a 20 (de 0 a 19) on voit bien que le siganl end_counter passe à 1 une fois que le nombre d'impulsion est atteint.

Question 8

Modification du fichier de contraintes.

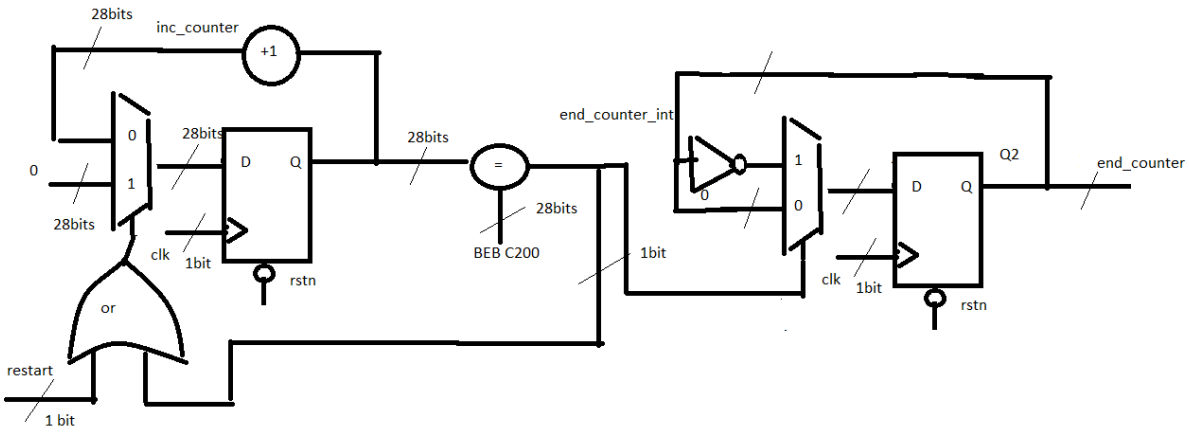
```
# PL System Clock
set_property -dict { PACKAGE_PIN H16   IOSTANDARD LVCMOS33 } [get_ports { clk }];
#IO_L13P_T2_MRCC_35 Sch=sysclk
create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4} [get_ports { clk }];#set

# RGB LEDs
set_property -dict { PACKAGE_PIN L15   IOSTANDARD LVCMOS33 } [get_ports { end_counter
}];
```

Le signal d'horloge est généré sur notre entrée clk.

La sortie end_counter est reliée sur la première led bleu du circuit.

Question 9

Schéma RTL modifié pour piloter une led.



Le circuit rajouté permet d'inverser l'état de la sortie end_counter à chaque fois que le signal end_counter_int est à 1.

Le signal end_counter_int est à 1 pendant 1 cycle d'horloge. Si on l'appliquait directement sur la led on ne verrait pas le clignotement.

Dans le cas ou ce signal declenche une invertion du signal de end_counter, le signal end_counter maintiendra sont état pendant un cycle de comptage.

Vu qu'un cycle de comptage est calculé pour durée 2 secondes la led restera allumée puis éteinte pendant 2 secondes.

Le signal restart passe par une porte or pour réinitialiser le compteur.

Question 10

```
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;
                rstn : 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 := 200000000;
    --constant cte : positive := 20;

        signal Q : std_logic_vector(27 downto 0);
        signal end_counter_int : std_logic;
        signal Q2 : std_logic;



begin
process(clk, rstn)
begin
        if (rstn = '1') then
                Q <= (others => '0');
                Q2 <= '0';

        elsif (rising_edge(clk)) then

                if (end_counter_int='1' or restart='1') then
                        Q <= (others => '0');

                else
                        Q <= Q + 1;
                end if;

                if(end_counter_int='1') then
        Q2 <= not Q2;
                else
                   Q2 <= Q2;
                end if;

        end if;
end process;


        end_counter_int <= '1' when Q =  cte - 1
                                else '0';


        end_counter <= Q2;


end behavioral;
```

Dans notre process initial nous rajoutons if else sur le signal end_counter_int qui permet d'affecter Q2 à not Q2 ou bien Q2 en fonction de la valeur de end_counter_int.

<u>test bench</u>

```
library ieee;
```

```
use ieee.std_logic_1164.all;

entity tb_counter is
end tb_counter;

architecture behavioral of tb_counter is

	signal rstn      : 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
	--constant long_time : time := 2000 ms;

	--Declaration de l'entite a tester
	component counter_unit
		port (
			clk                     : in std_logic;
			rstn                : 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,
	rstn=>rstn,
	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
```

```
        rstn <= '1';
        wait for 10ns;
        rstn <= '0';

        wait for 1000ns;
--      wait for long_time;
--      assert end_counter='1' report "test fail" severity failure;


    end process;


end behavioral;
```

Le fichier test bench n'a pas de reçu de modification.

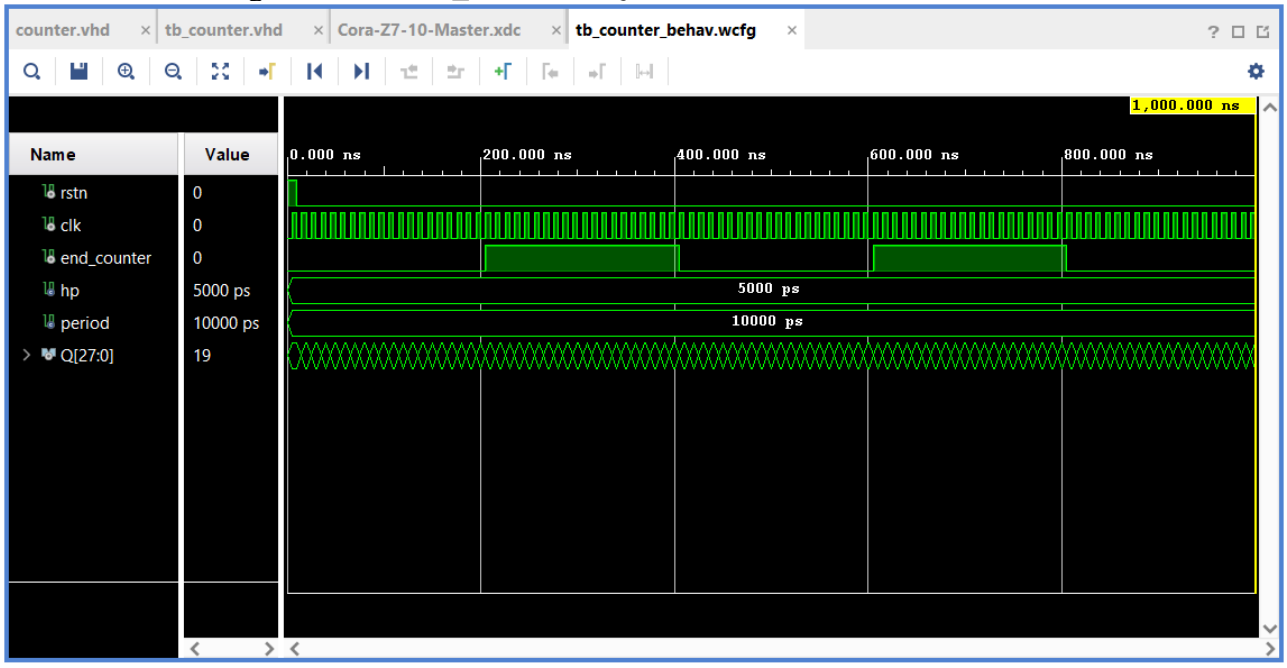
Question 11

Modification du fichier de contrainte pour ajouter l'entrée restart sur le deuxième bouton poussoir.

```
set_property -dict { PACKAGE_PIN D20   IOSTANDARD LVCMOS33 } [get_ports { restart }];
```

Question 12

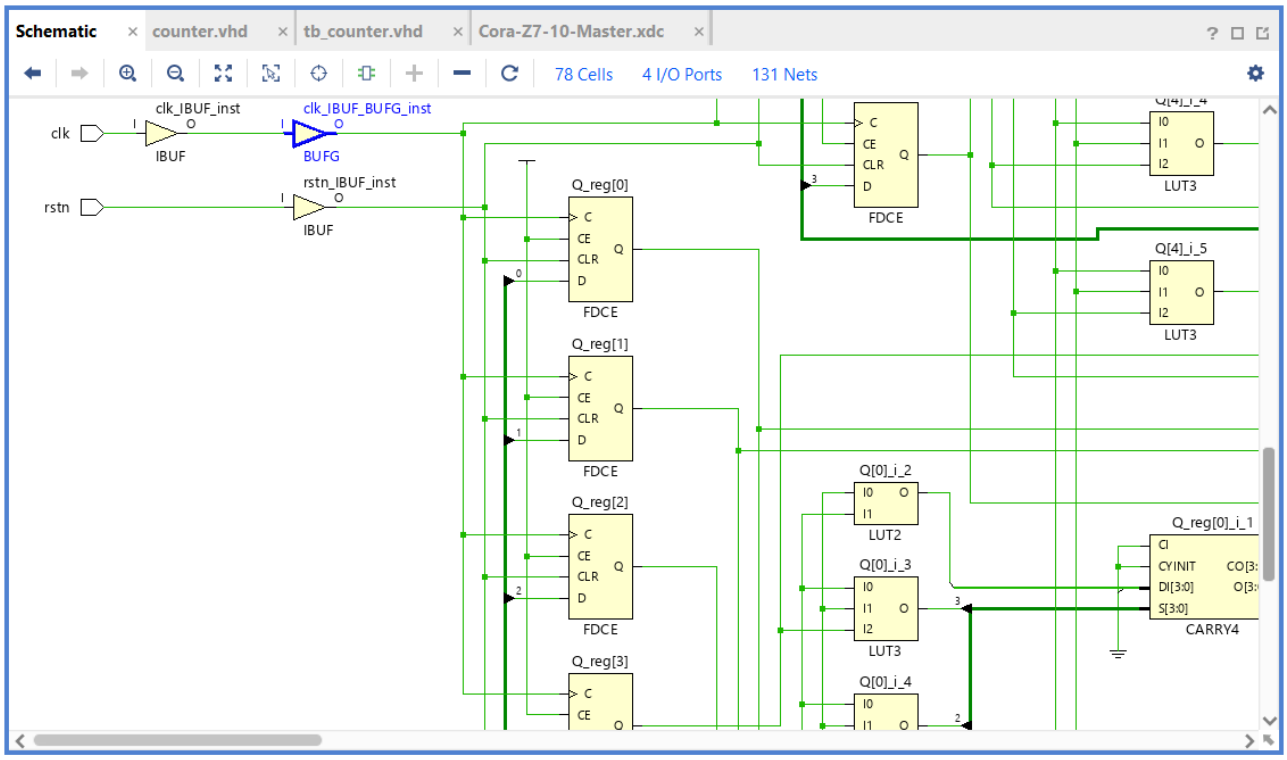
Visualisation des signaux : clk, end_counter et Q.



Contrairement au circuit précedent, End_counter maintient son état sur plusieurs cycles d'horloges. Ce qui permettra d'avoir un clignotement visible sur la led.

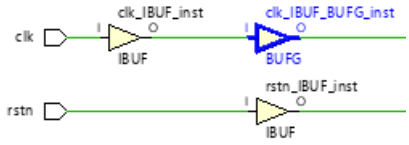
Question 13

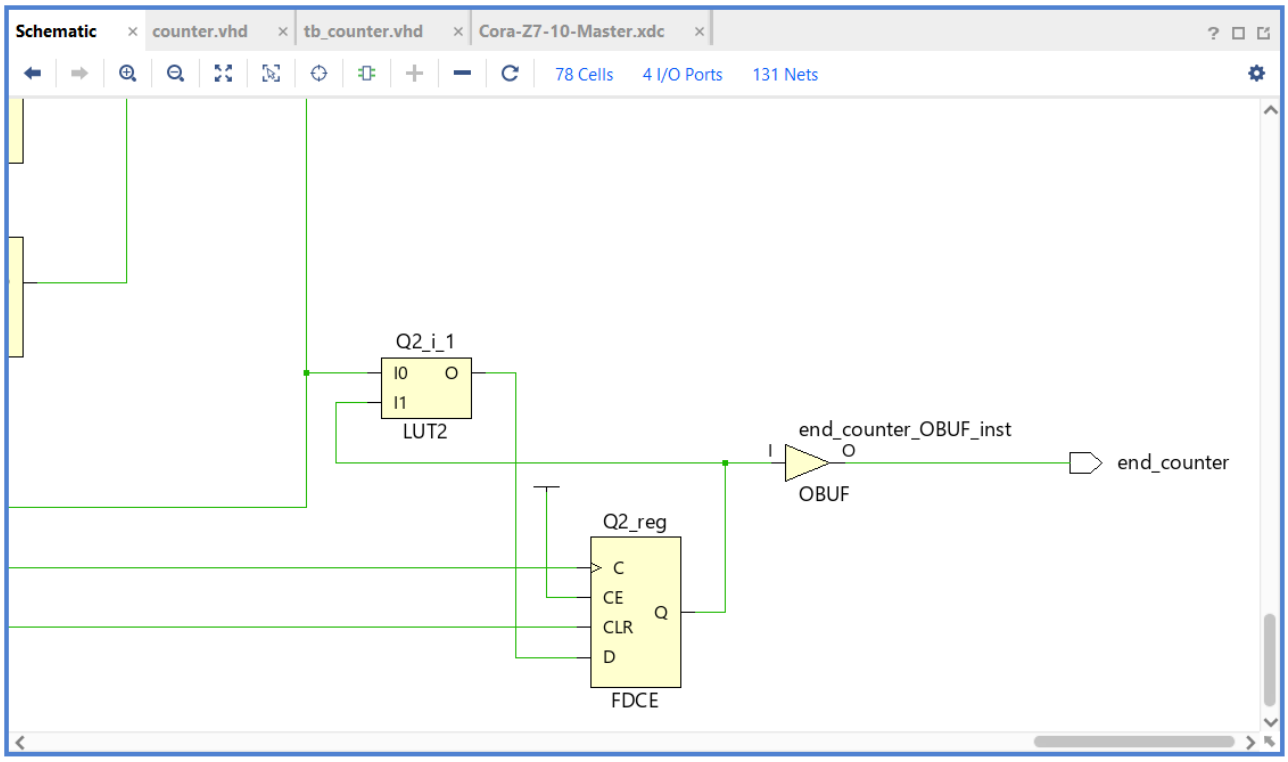
Visualisation du schematic:



On peut voir les premiers registres (FCDE) composant les 4 premiers bits de notre signal Q (Q_reg[0], Q_reg[1], Q_reg[2], Q_reg[3]).

Nous constatons que les entrées et sorties sont conformes à notre description.





Nous voyons notre sortie end_counter qui est en sortie de la bacule Q2 (Q2_reg) comme dans notre description vhdl.

Question 14

Rapport de synthèse :

	Cell	Count
1	BUFG	1
2	CARRY4	7
3	LUT2	2
4	LUT3	28
5	LUT4	4
6	LUT6	3
7	FDCE	29
8	IBUF	3
9	OBUF	1

Il y a 29 FCDE (registre à reset asynchrone) ce qui est conforme à notre description vhdl.
28 registres pour le signal Q (27 down to 0) et 1 registre supplémentaire pour le signal Q2 sur un bit.
3 IBUF correspond à nos 3 entrées : clk, rstn, restart
1 OBUF correspond à notre sortie end_counter

Question 15

Placement des sondes pour observer Q, end_counter.

Ne pas choisir clk permet d'éviter des problèmes de timing.

Question 16

Etude des rapports de timming:

WNS(ns)	TNS(ns)	TNS Failing Endpoints	TNS Total Endpoints
2.200	0.000	0	3806

WHS(ns)	THS(ns)	THS Failing Endpoints	THS Total Endpoints
0.030	0.000	0	3790

Il n'y a pas de problème de timing TNS et THS sont à zero ; pas de violation de setup ni de hold.

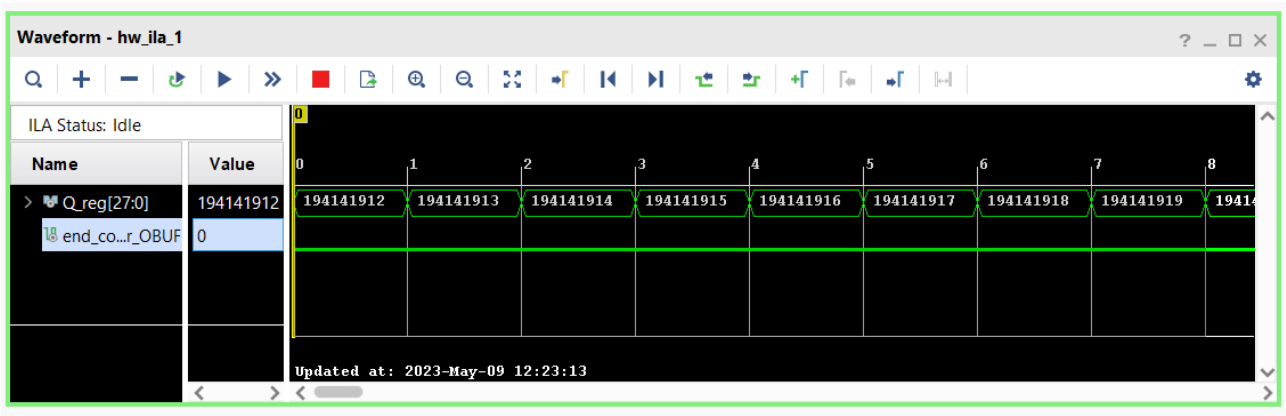
Identification du chemin critique :

Max Delay Paths

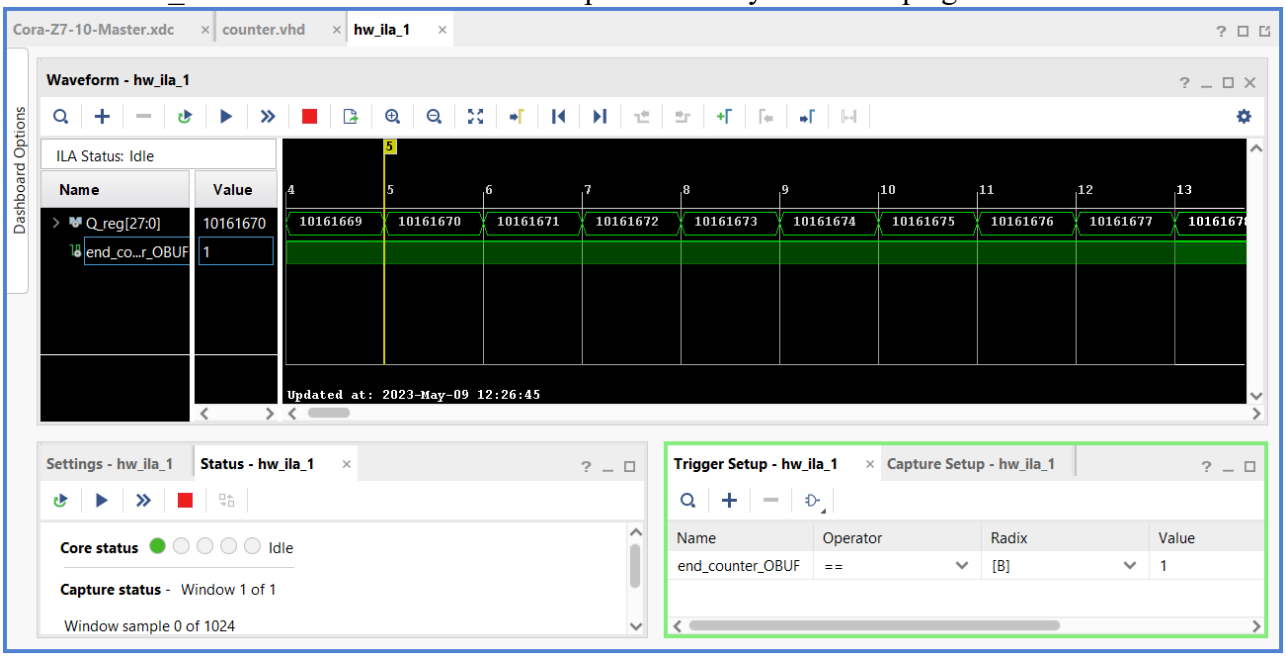
Slack (MET) : 25.651ns (required time - arrival time)
Source:
dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_switch/state_reg[1]/C
(rising edge-triggered cell FDRE clocked by
dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_inst/SERIES7_BSCAN.bscan_inst/TCK {rise@0.000ns fall@16.500ns period=33.000ns})
Destination:
dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_switch/portno_temp_reg[5]/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})

Question 17

Il a été nécéssaire de supprimer la clk des sondes pour eviter des erreurs de timing.

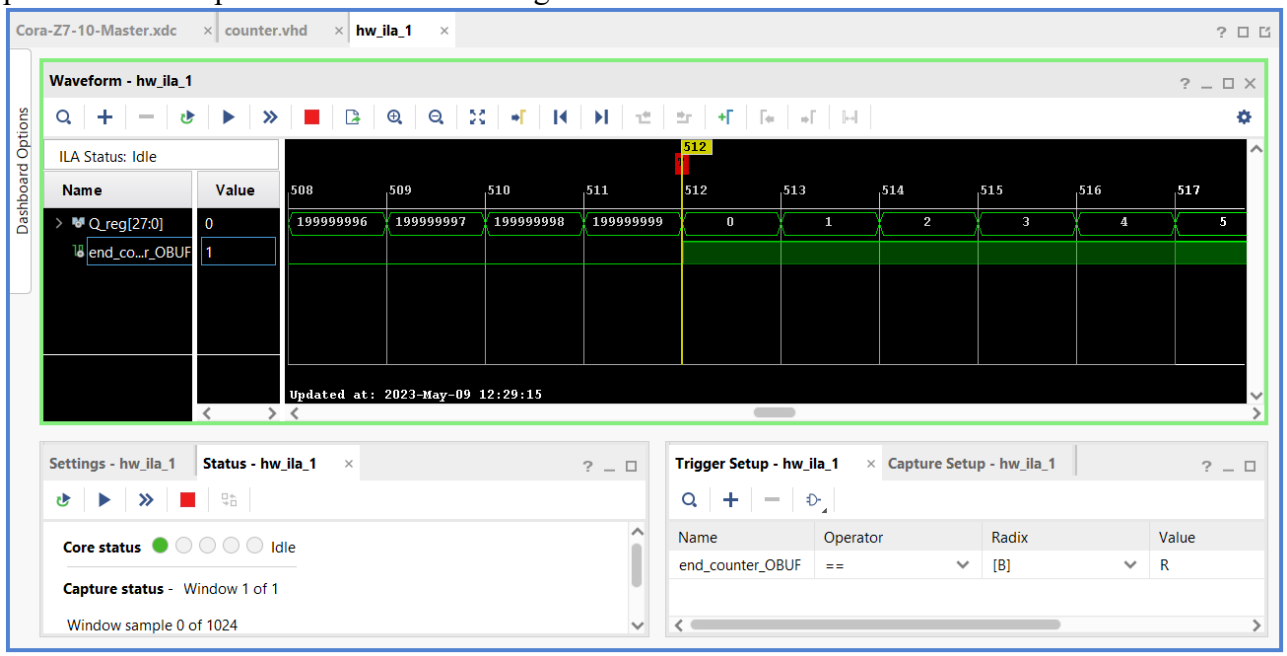


La sortie end_conter est maintenu à l'état bas pendant un cycle de comptage.

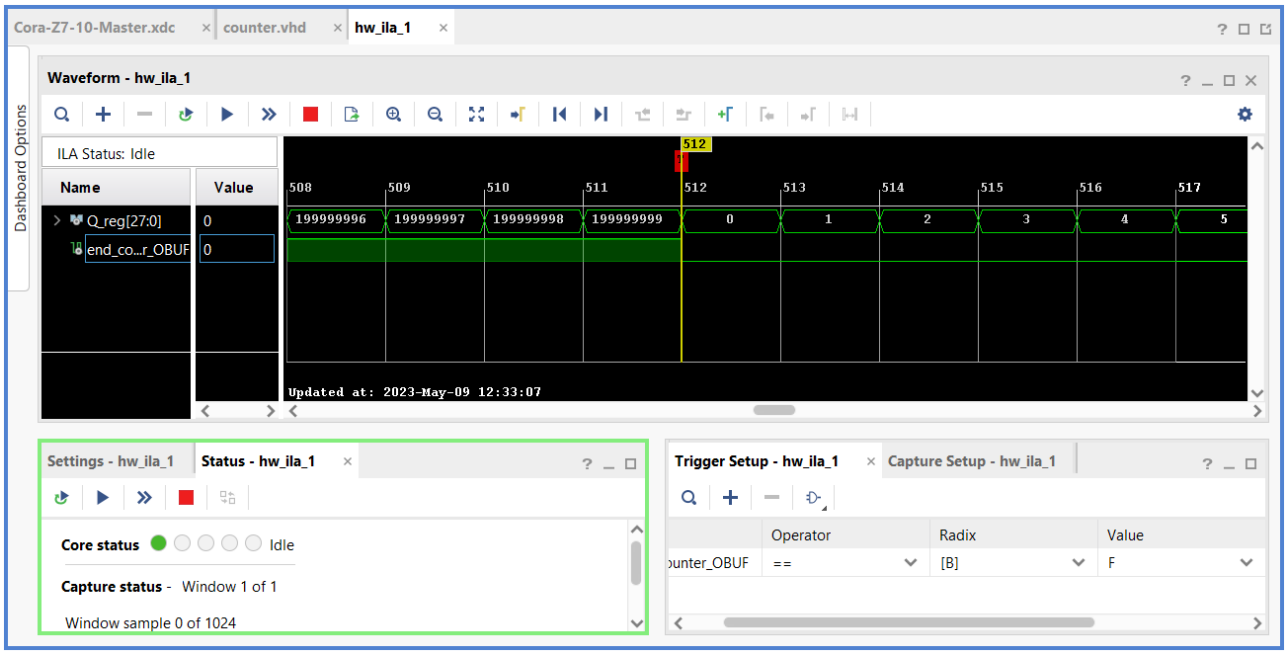


La sortie end_conter est maintenue à l'état haut pendant un cycle de comptage.

Pour obtenir les deux visualisations suivantes réglage du trigger sur end_counter au front montant puis descendant pour le deuxième chronogramme.



A la fin du comptage il y a bien une transition de la sortie à létat haut.



A la fin du comptage il y a bien une transition de la sortie à létat bas.

Ces quatre phases sont comformes au fonctionement attendu du compteur.
Avec les sondes nous avons pu vérifier le fonctionnement de notre compteur une fois implementé sur la carte.

Après programmation de la carte la led clignote. L'appui sur les boutons poussoir permet l'extinction de la led.

A travers ce tp nous avons realisé un prise en mains de vivado et nous avons conçu et vérifié le fonctionnement d'un compteur pour faire clignoter une led.