Academic year 2020-2021

REPORT – TP FPGA

M1 EEA/TSI
UFER Sciences et Techniques
Département IEM
Academic year

2020 – 2021

Realized by :

Tshala Tshibumbu Benjamin

TP instructor :

Neguissi Solomon

Table des matières

I. Tutorial, first exemple : new project in ISE

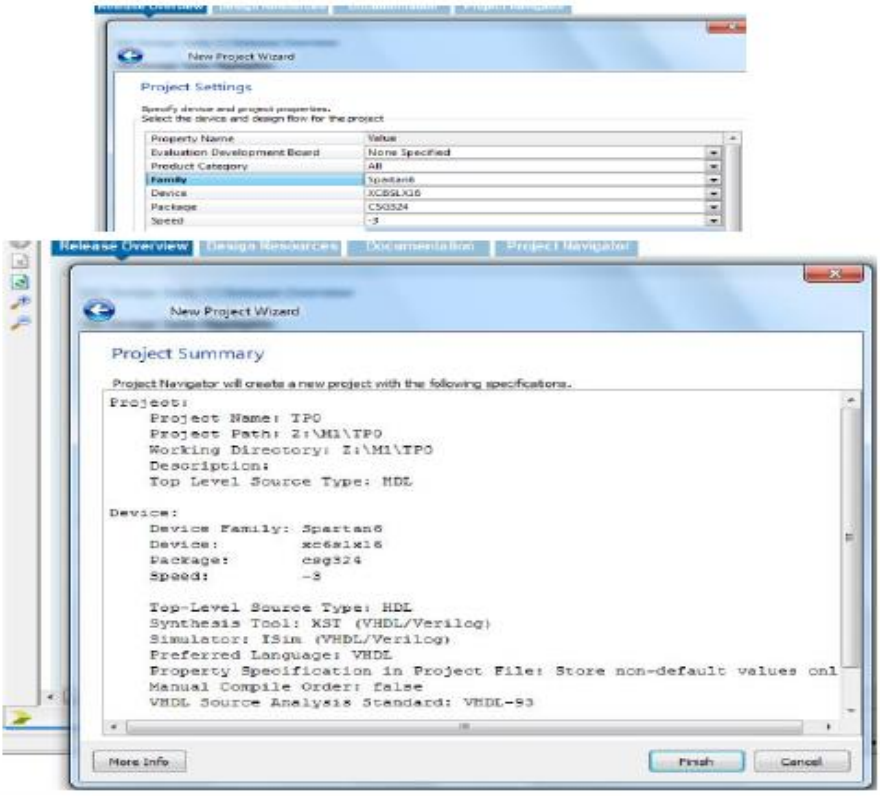
This tutorial allowed us to carry out all the requests of Tp either at the level of creation of a .vhd file, simulation.

Goal :

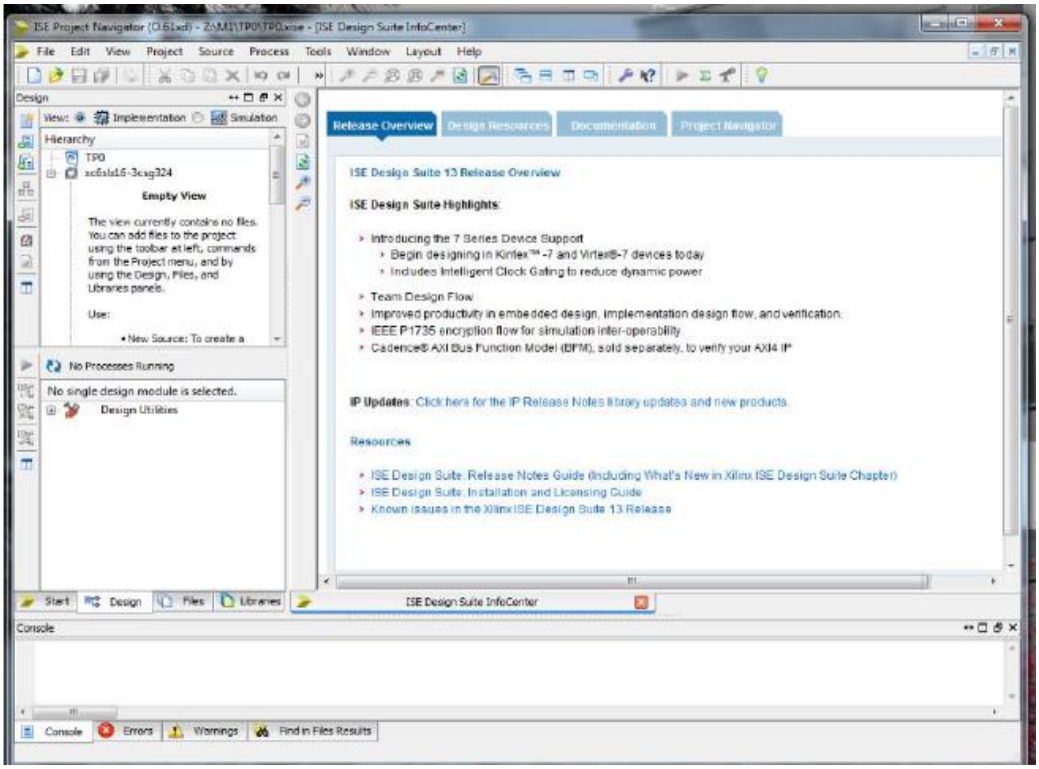
This first part will allow us to create a new ISE project that targets the Spartan 6 FPGA in the Nexys 3 board.

1. To launch Xilinx ISE, double-click on the Xilinx ISE icon in the desktop.
2. In the ISE project browser, click on File> New project.
3. In the New Project Wizard window, type TP0 as the project name. Note that TP0 is added to the initial path contained in the location field.
4. Let's select HDL for the Top-Level Source Type field and click Next.
5. Enter the following values in the New Project Wizard - Project Sittings window. at.
    a. Product Category: All
    b. Family: Spartan6
    c. Device: XC6SLX16
    d. Package: CSG324
    e. Speed: -3
    f. Synthesis Tool: XST (VHDL / Verilog)
    g. Simulator: Isim (VHDL / Verilog)
    h. Preferred Language: VHDL
6. Finally, click on Next.
7. Finally, we click on Finish in the Project summary window.

At this point, we get a new ISE project that does not yet contain any HDL files.



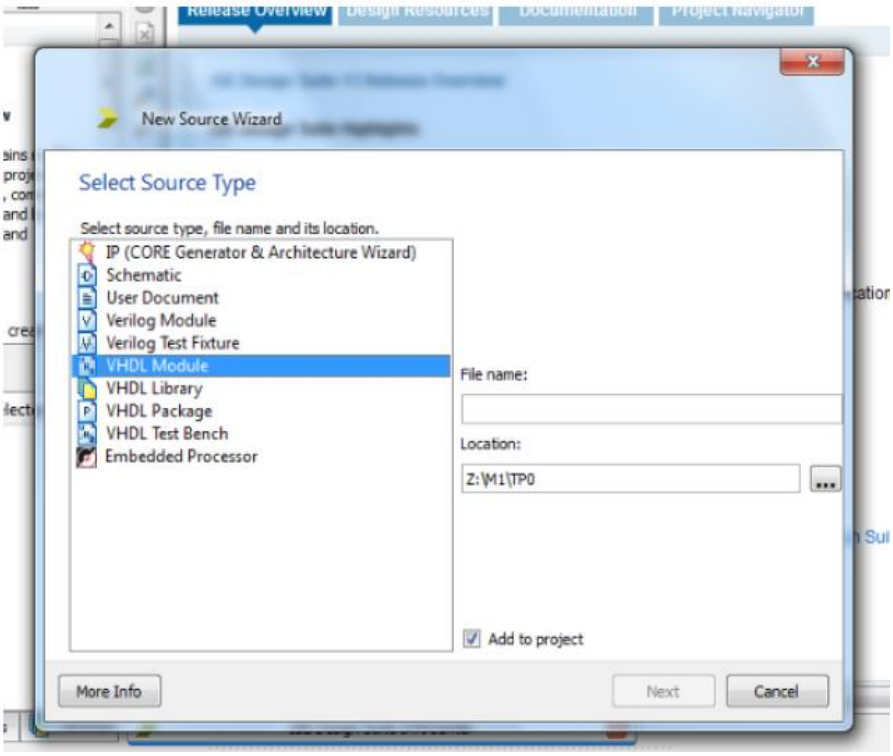
With this image above, we can realize the summary of the creation of our new project by integrating all the characteristics that we have chosen.



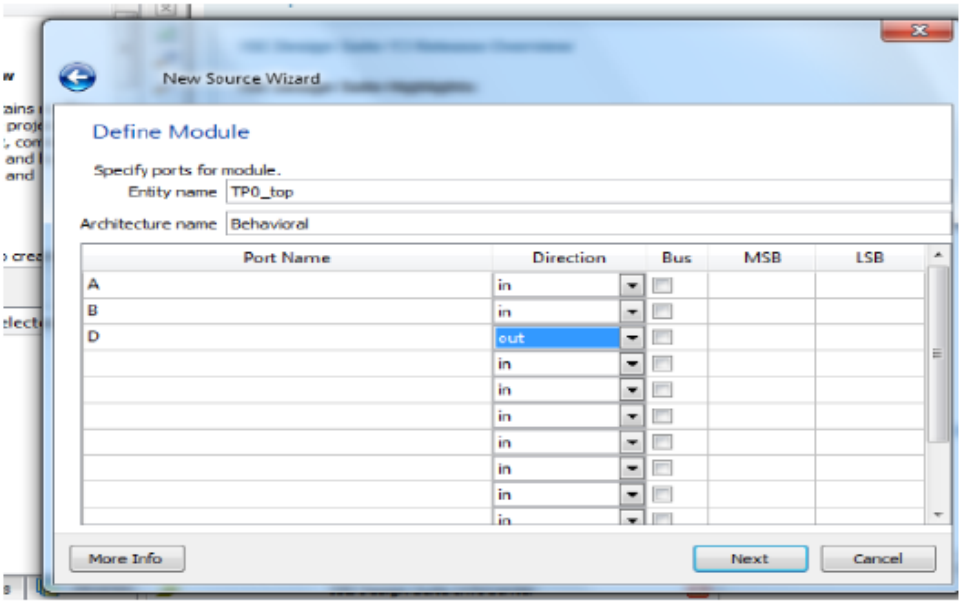
On the right, we have implementation and simulation as a selection mode for the use of our TP0.

Creation of a top level HDL

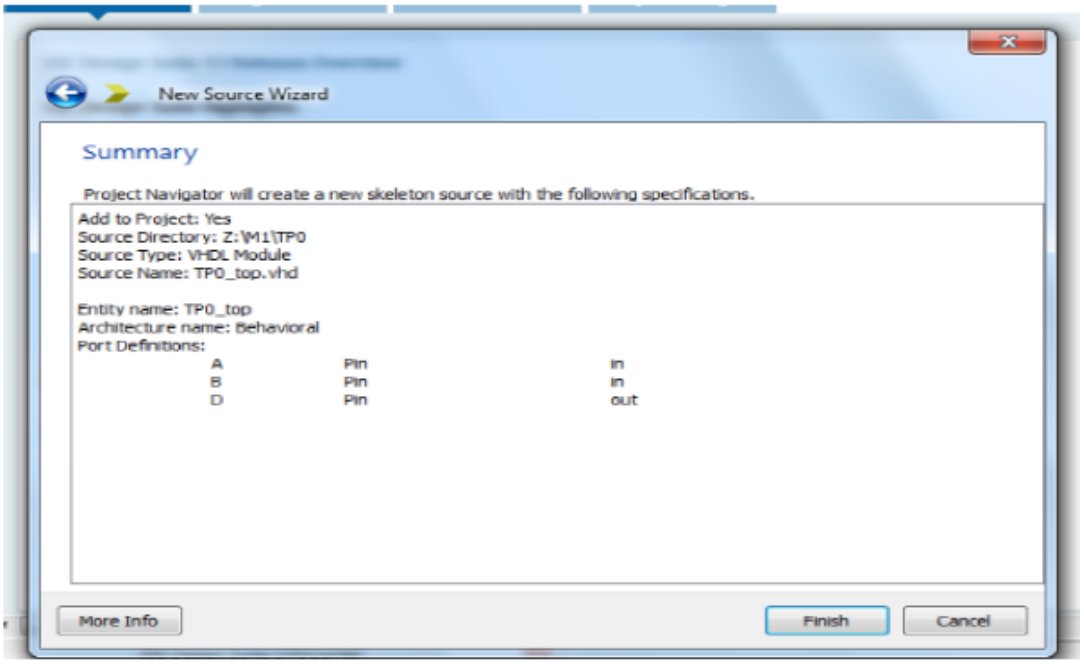
- Let's click on Project> NEW Source.
- Select VHDL module in the Project Wizard - Select Source Type window.
- Let's type TP0_top as the file name and make sure the Add to project box is checked. Let's click on Next.
- Let's create an input port A, B and an output D as shown in the figure
    - o Enter A, B, D in the first 3 Port Name fields.
    - o In the Direction field, choose in A, B and out for D
    - o Do not pig the Bus fields.
- Click on Next and then on Finish to complete this configuration step
- A dialog box displays the VHDL description generated by the Wizard. Let's click on Finish.



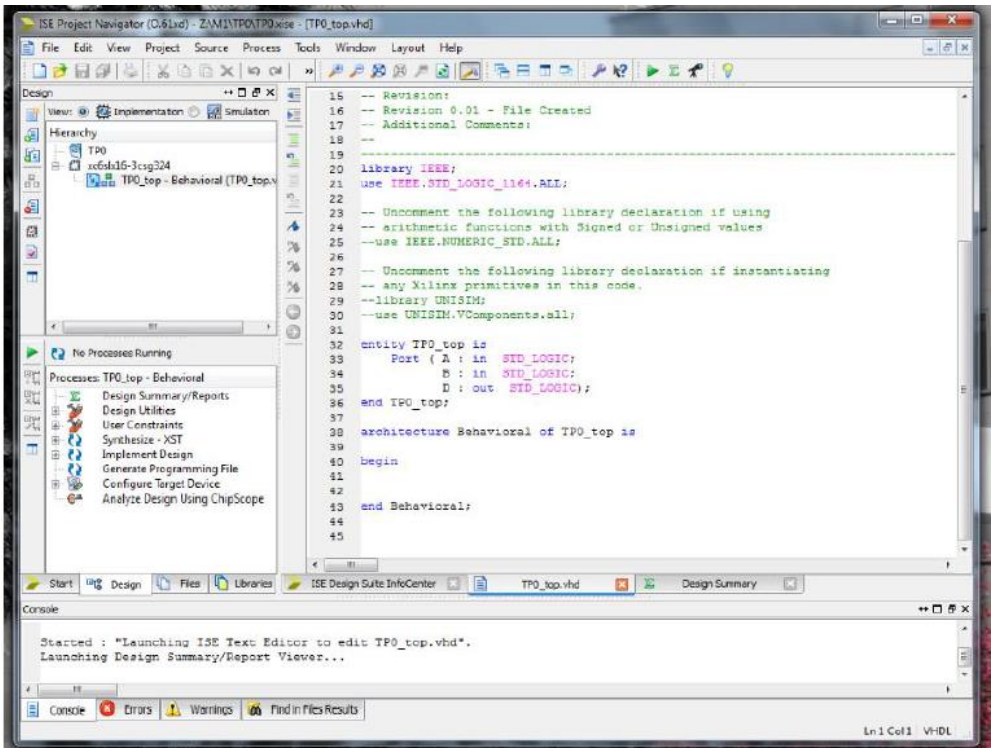
After selecting VHDL Module later, we will have this :



We have the input ports (In) denoted A and B and an output port (Out) denoted D.



The empty VHDL file describing the TP0_top module opens in the workspace as shown in the figure below. This file contains the entity of the module but has an empty architecture to complete.



We can see that our ports are present in the VHDL module that we have generated.

Let's complete the description of the VHDL of TP0_top, by performing the following steps:

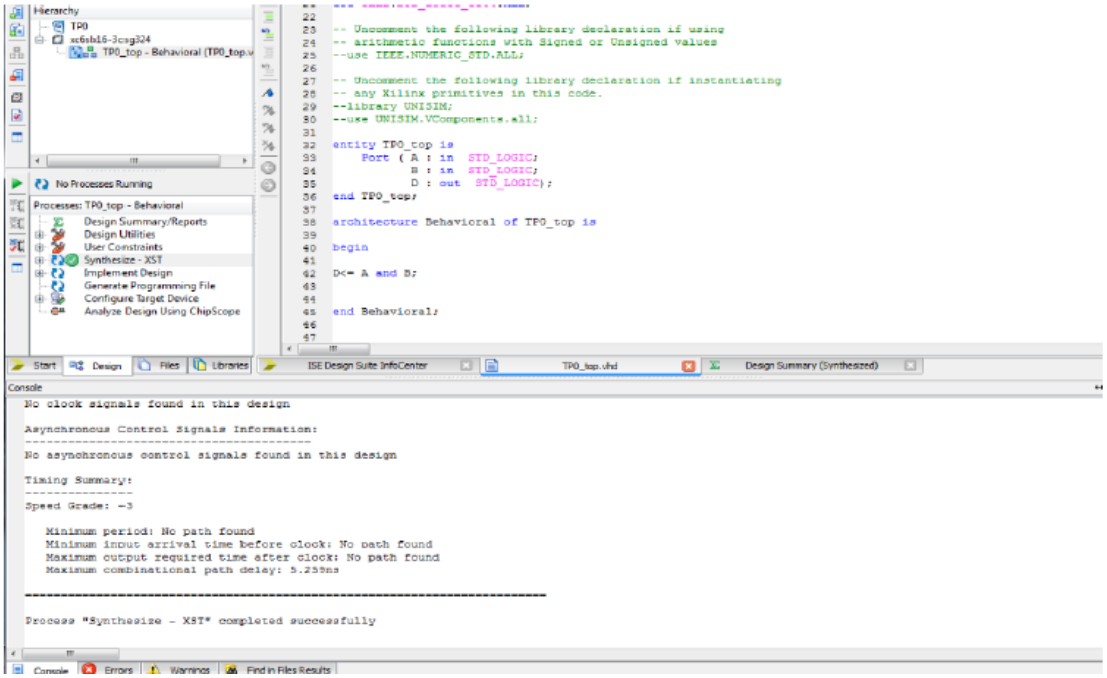
1. In the TP0_top file opened in the ISE Project Browser workspace, erase all lines of code below the line Bahavioral architecture of TP0_top is.

2. Suppose the circuit of TP0_top is similar to the following circuit

Which is an AND gate whose output will give us the product of the two input ports.

The behavior of the circuit will be written as follows :

D <= A and B ;



We can notice that our module is now complete, because we have inserted our AND function in the module.

We can observe the VHDL code with more precision and clarity below:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity TP0_top is
Port ( A : in STD_LOGIC;
B : in STD_LOGIC;
D : out STD_LOGIC);
end TP0_top;
architecture Behavioral of TP0_top is
begin
D<= A and B;
end Behavioral;
```

II. Functional simulation with Isim

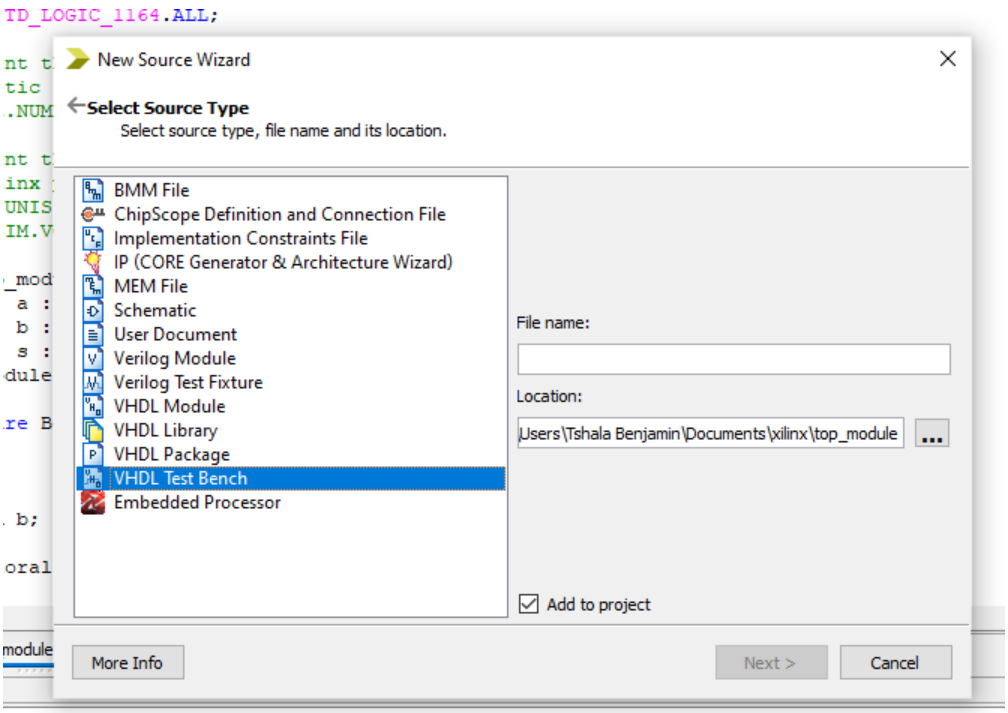
The purpose of this part is to set up a functional test of a VHDL program. To do this, it will be possible to visualize the simulation result with the ISIM tools (present in Xilinx's ISE). The test stimuli are created either using TCL command (directly in ISIM), or using "testbench" file. We will create for this lab a testbench file (in VHDL) and we will test the state machine code.

- Creation of a Testbench :

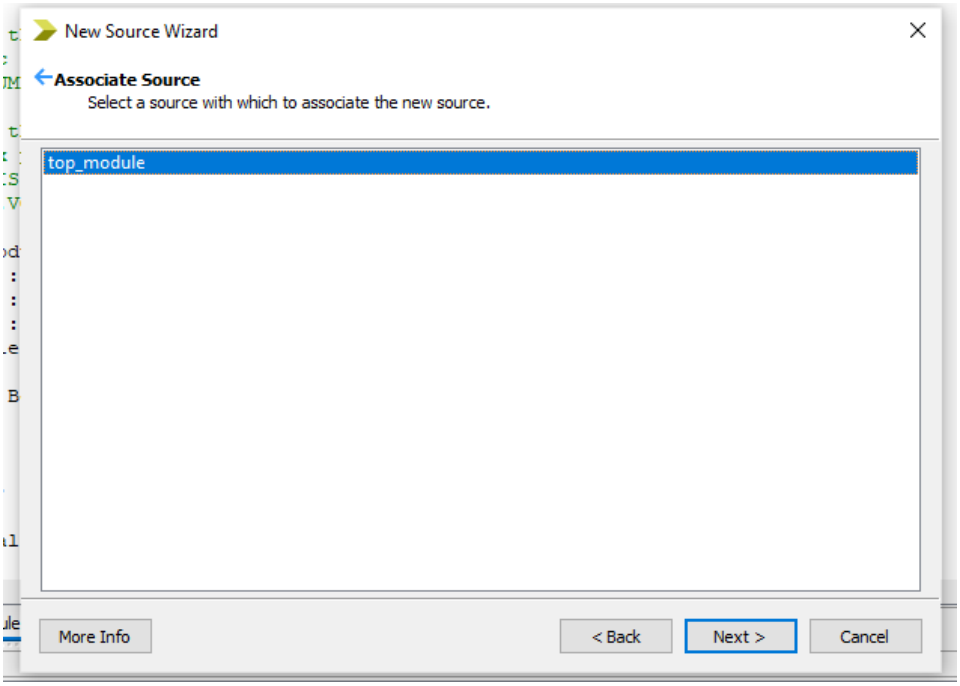
A testbench is a non-synthetic VHDL file used as a top-level framework. It allows you to instantiate VHDL modules to be tested, to connect them to each other and to assign test signal values to them. The testbench file can be simulated with an HDL simulation tool like Isim or Modelsim. In the next section, we will add a testbench to the project and simulate it with Isim to test the sequence detector.

In the previous lab let's add a VHDL testbench file to the project as follows:

1. Select Project> New Source.
2. Select VHDL Test Bench from the select source Type dialog box.
3. Let's enter the name of the test bench file and click next.



4. In the Associate Source dialog box, select the name of the main file to associate the testbench file to simulate.



5. Let's click on Next and then Finish to create our testbench file.

```
81     -- Stimulus process
82     stim_proc: process
83     begin
84        -- hold reset state for 100 ns.
85        wait for 100 ns;
86
87        wait for <clock>_period*10;
88
89        -- insert stimulus here
90
91        wait;
92     end process;
93
94  END;
95
```

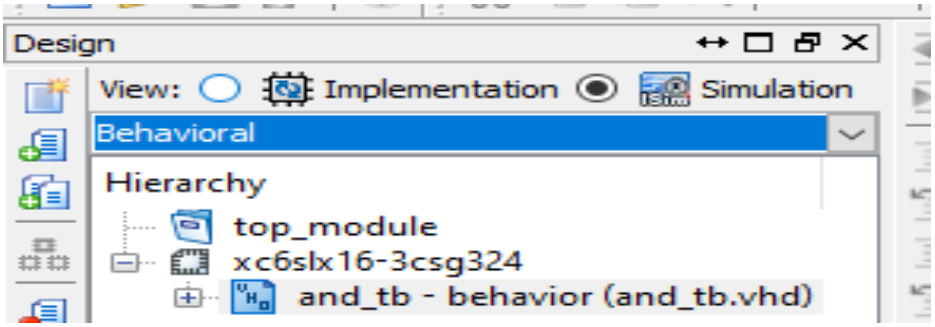
This file is mainly used to instantiate the top level machine_top module using the component and port map tags that we will see when we move through the file.

This file will also allow us to instantiate the clock periods first, and assign values to signals for different periods.

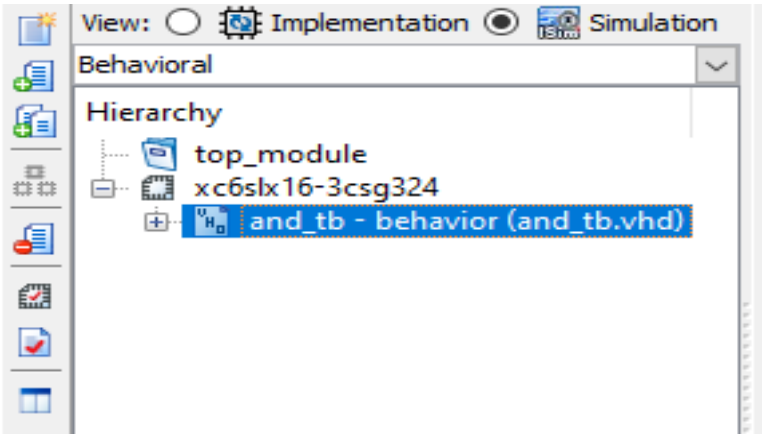
III. Project simulation with Isim

We will run the testbench simulation in Isim from the ISE Navigator. First, let's set up the simulation by performing the following steps:

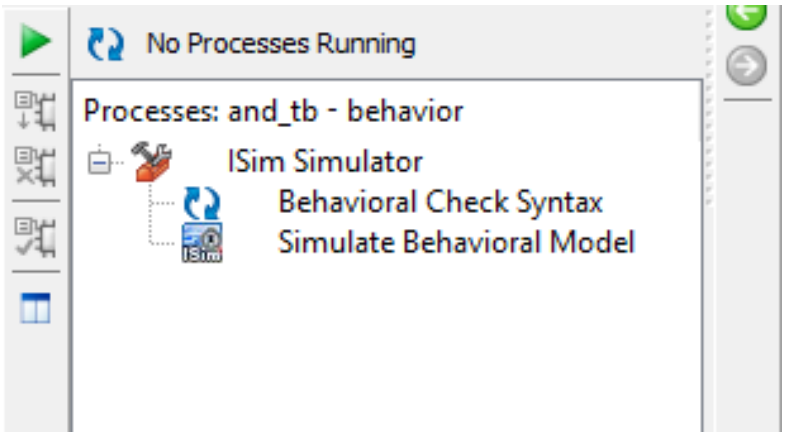
1. Select the Design tab in the Hierarchy area located at the top left of the Project Navigator.
2. Select the Behavioral item after selecting Simulation just above.



3. Select the file in the project tree.



4. Deploy the Isim Simulator item in the list of the Process space
5. Right-click on Simulate Behavioral Model.



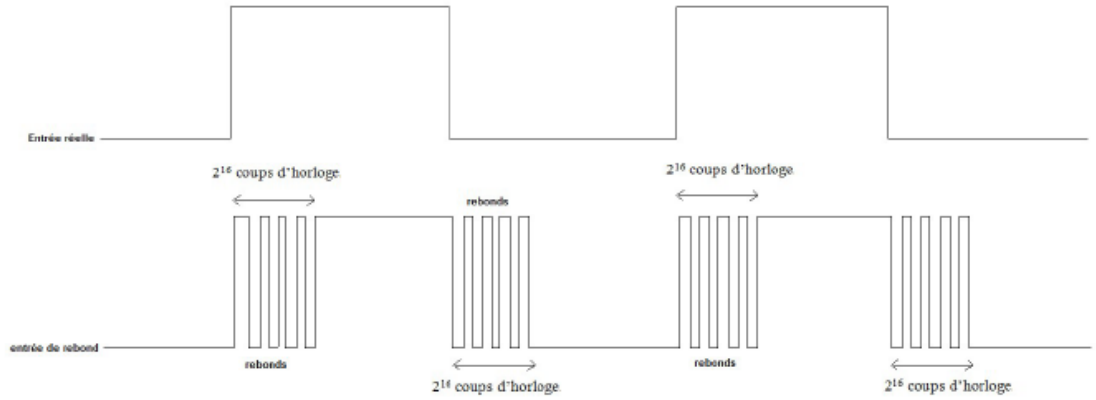
6. Let's start the simulation by double-clicking on Simulate Behavioral Model with the left mouse button.

IV. Button use

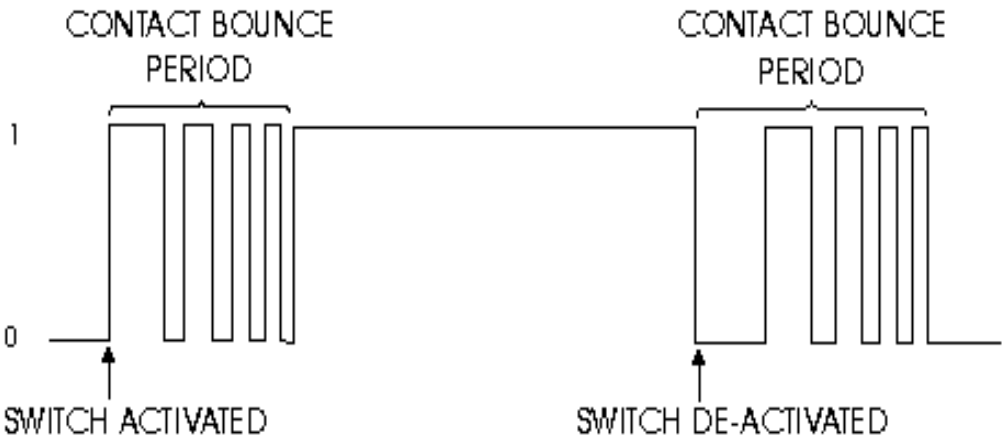
Introduction

There is a bouncing phenomenon when using mechanical buttons. Indeed when we press a button, the contact that occurs to generate "bounces" as shown in the image below. These bounces can be harmful if ever a counter is connected to a rising edge of the signal created by the button

bounce : it is a series of parasitic impulses when the push buttons are actuated.



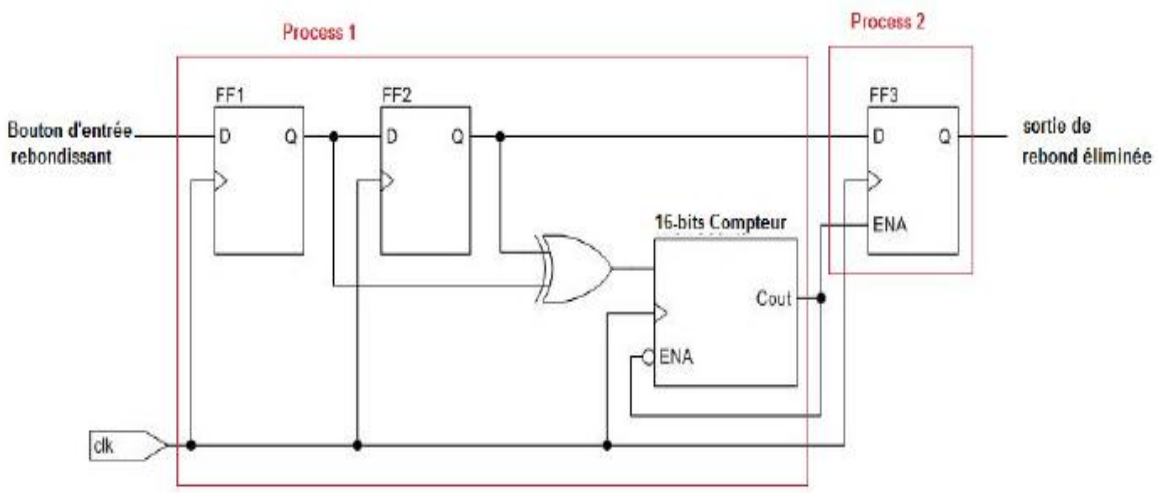
We can see more about the bouncing effect on the photo below:



Our goal :

The goal is to create a VHDL program with two levels of hierarchy. First, create a component that will eliminate bounces on 5 different buttons. This component will receive the states of the 5 buttons and the global clock at 100 MHz as input, and will output the states of the 5 buttons after filtering.

A stable state will be considered for a button if it does not change 2^{16} clock strokes :



We start by creating for each button, two processes:

a) The first process consists in incrementing a counter when the current value of the button is the same as that of the memorized button (if not the counter is reset to 0).

b) The second process changes the value of the memorized button if the previous counter has reached its maximum value.

Then, you have to validate the operation on the card by turning the LEDs on and off according to the button presses in real time.

VHDL code :

For the realization of the VHDL program, we made the VHDL code of each component described in the diagram above:

Latch :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
Use ieee.std_logic_unsigned.all;
Use ieee.std_logic_arith.all;

entity Bascule_D is
port(
            D: in STD_LOGIC;
            Q: out STD_LOGIC;
            clk: in STD_LOGIC;
            Raz: in STD_LOGIC);
end Bascule_D;

architecture Behavioral of Bascule_D is

begin

process(D,clk)
      begin
      if Raz = '1' then
            Q <= '0';
      elsif rising_edge(clk) then
      Q <= D;
      end if;
end process;

end Behavioral;
```

XOR GATE :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
Use ieee.std_logic_unsigned.all;
Use ieee.std_logic_arith.all;

entity XOR_GATE is
    Port ( A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           S : out  STD_LOGIC);
end XOR_GATE;

architecture Behavioral of XOR_GATE is

begin

S <= A xor B;

end Behavioral;
```

COUNTER 16bits :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
Use ieee.std_logic_unsigned.all;
```

Use ieee.std_logic_arith**.all;**

```
entity Counter is
    Port ( A : in  STD_LOGIC;
           ENA : in  STD_LOGIC;
           Cout : out  STD_LOGIC_VECTOR (15 downto 0);
           clk : in  STD_LOGIC);
end Counter;

architecture Behavioral of Counter is

signal s_init: std_logic_vector (15 downto 0):=(others =>'0');

begin

process(A,clk)
begin
      if ENA = '0' then
            if A = '1' then
            s_init <= (others => '0');
                  elsif rising_edge(clk) then
            s_init <= s_init + 1;
                  if s_init<="0000000000000001" then
                        Cout <= s_init;
                  end if;
            end if;
      end if;
end process;

end Behavioral;
```

Latch with enable :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
Use ieee.std_logic_unsigned.all;
Use ieee.std_logic_arith.all;

entity Latch_enable is
    Port ( D : in  STD_LOGIC;
           Q : out  STD_LOGIC;
                  Raz: in STD_LOGIC;
           clk : in  STD_LOGIC;
           ENA : in  STD_LOGIC);
end Latch_enable;

architecture Behavioral of Latch_enable is

begin
process(clk,Raz)
      begin
      if Raz = '1' then
       Q <= '0';
      elsif rising_edge(clk) then
       if ENA ='0' then
        Q <= '0';
```

```
        elsif ENA ='1' then
        Q <= '1';
        end if;
       end if;
end process;

end Behavioral;
```

Next, we have bundled all the VHDL codes into one package below:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
Use ieee.std_logic_unsigned.all;
Use ieee.std_logic_arith.all;
use work.ensemble.all;
entity finale is
      port ( a: in STD_LOGIC;
                  s: out STD_LOGIC;
                  clk: in STD_LOGIC);
end finale;

architecture Behavioral of finale is

signal s1,s2,s3,s5: std_logic :='0';
signal s4: STD_LOGIC_VECTOR (15 downto 0):=(others=>'0');
signal ena : std_logic :='1';
begin

inst_FF1: Bascule_D
port map(
D => a,
clk=> clk,
Raz=> '0',
Q=> s1 );

inst_FF2: Bascule_D
port map(
D=> s1,
clk=> clk,
Raz=> '0',
Q=>s2 );

inst_porte_logique: XOR_GATE
port map(
A=> s2,
B=> s1,
S=> s3 );

inst_compteur: counter
port map(
A=> s3,
ENA=> ena,
clk=> clk,
Cout=> s4 );

inst_process2: Latch_enable
port map (
D=> s2,
ENA=>ena, clk=> clk,
```

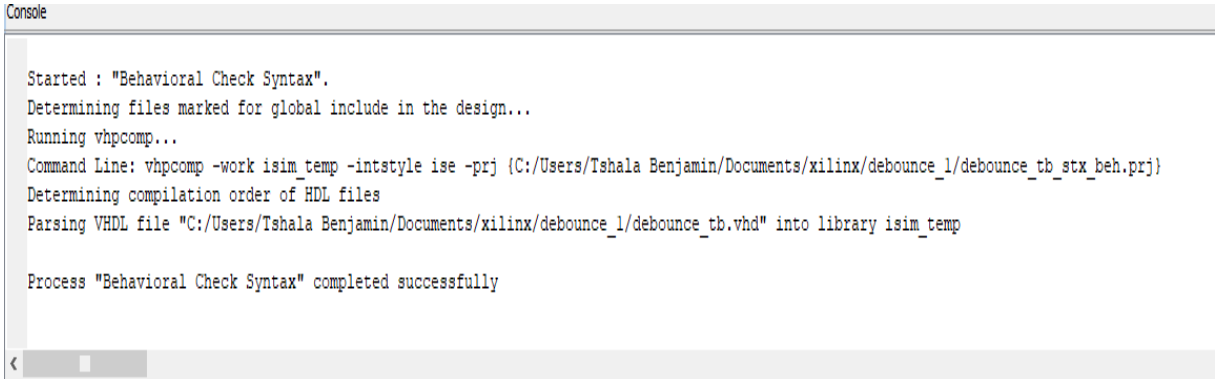
```
Raz=> '0',
Q=> s5 );

end Behavioral;
```

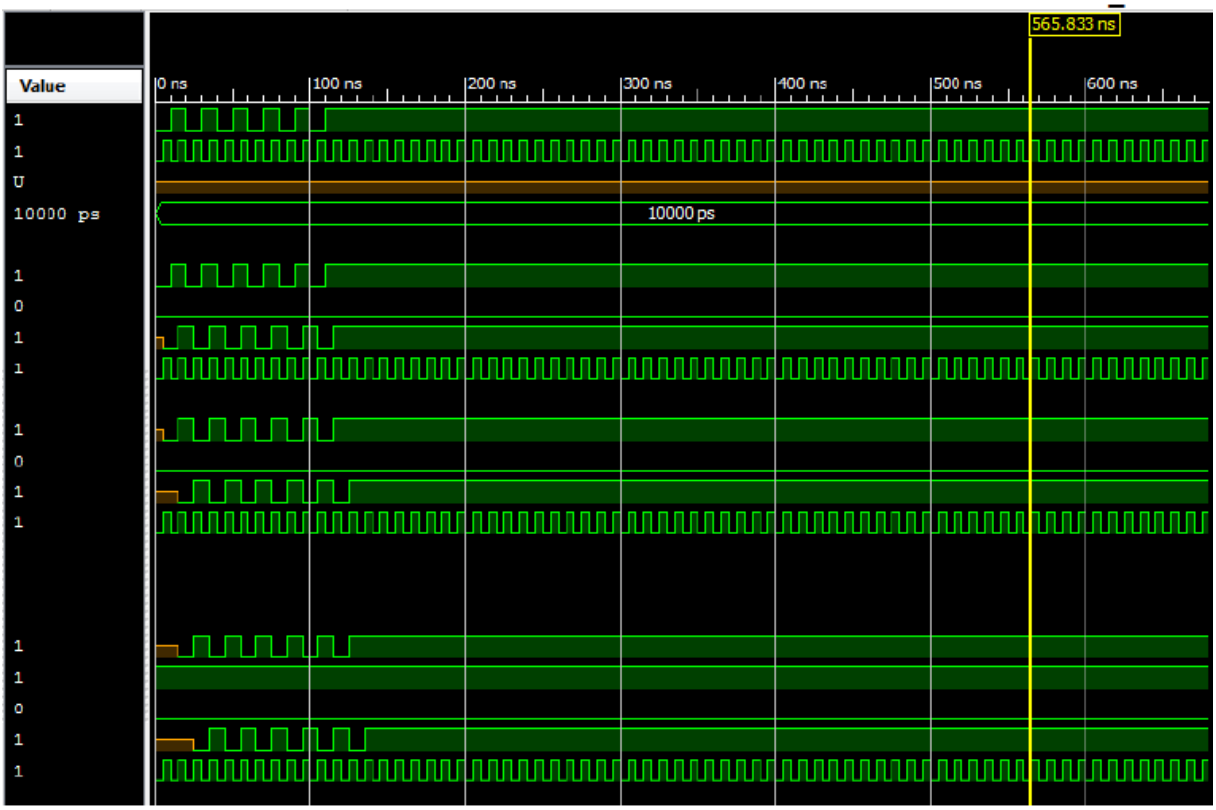
Test bench

```
USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY debounce_tb IS
END debounce_tb;

ARCHITECTURE behavior OF debounce_tb IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT debounce
    PORT(
         clk : IN  std_logic;
         button : IN  std_logic;
         result : OUT  std_logic
        );
    END COMPONENT;


   --Inputs
   signal clk : std_logic := '0';
   signal button : std_logic := '0';

      --Outputs
   signal result : std_logic;

   -- Clock period definitions
   constant clk_period : time := 10 ns;

BEGIN

      -- Instantiate the Unit Under Test (UUT)
   uut: debounce PORT MAP (
          clk => clk,
          button => button,
          result => result
        );

   -- Clock process definitions
   clk_process :process
   begin
            clk <= '0';
            wait for clk_period/2;
            clk <= '1';
            wait for clk_period/2;
   end process;
```

```
    stim_proc: process
    begin
       -- hold reset state for 100 ns.
       wait for clk_period;
                   button <= '1';
                   wait for clk_period;
                   button <= '0';
                    wait for clk_period;
                   button <= '1';
                   wait for clk_period;
                   button <= '0';
                    wait for clk_period;
                   button <= '1';
                   wait for clk_period;
                   button <= '0';
                    wait for clk_period;
                   button <= '1';
                   wait for clk_period;
                   button <= '0';
                    wait for clk_period;
                   button <= '1';
                   wait for clk_period;
                   button <= '0';
                    wait for clk_period;
                   button <= '1';
                   wait for clk_period;
                   button <= '0';
       wait for clk_period*100;

    end process;

END;
```

code test bench verification :





We simulated well and found that the output signal bounces were eliminated but was shifted due to a clock sync effect.

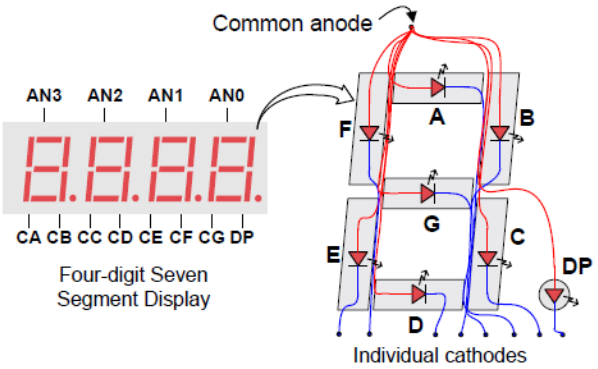
V. Using the 7-segment displays

Goal :

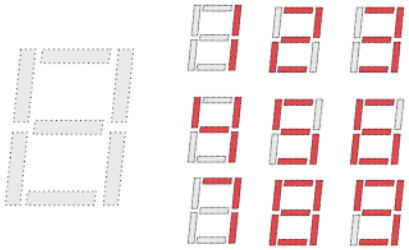
The goal is to use the 7-segment displays of the Xilink Spartan 6 FPGA board.

To achieve these goals, you have to follow certain steps:

1) Using the documentation for the Nexys 3 card, find out how to display a value between 0 and 15 on a display:



Four-digit Seven Segment Display

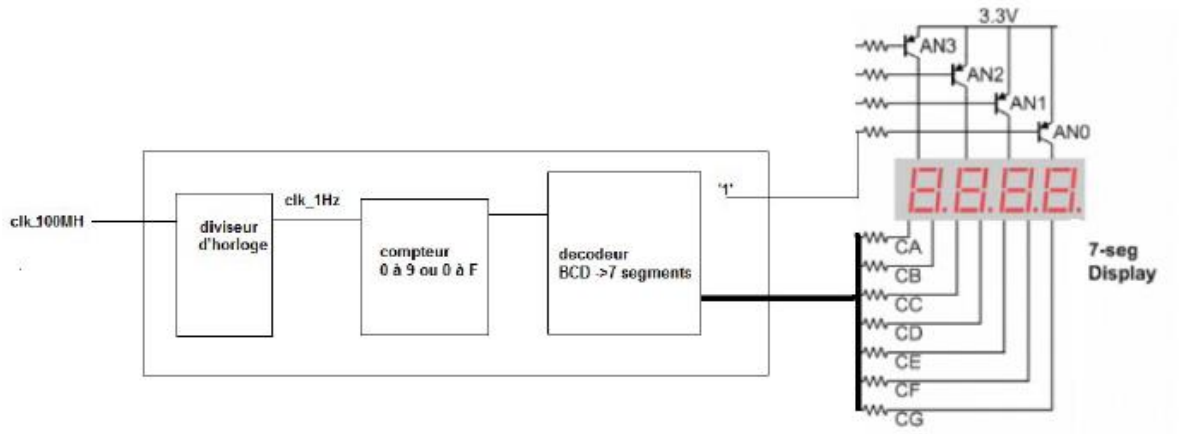


An un-illuminated seven-segment display, and nine illumination patterns corresponding to decimal digits

Then, fill in the values of the segments on the following truth table :

digit	display	A	b	c	d	e	f	g
0	0	1	1	1	1	1	1	0
1	1	0	1	1	0	0	0	0
2	2	1	1	0	1	1	0	1
3	3	1	1	1	1	0	0	1
4	4	0	1	1	0	0	1	1
5	5	1	0	1	1	0	1	1
6	6	1	0	1	1	1	1	1
7	7	1	1	1	0	0	0	0
8	8	1	1	1	1	1	1	1
9	9	1	1	1	1	0	1	1

Diagram to follow :



2) Create a VHDL design that displays the result of a counter that will need to be incremented if the user presses the top button, and decremented with the below. Use the right and left buttons to move the number displayed on one of the 4 digits

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity disp_hex_mux is
   port(
      clk, reset: in std_logic;
      hex3, hex2, hex1, hex0: in std_logic_vector(3 downto 0);
      dp_in: in std_logic_vector(3 downto 0);
      an: out std_logic_vector(3 downto 0);
      sseg: out std_logic_vector(7 downto 0)
   );
end disp_hex_mux ;

architecture arch of disp_hex_mux is
   -- each 7-seg led enabled (2^18/4)*25 ns (40 ms)
   constant N: integer:=19;
   signal q_reg, q_next: unsigned(N-1 downto 0);
   signal sel: std_logic_vector(1 downto 0);
   signal hex: std_logic_vector(3 downto 0);
   signal dp: std_logic;
begin
   -- register
   process(clk,reset)
   begin
      if reset='1' then
         q_reg <= (others=>'0');
      elsif (clk'event and clk='1') then
         q_reg <= q_next;
      end if;
   end process;

   -- next-state logic for the counter
   q_next <= q_reg + 1;

   -- 2 MSBs of counter to control 4-to-1 multiplexing
   sel <= std_logic_vector(q_reg(N-1 downto N-2));
   process(sel,hex0,hex1,hex2,hex3,dp_in)
   begin
      case sel is
         when "00" =>
            an <= "1110";
            hex <= hex0;
            dp <= dp_in(0);
         when "01" =>
            an <= "1101";
            hex <= hex1;
            dp <= dp_in(1);
         when "10" =>
            an <= "1011";
            hex <= hex2;
            dp <= dp_in(2);
         when others =>
            an <= "0111";
            hex <= hex3;
```

```
            dp <= dp_in(3);
      end case;
   end process;
   -- hex-to-7-segment led decoding
   with hex select
      sseg(6 downto 0) <=
         "0000001" when "0000",
         "1001111" when "0001",
         "0010010" when "0010",
         "0000110" when "0011",
         "1001100" when "0100",
         "0100100" when "0101",
         "0100000" when "0110",
         "0001111" when "0111",
         "0000000" when "1000",
         "0000100" when "1001",
         "0001000" when "1010", --a
         "1100000" when "1011", --b
         "0110001" when "1100", --c
         "1000010" when "1101", --d
         "0110000" when "1110", --e
         "0111000" when others; --f
   -- decimal point
   sseg(7) <= dp;
end arch;
```

Disp_hex_mux is based on a multiplexer that will help us to select the anode of the four 7 segments so we can switch it on and use it for our bcd counter.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.std_logic_unsigned.all;
entity hex_mux_test is
   generic (
            N :  std_logic_vector:=x"5F5E0FF"; -- count from 0 to M-1
            M : integer := 26   -- M bits required to count upto N i.e. 2**M
>= N
    );
    Port ( clk : in  STD_LOGIC;
           an : out  STD_LOGIC_vector(3 downto 0);
           sseg : out  STD_LOGIC_vector(7 downto 0);
           rst : in  STD_LOGIC);
end hex_mux_test;

architecture arch of hex_mux_test is
   signal bcd: std_logic_vector(15 downto 0);
      signal one_sec_counter: std_logic_vector(M downto 0);
      signal one_sec_en: std_logic;
      signal clk_in: std_logic;
```

```
      signal a,b,c,d: std_logic_vector(3 downto 0);
begin

       disp_unit: entity work.disp_hex_mux
                  port map(
                        clk=>clk_in, reset=>'0',
                        hex3=>d, hex2=>c,
                        hex1=>b, hex0=>a,
                        dp_in=>"1111", an=>an, sseg=>sseg);

-- One second Pulse or tick
process(clk,rst)
begin
  if rst='1' then
      one_sec_counter <= (others=>'0');
  elsif rising_edge(clk) then
      if one_sec_counter=N then
          one_sec_counter <= (others=>'0');
       else
          one_sec_counter <= one_sec_counter +"1";
      end if;
   end if;
end process;
one_sec_en <= '1' when one_sec_counter=N else '0';

--- BCD counter
process(clk_in,rst)
 begin
    if rst='1' then
       bcd<=(others=>'0');
    elsif rising_edge(clk_in) then
        if one_sec_en='1' then
             if  bcd(15 downto 0)="1001100110011001" then
         bcd(15 downto 0)<=(others=>'0');
      elsif bcd(11 downto 0)="100110011001" then
           bcd(11 downto 0)<=(others=>'0');
        elsif bcd(7 downto 0)="10011001" then
            bcd(7 downto 8)<=(others=>'0');
         elsif bcd(3 downto 0)="1001" then
                     bcd(3 downto 0)<=(others=>'0');
          else
         bcd(3 downto 0)<=bcd(3 downto 0) +"1";
            end if;
      end if;
      end if;
end process;

            a<=bcd(3 downto 0);
            b<=bcd(7 downto 4);
            c<=bcd(11 downto 8);
            d<=bcd(15 downto 12);

 end arch;
```

VI. Finite state machine

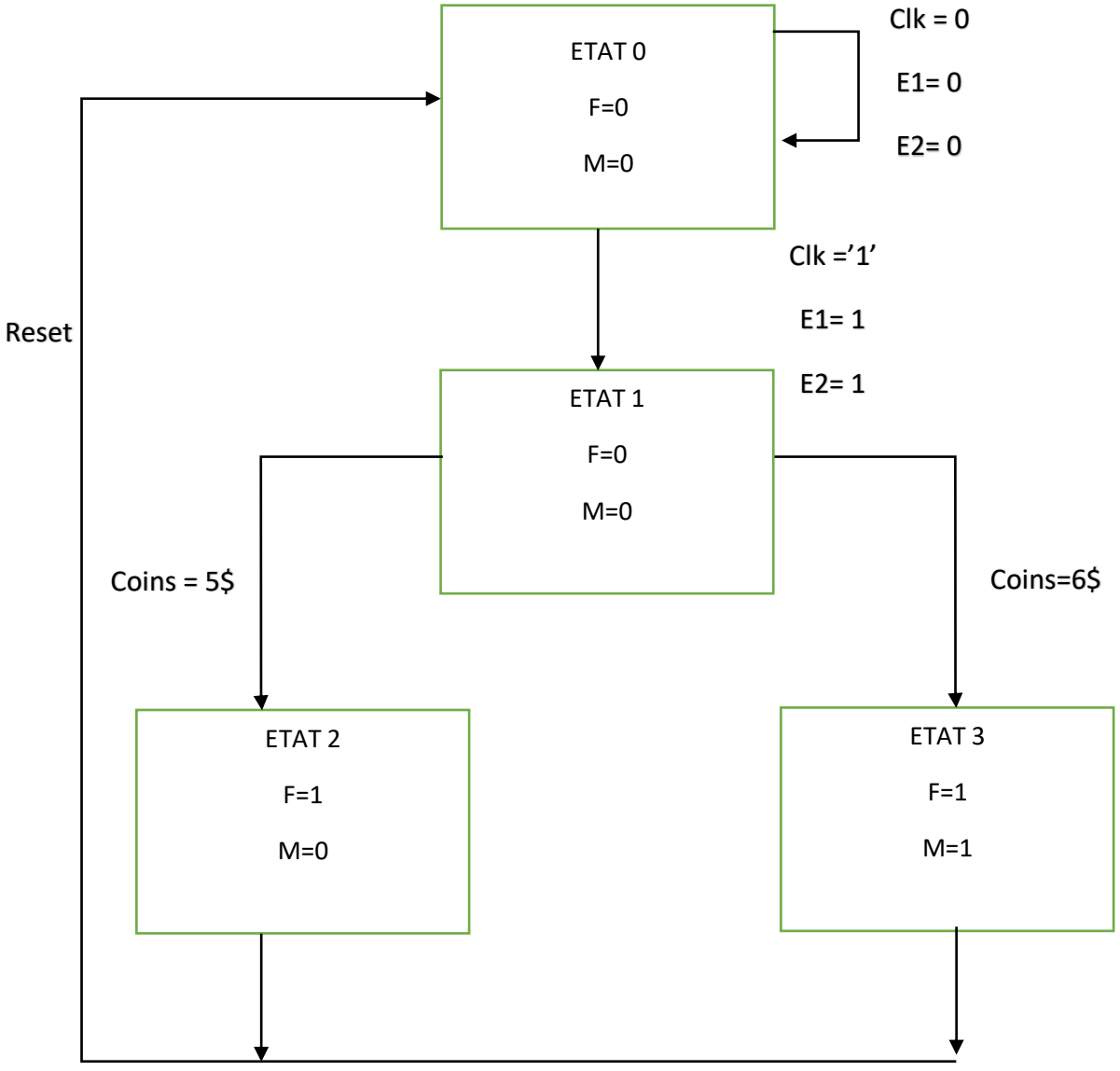
In this example, we are going to talk about state machines. We will consider the situation where there is a machine that sells the movies for $ 5. The user has the right to put in $ 1 coins or of $ 2 and as soon as the amount gets to $ 5 or more, a movie comes out. It is also possible to the user puts in $ 6 and in this case the machine gives a movie and also gives change.

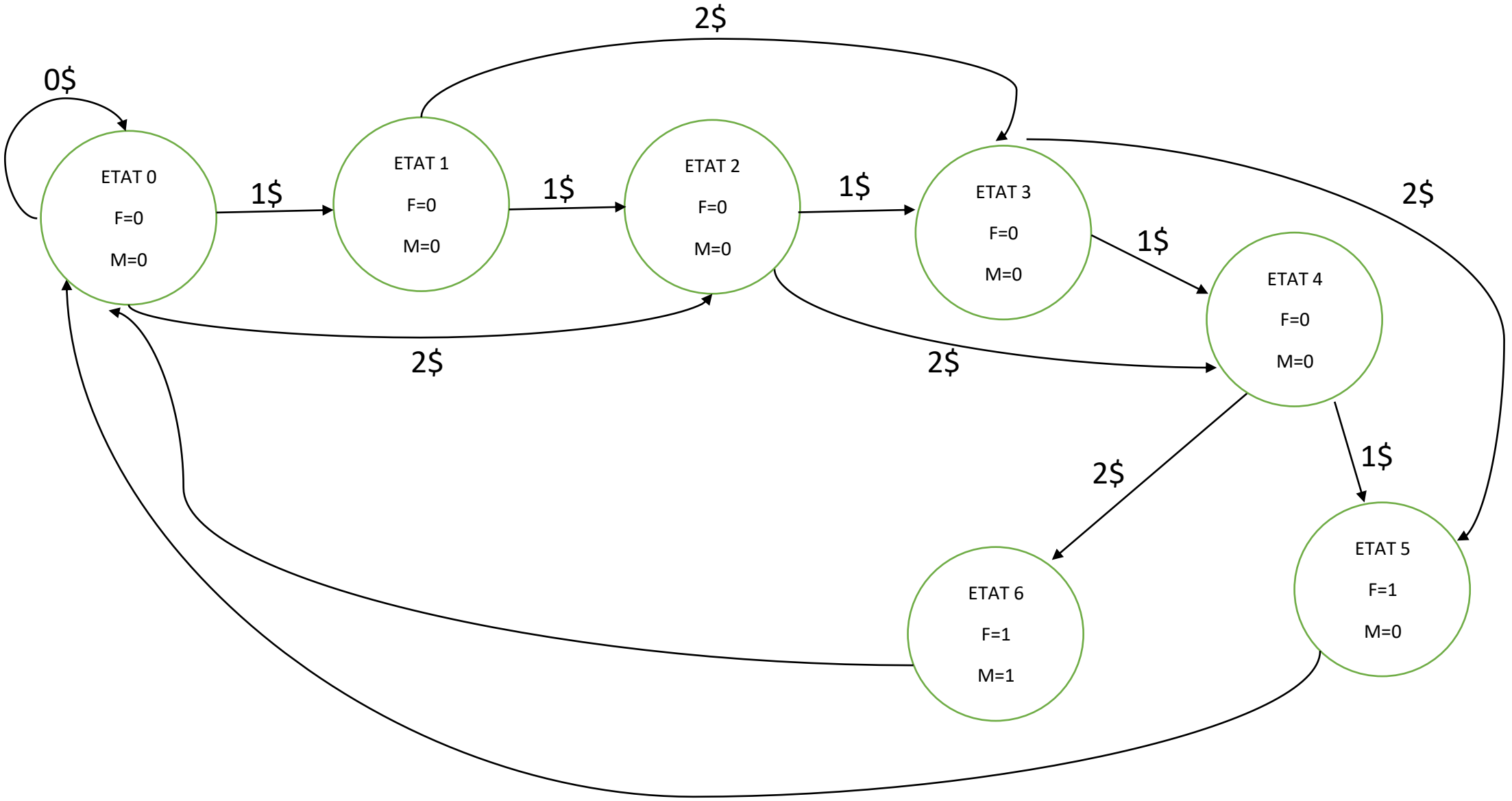
Goal :

The goal is to make a MOORE machine and so the outputs are state dependent. One more piece of information is that if the person puts money in while the movie is being released, the money will be lost.

Before moving on to VHDL code, determine the state graph. More details on the code can be found by viewing the course slides.

State transition graph will be as shown in the figures below:




0$
ETAT 0
F=0
M=0
1$
ETAT 1
F=0
M=0
1$
ETAT 2
F=0
M=0
1$
ETAT 3
F=0
M=0
2$
1$
ETAT 4
F=0
M=0
2$
2$
2$
2$
1$
ETAT 5
F=1
M=0
ETAT 6
F=1
M=1

How the program works :

- If the user presses three times the button corresponding to two_dollars, the machine will be in the state corresponding to film = "1" and change = "1".
- If the user presses five time in a row the button corresponding to un_dollar, the machine will be in the state corresponding to film = "1" and change = "0".

You must use the 1Hz clock to view the result on the map. Therefore, we must create a clock divider allowing the creation of a new 1Hz clock from the main clock of the FPGA.

VHDL :

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY machine_etats_top IS
      PORT (
                  clk : IN STD_LOGIC;
                  un_dollar : IN STD_LOGIC;
                  deux_dollars : IN STD_LOGIC;
                  film : OUT STD_LOGIC;
                  monnaie : OUT STD_LOGIC;
                  etat : OUT std_logic_vector (2 downto 0)
                                    );
END machine_etats_top;
ARCHITECTURE Behavioral OF machine_etats_top IS
            TYPE etats IS (etat_0, etat_1, etat_2, etat_3, etat_4, etat_5,
etat_6);
            SIGNAL etat_present : etats;
            SIGNAL etat_prochain : etats;
BEGIN
      Pro_1:PROCESS (etat_present, un_dollar, deux_dollars)
            BEGIN
                  CASE etat_present IS
                        WHEN etat_0 =>
                              film <= '0';
                              monnaie <= '0';
                              etat <= "000";
                              IF un_dollar = '1' THEN
                              etat_prochain <= etat_1;
                              ELSIF deux_dollars = '1' THEN
                              etat_prochain <= etat_2;
                              ELSE
                              etat_prochain <= etat_0;
                              END IF;
                        WHEN etat_1 =>
                              film <= '0';
                              monnaie <= '0';
                              etat <= "001";
                              IF un_dollar = '1' THEN
                              etat_prochain <= etat_2;
                              ELSIF deux_dollars = '1' THEN
```

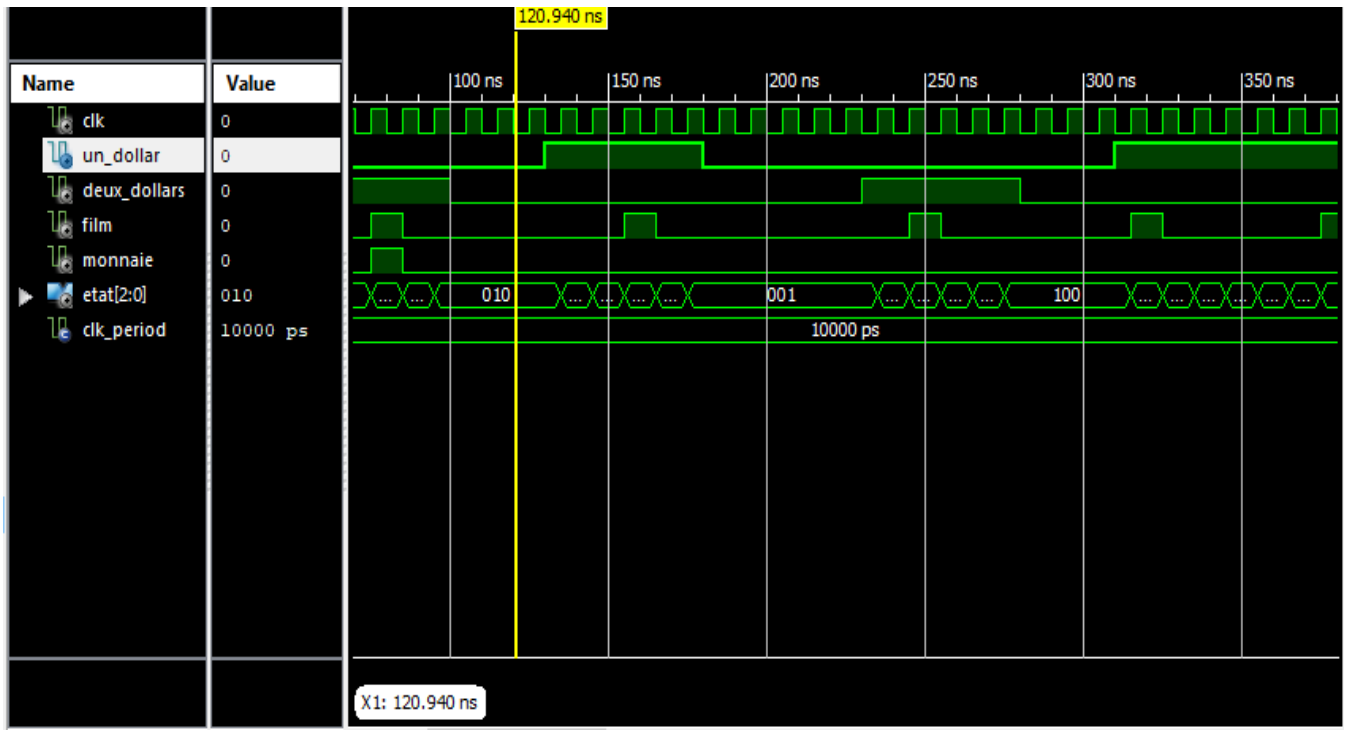
```
                              etat_prochain <= etat_3;
                              ELSE
                              etat_prochain <= etat_1;
                              END IF;
                        WHEN etat_2 =>
                              film <= '0';
                              monnaie <= '0';
                              etat <= "010";
                              IF un_dollar = '1' THEN
                              etat_prochain <= etat_3;
                              ELSIF deux_dollars = '1' THEN
                              etat_prochain <= etat_4;
                              ELSE
                              etat_prochain <= etat_2;
                              END IF;
                        WHEN etat_3 =>
                              film <= '0';
                              monnaie <= '0';
                              etat <= "011";
                              IF un_dollar = '1' THEN
                              etat_prochain <= etat_4;
                              ELSIF deux_dollars = '1' THEN
                              etat_prochain <= etat_5;
                              ELSE
                              etat_prochain <= etat_3;

                              END IF;
                        WHEN etat_4 =>
                              film <= '0';
                              monnaie <= '0';
                              etat <= "100";
                              IF un_dollar = '1' THEN
                              etat_prochain <= etat_5;
                              ELSIF deux_dollars = '1' THEN
                              etat_prochain <= etat_6;
                              ELSE
                              etat_prochain <= etat_4;
                              END IF;
                        WHEN etat_5 =>
                              film <= '1';
                              monnaie <= '0';
                              etat <= "101";
                              etat_prochain <= etat_0;
                        WHEN etat_6 =>
                              film <= '1';
                              monnaie <= '1';
                              etat <= "110";
                              etat_prochain <= etat_0;
      END CASE;
END PROCESS;
Pro_2:PROCESS (clk)
      BEGIN
      IF clk'EVENT AND clk = '1' THEN
      etat_present <= etat_prochain;
      END IF;
END PROCESS;
END Behavioral;
```

Test bench :

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY machine_etat_tb IS
END machine_etat_tb;

ARCHITECTURE behavior OF machine_etat_tb IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT machine_etats_top
    PORT(
         clk : IN  std_logic;
         un_dollar : IN  std_logic;
         deux_dollars : IN  std_logic;
         film : OUT  std_logic;
         monnaie : OUT  std_logic;
         etat : OUT  std_logic_vector(2 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal clk : std_logic := '0';
   signal un_dollar : std_logic := '0';
   signal deux_dollars : std_logic := '0';

      --Outputs
   signal film : std_logic;
   signal monnaie : std_logic;
   signal etat : std_logic_vector(2 downto 0);

   -- Clock period definitions
   constant clk_period : time := 10 ns;

BEGIN

      -- Instantiate the Unit Under Test (UUT)
   uut: machine_etats_top PORT MAP (
          clk => clk,
          un_dollar => un_dollar,
          deux_dollars => deux_dollars,
          film => film,
          monnaie => monnaie,
          etat => etat
        );

   -- Clock process definitions
   clk_process :process
   begin
            clk <= '0';
            wait for clk_period/2;
            clk <= '1';
            wait for clk_period/2;
   end process;
```

```
  -- Stimulus process
   stim_proc: process
   begin

wait for 50 ns;
un_dollar <= '0';
deux_dollars <= '1';
wait for clk_period*5;
un_dollar <= '0';
deux_dollars <= '0';
wait for clk_period*3;
un_dollar <= '1';
deux_dollars <= '0';
wait for clk_period*5;
un_dollar <= '0';
deux_dollars <= '0';
wait for clk_period*5;
un_dollar <= '0';
deux_dollars <= '1';
wait for clk_period*5;
un_dollar <= '0';
deux_dollars <= '0';
wait for clk_period*3;
un_dollar <= '1';
deux_dollars <= '0';
wait for clk_period*3;

      wait;
   end process;

END;
```

Simulation with isim



Conclusion TP final state machine :

For this tp we were unable to complete it due to the current circumstances regarding access to the tp room and the use of nexus cards for the implementation of our program.