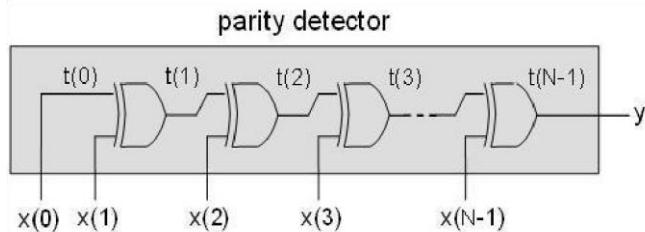


Parity Checker

A parity check is the process that ensures accurate data transmission between nodes during communication. Parity checker output = the xor operation of all the inputs.

The same can be seen in the code where $a(0) \oplus a(1) \oplus a(2)$ and $a(3)$



$$y = x(0) \oplus x(1) \oplus x(2) \oplus \dots \oplus x(N-1)$$

Source

The screenshot shows the ModelSim PE Student Edition 10.4a interface. The top menu bar includes File, Edit, View, Compile, Simulate, Add, Source, Tools, Layout, Bookmarks, Window, and Help. The toolbar contains various simulation and design tools. The left pane displays a hierarchical tree view of the project structure under 'sim - Default'. The right pane shows the VHDL source code for 'even_detector' and 'parity_4b.vhd'. The bottom pane features tabs for Library, Project, sim, Transcript, Wave, and several open files including tb_reg16b.vhd, parity_generator.vhd, Parity_4b.vhd, tb_parity.vhd, and reg_16b.vhd. The Windows taskbar at the bottom shows the current time as 3:35 AM and the date as 9/21/2018.

TestBench

The screenshot shows the ModelSim PE interface. The top menu bar includes File, Edit, View, Compile, Simulate, Add, Source, Tools, Layout, Bookmarks, Window, and Help. The toolbar contains various simulation and design tools. The left pane displays the project hierarchy under 'sim - Default' with nodes like tb_parity, DUT, line_17, standard, textio, and std_logic_1164. The right pane shows the VHDL code for tb_parity.vhd:

```

library ieee;
use ieee.std_logic_1164.all;
entity tb_parity is
end tb_parity;
architecture tbp of tb_parity is
component even_detector is
port( a: in std_logic_vector(3 downto 0);
      even : out std_logic);
end component;
signal a: std_logic_vector(3 downto 0);
signal even: std_logic;
begin
DUT: even_detector port map (a,even); --portmapping
process
begin
  a<="0001"; -- tests
  wait for 5ns;
  a<="1010";
  wait for 10ns;
  a<="0101";
  wait for 15ns;
  a<="1110";
  wait for 20ns;
end process;
end tbp;

```

Below the code editor is a transcript window showing simulation commands:

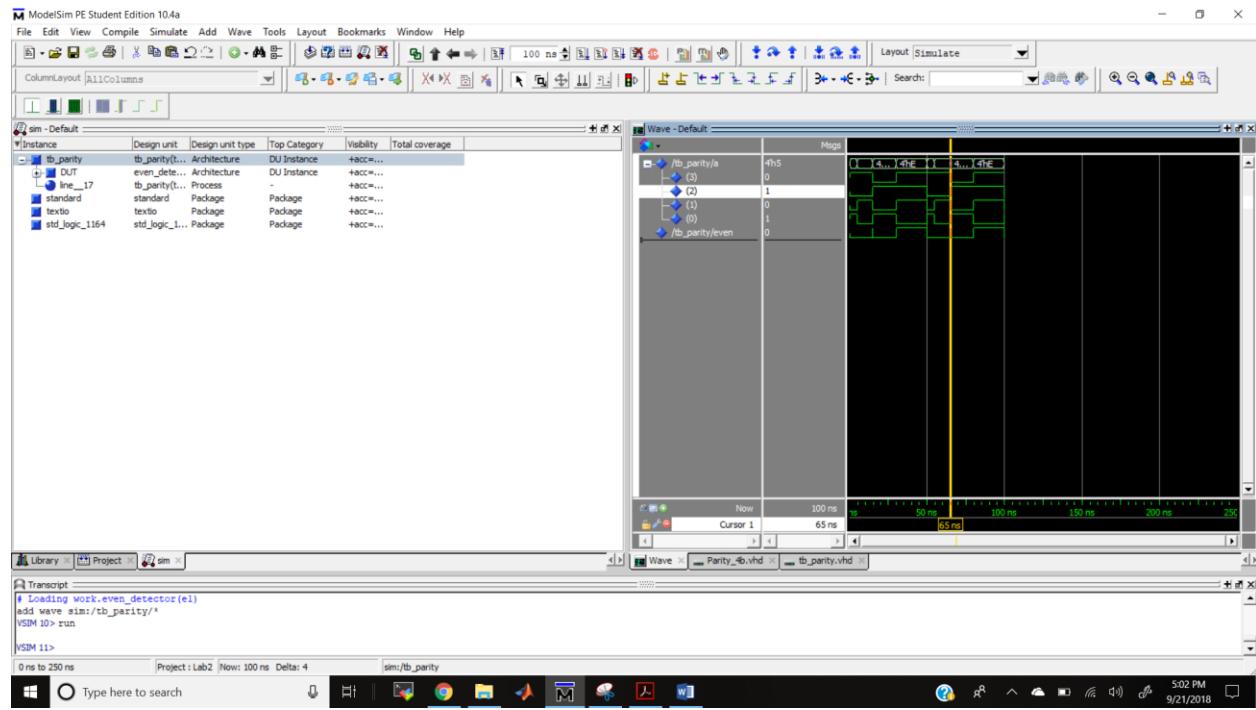
```

# Loading work.even_detector();
add wave sim:/tb_parity/*
VSM ID: run

```

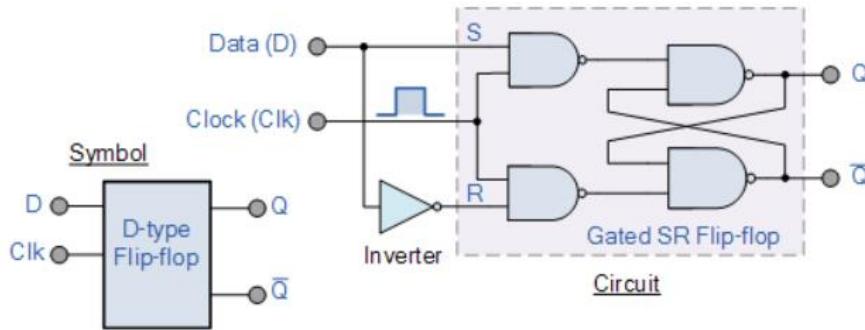
The bottom status bar indicates the project is Lab2, now at 100 ns, and the date is 9/21/2018.

Waveform



D flipflop

The “D flip flop” will store and output whatever logic level is applied to its data terminal so long as the clock input is HIGH.



Source code

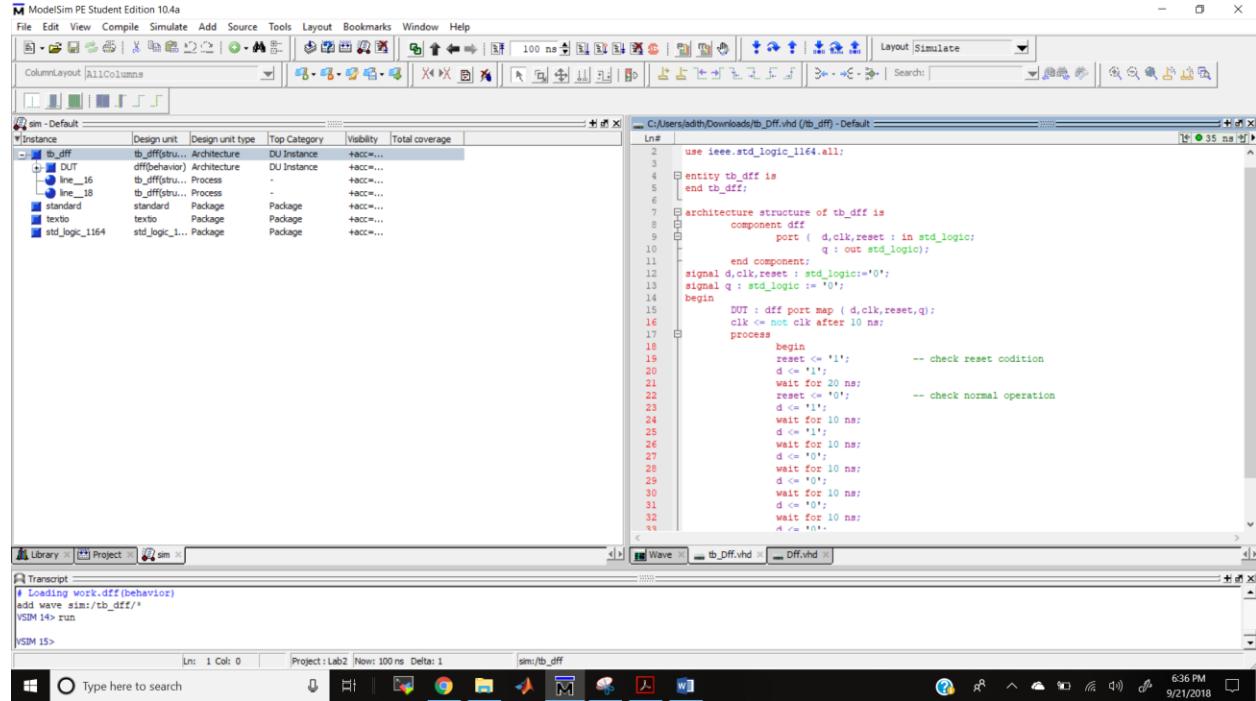
The screenshot shows the ModelSim PE interface with the source code for a D-type flip-flop. The code is written in VHDL and defines an entity 'dff' with a single port: 'd, clk, reset' and 'q' as output. The architecture 'behavior' contains a process that updates 'q' based on 'clk' and 'reset' signals. The code also includes standard library declarations and package imports.

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

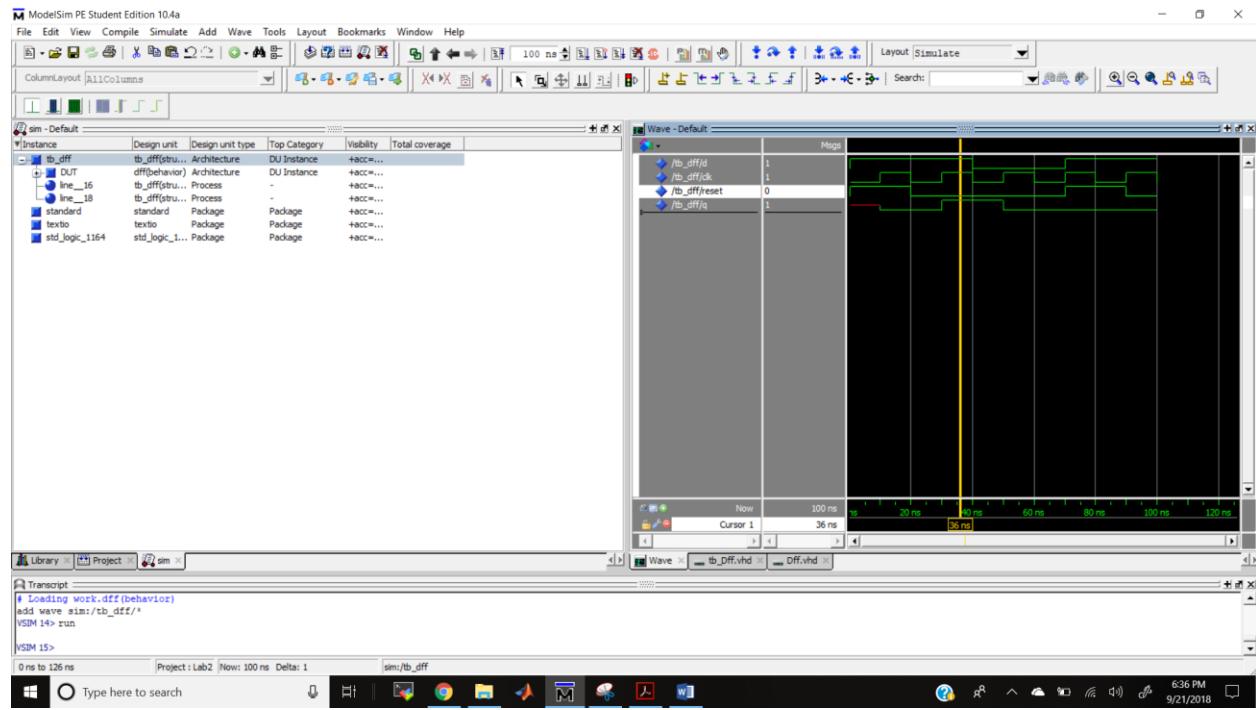
entity dff is
port ( d,clk,reset : in std_logic;
       q : out std_logic);
end dff;

architecture behavior of dff is
begin
process(clk)
begin
if (clk'event and clk='1') then
  if (reset = '1') then
    q <= '0';
  else
    q <= d;
  end if;
end if;
end process;
end behavior;
```

The interface also shows the project structure, simulation results, and transcript window.



Waveform



Register – 4Bit

Registers are used for data storage or for the movement of data and are therefore commonly used inside calculators or computers to store data

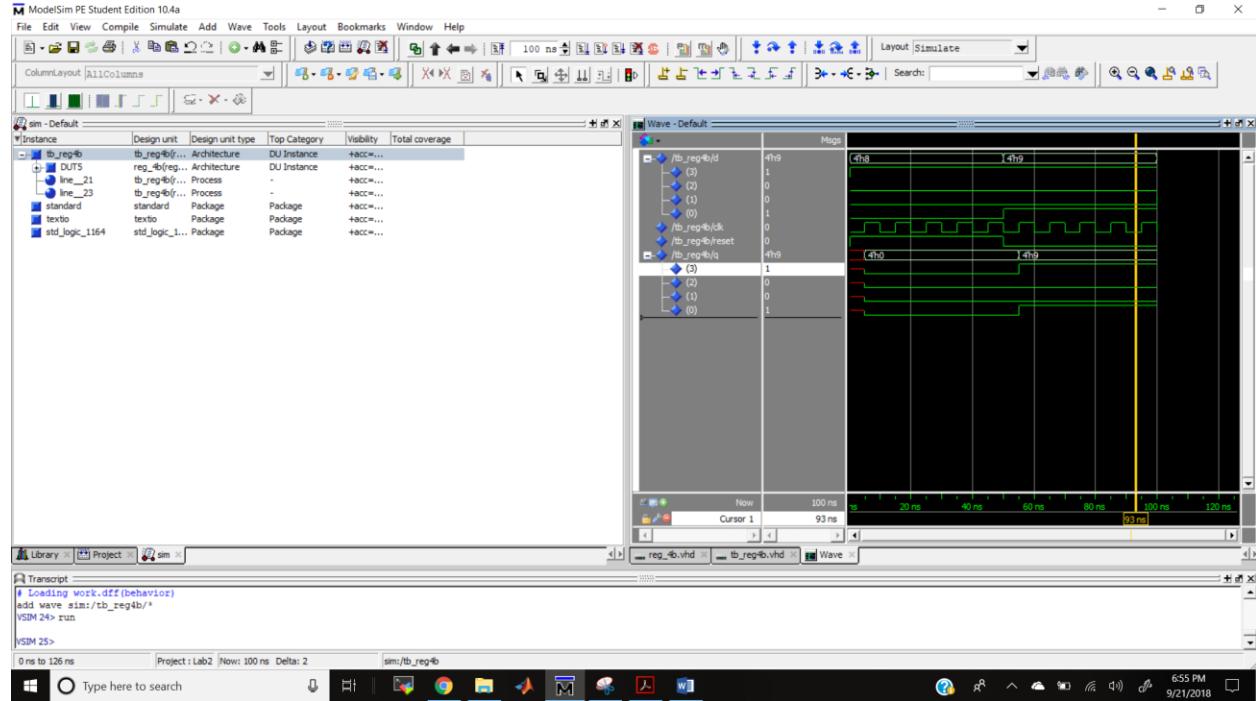
Source code

The screenshot shows the ModelSim PE interface with the following details:

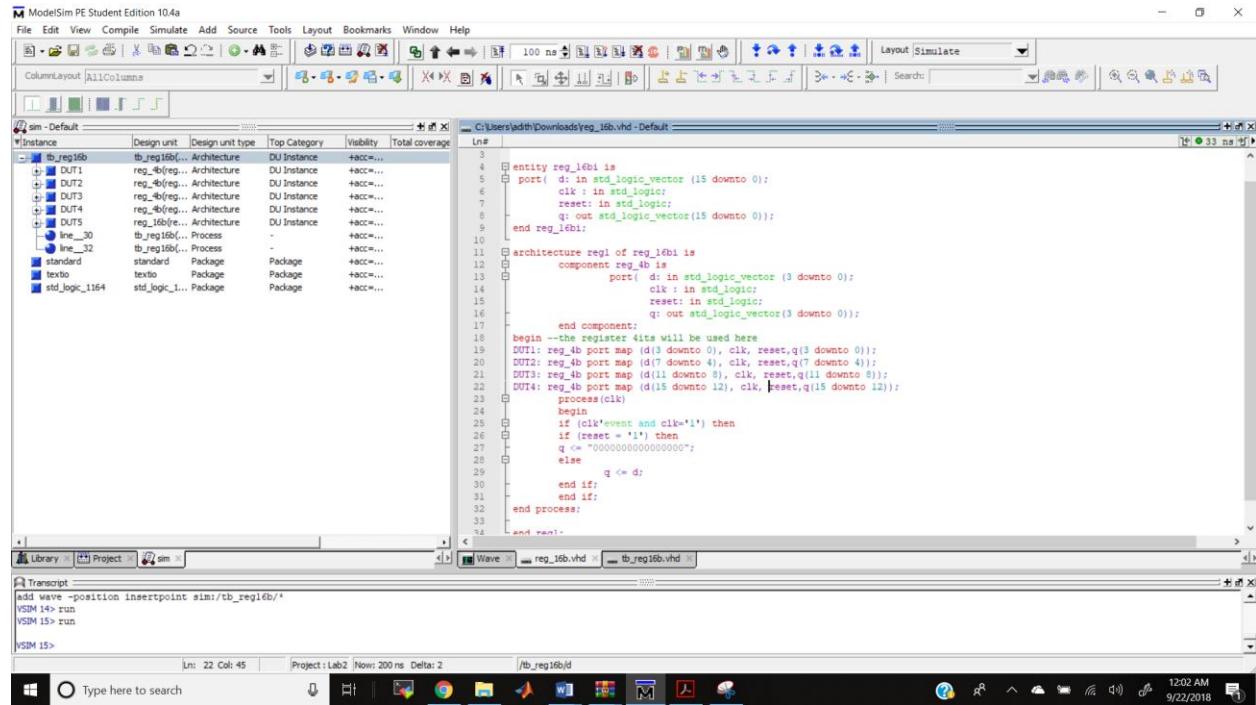
- File Menu:** File, Edit, View, Compile, Simulator, Add, Source, Tools, Layout, Bookmarks, Window, Help.
- Toolbar:** Includes icons for New, Open, Save, Run, Stop, Break, Step, Zoom, and various simulation controls.
- Search Bar:** Located at the top right.
- Column Layout:** Set to "All columns".
- Simulator View:** "sim - Default" tab. It lists components and packages:
 - tb_reg4b (DU Instance)
 - DUTs:
 - tb_reg4b (Architecture)
 - reg_4b (Architecture)
 - line_23 (Process)
 - standard (Package)
 - textio (Package)
 - std_logic_1164 (Package)
- Code Editor:** Displays the VHDL code for tb_reg4b.vhd. The code defines a process that maps four DFFs (DUT1-DUT4) to a register (reg_4b). It includes a synchronous reset logic where if both clk and reset are high, q is set to 0000.
- Library View:** Shows the project structure with tb_reg4b.vhd as the active file.
- Transcript:** Shows the command history for loading the work library and adding the tb_reg4b waveform.
- Bottom Status Bar:** Shows the current line (Ln: 25), column (Col: 10), project (LabZ), now time (100 ns), delta (Delta: 2), and the current simulation file (sim/tb_reg4b).

Test Bench

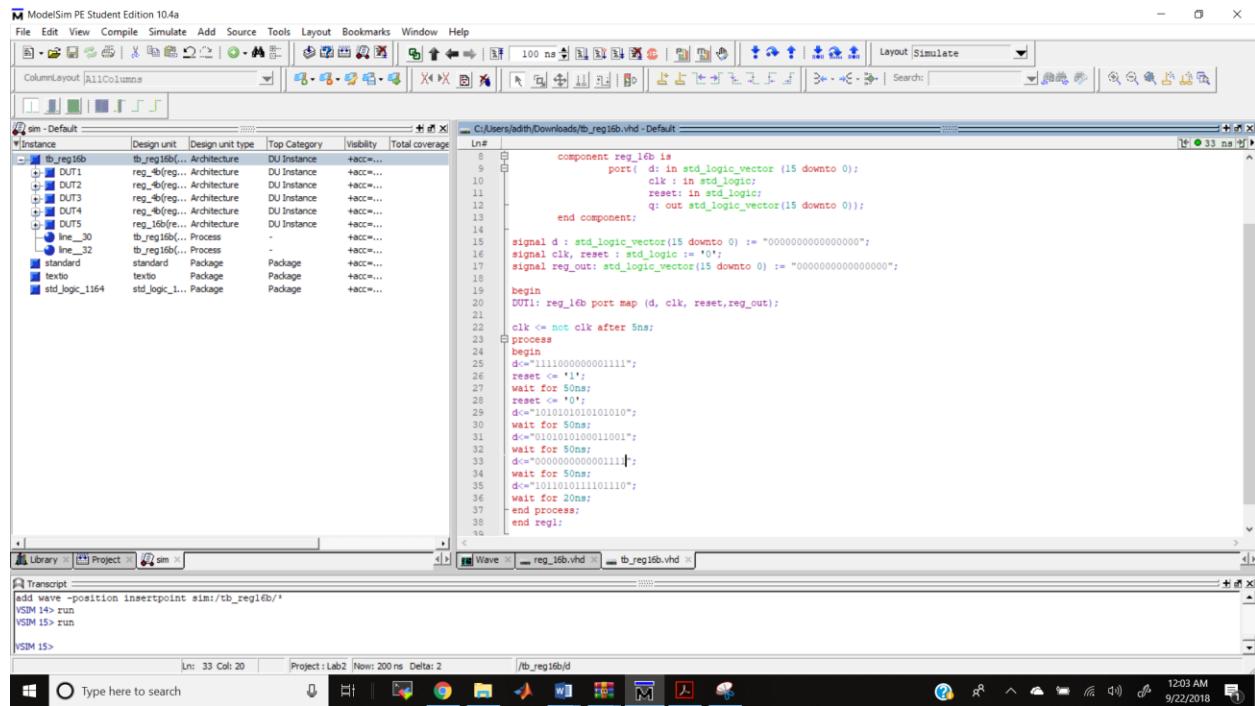
The screenshot shows the ModelSim PE software interface. The top menu bar includes File, Edit, View, Compile, Simulator, Add, Source, Tools, Layout, Bookmarks, Window, and Help. The toolbar contains various icons for simulation, waveform viewing, and file operations. The left pane displays a hierarchical tree of design units under 'sim - Default', including 'tb_reg4b' as the selected architecture. The right pane shows the VHDL source code for 'tb_reg4b.vhd'. The bottom pane features tabs for 'Library', 'Project', 'sim', 'Transcript', and 'Wave'. The Transcript window shows command-line interactions for loading work files and setting up the simulation. The Wave window is currently empty.



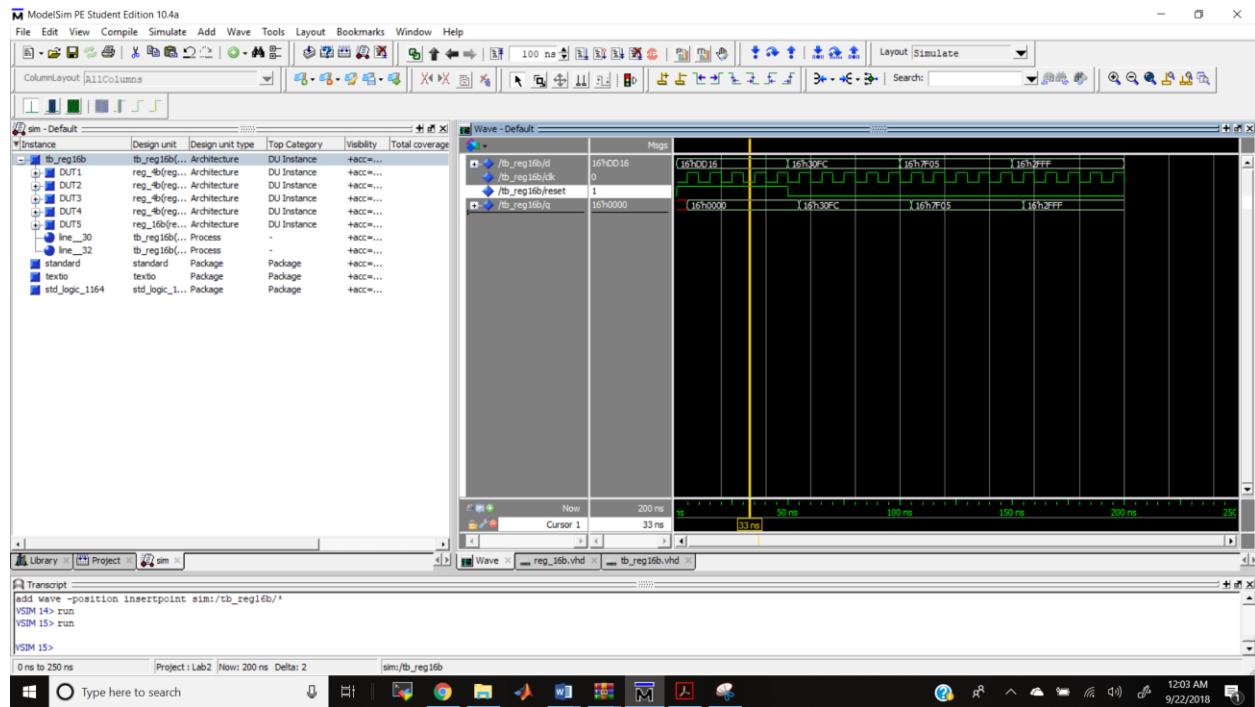
Register 16bit.



Testbench



Waveform



Counter

A 4 bit counter is as shown below. The incrementation of a bit takes place in every cycle.

For example, the output when the input is 0111 is 1000. i.e from 3 to 4.

The vhdl code for it is as shown in the image. Waveform is also obtained

Source code

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity COUNT is
    Port ( clk: in std_logic;
           reset: in std_logic;
           counter: out std_logic_vector(3 downto 0) );
end COUNT;
architecture Behavior of COUNT is
begin
    process(clk,reset)
    begin
        if(rising_edge(clk)) then
            if(reset='1') then
                outl <= x"0";
            else
                outl <= outl + x"1";
            end if;
        end process;
        counter <= outl;
    end Behavioral;
end;

```

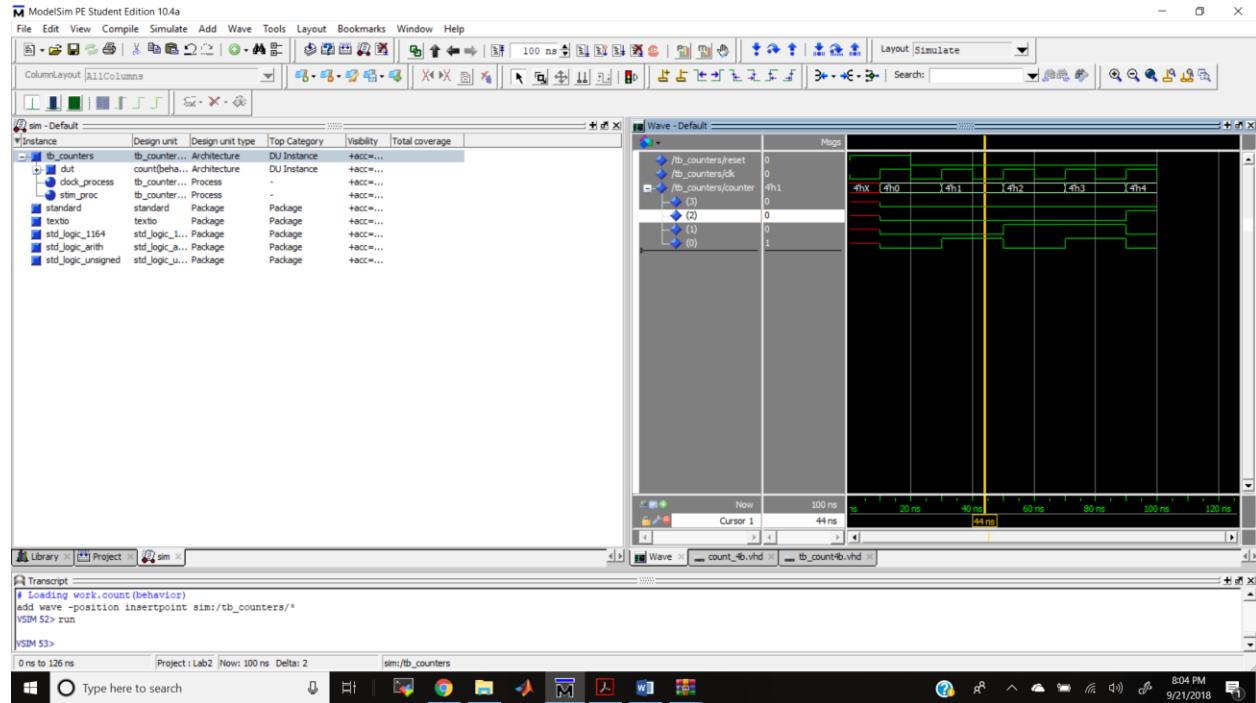
Test bench

```

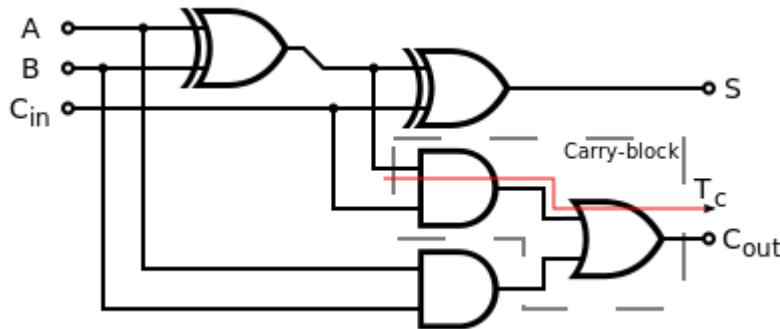
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity tb_counters is
end tb_counters;
architecture Behavioral of tb_counters is
component COUNT
    Port ( clk: in std_logic;
           reset: in std_logic;
           counter: out std_logic_vector(3 downto 0) );
end component;
signal reset,clk: std_logic;
signal counter:std_logic_vector(3 downto 0);
begin
    dut: COUNT port map (clk => clk, reset=>reset, counter => counter);
    clock_process :process -- two process, clk and rst
    begin
        clk <='0';
        wait for 10 ns;
        clk <='1';
        wait for 10 ns;
    end process;
    reset_process :process
    begin
        reset <='1';
        wait for 20 ns;
        reset <='0';
        wait;
    end process;
end Behavioral;

```

Waveform



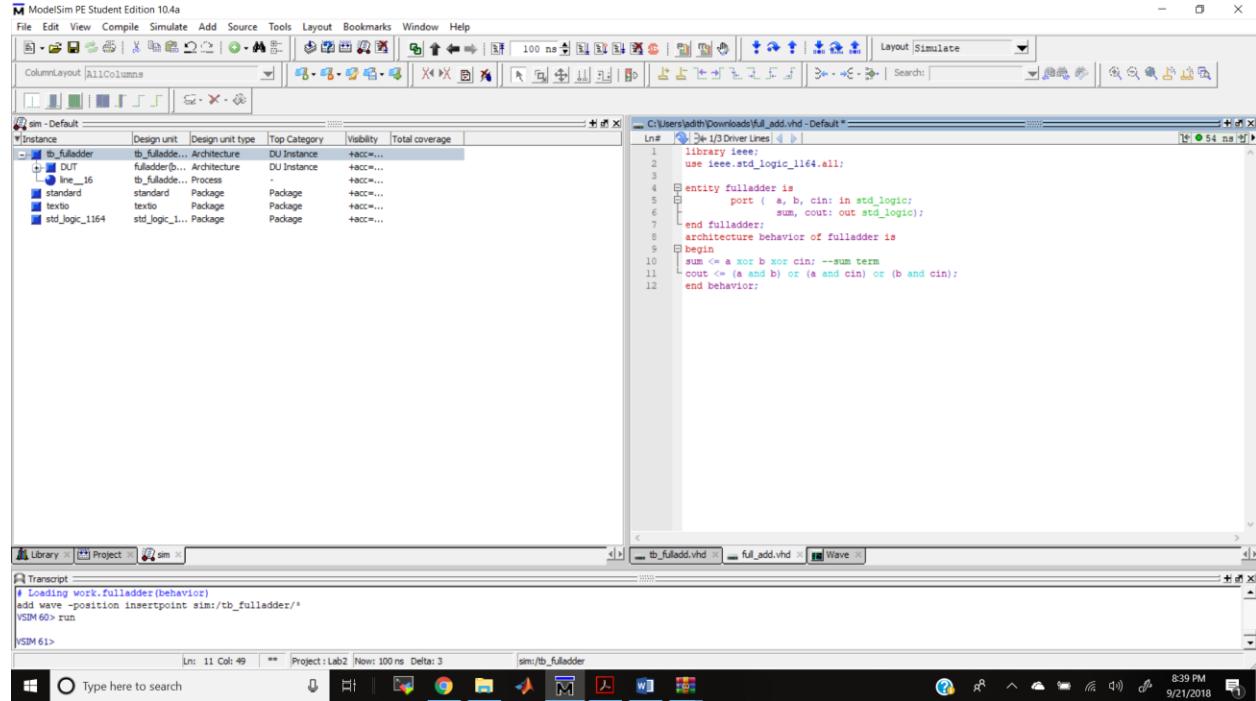
Full adder



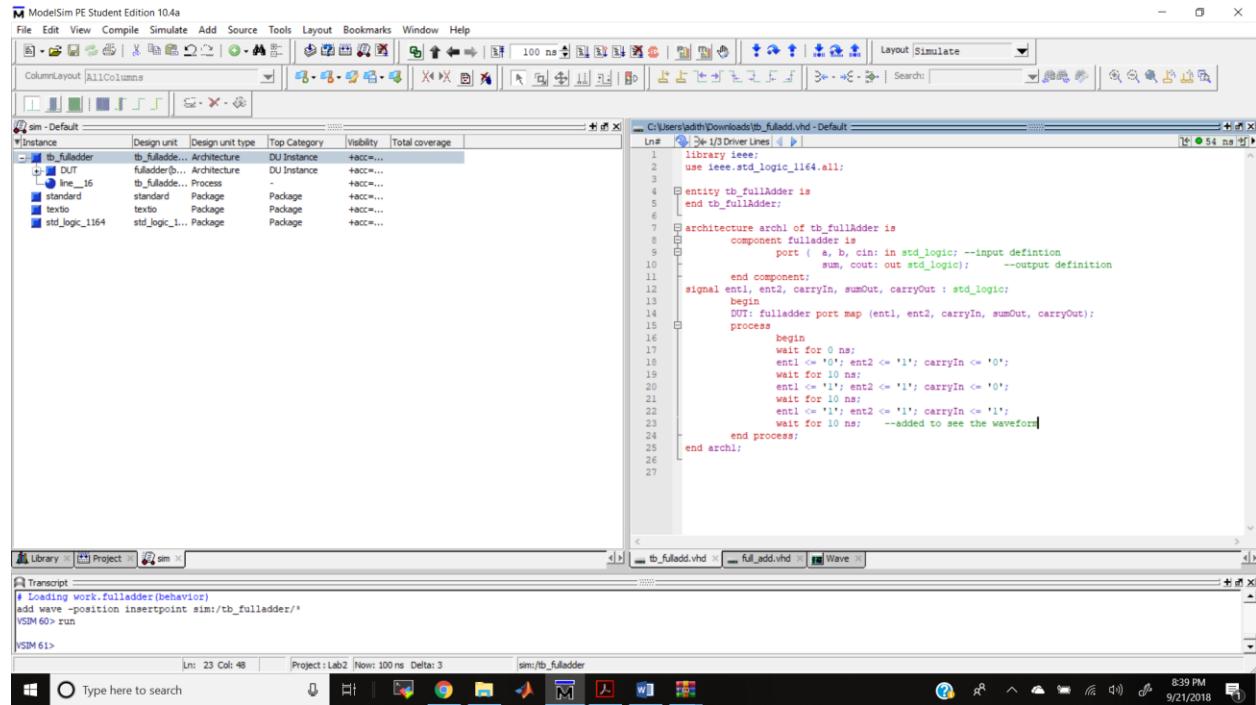
$$S = A \text{ xor } B \text{ xor } C_{in}$$

$$C_{out} = A \cdot B + (C_{in} \text{ xor } (A \text{ xor } B))$$

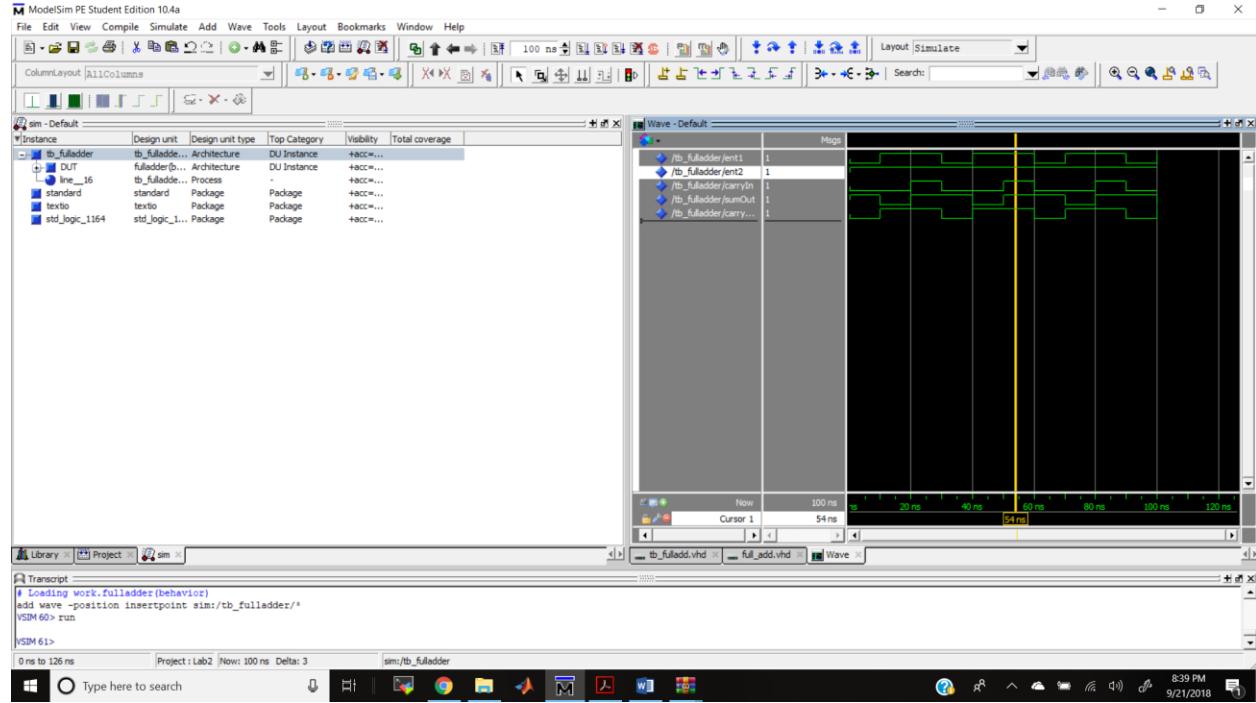
Source



Test bench

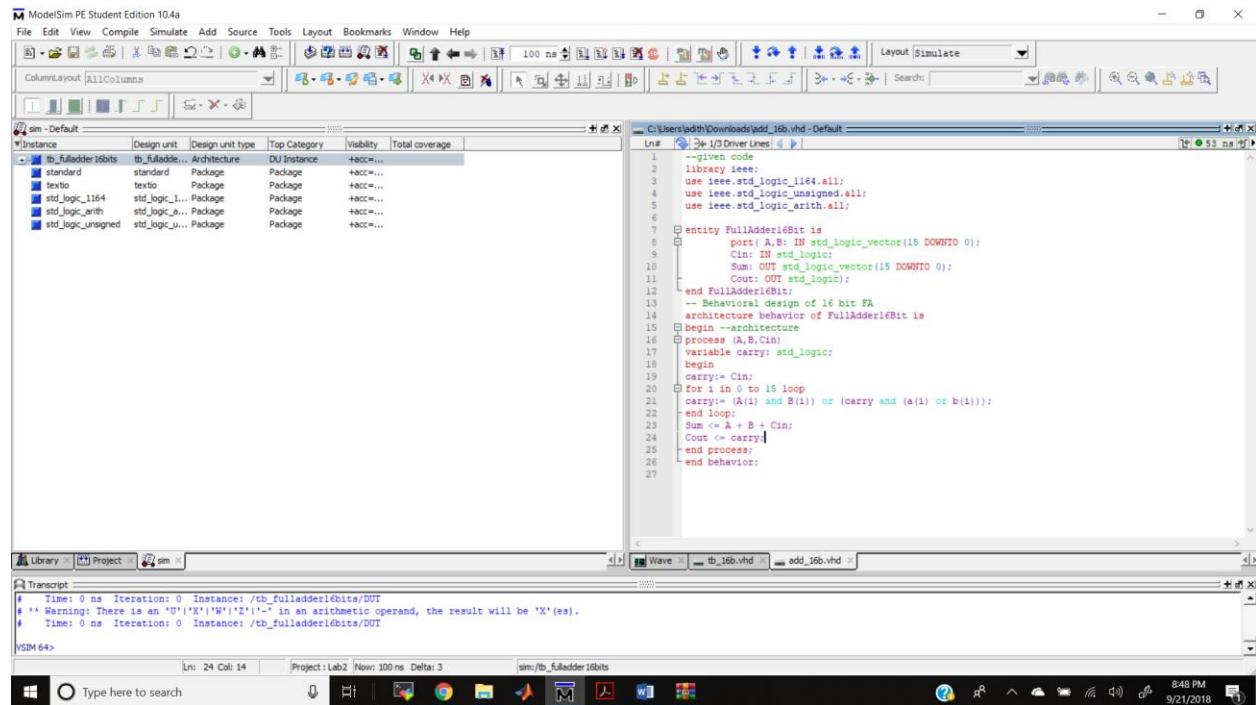


Waveform



Full adder – 16bits

Similarly a 16 bits full adder is created using full adders



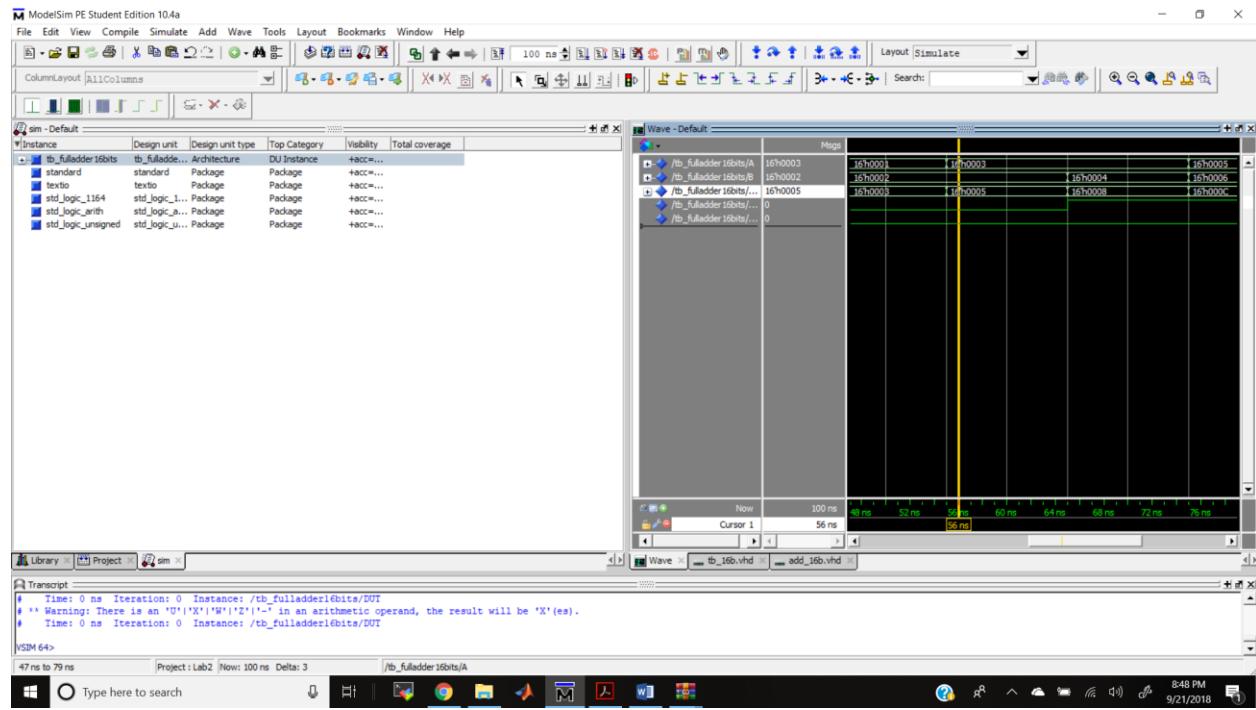
Test bench

```

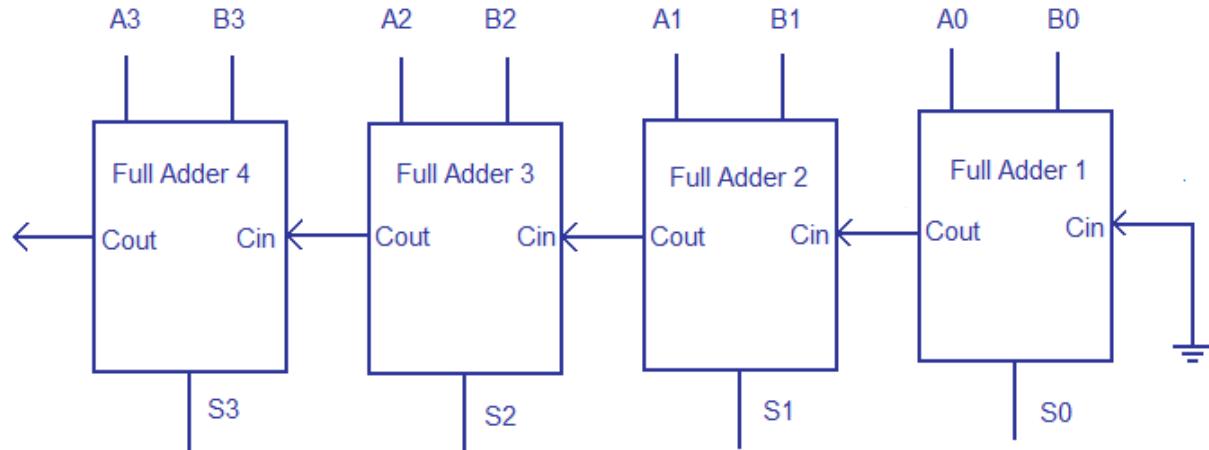
entity tb_FullAdder16Bits is
end entity tb_FullAdder16Bits;
architecture tb_FullAdder16Bits of tb_FullAdder16Bits is
begin
    component FullAdder16Bit
        port ( A,B: IN std_logic_vector(15 downto 0);
               Cin: IN std_logic;
               Sum: OUT std_logic_vector(15 downto 0);
               Cout: OUT std_logic);
    end component;
    signal A,B,Cin: std_logic_vector(15 downto 0);
    signal Cin,Cout : std_logic;
begin
    DUT: FullAdder16Bit port map (A,B,Cin,Cout);
    process
        begin
            --provide input stimuli
            wait for 0 ns;
            A <= "x0000"; B <= "x0002"; Cin <= '0';
            wait for 10 ns;
            A <= "x0003";
            wait for 10 ns;
            B <= "x0004"; Cin <= '1';
            wait for 10 ns;
            A <= "x0005"; B <= "x0006";
            wait for 10 ns;
            Cin <= '0';
            wait for 5ns;
        end process;
    end structure;

```

Waveform



Ripple carry adder - 4bit

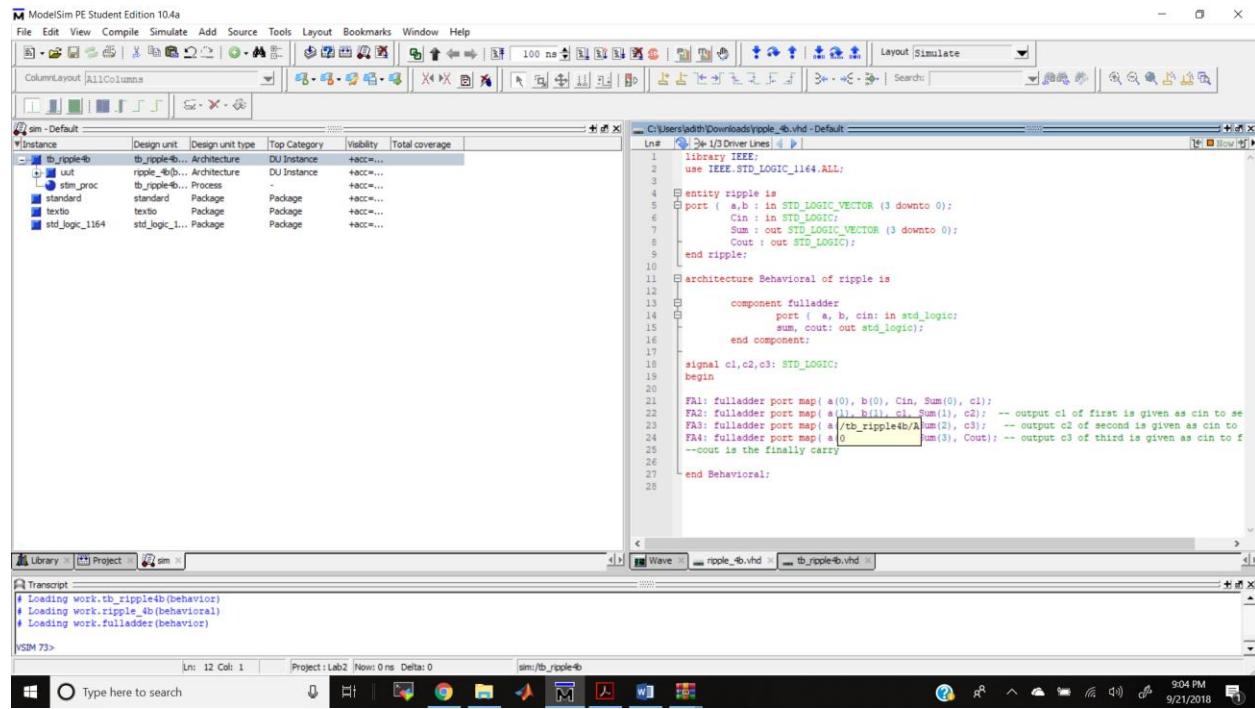


4 bit ripple carry adder

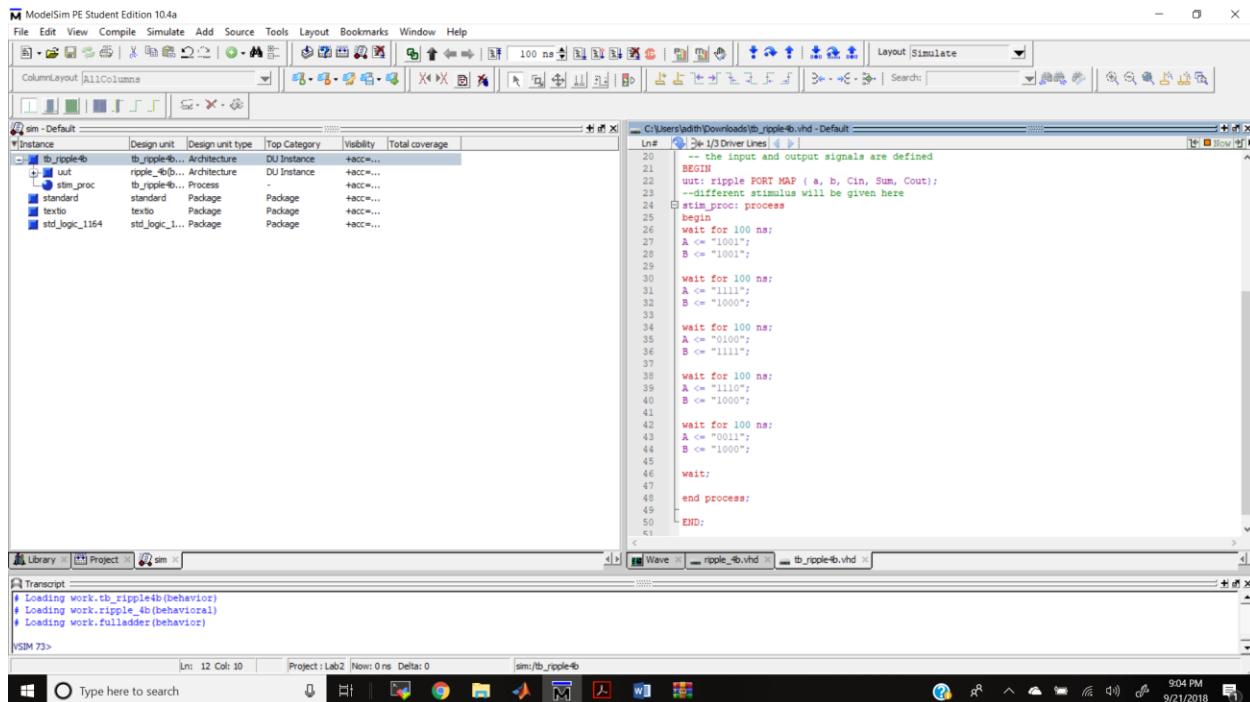
www.circuitstoday.com

As seen above the cout of first is given as an input to the next and so on.

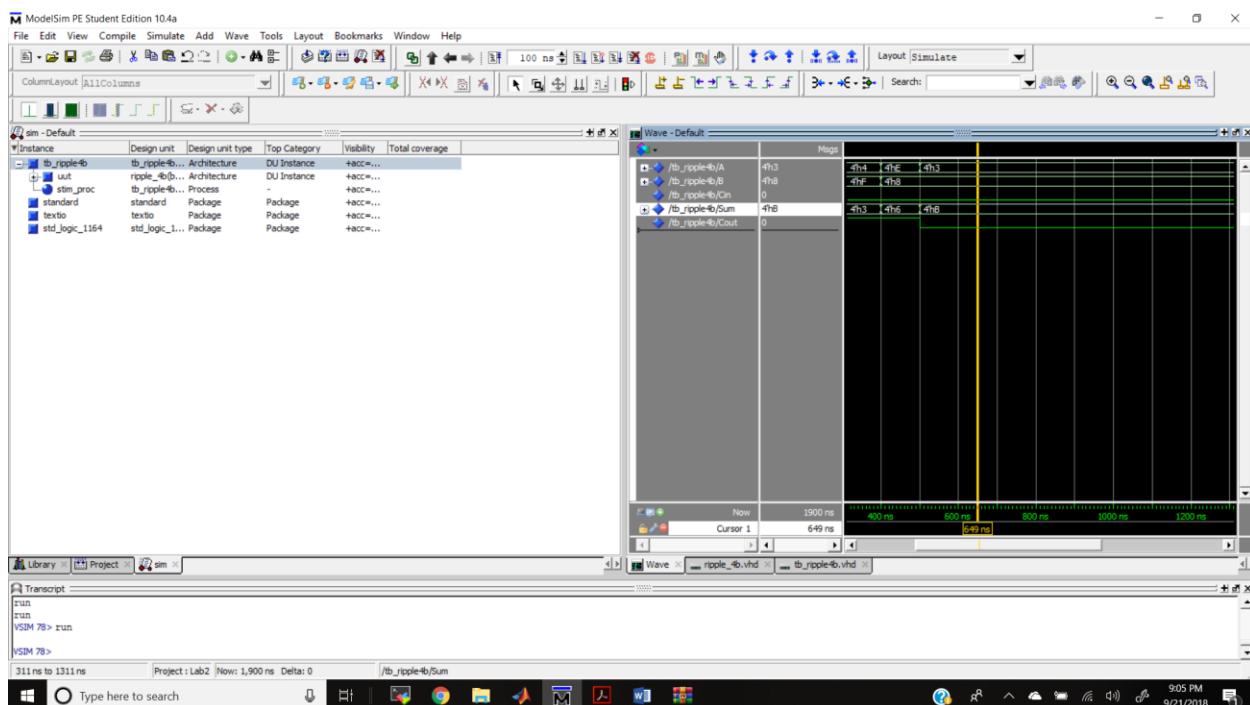
This is how a ripple carry adder is created.



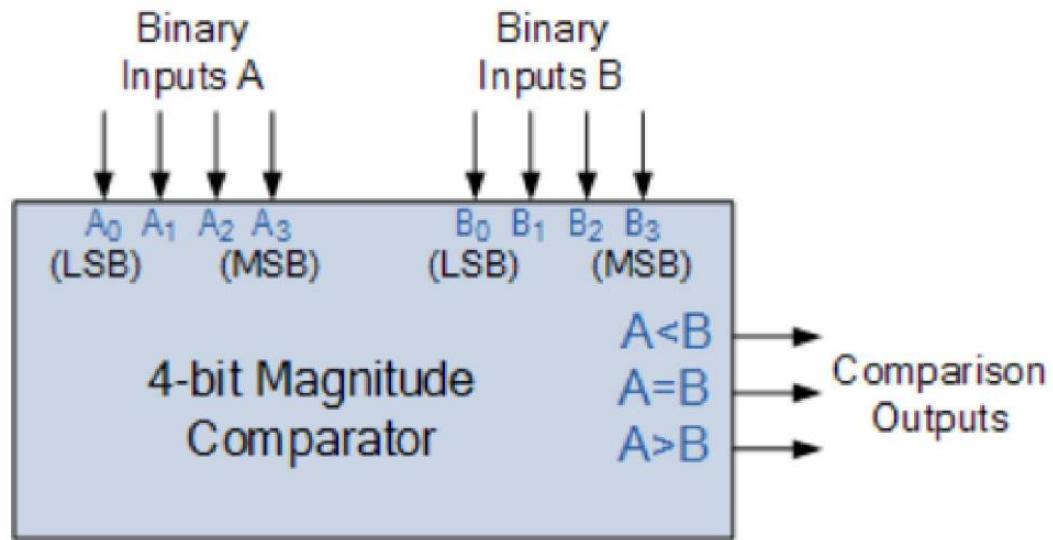
Test bench



Waveform



Comparator



Comparator Is basically used to compare the two inputs.

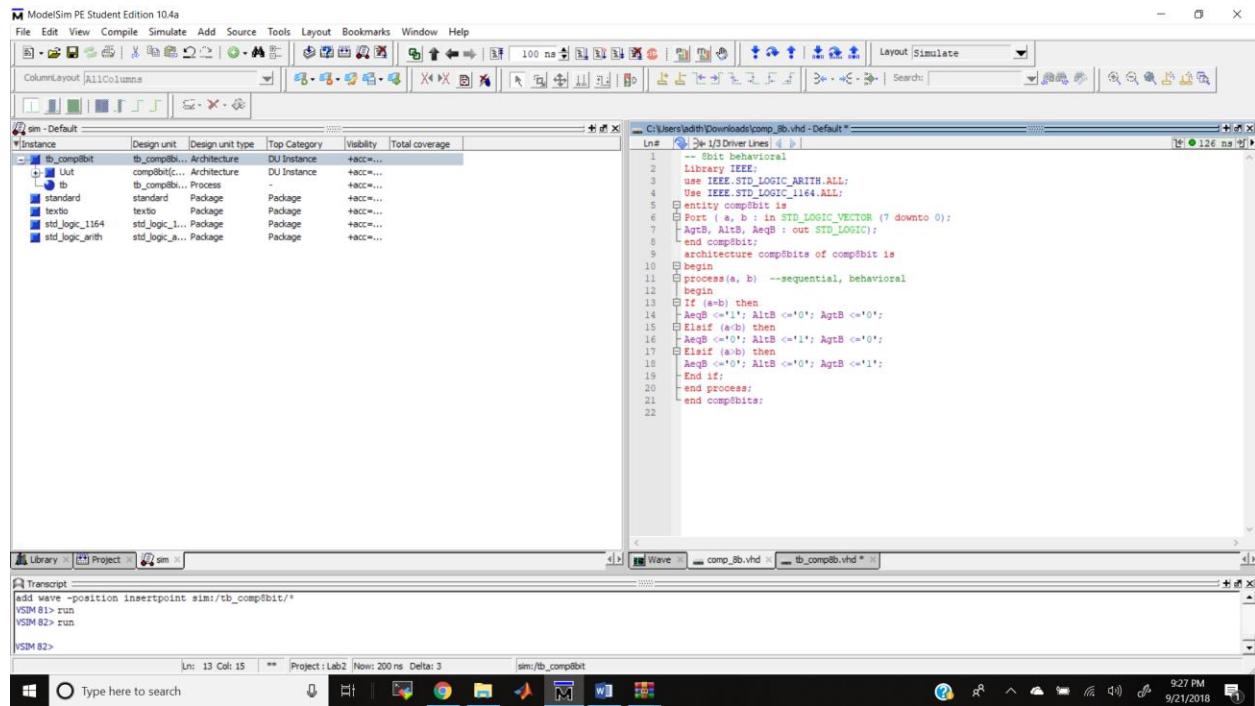
We have 3 inputs,

$a > b$, $a = b$, $a < b$. Depending on the inputs the value of the output will be high.

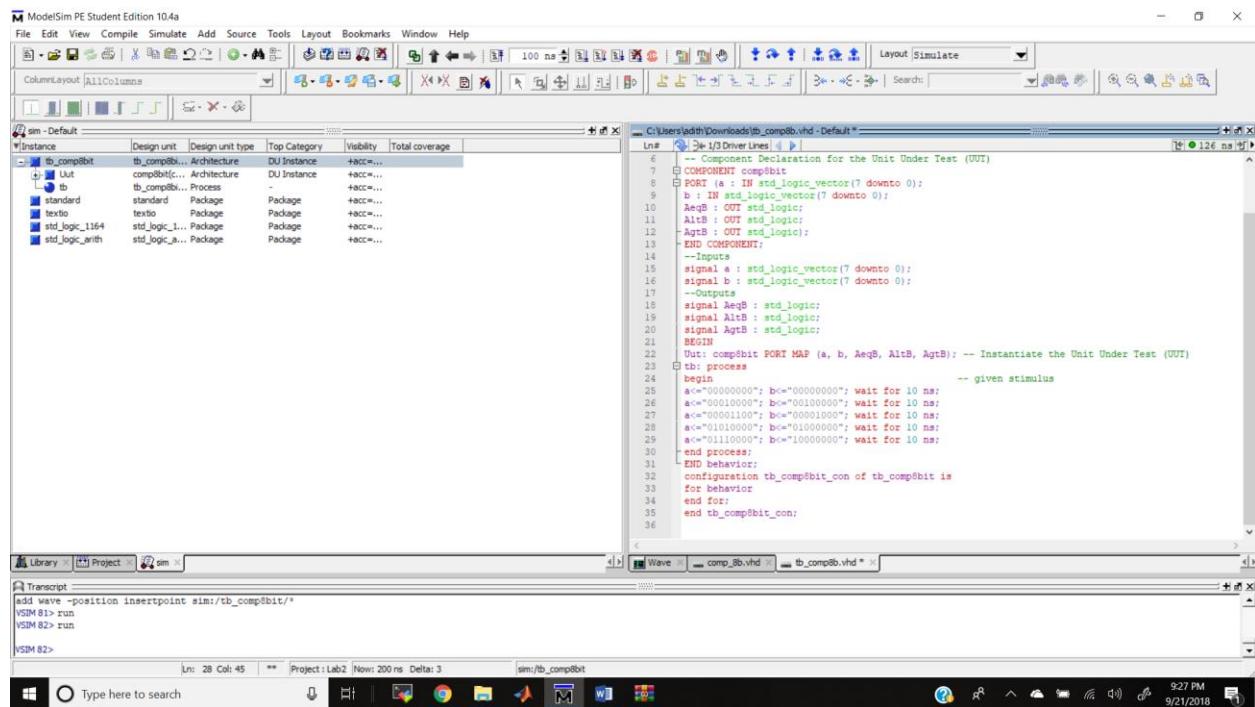
Inputs				Outputs		
A_1	A_0	B_1	B_0	$A > B$	$A = B$	$A < B$
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

- behavioral, sequential

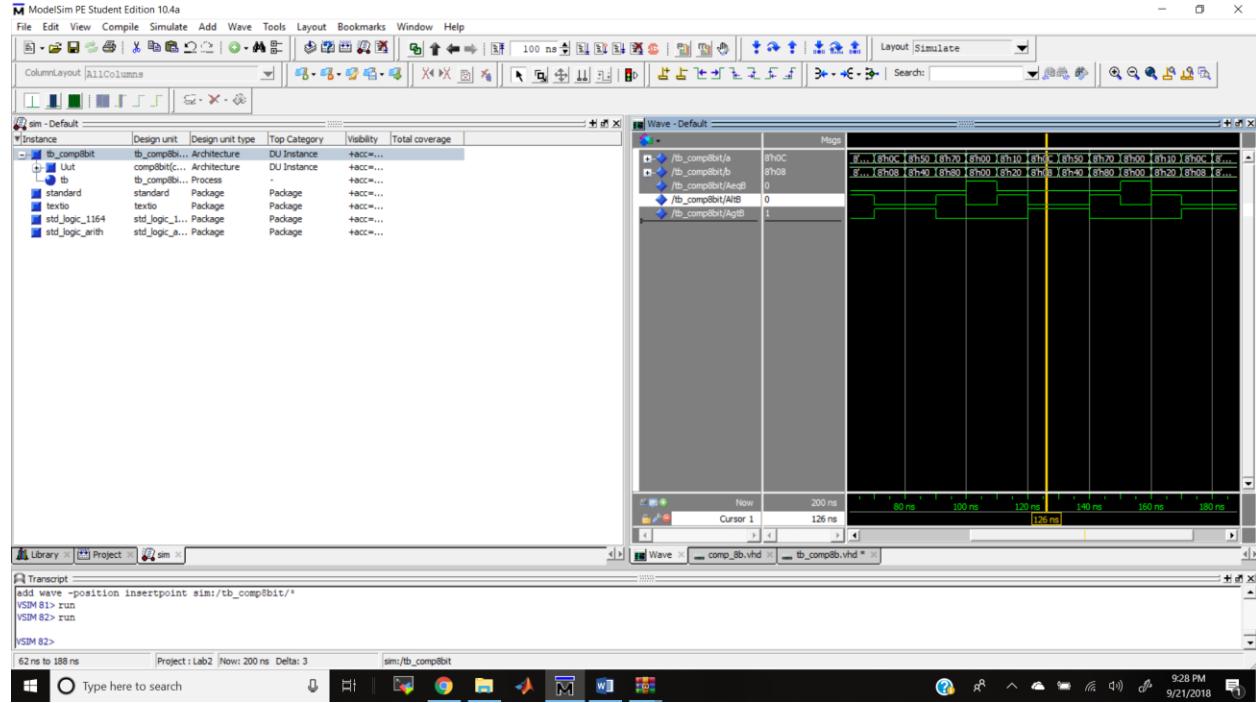
Here the process is used, which makes the lines to be executed one after the other or sequentially.



Testbench

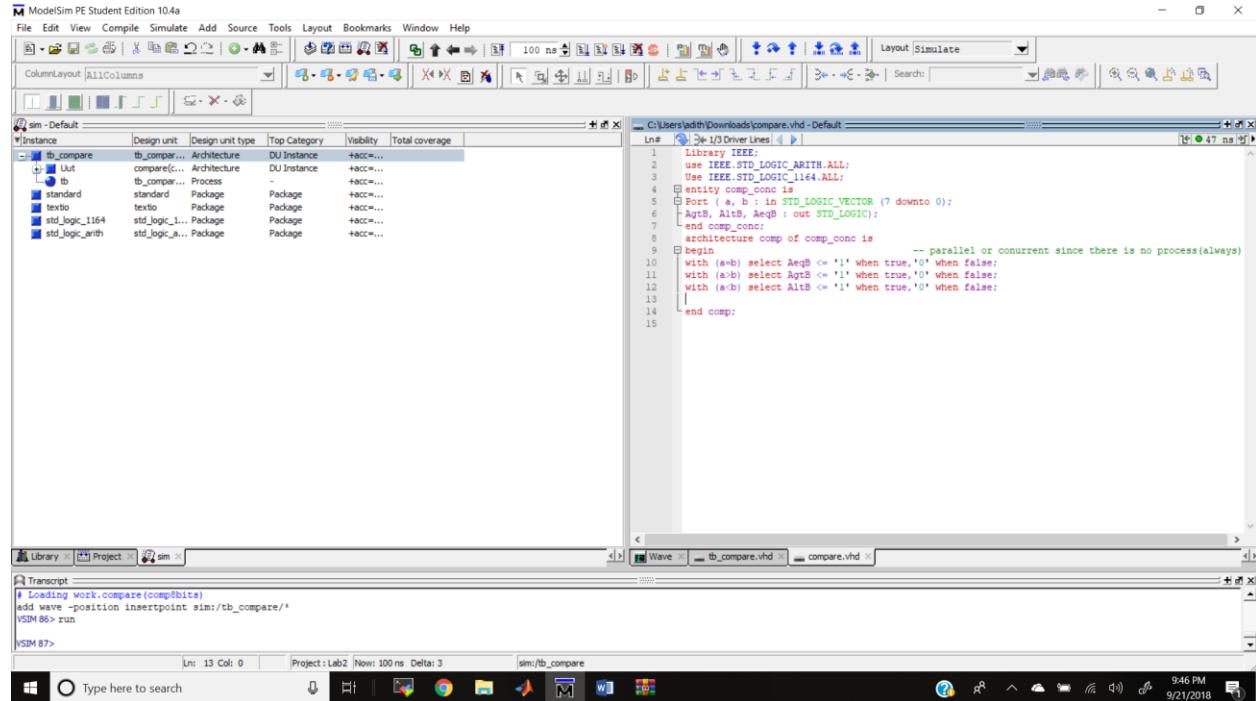


Waveform

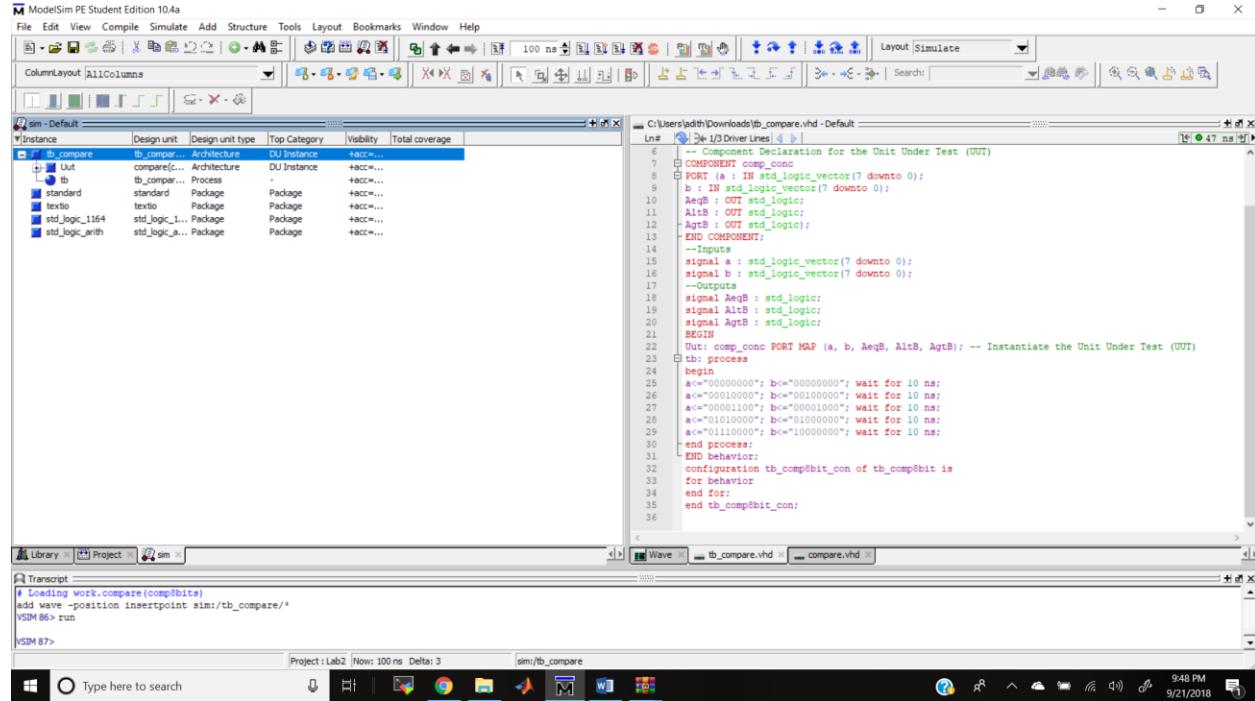


Comparator concurrent

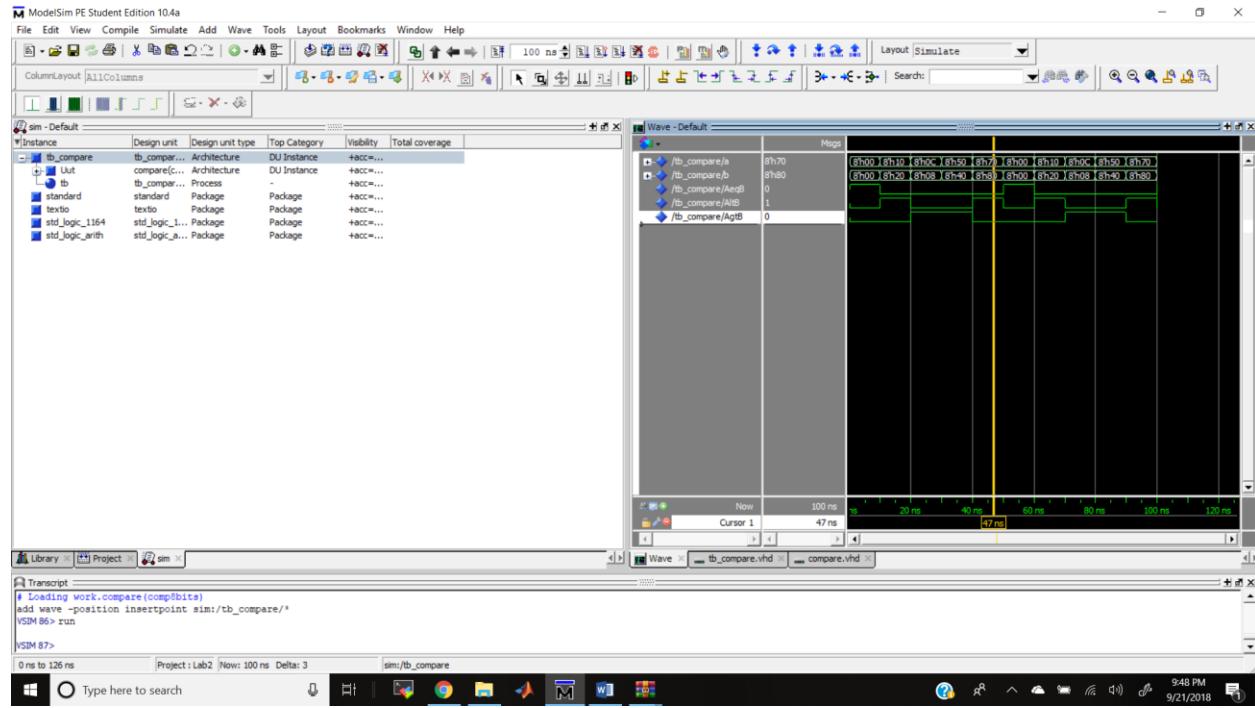
Here the process is not used, the when and select statements are used, which makes the codes to be executed in parallel or concurrently.



Test bench



Waveform



References,

Google images and other websites were used for description.