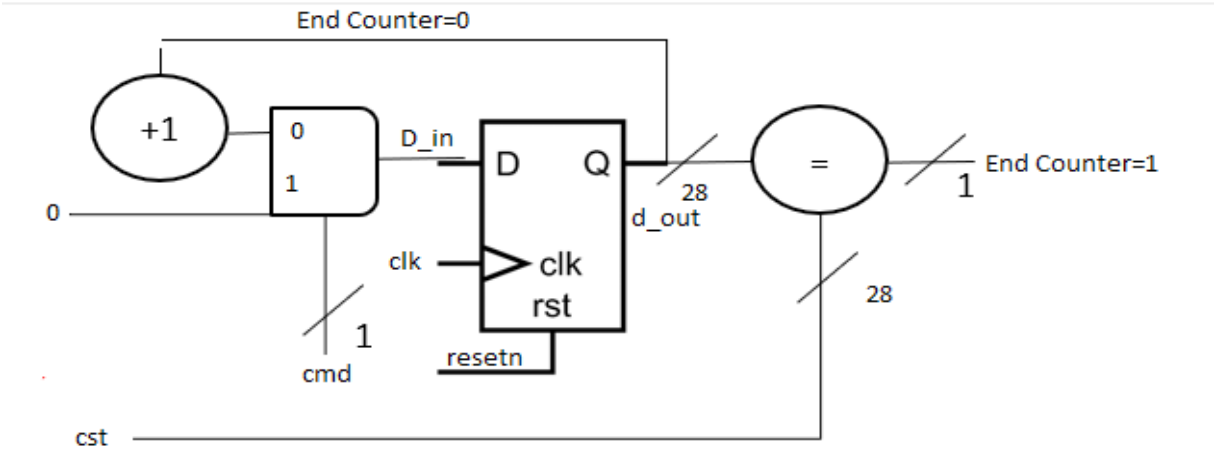
Atmani Hicham TP2

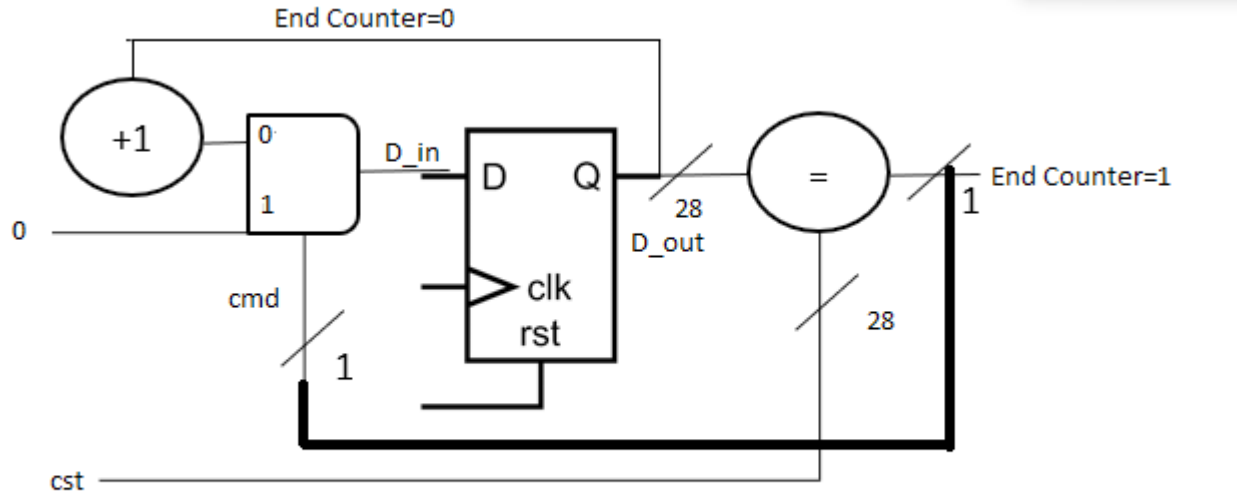
1. L'horloge du système est fixée à 100MHz. Combien de période faut-il compter pour attendre 2 secondes ? Combien de bits faut-il au minimum pour représenter cette valeur ?

On sait que l'on a 100 millions d'impulsions par seconde, il suffit de faire :
2 secondes x 100 millions d'impulsions par seconde = 200 millions d'impulsions pour attendre 2 secondes. Il faut 200 millions d'impulsions pour attendre les deux secondes.
Pour le nombre de bit, il faut : Nbr de bit = Log2(200x10^6)=28 Bits. Il nous faut donc au minimum 28 Bits.

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.



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.

Entrées : clk, Resetn

Sorties : End_Counter

Signaux interne : Cst, D_out, Cmd

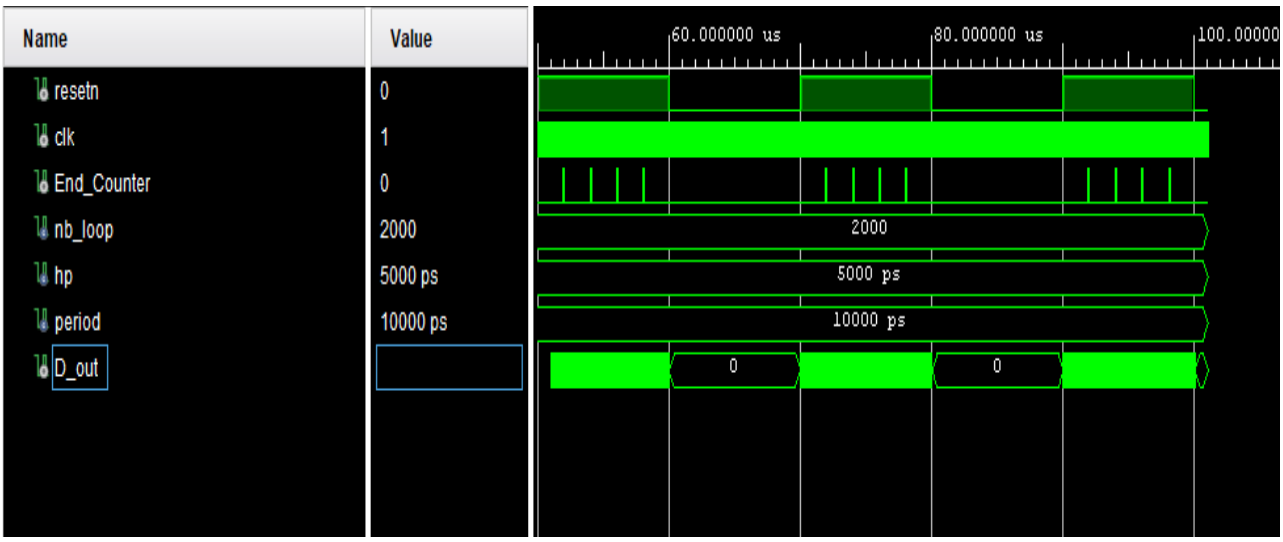
5. Ecrivez à présent le compteur en VHDL en suivant le schéma RTL, faites attention de bien faire correspondre les noms des signaux de votre code VHDL avec ceux de votre schéma RTL.

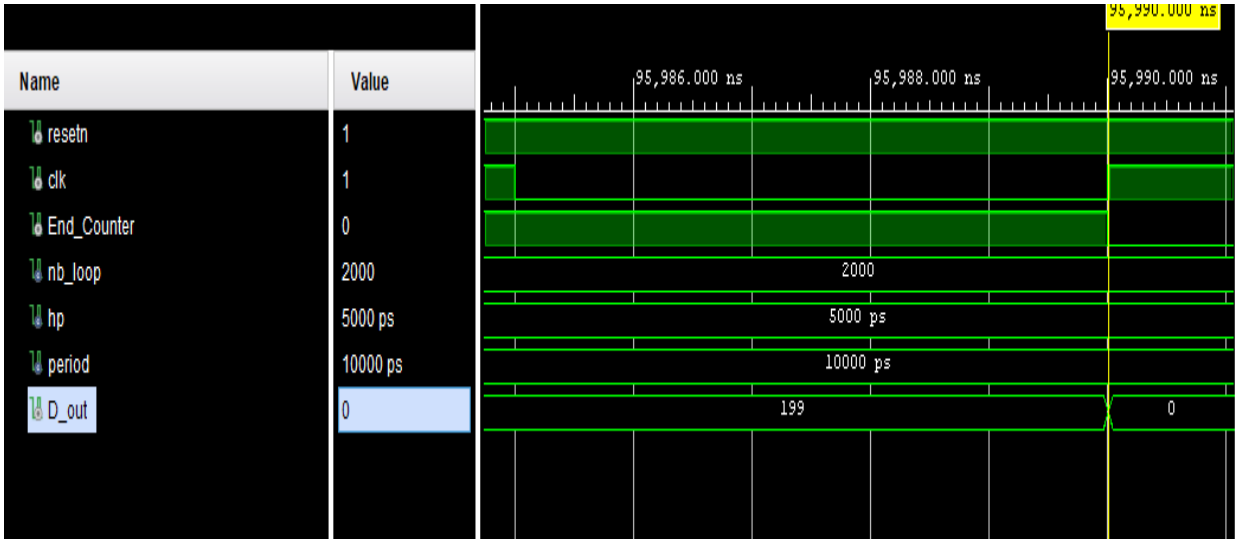
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.NUMERIC_STD.ALL;

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 Cst : positive := 200; --200000000 --s_TimeOut_2s
    signal D_out : positive := 0; --s_Counter_2s  --s_Counter_2s
    Signal Cmd   : std_logic := '0';

    begin

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

    --Partie combinatoire


    Cmd <= '1'  when (D_out = Cst-1 )
           else '0';
    End_Counter <= Cmd;
end behavioral;
```

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

```vhdl
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';

    constant nb_loop : integer := 2000;

    -- 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 100MHz

    --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

       -- TESTS A EFFECTUER

       -- Compteur jusqu'Ã  2 seconde avec Reset active
       resetn <= '0';
       for i in 1 to nb_loop loop
            clk <= not clk;
            wait for hp;
            if (nb_loop = 400) then
                assert End_Counter = '0'
                report "End_Counter : test failed, End_coder=0" severity failure;
            end if;
        end loop;

       -- Compteur jusqu'Ã  2 seconde avec Reset non active
       resetn <= '1';


       for j in 1 to nb_loop loop
            clk <= not clk;
            wait for hp;
                assert End_Counter = '1'
                report "End_Counter : test failed, End_coder=1" severity error;
        end loop;



    end process;
end behavioral;
```

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



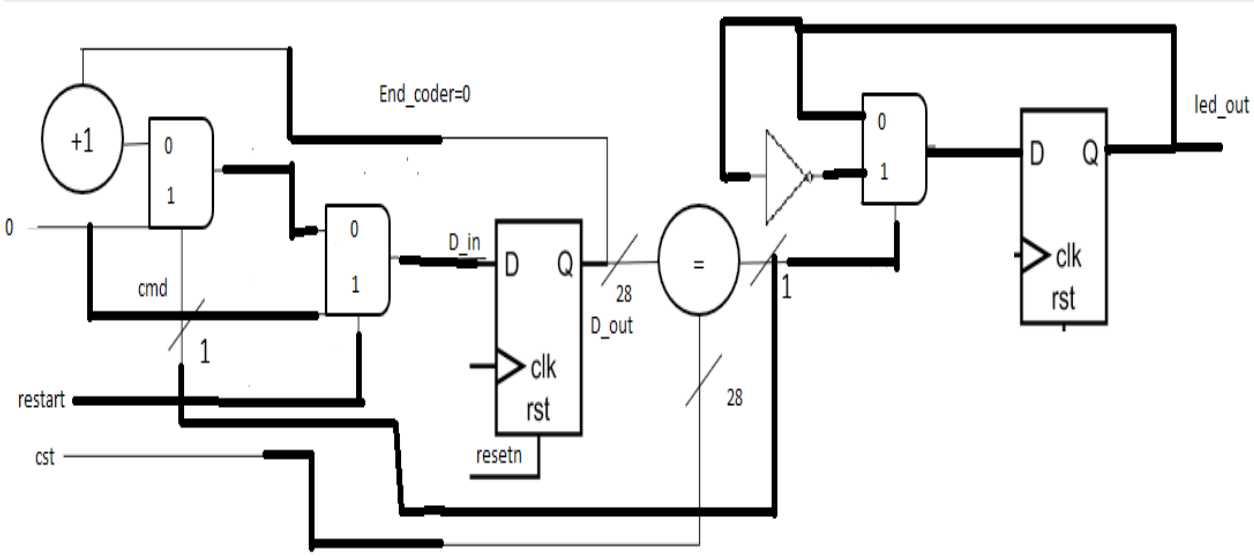


On peut voir que lorsque Resetn=0, on a bien End_counter à 0 et D_out à 0 donc le reset fonctionne correctement. Ensuite lorsque Resetn=1, on a notre D_Out qui compte, lorsqu'on, arrive à 200 cycles d'horloge (Réglage pour la simulation) on a bien End_counter qui passe à 1 et notre d_out se réinitialise. On observe bien le comportement que l'on recherche depuis le début.

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.**

```
## This file is a general .xdc for the Cora Z7-07S Rev. B
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project

# 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 }]; #IO_L22N_T3_AD7N_35 Sch=led0_b
#set_property -dict { PACKAGE_PIN G17   IOSTANDARD LVCMOS33 } [get_ports { led0_g }]; #IO_L16P_T2_35 Sch=led0_g
#set_property -dict { PACKAGE_PIN N15   IOSTANDARD LVCMOS33 } [get_ports { led0_r }]; #IO_L21P_T3_DQS_AD14P_35 Sch=led0_r
#set_property -dict { PACKAGE_PIN G14   IOSTANDARD LVCMOS33 } [get_ports { led1_b }]; #IO_0_35 Sch=led1_b
#set_property -dict { PACKAGE_PIN L14   IOSTANDARD LVCMOS33 } [get_ports { led1_g }]; #IO_L22P_T3_AD7P_35 Sch=led1_g
#set_property -dict { PACKAGE_PIN M15   IOSTANDARD LVCMOS33 } [get_ports { led1_r }]; #IO_L23N_T3_35 Sch=led1_r

```

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.**



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

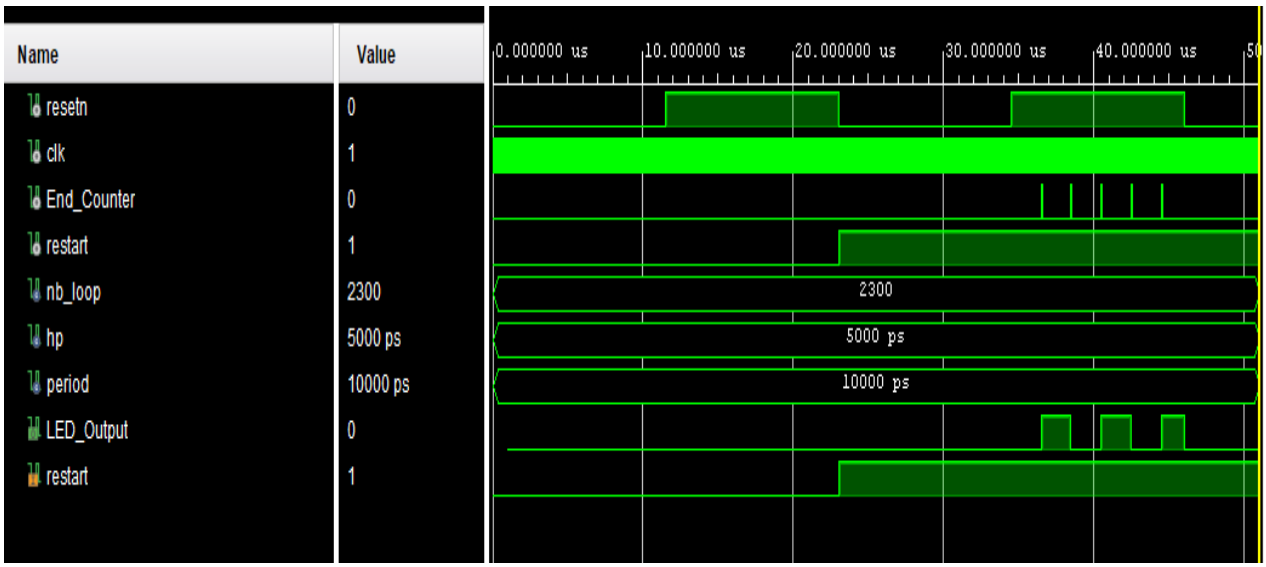
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.NUMERIC_STD.ALL;

entity counter_unit is
    port (
        clk          : in std_logic;
        resetn       : in std_logic;
        restart      : in std_logic;
        End_Counter  : out std_logic;
        LED_Output   : out std_logic
    );
end counter_unit;

architecture behavioral of counter_unit is

    --Declaration des signaux internes
    constant Cst : positive := 200; --200000000 --s_TimeOut_2s
    signal D_out : positive range 0 to Cst ; --s_Counter_2s  --s_Counter_2s
    Signal Cmd   : std_logic := '0';
    Signal s_LED_Output : std_logic := '0';

    begin

        --Partie sequentielle
        process(clk,resetn, restart)
        begin
            if(resetn = '0') then
                D_out <= 0;
                s_LED_Output <= '0';
            elsif(rising_edge(clk)) then
                if (restart = '1') then
                    D_out <= 0;
                    s_LED_Output <= '0';
                else
                    D_out <= D_out + 1 ;
                    if (Cmd = '1') then
                        D_out <= 0;
                        s_LED_Output <= not(s_LED_Output);
                    end if;
                end if;
            end if;
    end process;

    --Partie combinatoire
    Cmd <= '1'  when (D_out = Cst-1 )
           else '0';
    End_Counter <= Cmd;

    LED_Output <= s_LED_Output;

end behavioral;
```

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

```
# Buttons
set_property -dict { PACKAGE_PIN D20   IOSTANDARD LVCMOS33 } [get_ports { restart }]; #IO_L4N_T0_35 Sch=btn[0]
#set_property -dict { PACKAGE_PIN D19   IOSTANDARD LVCMOS33 } [get_ports { btn[1] }]; #IO_L4P_T0_35 Sch=btn[1]
```

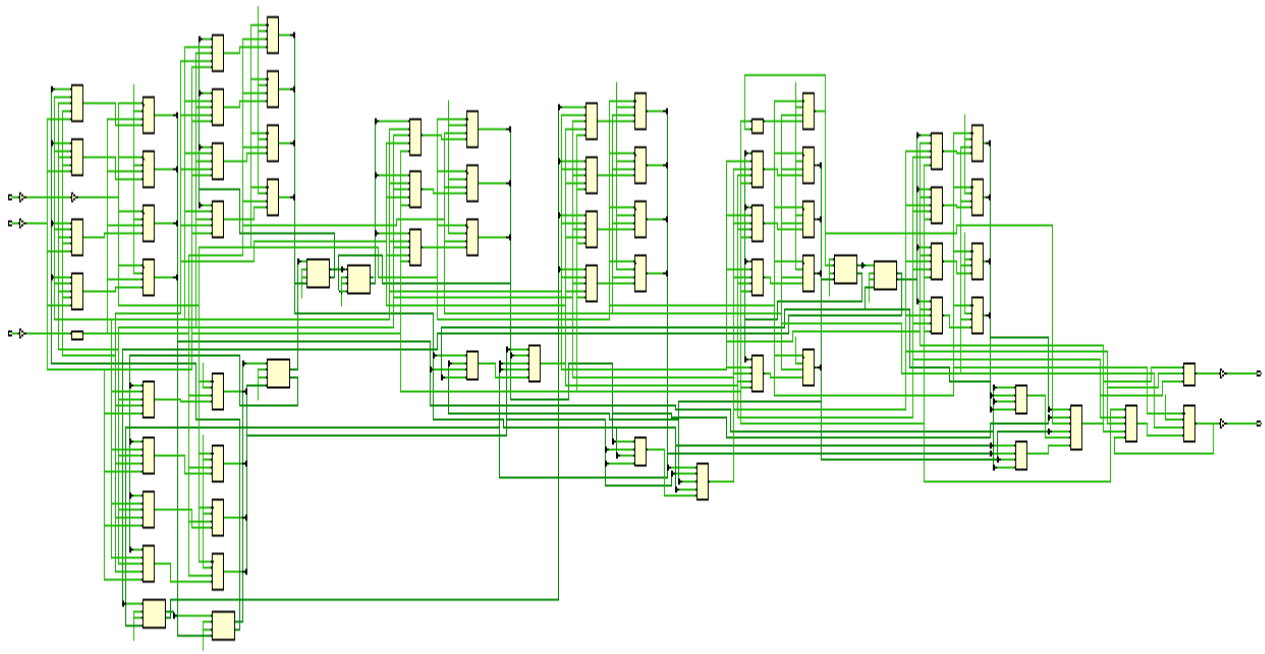
12. Mettez à jour votre testbench puis vérifier votre design avec une simulation. Quels sont les signaux que vous devez observer ?

```
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';
    signal restart     : std_logic := '0';

    constant nb_loop : integer := 2300;

    -- 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 100MHz

    --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

      -- TESTS A EFFECTUER

      -- Compteur jusqu'Ã  2 seconde avec Reset active

      for k in 1 to 4 loop
       if (k>=2) then
       restart<='1';
       end if;
       resetn <= '0';
       for i in 1 to nb_loop loop
            clk <= not clk;
            wait for hp;
            if (nb_loop = 400) then
                assert End_Counter = '0'
                report "End_Counter : test failed, End_coder=0" severity failure;
            end if;
        end loop;

      -- Compteur jusqu'Ã  2 seconde avec Reset non active
      resetn <= '1';


      for j in 1 to nb_loop loop
            clk <= not clk;
            wait for hp;
                assert End_Counter = '1'
                report "End_Counter : test failed, End_coder=1" severity error;
       end loop;
      end loop;


    end process;
end behavioral;
```



Les signaux qu'il est important de surveiller sont : le D_Out, Led_Output, le End_counter et la clk.

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



Sur le schéma, on peut voir que nos entrées/sorties correspondent bien à ce que l'on a préparé dans l'architecture.On peut voir que l'on a un registre par bit donc 28 registres pour 28 bits. Lorsque l'on clique sur un registre on peut voir où il est sur notre schéma. On peut voir les entrées à gauche, les sorties à droites. On peut également voir la présence de lut pour la partie combinatoire.

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

```
Detailed RTL Component Info :
+---Adders :
       2 Input   28 Bit       Adders := 1
+---Registers :
                 28 Bit    Registers := 1
                  1 Bit    Registers := 1
+---Muxes :
       2 Input   28 Bit        Muxes := 2
       2 Input    1 Bit        Muxes := 1
```

On peut voir que l'on a bien un registre par bit soit 28 registres comme dans l'architecture RTL. On a bien 2 Muxes avec 2 entrées et le un adder avec deux entrées. Le Adder permet de faire la comparaison. On a bien la même chose que sur le RTL.

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

J'ai placé mes sondes sur ma clk, sur end_counter, sur led_output et sur mon D_out.

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

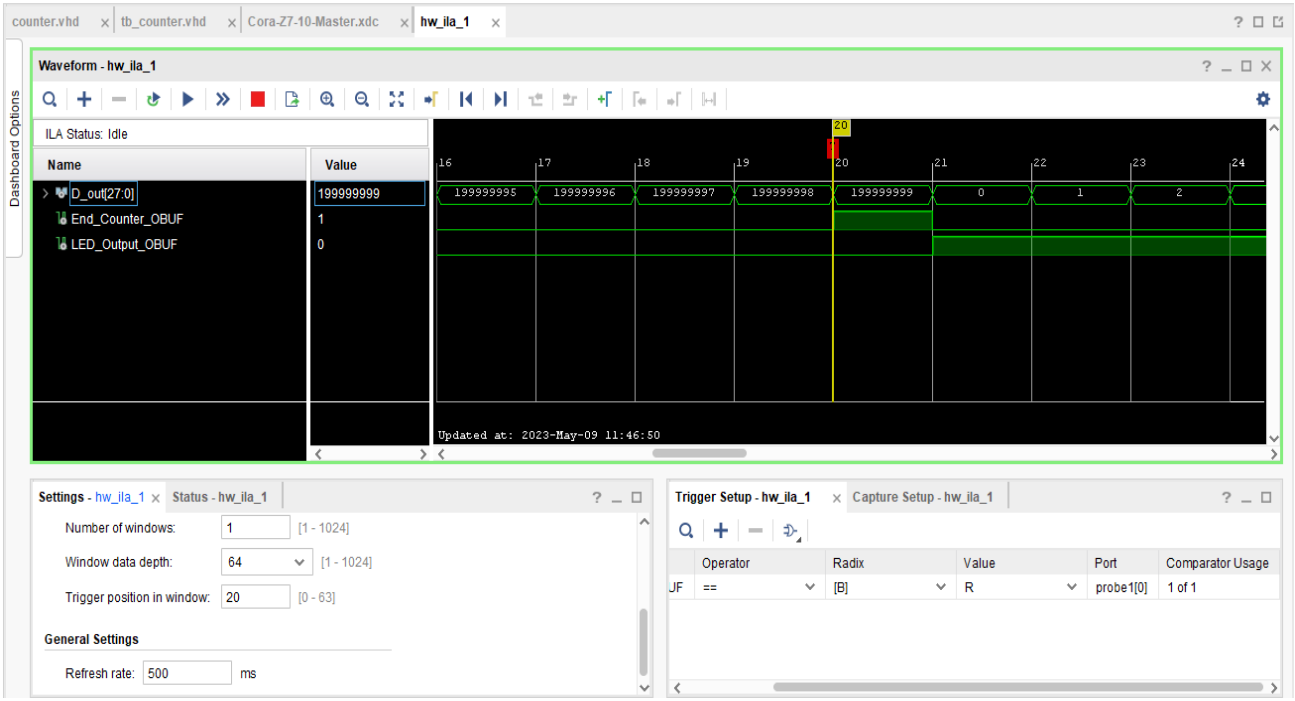
```
------------------------------------------------------------------------------------------------
| Design Timing Summary
| ---------------------
------------------------------------------------------------------------------------------------

    WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints      WHS(ns)      THS(ns)  T
    -------      -------  ---------------------  -------------------      -------      -------  -
      1.771        0.000                      0                 3835        0.014        0.000
```

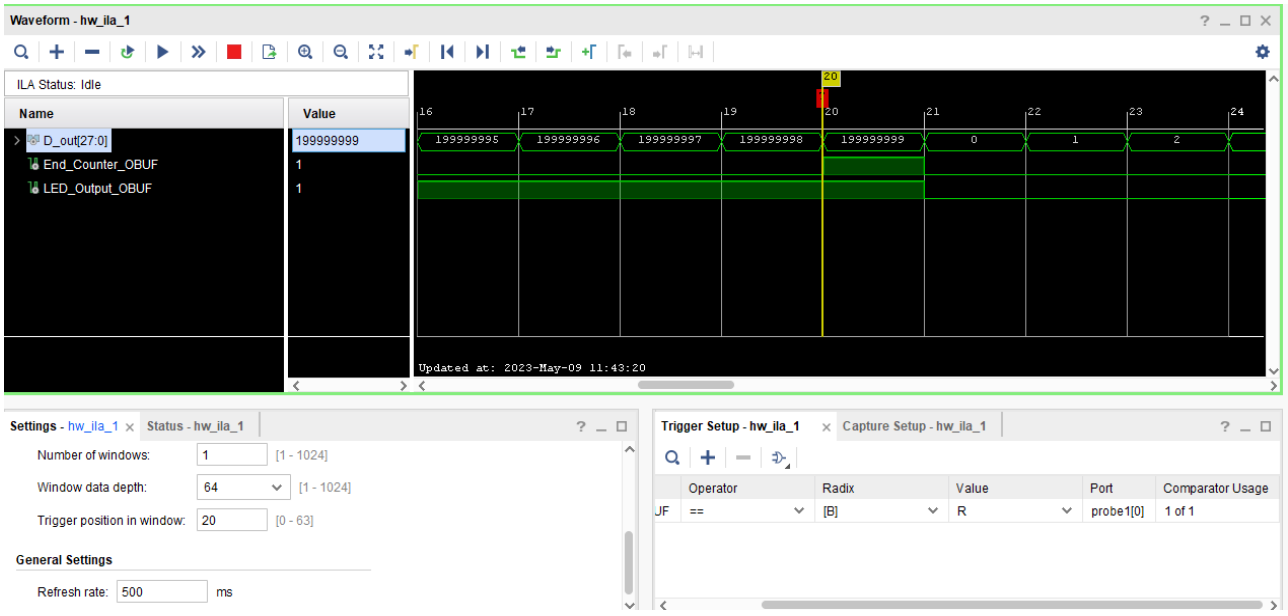
On peut voir que l'on n'a aucune violation (pas de slack) donc il n'y aura pas de métastabilité.

Le chemin critique :

```
Slack (MET) :          26.174ns  (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  {rise@0
  Destination:         dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_switch/state_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  {rise@0
```

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





On peut voir qu'à chaque front de end counter notre led change d'état, de plus on peut voir que notre compteur compte bien jusqu'à 200000000 donc qu'on attend bien les 2 secondes.