Rapport TP 02

Samory DIABY

Table des matières

1 Question 1 : Combien de période pour attendre 2 secondes ?

L'horloge du système est fixée à 100MH. On a donc une période de 10ns.

Soit 10ns $= 1 \times 10^{-8}$ s, pour attendre 2s on a : $\frac{2}{1 \times 10^{-8}} = 2 \times 10^{8}$ périodes.
On doit attendre 200 millions de périodes pour attendre 2 secondes.

Combien de bits faut-il pour représenter cette valeur ?

Pour représenter le nombre 200000000 en binaire il nous faut au moins 28 bits.

2 Question 2 : dessiner le schéma RTL du compteur

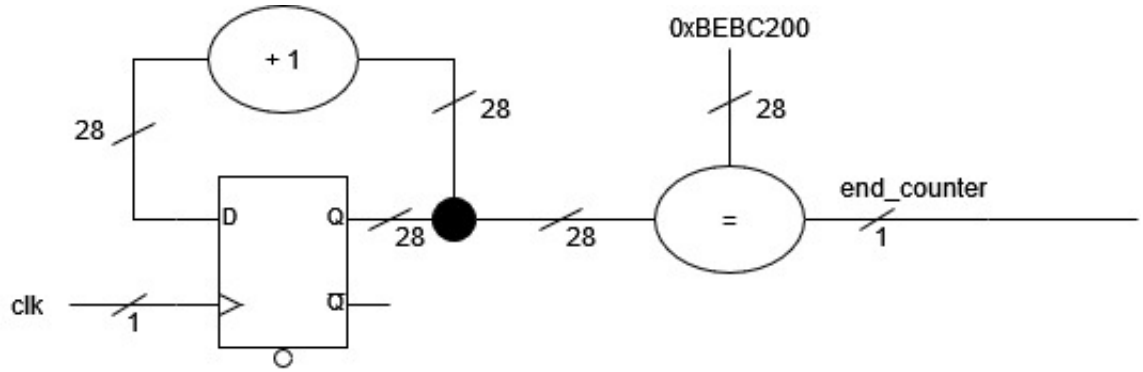

Figure 1 – Schéma RTL du compteur

3 Question 3 : Ajouter une condition pour que le compteur soit remis à 0 lorsqu'il atteint la valeur souhaitée

Si le compteur arrive à la valeur souhaitée, le signal **end_counter** passe à 1 et on utilise ce signal pour changer la valeur de commande d'un multiplexeur. Le multiplexeur utilisera alors l'ancienne valeur de "D" + 1 si le signal de commande **cmd** est à "0", et utilisera un signal constant à "0" sur 28 bits si le signal **cmd** est à "1".

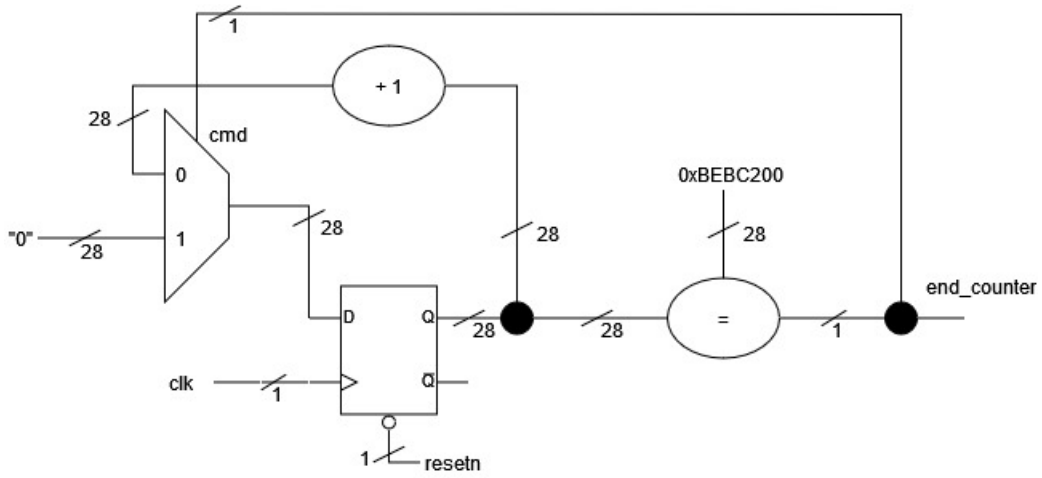

Figure 2 – Schéma RTL du compteur avec remise à zéro

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

Il y a 2 signaux d'entrées qui sont **clk** et **resetn** (à partir de la question 9, il y aura une troisième entrée pour le signal **restart**).
Il y a 1 signal de sortie qui est **end_counter**.
Il y a 3 signaux internes qui sont la sortie **Q** de la bascule D et le signal contenant la valeur limite du compteur et le signal **cmd** de commande du multiplexeur (à partir de la question 9, il y aura 2 entrées pour gérer le stockage de l'état du signal de sortie **end_counter**).

5 Question 5 : Ecrivez à présent le compteur en VHDL en suivant le schéma RTL.

```vhdl
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;
        end_counter : out std_logic
    );
end counter_unit;

architecture behavioral of counter_unit is

  --Declaration des signaux internes
  constant cnt_limit    : positive := 10#4#; -- 16#BEBC1FF#; -- counter limit value : 200 000 000 - 1
  -- counter signal
  signal cnt            : std_logic_vector(27 downto 0) := (others => '0');
  -- command signal for the mux
  signal cmd            : std_logic := '0';
  -- internal signal to store the comparison between 'cnt' and 'cnt_limit'
  signal cmp_cnt        : std_logic := '0';

  begin

    --Partie sequentielle
    process(clk, resetn)
    begin
        -- begin counter logic
      if(resetn = '1') then
          -- reset triggered -> clear the counter
          cnt <= (others => '0');
      elsif(rising_edge(clk)) then
          if cmd = '0' then
              -- we add '1' to the counter at each rising_edge of clk
              cnt <= std_logic_vector(unsigned(cnt) + 1);
          else
              -- the counter reached the limit value
              -- -> we choose the '0' entry of the mux
              cnt <= (others => '0');
          end if;
      end if;
      -- end counter logic
    end process;

    --Partie combinatoire

```

```
50      -- if cnt >= cnt_limit -> cmp_cnt = 1 -> the counter reached the
            allowed limit
51      -- in every other case -> cmp_cnt = '0'
52      -- we don't set 'end_counter' directly because
53      -- we won't be able to use it later to set the 'cmd' signal
54      -- (because it's an output signal)
55      cmp_cnt <= '1' when cnt >= cnt_limit else '0';
56
57          -- set the cmd bit according to the cnt_limit const
58          cmd <= cmp_cnt;
59
60        -- set 'end_counter' output signal
61      end_counter <= cmp_cnt;
62
63  end behavioral;
```

6 Question 6 : Ecrivez un fichier de testbench pour tester votre design

```
library ieee;
use ieee.std_logic_1164.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 end_counter : std_logic := '0';

  -- 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;
      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,
        end_counter => end_counter
    );

  --Simulation du signal d'horloge en continue
  process
  begin
    wait for hp;
    clk <= not clk;
  end process;

  -- resetn signal simulation (Q6)
  process
  begin
    wait for 60 ns;
    resetn <= not resetn;
  end process;

```

```
50
51  process
52     variable cnt_period : time := 45 ns;
53  begin
54     -- TESTS A EFFECTUER
55
56       -- handle the initial case when there is
57       -- a "missing" rising edge
58       if now > 45 ns then
59           -- when we pass the first count we wait for 50 ns
60           -- to test end_counter
61           cnt_period := 50 ns;
62       end if;
63
64     wait for cnt_period;
65
66     assert end_counter = '1'
67         report "Q6 test failed : 'end_counter' is supposed to be 1 when '
               cnt' is equal to '4'"
68         severity failure;
69
70  end process;
71
72
73 end behavioral;
```

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

On doit observer un retour à 0 du compteur lorsqu'il a atteint la valeur limite définie (dans notre cas 199 999 999) qui correspond à 2s ainsi que le passage à "1" du signal de sortie **end_counter** le temps d'un coup d'horloge.

Pour tester dans vivado, j'ai réduit la valeur limite à 4 pour observer une remise à 0 du compteur toutes les 5 périodes.

Figure 3 – Chronogramme du test bench Q7

On peut voir sur le chronogramme que le compteur est remis à zéro (au front montant suivant) lorsqu'il atteint la valeur '4'. On remarque aussi que le testbench s'arrête au bout de 90 ns car **end_counter** n'est pas à '1' à cause du signal **resetn** qui empêche le compteur de continuer à compter.

8 Question 8 : Associez une LED avec le signal de test d'arrêt du compteur.

LED modifiée : led0_b.

Ligne correspondante dans le fichier **Cora-Z7-10-Master.xdc**

```
set_property -dict {PACKAGE_PIN L15 IOSTANDARD LVCMOS33} [get_ports end_counter]
```

9 Question 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.

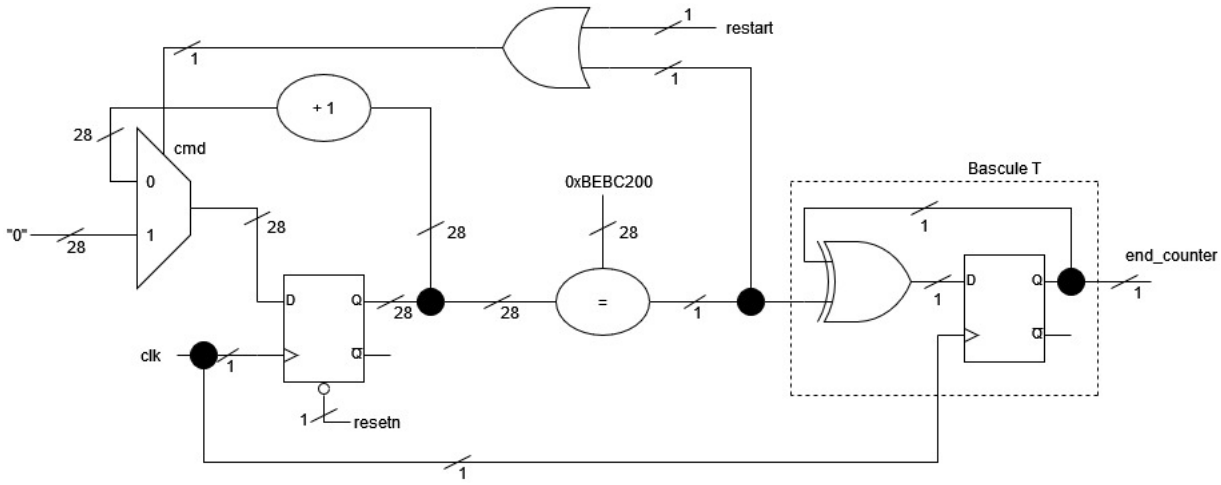

Figure 4 – Schéma RTL du compteur avec signal restart

10 Question 10 : Faites les mises à jour nécessaires sur le code VHDL pour correspondre au nouveau schéma.

Cf. fichier : counter.vhd

11 Question 11 : Associez la nouvelle entrée restart à un bouton.

Bouton associé : btn[0].

Ligne correspondante dans le fichier **Cora-Z7-10-Master.xdc**

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

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

Il faut observer les signaux **end_counter**, **restart**, **cnt** et **cmd**.

Au début de la simulation (t= 0ns) les signaux sont initialisés à 0, puis à chaque front montant de l'horloge le compteur **cnt** est incrémenté de 1. Au moment où **cnt** atteint la valeur limite (dans l'exemple ci-dessus 4) on passe le signal **cmd** à 1, ce qui va remettre à 0 le compteur au prochain front montant d'horloge et changer la valeur du signal **end_counter** pour lui assigner $\overline{\textbf{end\_counter}}$ (effectué par la bascule T) jusqu'à ce que le compteur atteigne une nouvelle fois sa valeur limite. On peut également observer que le compteur ne s'incrémente pas tant que le bouton **restart** est pressé.

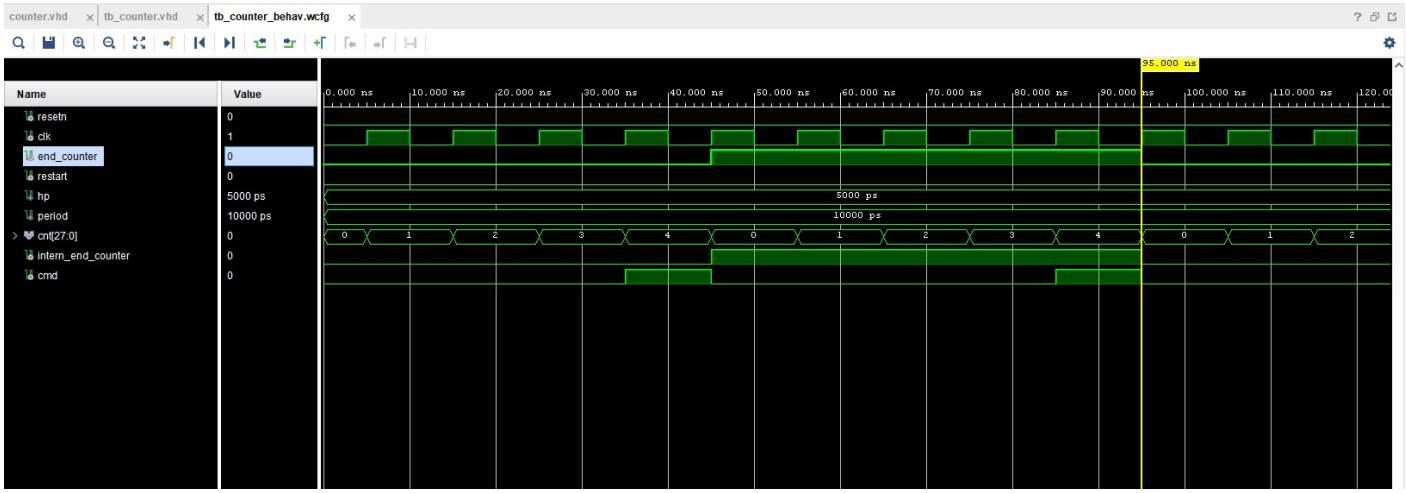

Figure 5 – Chronogramme testbench avec bouton restart

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

Les différents éléments de notre schéma RTL que l'on retrouve dans le schematic sont :

- Pour la partie compteur, on a 28 registres nommés **cnt_reg[0...27]** ainsi que des **carry_4** associées nommées **cnt_reg[0...24]_i_1** (les carry 4 vont prendre les registres cnt 4 par 4).
- Pour la partie de mémorisation du signal de sortie (pour que la led reste allumée ou éteinte 2s) j'ai essayé de "recréer" une bascule T. On retrouve donc 1 registre nommé **intern_end_counter_reg** ainsi qu'une LUT associée nommées **intern_end_counter_i_1** qui sert pour la partie Xor de la bascule T. La bascule T est représentée dans le code comme un signal interne **intern_end_counter**.
- On retrouve un buffer **end_counter_OBUF_inst** qui permet de passer du signal interne à la sortie **end_counter**.

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

En regardant le rapport de synthèse, on observe :

- Dans le **RTL components statistics** qu'on a 1 XOR à 2 entrées sur 1 bit ainsi qu'1 registre sur 1 bit, ce qui semble correspondre à la bascule T.
- Dans le **Report cell usage** (ci-dessous) qu'on a 29 FDCE ce qui semble correspondre aux 28 bascules D de notre compteur 28 bits ainsi qu'1 FDCE qui semble

être le registre de la bascule T.

— On voit qu'on a bien 3 **IBUF** pour les 3 signaux d'entrées **clk**, **restart** et **resetn**.

— On voit qu'on a bien 1 signal de sortie pour le signal **end_counter**.

```
+------+-------+------+
|      |Cell   |Count |
+------+-------+------+
|1     |BUFG   |     1|
|2     |CARRY4 |    11|
|3     |LUT2   |    28|
|4     |LUT3   |    29|
|5     |FDCE   |    29|
|6     |IBUF   |     3|
|7     |OBUF   |     1|
+------+-------+------+
```

Le résultat observé est proche du résultat attendu par rapport au schéma de la question 9

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

16 Question 16 : Lancez l'implémentation puis étudiez le rapport de timing.

Design Timing Summary

WNS(ns)	TNS(ns)	TNS Failing Endpoints	TNS Total Endpoints	WHS(ns)	THS(ns)	THS Failing Endpoints	THS Total Endpoints	WPWS(ns)	TPWS(ns)
4.179	0.000	0	29	0.263	0.000	0	29	4.500	0.000

Figure 6 – Design Timing Summary

— On remarque qu'il n'y a pas de violation de setup -> TNS = 0
— On remarque qu'il n'y a pas de violation de hold -> THS = 0

Chemin critique

Dans la section **Max delays**, vivado indique que le chemin critique démarre à partir du registre FDCE pin C et termine au pin D de ce même registre. On peut donc en conclure que le chemin critique est le chemin qui part de l'entrée du multiplexeur et qui passe par le registre 28 bits du compteur, le comparateur et qui se termine au niveau de la bascule T (Cf. schéma ci-dessous).

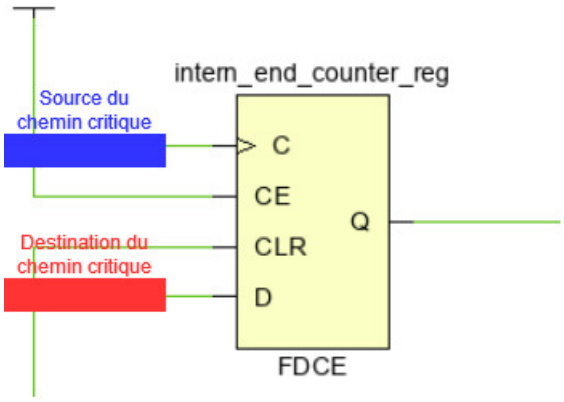

Figure 7 – Chemin critique décrit par vivado

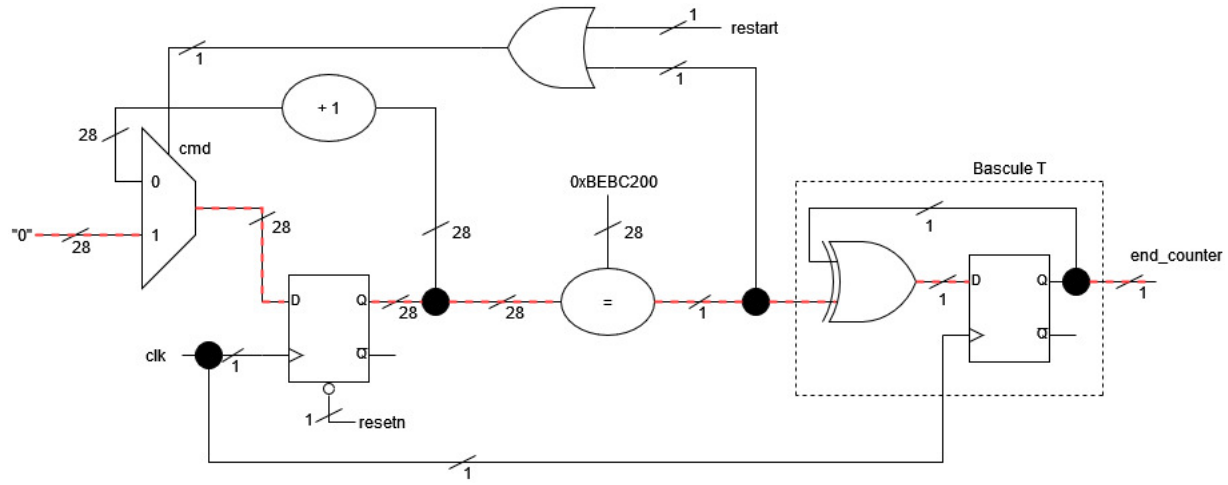

Figure 8 – Schéma RTL + chemin critique

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

On peut observer sur les captures ci-dessous que le signal **end_counter** change de valeur quand le compteur arrive à 199 999 999, et garde l'état jusqu'à ce que cette condition soit respectée à nouveau.

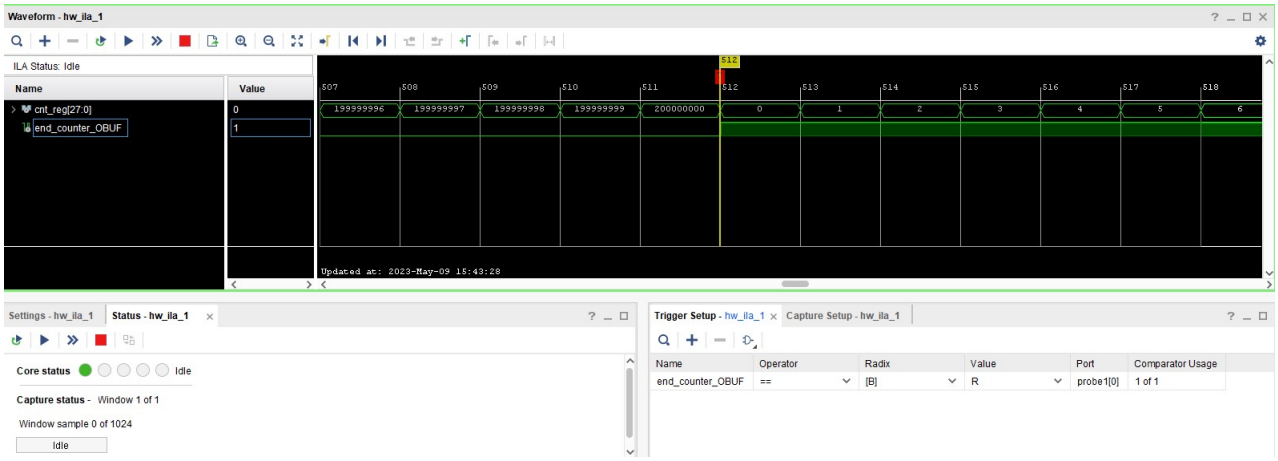

Figure 9 – Chronogramme ILA front montant

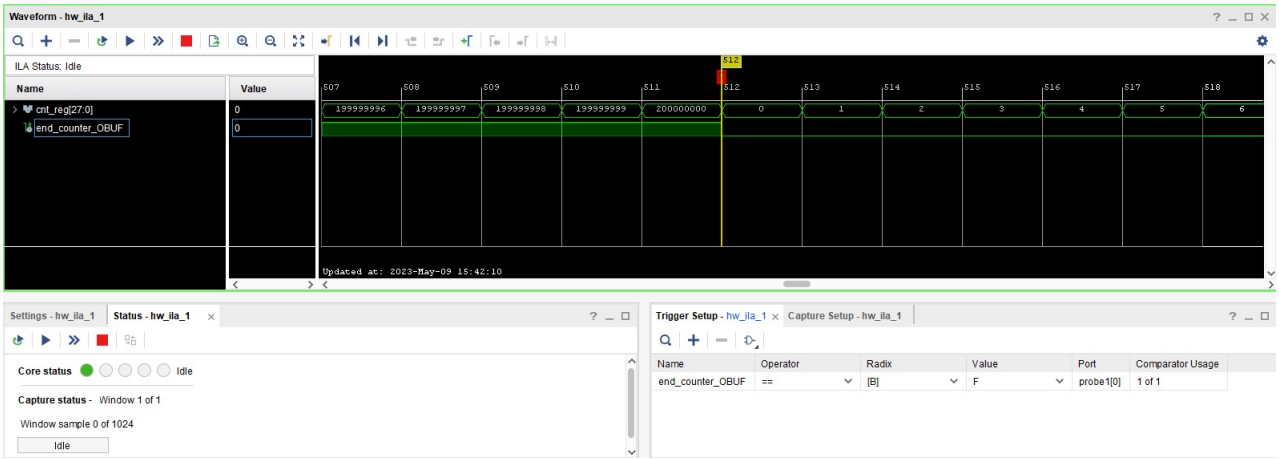

Figure 10 – Chronogramme ILA front descendant

On peut également voir que le compteur **cnt** et **end_counter** sont remis à 0 quand le bouton **restart** est pressé.

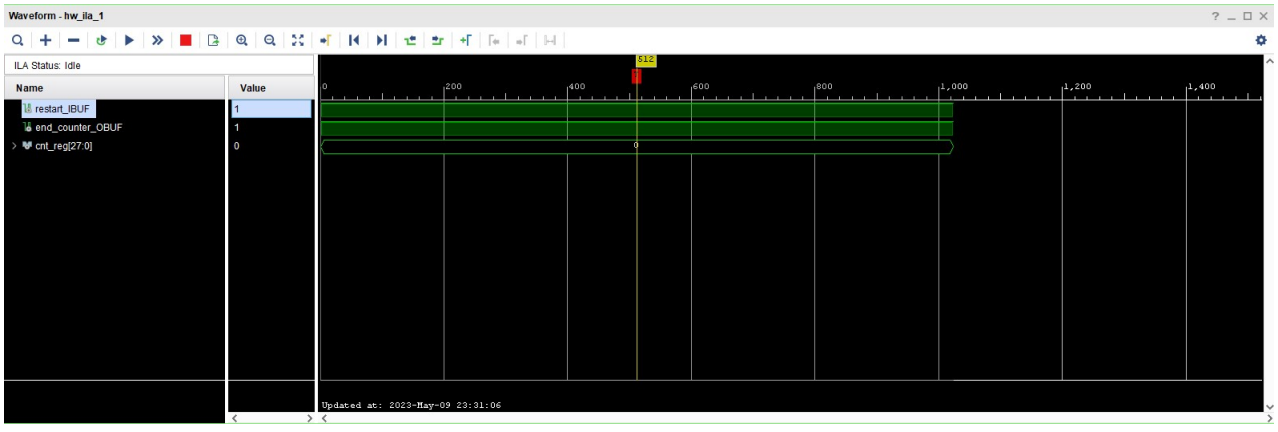

Figure 11 – Chronogramme ILA bouton restart