

TP2 COMPTEUR

Rendu

Votre rapport devra contenir :

- Vos schéma RTL
- Vos résultats de simulation avec vos chronogrammes commentés
- Vos résultats de mesure ILA
- Vos résultats de synthèse (analyse de vos ressources utilisées)
- Vos résultats de STA (analyse du rapport de timing)
- Une démonstration de votre design

Vous fournirez également vos codes source commentés.

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 ?

$$T = 1/f \Rightarrow T = 10\text{ns}$$

$$\text{timer} = 2\text{s}$$

$$\text{timer} / T = 200\ 000\ 000 \text{ périodes}$$

$$2^{\text{puissance } 27} = 134\ 217\ 728$$

$$2^{\text{puissance } 28} = 268\ 435\ 456$$

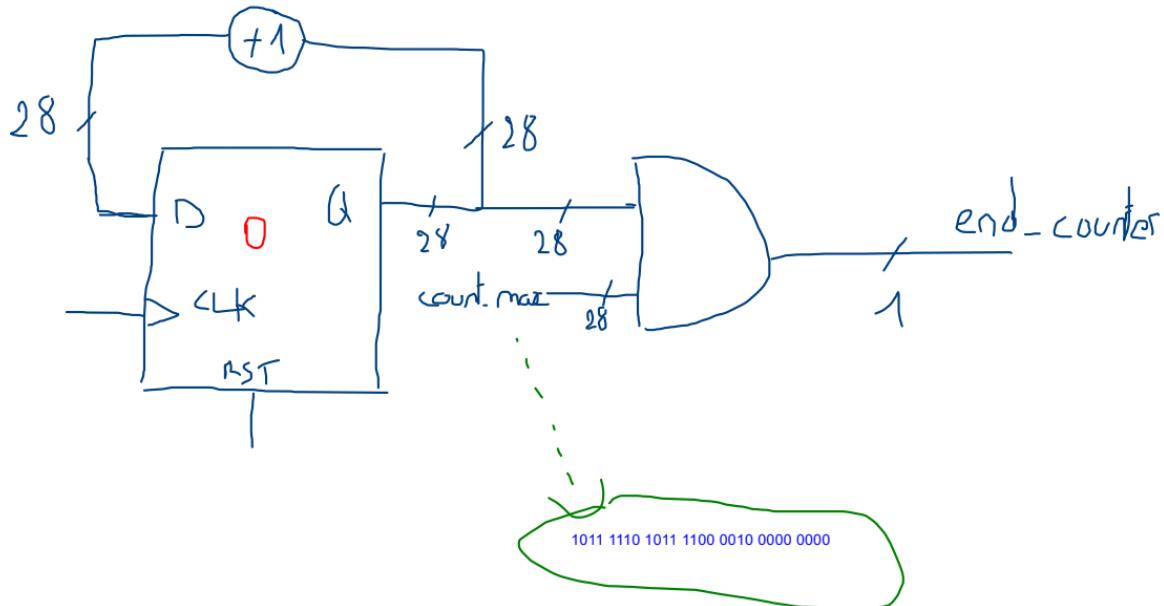
=> Il faut attendre 28 périodes.

$$200\ 000\ 000 = 1011\ 1110\ 1011\ 1100\ 0010\ 0000\ 0000$$

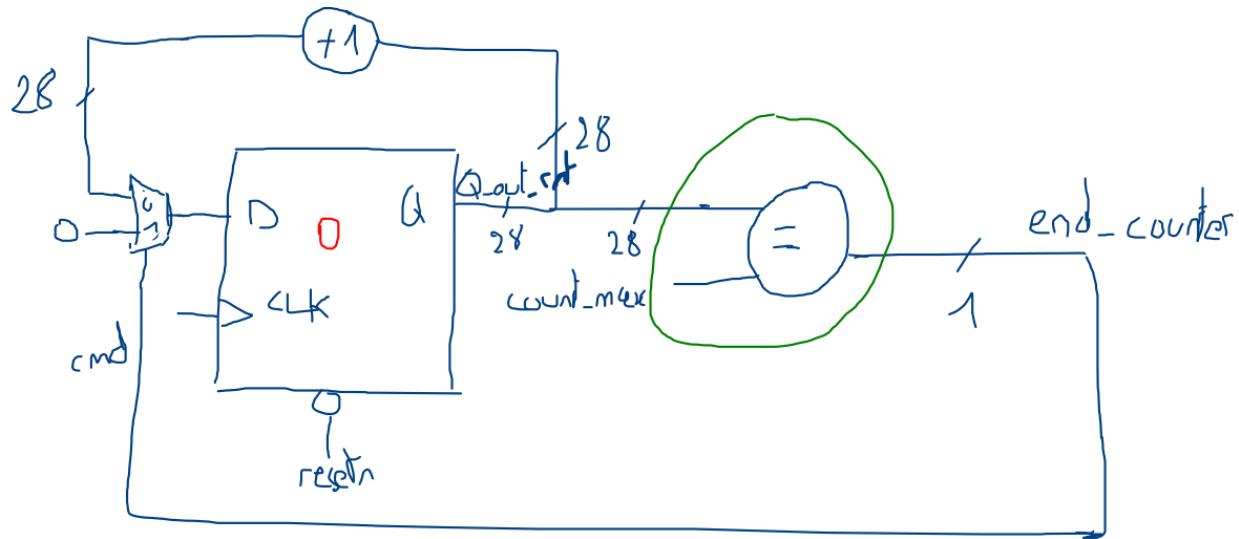
NB: Pour trouver 28, on peut réaliser le calcul:
 $\ln(200\ 000\ 000) / \ln(2)$
autrement dit le logarithme en base deux de 200 000 000

2. 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.

Registre initialisé à 0



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



4. Listez les signaux d'entrée, de sortie et les signaux internes de votre architecture.

Signaux d'entrée:

- clk : clock
- resetn : reset du registre

Signaux internes:

- Q_out_cnt : compteur (signal en sortie du registre)
- count_max : 200 000 000 périodes (constante)
- cmd : entrée du multiplexeur

Signaux de sortie:

- end_counter : fin du compteur (Passe à 1 lorsque le compteur atteint 199 999 999, on compte de 0 à 199 999 999. Soit 200 000 000 périodes)

5. 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.

counter.vhd

C:/FORMATION_SAFRAN/TP/TP2_Compteur/counter.vhd

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

entity counter_unit is
    port (
        clk : in std_logic;
        resetn : in std_logic;
        end_counter : out std_logic
    );
end counter_unit;

architecture behavioral of counter_unit is

    --Declaration des signaux internes
    constant count_max : positive := 2000;          -- Nombre de periodes correspondant a 2 secondes *** A REMPLACER PAR 2E8 ***
    signal Q_out : positive := 0;                   -- Signal en sortie du registre
    signal cmd : std_logic := '0';                  -- Recopie du signal end_counter pour utilisation dans partie sequentielle

begin

    begin

        --Partie sequentielle
        process(clk,resetn)
        begin
            if(resetn = '0') then
                Q_out <= 0;
            elsif(rising_edge(clk)) then
                if (cmd = '1') then Q_out <= 0;
                else Q_out <= Q_out + 1;
                end if;
            end if;
        end process;

        --Partie combinatoire
        cmd <= '1' when (Q_out = (count_max-1))
        else '0';
        end_counter <= cmd;                         -- Copie du signal end_counter
    end behavioral;
```

Pour tester, petite valeur de count_max
(2000 périodes au lieu de 200 000 000).

NB: Cette constante doit être en phase avec la constante du fichier tb_counter.vhd

NB: On ne peut pas utiliser un signal de sortie dans la partie séquentielle. D'où, l'intérêt d'ajouter un signal interne (ici cmd).

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

tb_counter.vhd

C:/FORMATION_SAFRAN/TP/TP2_Compteur/tb_counter.vhd

Q | | | | | | | | | | | | ? |

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity tb_counter is
5  end tb_counter;
6
7  architecture behavioral of tb_counter is
8
9    signal resetn      : std_logic := '1';
10   signal clk         : std_logic := '0';
11   signal end_counter : std_logic;
12
13  -- Les constantes suivantes permettent de definir la frequence de l'horloge
14  constant hp : time := 5 ns;           -- demi periode de 5ns
15  constant period : time := 2*hp;       -- periode de 10ns, soit une frequence de 100MHz
16  constant count_max_tb : positive := 2000; -- nombre de periodes correspondant au compteur
17
18
19  --Declaration de l'entite a tester
20  component counter_unit
21    port (
22      clk      : in std_logic;
23      resetn   : in std_logic;
24      end_counter : out std_logic
25    );
26  end component;
27
```

Pour tester, petite valeur de count_max_tb
(2000 périodes au lieu de 200 000 000).

NB: Cette constante doit être en phase avec la constante du fichier counter.vhd

```
begin  
  
    --Affectation des signaux du testbench avec ceux de l'entite a tester  
    uut: counter_unit  
        port map (  
            clk => clk,  
            resetn=>resetn,  
            end_counter => end_counter  
        );  
  
    --Simulation du signal d'horloge en continue  
    process  
    begin  
        wait for hp;  
        clk <= not clk;  
    end process;
```

Simulation de la clock.
La clock change d'état toutes les demi-périodes.

```

process
begin
-----
-- FIRST TEST TO CHECK end_counter IS SET TO '1' AFTER THE TIMER      --
-- AND SWITCH BACK TO '0' AFTER                                         --
-----
-- counting for 2 periods - Waiting before reset
wait for (2 * period);
-- signal reset a 1, ie resetn a 0
resetn <= '0';
-- counting for 1 period
wait for period;
-- signal reset a 0, ie resetn a 1
resetn <= '1';           -- counting for (count_max_tb-2) periods
wait for ((count_max_tb-2) * period);
-- Valeurs des sorties attendues
-- end_counter = 0
assert end_counter = '0'
report "ERROR: end_counter not equals to 0" severity failure;

-- counting for 1 additionnal period (Total: (count_max_tb-1) periods)
wait for period;
-- Valeurs des sorties attendues
-- end_counter = 1
assert end_counter = '1'
report "ERROR: end_counter not equals to 1" severity failure;

-- counting for 1 additionnal period (Total: (count_max_tb) periods)
wait for period;
-- Valeurs des sorties attendues
-- end_counter = 0
assert end_counter = '0'
report "ERROR: end_counter not equals to 0" severity failure;

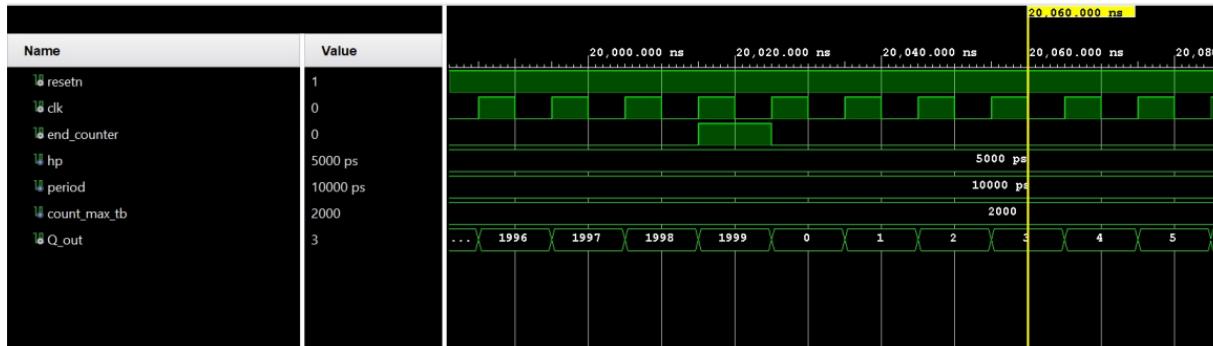
```

Principe:

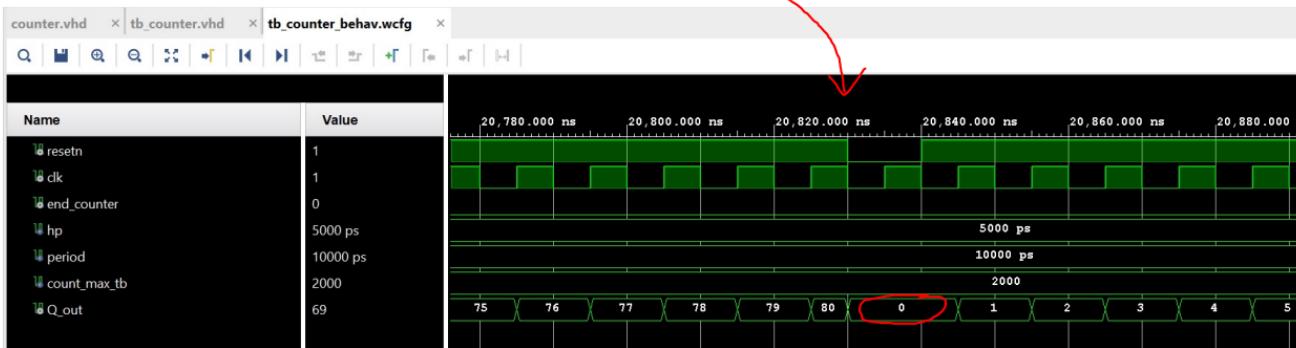
- On réalise un RESET au début afin d'initialiser le compteur.
- On teste la valeur de end_counter une période avant la fin du timer.
- On teste la valeur de end_counter à la fin du timer.
- On teste la valeur de end_counter une période après la fin du timer.

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

Suite au 2000ème front montant de l'horloge, le signal end_counter passe à 1 (0 à 1999).



Suite au RESET, le compteur est réinitialisé.



8. 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.

counter.vhd x tb_counter.vhd x tb_counter_behav.wcfg x Cora-Z7-10-Master.xdc

C:/FORMATION_SAFRAN/TP/TP2_Compteur/Cora-Z7-10-Master.xdc

Q | H | ← | → | X | // | ■ | ♀ |

```
1 ## This file is a general .xdc for the Cora Z7-07S Rev. B
2 ## To use it in a project:
3 ## - uncomment the lines corresponding to used pins
4 ## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project
5
6 # PL System Clock
7 set_property -dict { PACKAGE_PIN H16 IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L13P_T2_MRCC_35 Sch=sysclk
8 create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4} [get_ports { clk }];#set
9
10 # RGB LEDs
11 #set_property -dict { PACKAGE_PIN L15 IOSTANDARD LVCMOS33 } [get_ports { led0_b }]; #IO_L22N_T3_AD7N_35 Sch=led0_b
12 #set_property -dict { PACKAGE_PIN G17 IOSTANDARD LVCMOS33 } [get_ports { led0_g }]; #IO_L16P_T2_35 Sch=led0_g
13 #set_property -dict { PACKAGE_PIN N15 IOSTANDARD LVCMOS33 } [get_ports { led0_r }]; #IO_L21P_T3_DQS_AD14P_35 Sch=led0_r
14 #set_property -dict { PACKAGE_PIN G14 IOSTANDARD LVCMOS33 } [get_ports { led1_b }]; #IO_O_35 Sch=led1_b
15 #set_property -dict { PACKAGE_PIN L14 IOSTANDARD LVCMOS33 } [get_ports { led1_g }]; #IO_L22P_T3_AD7P_35 Sch=led1_g
16 #set_property -dict { PACKAGE_PIN M15 IOSTANDARD LVCMOS33 } [get_ports { led1_r }]; #IO_L23N_T3_35 Sch=led1_r
```

**Modification de la ligne ci-dessous dans le fichier de contrainte
=> Reprise de tout led dans le code défini dans les sorties**

```

10 # RGB LEDs
11 set_property -dict { PACKAGE_PIN L15 IOSTANDARD LVCMOS33 } [get_ports { out_led }];
12 #IO_L22N_T9_AD7N_35 Sch=out_led
13 #set_property -dict { PACKAGE_PIN G17 IOSTANDARD LVCMOS33 } [get_ports { led0_g }];
14 #IO_L16P_T2_35 Sch=led0_g

```

Modification du code compteur.vhd

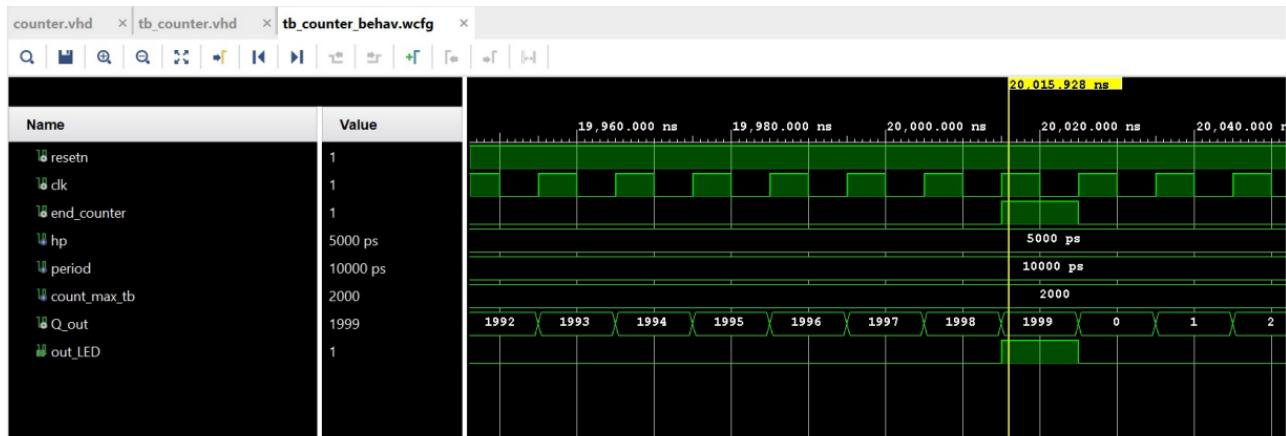
```
entity counter_unit is
    port (
        clk      : in std_logic;
        resetn   : in std_logic;
        end_counter : out std_logic;
        out_LED   : out std_logic
    );
end counter_unit;

--Partie combinatoire
cmd <= '1' when (Q_out = (count_max-1))
            else '0';
end_counter <= cmd;           -- Copie du signal end_counter
out_LED <= cmd;

end behavioral;
```

Penser à modifier la structure de l'entité dans le testbench également.

Le signal out_LED passe à 1 en même temps que end_counter.



9. 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.

Schéma initial pour le compteur

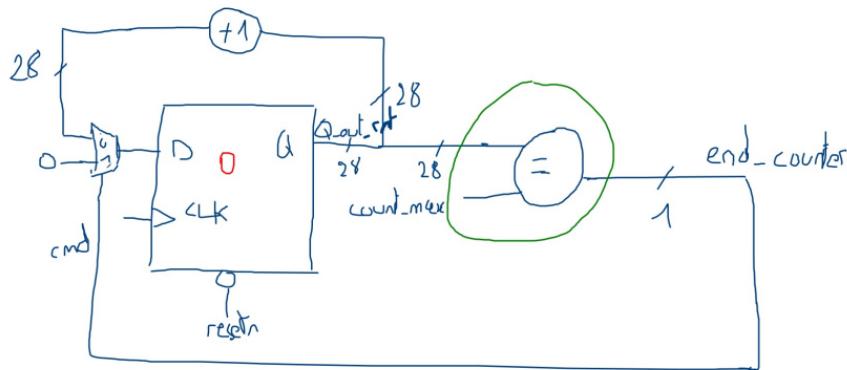


Table de vérité

end_counter	btn_restart	cmd
0	0	0
1	0	1
0	1	1
1	1	1

Cette table correspond à un OR.



Schéma final pour le compteur

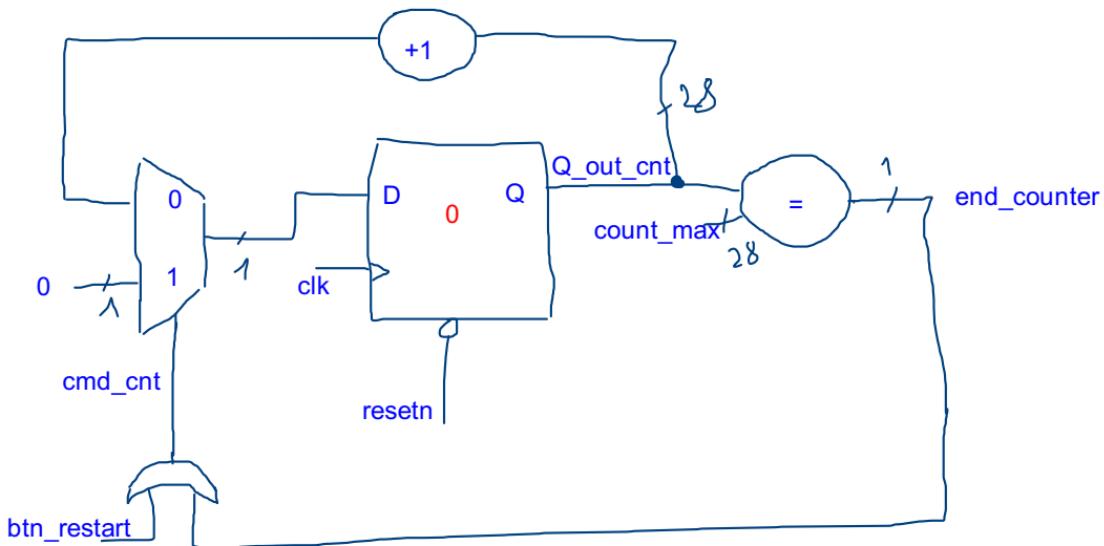
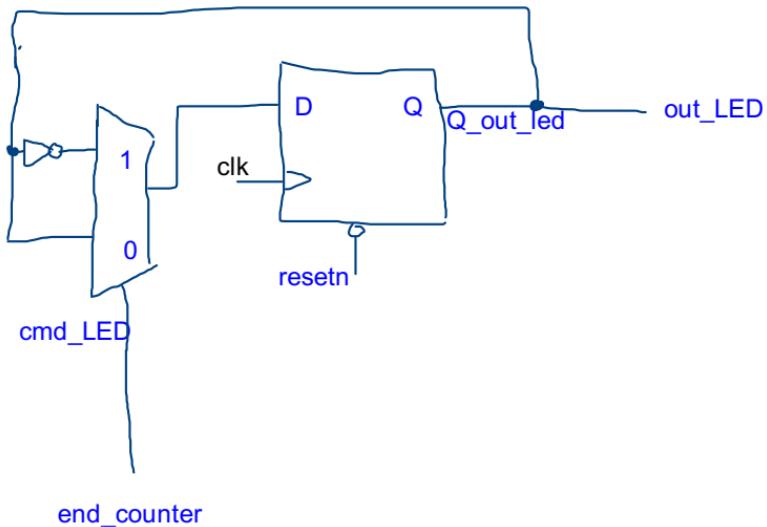


Schéma final pour la LED



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

Ajout de l'entrée `btn_restart` (bouton `restart`) et modifications des signaux internes de commande des multiplexeurs

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

entity counter_unit is
    port (
        clk           : in std_logic;          -- Horloge
        btn_restart   : in std_logic;          -- Entrée pour remise à zéro du compteur
        resetn        : in std_logic;          -- Entrée pour RESET des registres
        end_counter   : out std_logic;         -- Sortie indiquant la fin du comptage
        out_LED       : out std_logic          -- Sortie LED allumée ou éteinte
    );
end counter_unit;

architecture behavioral of counter_unit is

    --Déclaration des signaux internes
    constant count_max : natural := 2000;      -- Nombre de périodes correspondant à 2 secondes *** A remplacer par 2E8 ***
    signal Q_out_cnt : natural range 0 to count_max; -- Signal en sortie du registre du compteur
    signal cmd_cnt   : std_logic := '0';          -- Signal de commande pour la partie séquentielle
    signal cmd_LED   : std_logic := '0';          -- Recopie du signal end_counter pour utilisation dans partie séquentielle
    signal Q_out_LED : std_logic := '0';          -- Signal en sortie du registre de la LED (sortie LED)
```

J'ai séparé les signaux de commande des deux multiplexeurs.

Afin de limiter le nombre de bits des signaux, j'ai précisé le range de `Q_out_cnt`.

```
begin  
  
    --Partie sequentielle  
    process(clk,resetn)  
    begin  
        if(resetn = '0') then  
            Q_out_cnt <= 0;  
            Q_out_led <= '0';  
        elsif(rising_edge(clk)) then  
            if (cmd_cnt = '1') then  
                Q_out_cnt <= 0;  
            else  
                Q_out_cnt <= Q_out_cnt + 1;  
            end if;  
            if (cmd_LED = '1') then  
                Q_out_LED <= NOT(Q_out_LED);  
            end if;  
        end if;  
    end process;  
  
    --Partie combinatoire  
    cmd_LED <= '1' when (Q_out_cnt = (count_max-1))  
        else '0';  
    cmd_cnt <= '1' when ((Q_out_cnt = (count_max-1)) OR (btn_restart = '1'))  
        else '0';  
    end_counter <= cmd_LED;           -- Copie du signal end_counter  
    out_LED <= Q_out_led;  
  
end behavioral;
```

Multiplexeur en entrée du registre du compteur

Multiplexeur en entrée du registre du pilotage de la LED

Modifications dans le fichier du testbench

MATION_SAFRAN/TP/TP2_Compteur/tb_counter.vhd

library ieee;
use ieee.std_logic_1164.all;

entity tb_counter is
end tb_counter;

architecture behavioral of tb_counter is

signal btn_restart : std_logic := '0';
signal resetn : std_logic := '0';
signal clk : std_logic := '0';
signal end_counter : std_logic;

-- Les constantes suivantes permettent 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 100MHz
constant count_max_tb : natural := 2000; -- nombre de periodes correspondant au compteur

--Declaration de l'entite a tester
component counter_unit

port (
clk : in std_logic;
btn_restart : in std_logic;
resetn : in std_logic;
end_counter : out std_logic;
out_LED : out std_logic
);

end component;

Ajout de btn_restart dans les signaux à tester

```
begin  
  
    --Affectation des signaux du testbench avec ceux de l'entite a tester  
    uut: counter_unit  
        port map (  
            clk => clk,  
            resetn => resetn,  
            btn_restart => btn_restart,  
            end_counter => end_counter  
        );  
  
    --Simulation du signal d'horloge en continue
```

```
process  
begin  
    wait for hp;  
    clk <= not clk;  
end process;
```

```
-----  
-- THIRD TEST TO CHECK THE LED STOP BLINKING DURING RESET  
-- btn_restart SET TO '1' (active) && resetn SET TO '1' (inactive) --  
-----
```

```
-- INITIALISATION  
-- counting for 2 periods - Waiting before reset  
wait for (2 * period);  
-- signal reset a 1, ie resetn a 0  
resetn <= '0';  
btn_restart <= '0';  
-- counting for 1 period  
wait for period;  
-- signal reset a 0, ie resetn a 1  
resetn <= '1';  
-- counting for 80 periods - Waiting before pressing restart button  
wait for (80 * period);  
-- signal btn_restart a 1  
btn_restart <= '1';
```

```
assert end_counter = '0'  
report "ERROR: end_counter not equals to 0" severity failure;
```

```
-- STOP PRESSING button restart  
-- counting for 500 period  
wait for 500 * period;  
-- signal btn_restart a 0  
btn_restart <= '0';  
wait;
```

Ajout d'un troisième test avec activation du bouton btn_restart

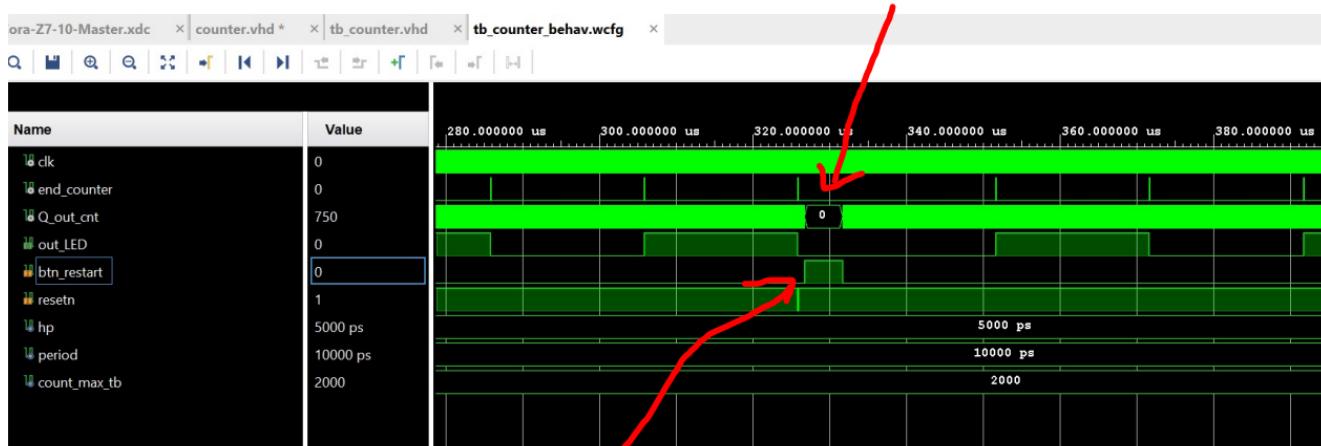
11. Associez la nouvelle entrée *restart* à un bouton.

Modifications du fichier de contrainte

```
Cora-Z7-10-Master.xdc x counter.vhd x tb_counter.vhd x tb_counter_behav.wcfg x  
C:/FORMATION_SAFRAN/TP/TP2_Compteur/Cora-Z7-10-Master.xdc  
Q | G | ← | → | X | ⌂ | □ | X | // | ☰ | ? |  
1 ## This file is a general .xdc for the Cora Z7-07S Rev. B  
2 ## To use it in a project:  
3 ## - uncomment the lines corresponding to used pins  
4 ## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project  
5  
6 # PL System Clock  
7 set_property -dict { PACKAGE_PIN H16 IOSTANDARD LVCMS33 } [get_ports { clk }]; #IO_L13P_T2_MRCC_35 Sch=sysclk  
8 create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4} [get_ports { clk }];#set  
9  
10 # RGB LEDs  
11 set_property -dict { PACKAGE_PIN L15 IOSTANDARD LVCMS33 } [get_ports { out_LED }]; #IO_L22N_T3_AD7N_35 Sch=out_led  
12 #set_property -dict { PACKAGE_PIN G17 IOSTANDARD LVCMS33 } [get_ports { led0_g }]; #IO_L16P_T2_35 Sch=led0_g  
13 #set_property -dict { PACKAGE_PIN N15 IOSTANDARD LVCMS33 } [get_ports { led0_r }]; #IO_L21P_T3_DQS_AD14P_35 Sch=led0_r  
14 #set_property -dict { PACKAGE_PIN G14 IOSTANDARD LVCMS33 } [get_ports { led1_b }]; #IO_O_35 Sch=led1_b  
15 #set_property -dict { PACKAGE_PIN L14 IOSTANDARD LVCMS33 } [get_ports { led1_g }]; #IO_L22P_T3_AD7P_35 Sch=led1_g  
16 #set_property -dict { PACKAGE_PIN M15 IOSTANDARD LVCMS33 } [get_ports { led1_r }]; #IO_L23N_T3_35 Sch=led1_r  
17  
18 # Buttons  
19 set_property -dict { PACKAGE_PIN D20 IOSTANDARD LVCMS33 } [get_ports { btn_restart }]; #IO_L4N_T0_35 Sch=btn[0]  
20 #set_property -dict { PACKAGE_PIN D19 IOSTANDARD LVCMS33 } [get_ports { btn[1] }]; #IO_L4F_T0_35 Sch=btn[1]  
21
```

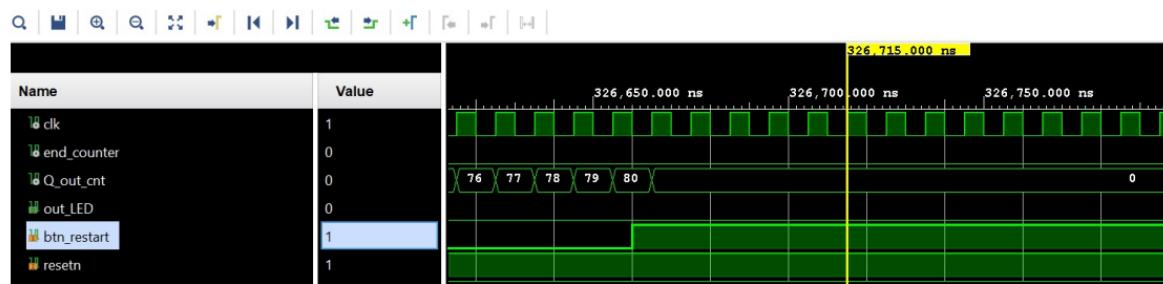
12. Mettez à jour votre testbench puis vérifier votre design avec une simulation. Quels sont les signaux que vous devez observer ?

Compteur remis à zéro

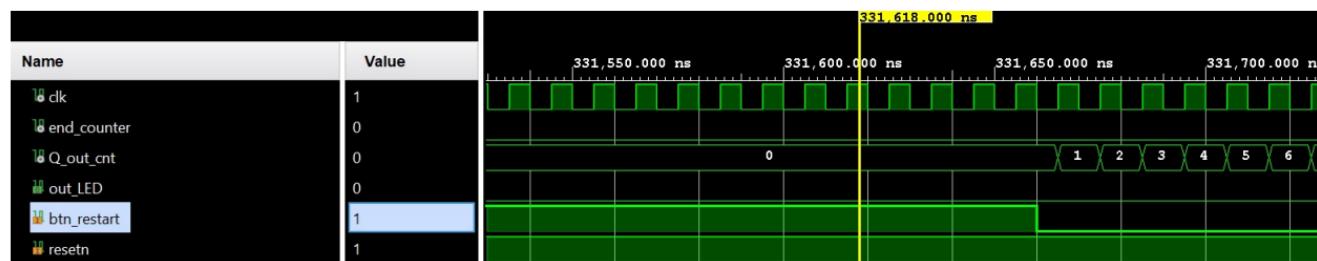


Bouton restart activé

EN ZOOMANT: APPUI SUR LE BOUTON btn restart



EN ZOOMANT: BOUTON btn restart relaché



TEST AVEC LES VALEURS REELS (LED allumée 2 secondes puis éteinte 2 secondes)

Je me suis aperçu tout à la fin du TP qu'il était préférable de faire fonctionner le resetn à l'inverse. Bref, la suite du TP est réalisé avec le code:

```
begin
    --Partie sequentielle
    process(clk,resetn)
    begin
        if(resetn = '1') then
            Q_out_cnt <= 0;
            Q_out_led <= '0';
        elsif(rising_edge(clk)) then
            if (cmd_cnt = '1') then
                Q_out_cnt <= 0;
            else
                Q_out_cnt <= Q_out_cnt + 1;
            end if;
            if (cmd_LED = '1') then
                Q_out_LED <= NOT(Q_out_LED);
            end if;
        end if;
    end process;

    --Partie combinatoire
    cmd_LED <= '1' when (Q_out_cnt = (count_max-1))
    else '0';
    cmd_cnt <= '1' when ((Q_out_cnt = (count_max-1)) OR (btn_restart = '1'))
    else '0';
    end_counter <= cmd_LED;           -- Copie du signal end_counter
    out_LED <= Q_out_led;

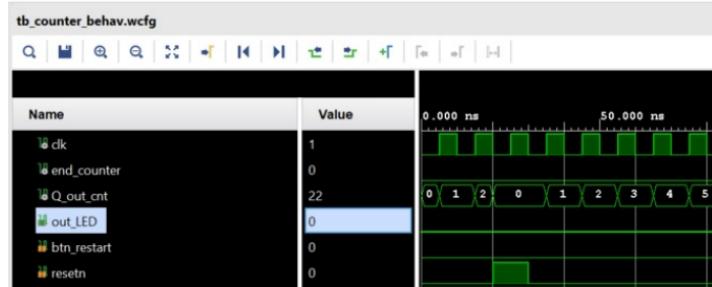
end behavioral;
```

1 au lieu de 0

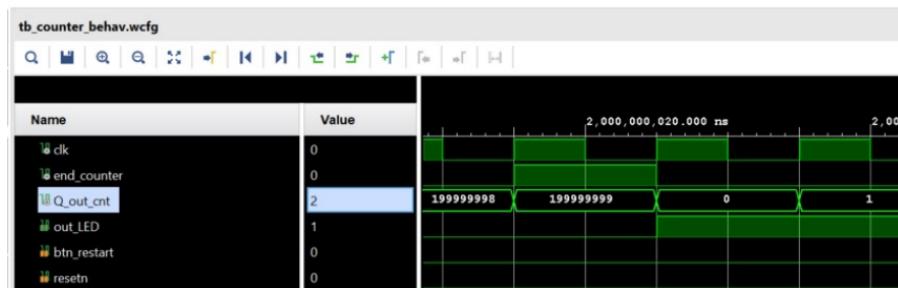
Premier test:

Le signal resetn passe de 0 à 1. Le compteur redémarre bien à 0 et la sortie out_LED est bien à 0 également.

Le signal resetn passe de 1 à 0. Le comptage démarre.



Le compteur atteint 199 999 999. Il redémarre à 0 et la sortie out_LED passe à 1.

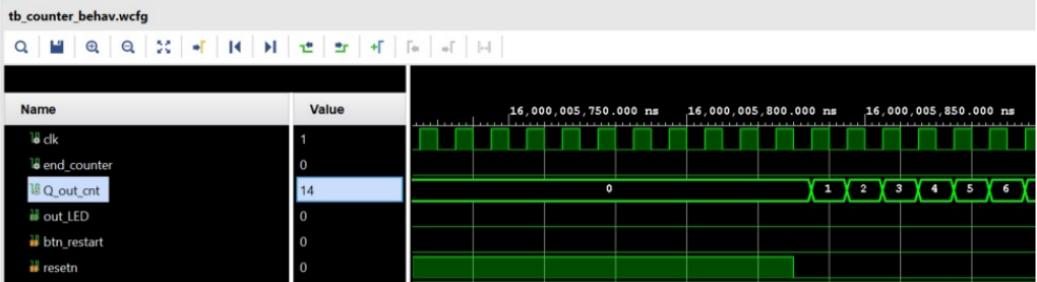


Deuxième test:

Test similaire en maintenant resetn à 1 pendant un certain temps.
On observe bien que le compteur reste à 0 durant cette période.



Puis, le compteur repart.

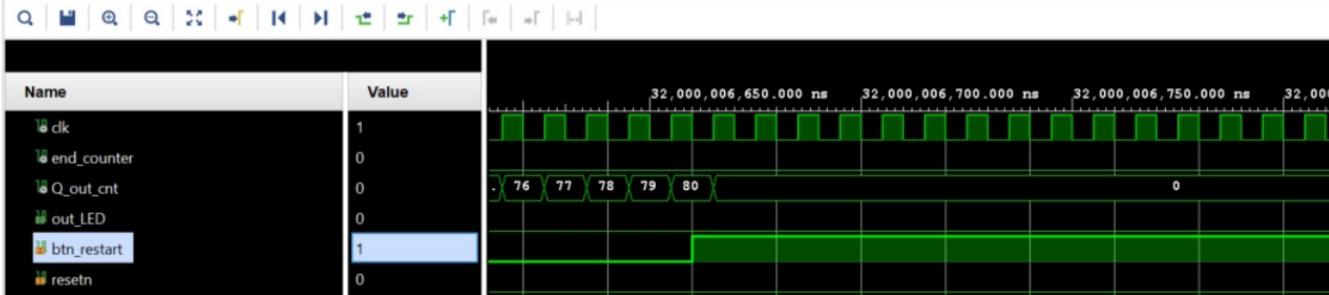


Troisième test:

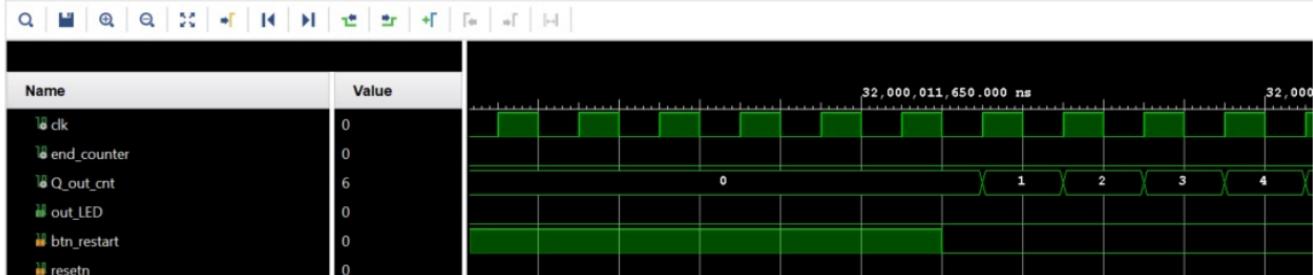
Le signal `btn_restart` passe de 0 à 1. Le compteur redémarre bien à 0 et la sortie `out_LED` est bien maintenu dans son état.

Le signal `btn_restart` passe de 1 à 0. Le comptage démarre.

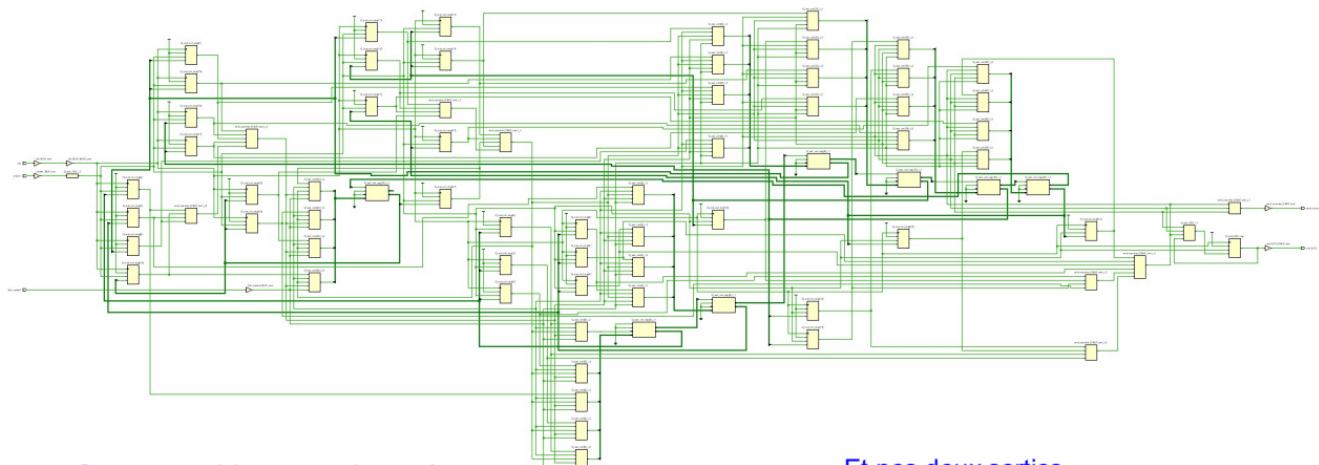
tb_counter_behav.wcfg



tb_counter_behav.wcfg

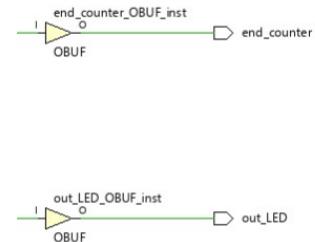
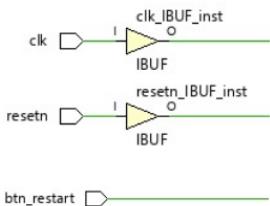


13. Exécutez la synthèse puis ouvrez la schématique. Identifiez sur la schématique les différents éléments de votre architecture RTL.

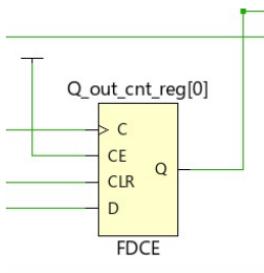


On retrouve bien nos trois entrées.

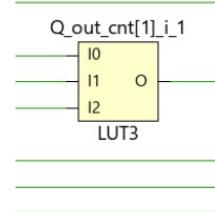
Et nos deux sorties.



Des registres.



Et des LUTs.



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

Synthesis report avec le compteur max count max égal à 2000

```
73 -----
74 Start RTL Component Statistics
75 -----
76 Detailed RTL Component Info :
77 +---Adders :
78     2 Input   11 Bit      Adders := 1
79 +---Registers :
80         11 Bit      Registers := 1
81         1 Bit       Registers := 1
82 +---Muxes :
83     2 Input   11 Bit      Muxes := 1
84 -----
85 Finished RTL Component Statistics
86 -----
```

178 Report Cell Usage:

Cell	Count
BUFG	1
LUT1	1
LUT2	1
LUT3	2
LUT4	5
LUT5	3
LUT6	6
FDCE	12
IBUF	3
OBUF	2

+-----+
|Cell |Count |
+-----+
1	BUFG	1
2	LUT1	1
3	LUT2	1
4	LUT3	2
5	LUT4	5
6	LUT5	3
7	LUT6	6
8	FDCE	12
9	IBUF	3
10	OBUF	2
+-----+

"+1" pour le comptage
Registre pour le comptage
11 bits pour coder 2000
Registre pour la LED
Je m'attendais à trouver 2 multiplexeurs, Vivado a réussi à les combiner en un seul.
LUT = Look Up Table
 $11+1 = 12 \text{ bits} \Rightarrow 12 \text{ registres } 1 \text{ bit}$
3 entrées
2 sorties

Synthesis report avec le compteur max count_max égal à 200 000 000

Remarque: Lorsqu'on augmente la valeur de count_max, le rapport "RTL Component Statistics" n'apparaît plus pour des raisons d'optimisation du logiciel.
Avec "count_max = 20 000", c'est déjà le cas.

Report Cell Usage:		
	Cell	Count
173	1	BUFG
174	2	CARRY4
175	3	LUT1
176	4	LUT3
177	5	LUT4
178	6	LUT5
179	7	LUT6
180	8	FDCE
181	9	IBUF
182	10	OBUF
183		
184		
185		
186		
187		

Annotations:

- A blue arrow points from the 'Count' column of the CARRY4 row to the text "Retenues".
- A blue arrow points from the 'Count' column of the FDCE row to the text "200 000 000 est codé sur 28 bits
 $28+1 = 29$ bits => 29 registres 1 bit".

A ce stade, il devient intéressant de tester sur la carte:

- Run implementation,
 - Generate Bitstream

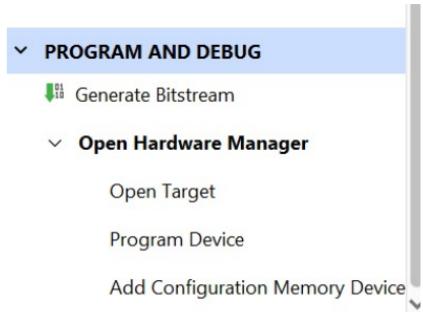
Une erreur critique apparait. Vivado m'indique que les signaux end_counter et resetn ne sont pas définis en entrées/ sorties de la carte. => Je les ajoute dans le fichier de contrainte.

Schematic x counter.vhd x tb_counter.vhd x Cora-Z7-10-Master.xdc x

C:/FORMATION_SAFRAN/TP/TP2_Compteur/Cora-Z7-10-Master.xdc

1 ## This file is a general .xdc for the Cora Z7-07S Rev. B
2 ## To use it in a project:
3 ## - uncomment the lines corresponding to used pins
4 ## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project
5
6 # PL System Clock
7 set_property -dict { PACKAGE_PIN H16 IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L13P_T2_MRCC_35 Sch=sysclk
8 create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4} [get_ports { clk }];#set
9
10 # RGB LEDs
11 set_property -dict { PACKAGE_PIN L15 IOSTANDARD LVCMOS33 } [get_ports { out_LED }]; #IO_L22N_T3_AD7N_35 Sch=out_led
12 set_property -dict { PACKAGE_PIN G17 IOSTANDARD LVCMOS33 } [get_ports { end_counter }]; #IO_L16P_T2_35 Sch=led0_g
13 #set_property -dict { PACKAGE_PIN N15 IOSTANDARD LVCMOS33 } [get_ports { led0_r }]; #IO_L21P_T3_DQS_AD14P_35 Sch=led0_r
14 #set_property -dict { PACKAGE_PIN G14 IOSTANDARD LVCMOS33 } [get_ports { led1_b }]; #IO_0_35 Sch=led1_b
15 #set_property -dict { PACKAGE_PIN L14 IOSTANDARD LVCMOS33 } [get_ports { led1_g }]; #IO_L22P_T3_AD7P_35 Sch=led1_g
16 #set_property -dict { PACKAGE_PIN M15 IOSTANDARD LVCMOS33 } [get_ports { led1_r }]; #IO_L23N_T3_35 Sch=led1_r
17
18 # Buttons
19 set_property -dict { PACKAGE_PIN D20 IOSTANDARD LVCMOS33 } [get_ports { btn_restart }]; #IO_L4N_T0_35 Sch=btn[0]
20 set_property -dict { PACKAGE_PIN D19 IOSTANDARD LVCMOS33 } [get_ports { resetn }]; #IO_L4P_T0_35 Sch=btn[1]

Ensuite, "Open Hardware Manager", puis "Open target" + "autoconnect".



Et enfin, "Program Device" en sélectionnant la carte.

A ce stade, pour que la LED clignote, il faut rester appuyer sur le bouton noir correspondant au signal d'entrée resetn. Pas très pratique pour réaliser des tests...
=> Modification du code pour un fonctionnement inverse du bouton resetn.

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

Set Up Debug

Nets to Debug

The nets below will be debugged with ILA cores. To add nets click "Find Nets to Add". You can also select nets in the Netlist or other windows, then drag them to the list or click "Add Selected Nets".

Name	Clock Domain	Driver Cell	Probe Type
> Q_out_cnt_reg (28)	clk_IBUF_BUFG	FDCE	Data and Trigger
btn_restart_IBUF	clk_IBUF_BUFG	IBUF	Data and Trigger
end_counter_OBUF	clk_IBUF_BUFG	LUT3	Data and Trigger
out_LED_OBUF	clk_IBUF_BUFG	FDCE	Data and Trigger
resetn_IBUF	clk_IBUF_BUFG	IBUF	Data and Trigger

Find Nets to Add...

ILA Core Options

Choose features for the ILA debug cores.

Sample of data depth: 1024

Input pipe stages: 0

Trigger and Storage Settings

Capture control

Advanced trigger

? **< Back** **Next >** **Finish** **Cancel**

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

Project Summary X | counter.vhd X | tb_counter.vhd X | Cora-Z7-10-Master.xdc X | Timing Summary - Route Design - impl_1 X
C:/FORMATION_SAFRAN/TP/TP2_Compteur/Compteur.rns/impl_1/counter_unit_timing_summary_routed.rpt

Q | L | ← | → | X | D | // | B | ? |

Clock summary Next | Previous | Highlight | Match Case | Whole Words | 1 Match(es)

```
135:  
136:  
137:-----  
138: | Clock Summary  
139: |-----  
140:  
141:  
142: Clock  
143: -----  
144: dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_inst/SERIES7_BSCAN.bscan_inst/TCK  
145: sys_clk_pin  
146:
```

	Waveform(ns)	Period(ns)	Frequency (MHz)
dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_inst/SERIES7_BSCAN.bscan_inst/TCK sys_clk_pin	{(0.000 16.500) (0.000 4.000)}	33.000 8.000	30.303 125.000

The diagram illustrates the timing characteristics of the clock signal. The horizontal axis represents time, and the vertical axis represents the state of the clock. The signal starts at a low level, transitions to a high level at 4 ns, remains high until 16.5 ns, and then returns to a low level. A red arrow indicates the period of 8 ns, which is the time between consecutive high-to-low transitions. A green circle highlights the 4 ns interval during which the signal is transitioning from low to high.

Project Summary X counter.vhd X tb_counter.vhd X Cora-Z7-10-Master.xdc X Timing Summary - Route Design - impl_1 X

C:/FORMATION_SAFRAN/TP/TP2_Compteur/Compteur.runs/impl_1/counter_unit_timing_summary_routed.rpt

Read-only

Design Timing Next Previous Highlight Match Case Whole Words 1 Match(es)

```
118 : 12. checking latch_loops (0)
119 -----
120 There are 0 combinational latch loops in the design through latch input
121
122
123
124 -----
125 | Design Timing Summary
126 |
127
128
129 WNS(ns) TNS(ns) TNS Failing Endpoints TNS Total Endpoints WHS(ns) THS(ns) THS Failing Endpoints THS Total Endpoints WPWS(ns) TPWS(ns)
130 ----- 0.000 0 3638 0.022 0.000 0 3622 2.750 0.000
131
132
133
```

Total Negative Slack = 0
=> Pas de violation de set up

Total Hold Slack = 0
=> Pas de violation de hold

SOURCE ET DESTINATION DU CHEMIN



Project Summary | counter.vhd | tb_counter.vhd | Cora-Z7-10-Master.xdc | Timing Summary - Route Design - impl_1 | C:/FORMATION_SAFRAN/TP/TP2_Compteur/Compteur/Compteur.rpt | Read-only | ...

C:/FORMATION_SAFRAN/TP/TP2_Compteur/Compteur/runs/impl_1/counter_unit_timing_summary_routed.rpt

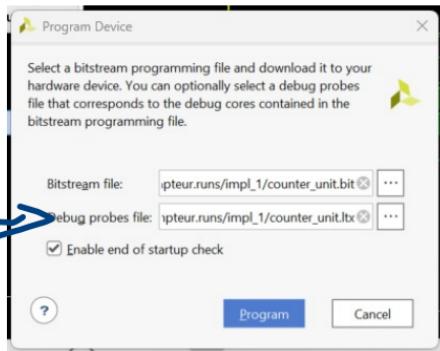
Max Delay | Next | Previous | Highlight | Match Case | Whole Words | 23 Match(es)

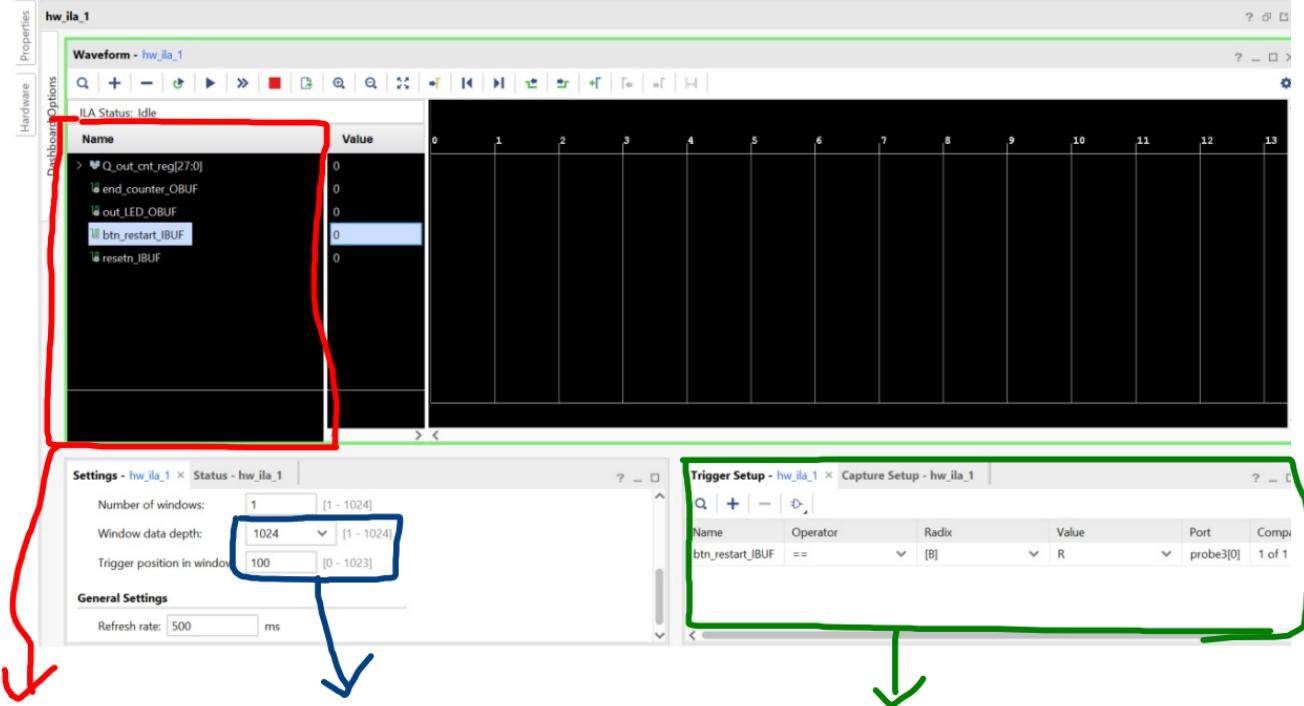
Max Delay Paths

Slack (MET) : 26.160ns (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
Destination: dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_switch/portno_temp_reg[2]/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
Path Group: dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_inst/SERIES7_BSCAN.bscan_inst/TCK
Path Type: Setup (Max at Slow Process Corner)
Requirement: 33.000ns (dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_inst/SERIES7_BSCAN.bscan_inst/TCK rise@33.000ns - dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_inst/TCK fall@33.000ns)
Data Path Delay: 6.814ns (logic 1.810ns (26.561%) route 5.004ns (73.439%))
Logic Levels: 6 (CARRY#2 LUT3#1 LUT4#1 LUT5#1 LUT6#1)
Clock Path Skew: -0.059ns (DDC - SCD + CPR)
Destination Clock Delay (DDC): 3.079ns = (36.079 - 33.000)
Source Clock Delay (SCD): 3.513ns
Clock Pessimism Removal (CPR): 0.376ns
Clock Uncertainty: 0.035ns ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
Total System Jitter (TSJ): 0.071ns
Total Input Jitter (TIJ): 0.000ns
Discrete Jitter (DJ): 0.000ns
Phase Error (PE): 0.000ns

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

Lorsqu'on génère le bitstream avec le debug, on a un fichier supplémentaire.



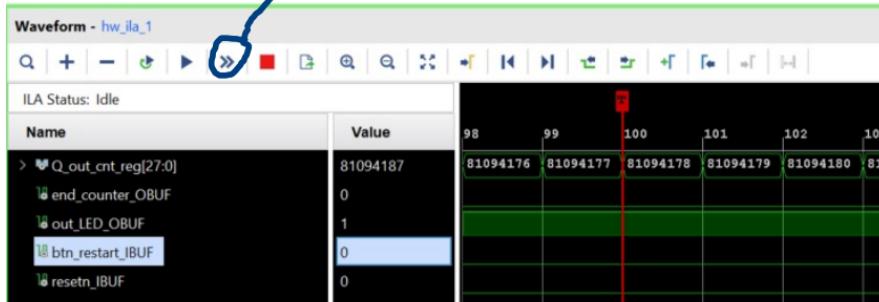


Signaux sous observation

A quel moment a lieu le trigger:
Ici au 100ème front montant sur 1024

Trigger btn_restart sur front montant

En cliquant sur >> lorsque la LED est allumée, on obtient:

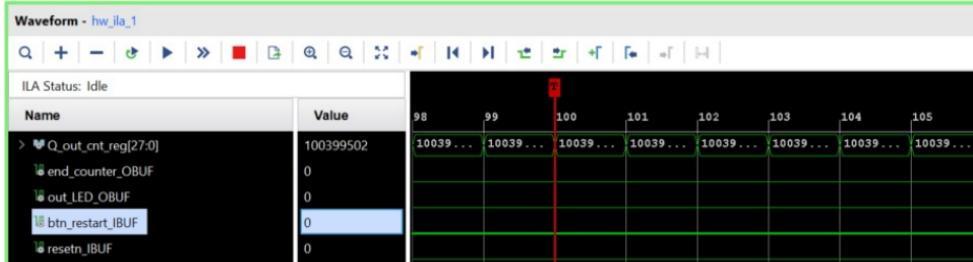


On observe bien que le compteur s'incrémente.

Le signal out_LED est bien à l'état haut.

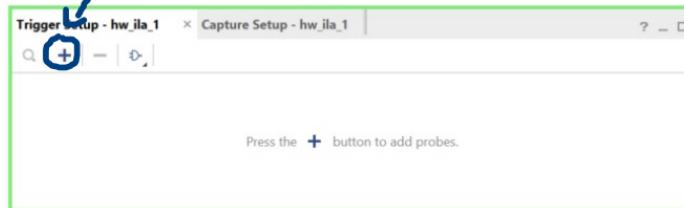
Et lorsque la LED est éteinte:

Le signal out_LED est bien à l'état bas.



ETUDE DU BOUTON RESTART SUR FRONT MONTANT (Utilisation de la partie Trigger de ILA)

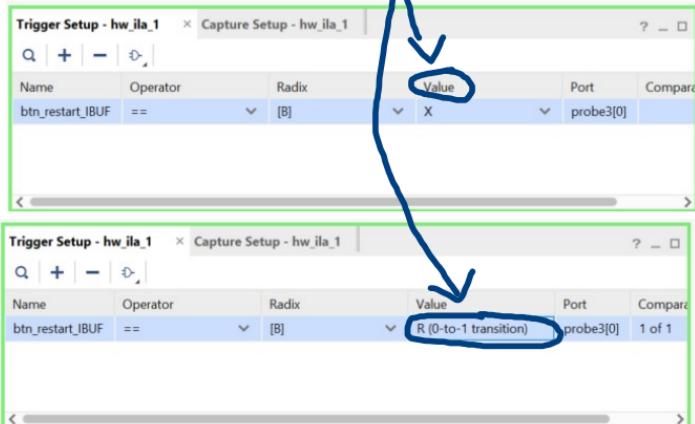
Cliquer sur +



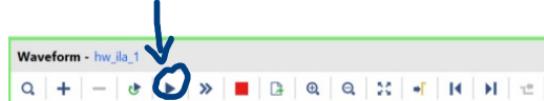
Choisir le
signal utilisé
comme Trigger



Modifier le paramètre Value en
choisissez un front montant



Lancer le Trigger avec un click droit en sélectionnant "Run Trigger" ou sur le triangle bleu, puis sur la carte appuyer sur le bouton `btn_restart`



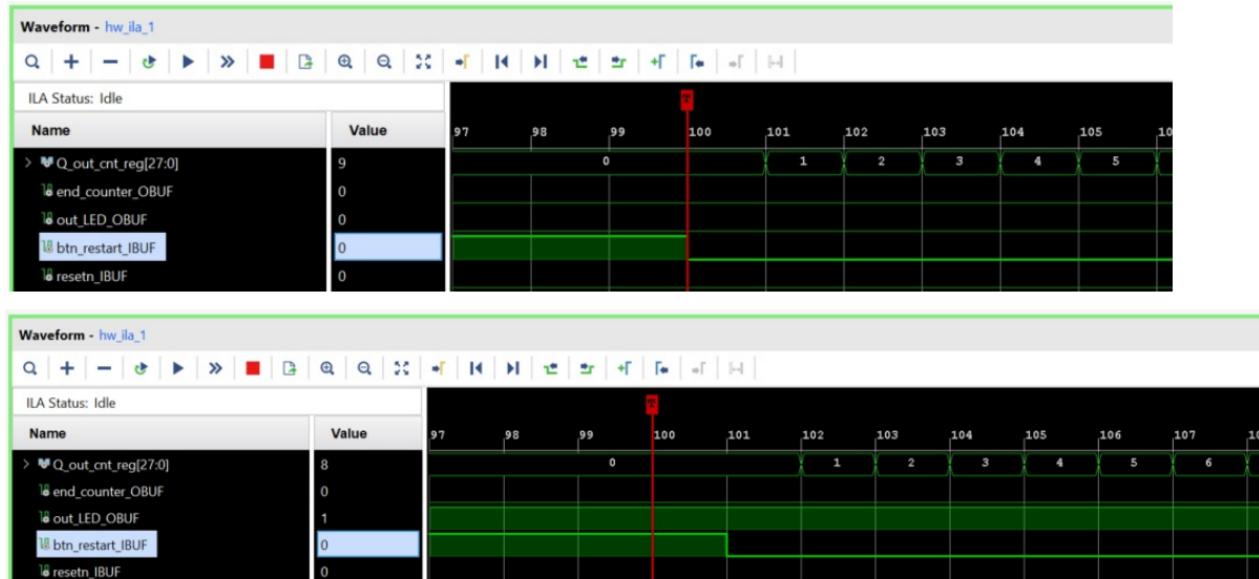
Suivant l'état de la LED, on obtient les chronogrammes ci-dessous:



On observe bien que suite au front montant de `btn_restart` le compteur est réinitialisé.

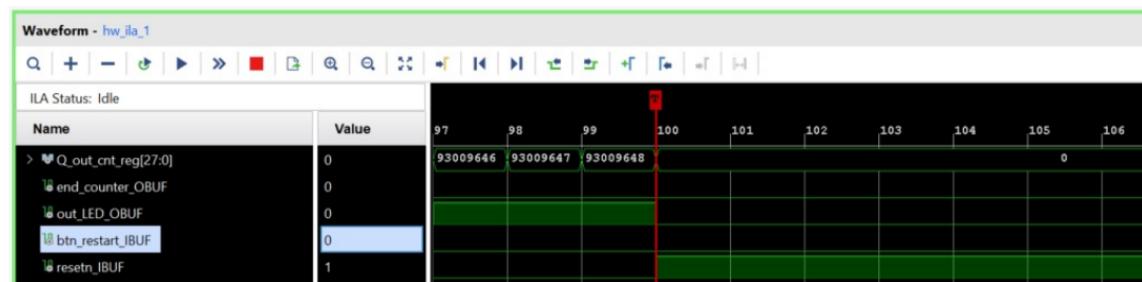
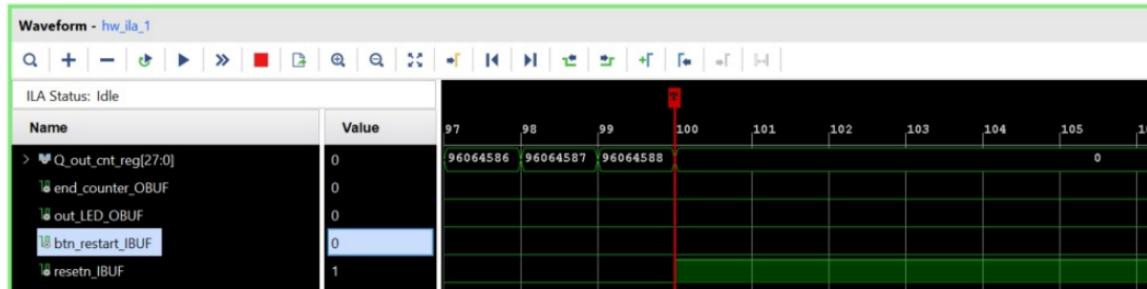
ETUDE DU BOUTON RESTART SUR FRONT DESCENDANT

Suivant l'état de la LED, on obtient les chronogrammes ci-dessous:



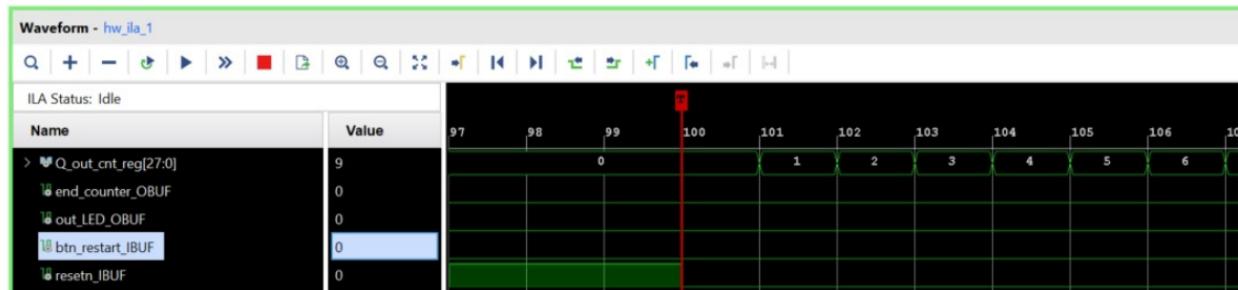
On observe bien que suite au front descendant de btn_restart le compteur redémarre.

ETUDE DU BOUTON RESET SUR FRONT MONTANT

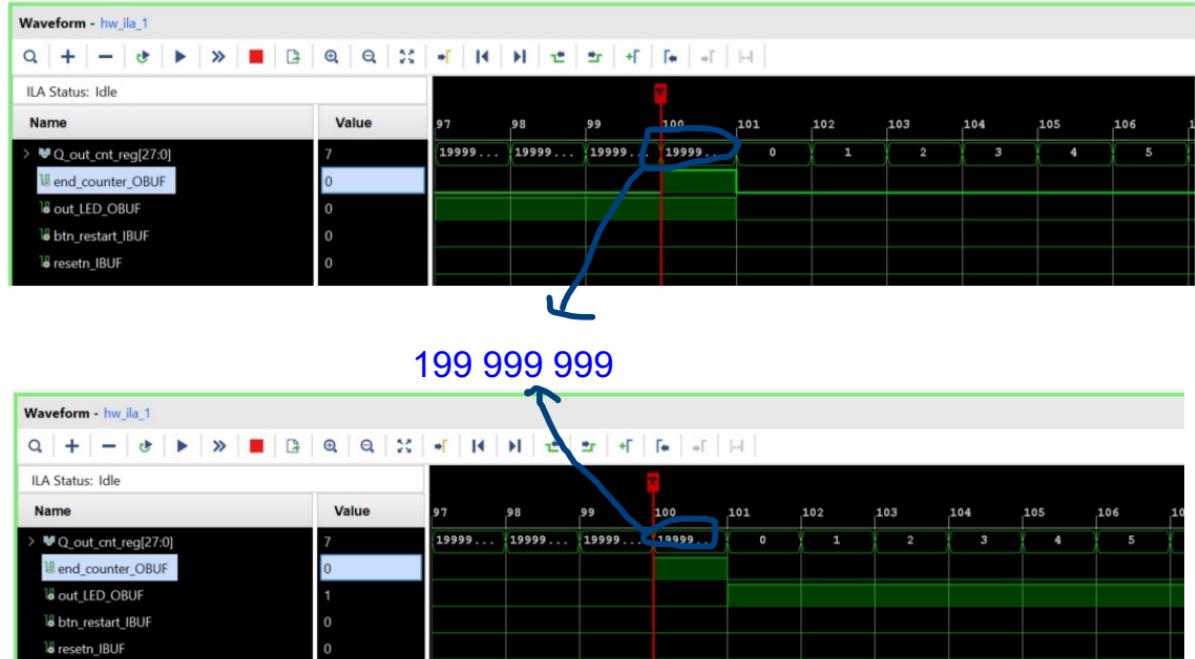


On observe bien une remise à 0 du compteur sur front montant de resetn.

ETUDE DU BOUTON RESET SUR FRONT DESCENDANT



ETUDE DU SIGNAL end_counter SUR FRONT MONTANT



On observe bien que lorsque le compteur atteint son max il repasse à 0 et que la LED change d'état (haut vers bas ou bas vers haut).