S. B. JAIN INSTITUTE OF TECHNOLOGY, MANAGEMENT & RESEARCH, NAGPUR.

Practical No. 01

Aim: Design and Simulation of all Logic gates and verify it using test bench

Name of Student	**: Aadesh Motghare**
Roll No.	**: 41(ET20065)**
Semester/Year	**: 6th Sem/3rd Year**
Academic Session	**: 2022-23**
Date of Performance	**:**
Date of Submission	**:**

AIM: Design and Simulation of all Logic gates and verify it usingtest bench

OBJECTIVE:

- To verify the functionality of all Logic gates.
- To design a digital system using Logic gates.

SOFTWARE: - Xilinx ISE14.7.

THEORY:

A logic gate is an elementary building block of a digital circuit. Most logic gates have two inputs and one output. At any given moment, every terminal is in one of the two binary conditions *low* (0) or *high* (1), represented by different voltage levels. The logic state of a terminal can, and generally does, change often, as the circuit processes data. In most logic gates, the low state is approximately zero volts (0 V), while the high state is approximately five volts positive (+5 V).

There are seven basic logic gates: AND, OR, XOR, NOT, NAND, NOR, and XNOR.

AND GATE

The *AND gate* is so named because, if 0 is called "false" and 1 is called "true," the gate acts in the same way as the logical "and" operator. The following illustration and table show the circuit symbol and logic combinations for an AND gate. (In the symbol, the input terminals are at left and the output terminal is at right.) The output is "true" when both inputs are "true." Otherwise, the output is "false."

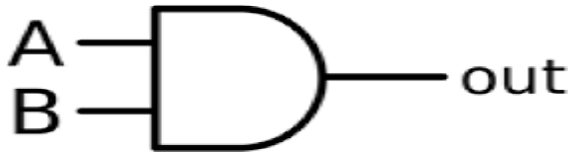


Fig 1. AND gate

A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

Table 1: Truth Table of AND Gate

OR GATE

The *OR gate* gets its name from the fact that it behaves after the fashion of the logical inclusive "or." The output is "true" if either or both of the inputs are "true." If both inputs are "false," then the output is "false."

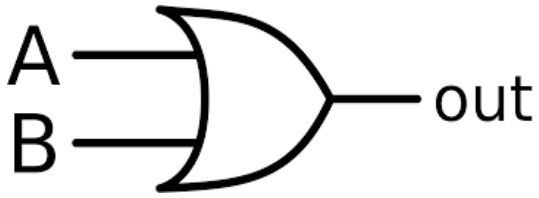


Fig 2. OR gate

A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

Table 2: Truth Table of OR Gate

EXCLUSIVE-*OR* GATE

The *XOR* (exclusive-*OR*) *gate* acts in the same way as the logical "either/or." The output is "true" if either, but not both, of the inputs are "true." The output is "false" if both inputs are "false" or if both inputs are "true." Another way of looking at this circuit is to observethat the output is 1 if the inputs are different, but 0 if the inputs are the same.

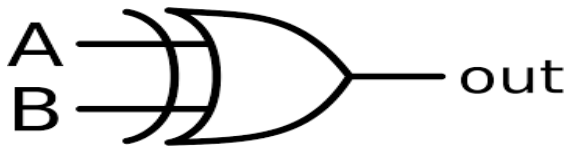


Fig 3. XOR gate

A	B	**Out**
0	0	0
0	1	1
1	0	1
1	1	0

Table 3: Truth Table of XOR Gate

NOT GATE

A logical *inverter*, sometimes called a *NOT gate* to differentiate it from other types ofelectronic inverter devices, has only one input. It reverses the logic state.

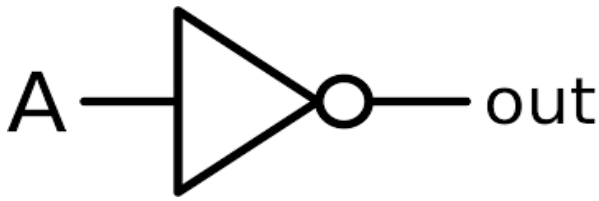


Fig 4. Inverter or NOT gate

A	Out
1	0
0	1

Table 4: Truth Table of NOT Gate

NAND GATE

The *NAND gate* operates as an AND gate followed by a NOT gate. It acts in the manner of the logical operation "and" followed by negation. The output is "false" if both inputs are "true." Otherwise, the output is "true."

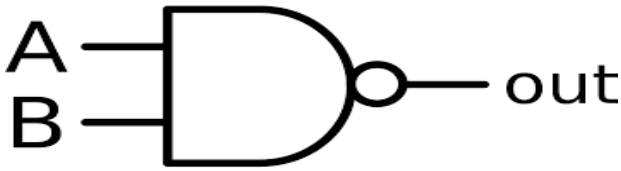


Fig 5. NAND gate

A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

Table 5: Truth Table of NAND Gate

NOR GATE

The *NOR gate* is a combination OR gate followed by an inverter. Its output is "true" ifboth inputs are "false." Otherwise, the output is "false."

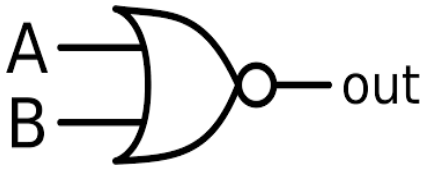


Fig 6. NOR gate

A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

Table 6: Truth Table of NOR Gate

XNOR GATE

The *XNOR (exclusive-NOR) gate* is a combination XOR gate followed by an inverter. Itsoutput is "true" if the inputs are the same and “false" if the inputs are different.

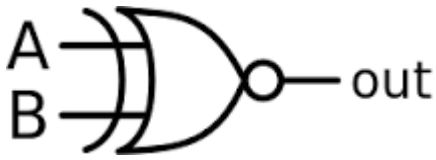


Fig 7. XNOR gate

A	B	Out
0	0	1
0	1	0
1	0	0
1	1	1

Table 7: Truth Table of XNOR Gate

Using combinations of logic gates, complex operations can be performed. In theory, thereis no limit to the number of gates that can be arrayed together in a single device. But in practice, there is a limit to the number of gates that can be packed into a given physical space. Arrays of logic gates are found in digital integrated circuits (ICs). As IC technology advances, the required physical volume for each individual logic gate decreases and digital devices of the same or smaller size become capable of performing ever-more-complicated operations at ever-increasing speeds.

VHDL CODE: -

STEPS FOR PROGRAM: -

Step 1. Library/Package Declaration: Involves declaration of all libraries and respective packages used in the design.

```
LIBRARY library_name;
USE library_name.package_name.all;
```

Step 2.Entity:

```
ENTITY entity_name is
        PORT(signal_name(s): mode signal_type;
                signal_name(s): mode signal_type;
                …);
end ENTITY entity_name;
```

Signals of the same mode and signal_type can be grouped on 1 line.

MODE describes the direction data is transferred through port

- in – data flows into the port
- out – data flows out of port *only*
- buffer – data flows out of port *as well as read*

internally.

- inout – bi-directional data flow into and out of port

SIGNAL_TYPE defines the data type for the signal(s)

- bit – single signals that can have logic values 0 and 1.
- bit_vector – bus signals(vector form of bit) that can have logic values 0 and 1.
- std_logic – part of std_logic_1164 package of IEEE library. Used to represent 2 value logical values i.e 0 and 1 as well as other values such as high impedance ,don't care and others as described below.
- std_logic_vector – bus signals (vector form of std_logic) but IEEE standard for simulation and synthesis note that all vectors must have a range specified example for a 4 bit bus: bit_vector (3 downto 0) or std_logic_vector (3 downto 0).

In order to use std_logic and std_logic_vector we must include the library and package usage declarations in the VHDL model before the entity statement as follows:

LIBRARY ieee;
USE ieee.std_logic_1164.all;

Values for std-logic:

U un-initialized (undefined logic value)
X forced unknown logic value
0 Logic low
1 Logic High
Z high impedance (tri-state)
W weak unknown
L weak 0
H weak 1
- don't care value (for synthesis minimization)

Step 3: Architecture Declaration

```
architecture architecture name of entity_name
 architecture_declarative_part;
begin
Statements;
 end architecture_name;
```

Here we should specify the entity name for which we are writing the architecture body. The architecture statements should be inside the begin and end keyword. Architecture declarative part may contain variables, constants, or component declaration.

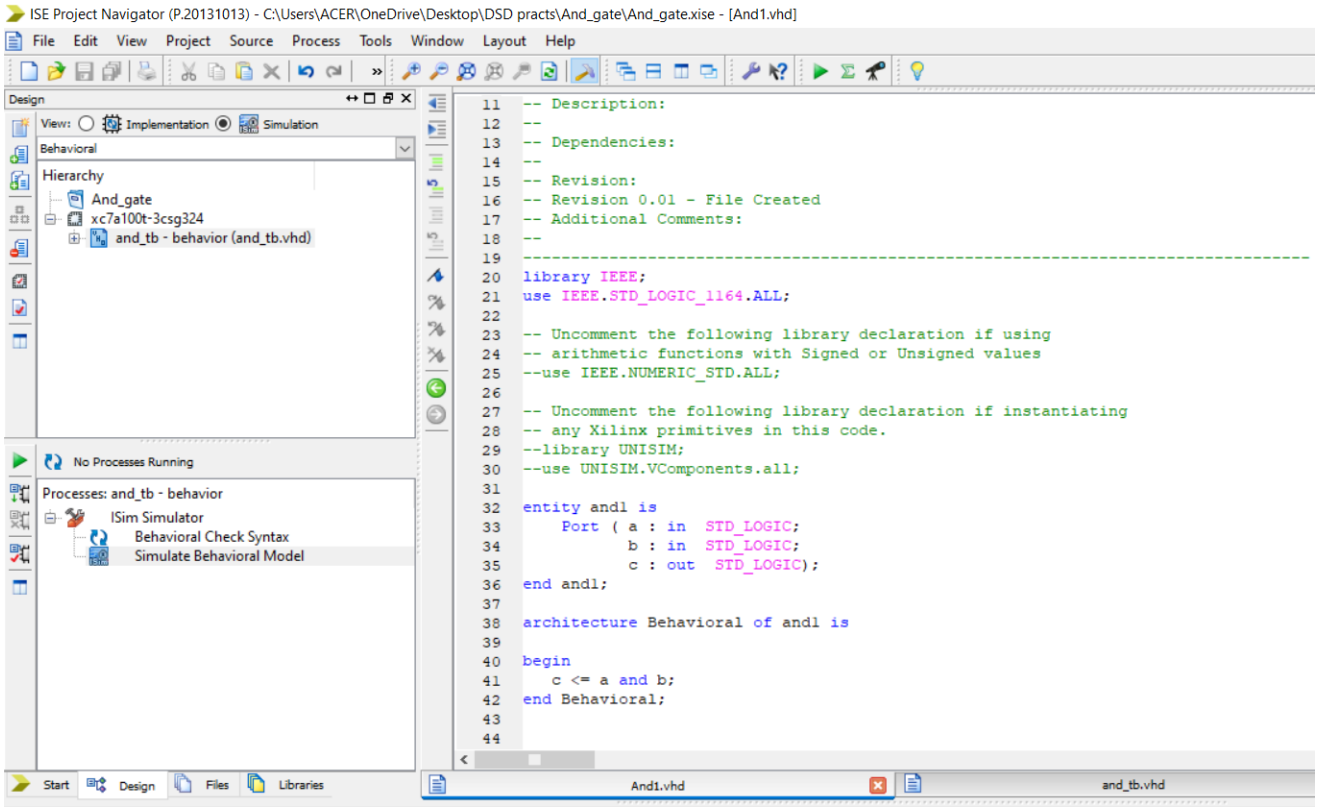
Step 4: Simulate the VHDL code and remove the syntax

errors if any.

Step 5: Write the testbench and verify the design .

CODE:

1. **AND Gate:**



TESTBENCH:

ISE Project Navigator (P.20131013) - C:\Users\ACER\OneDrive\Desktop\DSD practs\And_gate\And_gate.xise - [and_tb.vhd]

File Edit View Project Source Process Tools Window Layout Help

Design

View: Implementation Simulation

Behavioral

Hierarchy

And_gate

xc7a100t-3csg324

and_tb - behavior (and_tb.vhd)

No Processes Running

Processes: and_tb - behavior

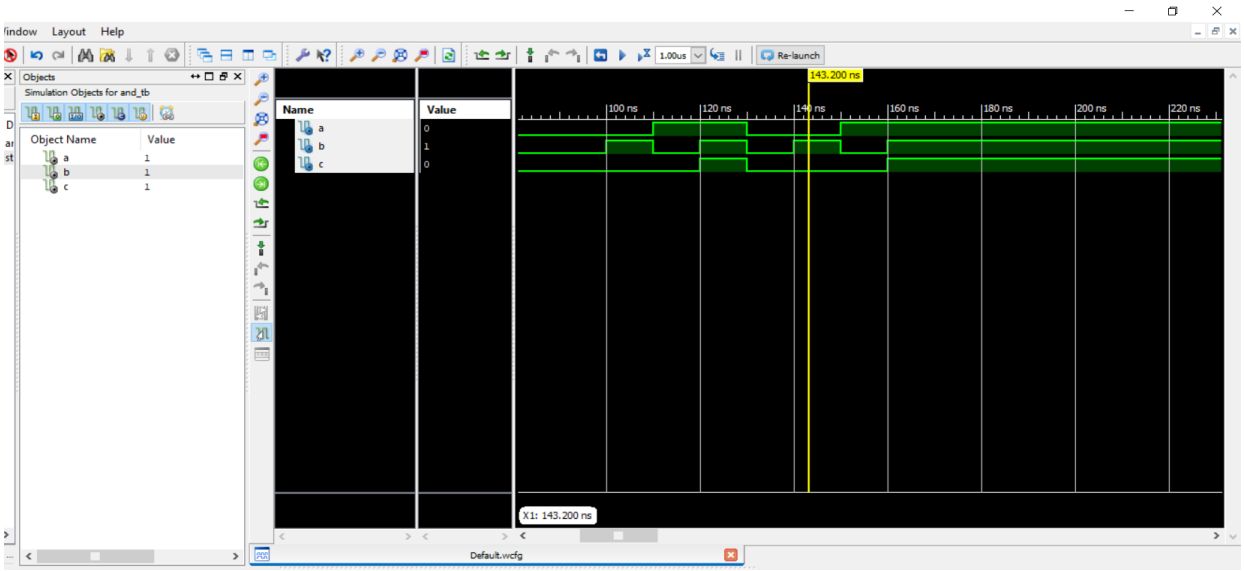
ISim Simulator

Behavioral Check Syntax

Simulate Behavioral Model

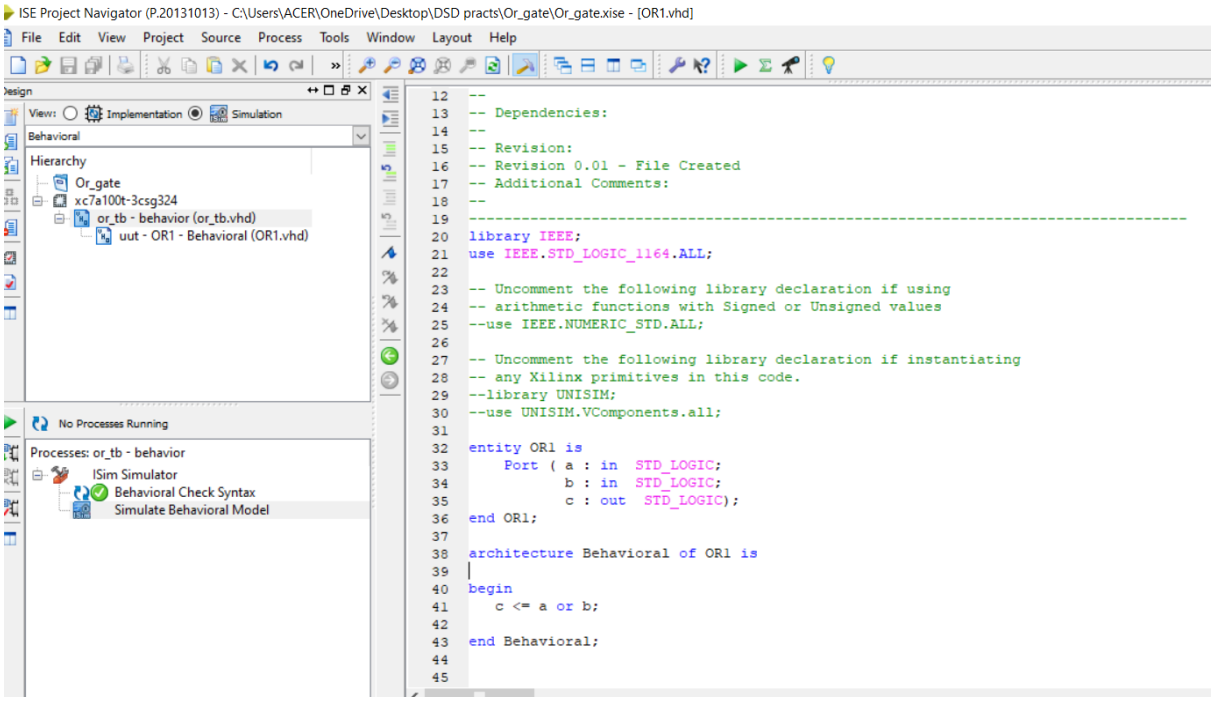
```vhdl
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

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

ENTITY and_tb IS
END and_tb;

ARCHITECTURE behavior OF and_tb IS

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

    COMPONENT and1
    PORT(
         a : IN  std_logic;
         b : IN  std_logic;
         c : OUT  std_logic
        );
    END COMPONENT;
   --Inputs
   signal a : std_logic := '0';
   signal b : std_logic := '0';

   --Outputs
   signal c : std_logic;
   -- No clocks detected in port list. Replace <clock> below with
   -- approp

BEGIN
   -- Instantiate the Unit Under Test (UUT)
   uut: and1 PORT MAP (
          a => a,
          b => b,
          c => c
        );
   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;
    -- insert stimulus here
      a <= '0';
      b <= '1';
      wait for 10 ns;

      a <= '1';
      b <= '0';
      wait for 10 ns;

      a <= '1';
      b <= '1';
      wait for 10 ns;

      a <= '0';
      b <= '0';
      wait for 10 ns;

      a <= '0';
      b <= '1';
      wait for 10 ns;

      a <= '1';
      b <= '0';
      wait for 10 ns;

      a <= '1';
      b <= '1';
      wait for 10 ns;
      wait;
   end process;
END;
```

WAVEFORM:

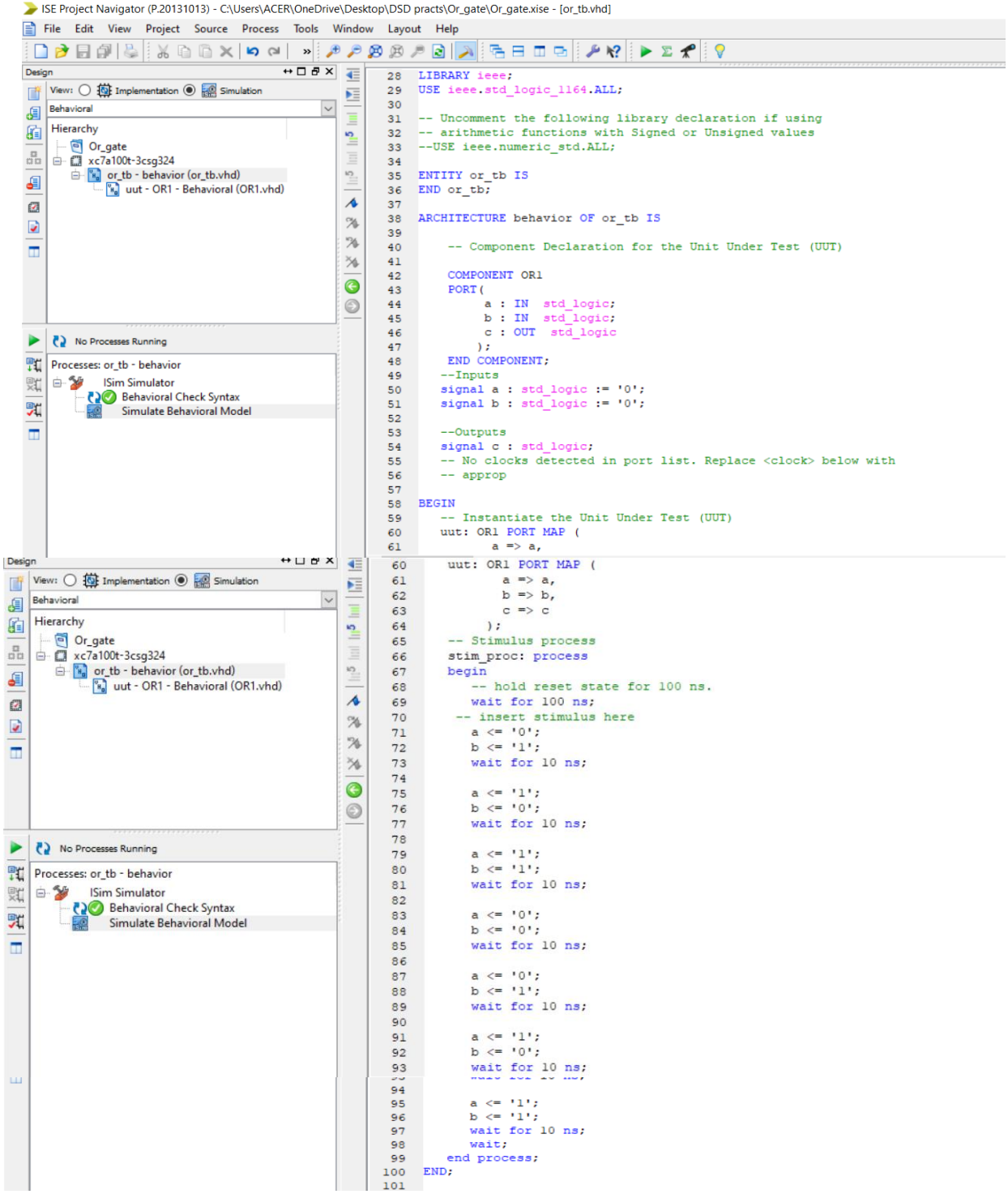


2. OR Gate:

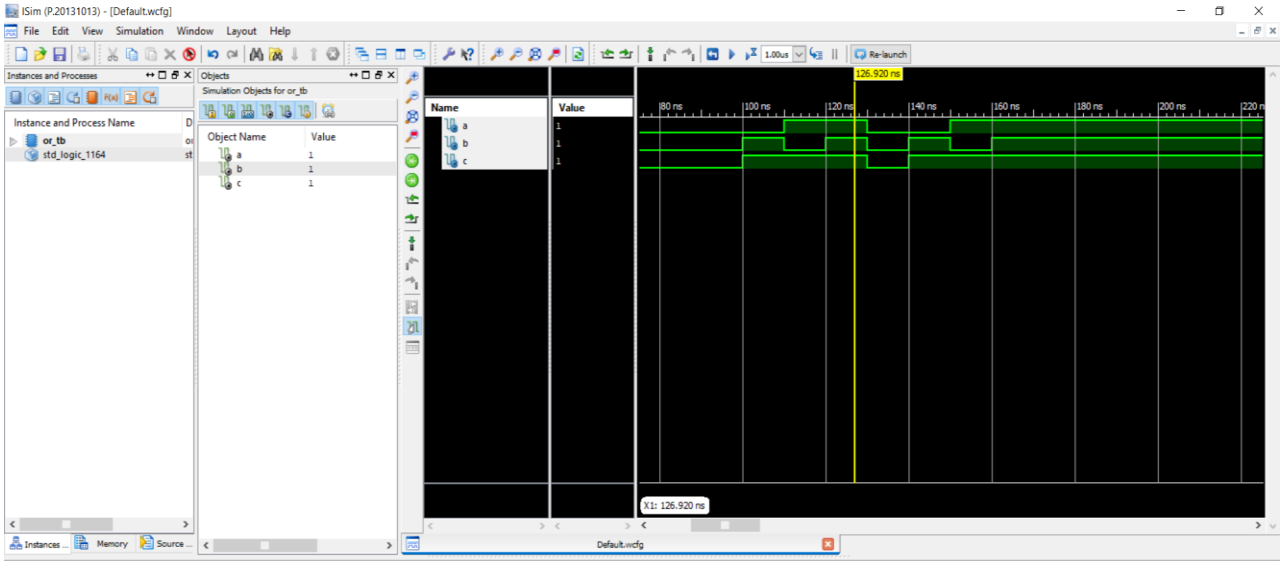
Code:



TESTBENCH:

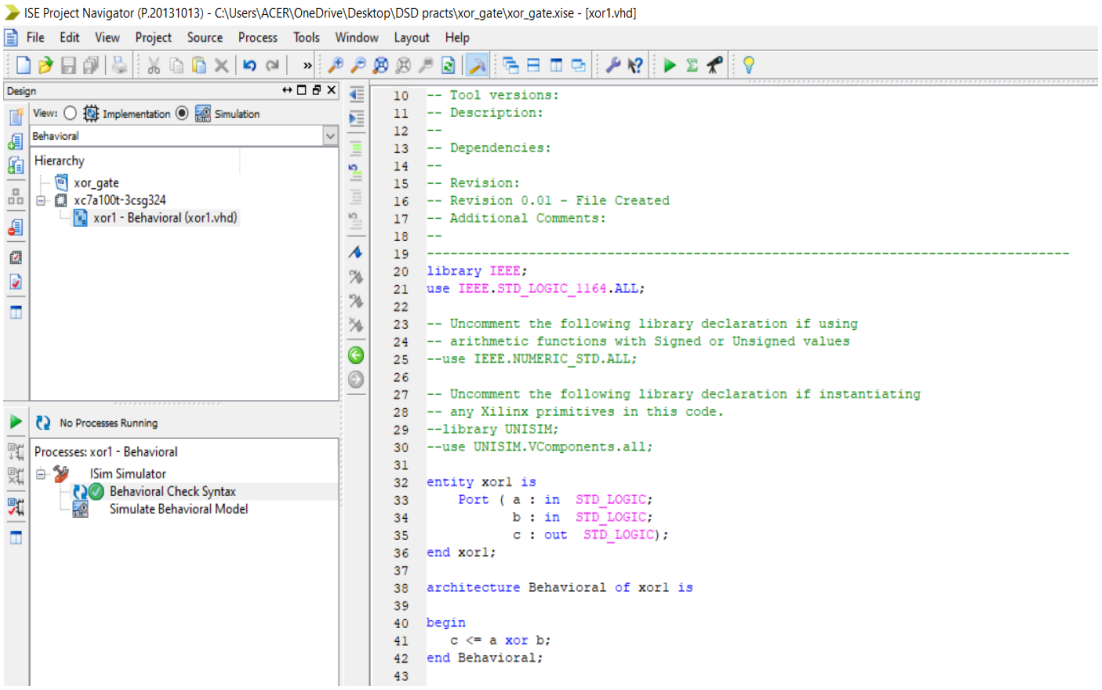

ISE Project Navigator (P.20131013) - C:\Users\ACER\OneDrive\Desktop\DSD practs\Or_gate\Or_gate.xise - [or_tb.vhd]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Behavioral
Hierarchy
Or_gate
xc7a100t-3csg324
or_tb - behavior (or_tb.vhd)
uut - OR1 - Behavioral (OR1.vhd)
No Processes Running
Processes: or_tb - behavior
ISim Simulator
Behavioral Check Syntax
Simulate Behavioral Model
LIBRARY ieee;
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 or_tb IS
END or_tb;
ARCHITECTURE behavior OF or_tb IS
-- Component Declaration for the Unit Under Test (UUT)
COMPONENT OR1
PORT(
a : IN std_logic;
b : IN std_logic;
c : OUT std_logic
);
END COMPONENT;
--Inputs
signal a : std_logic := '0';
signal b : std_logic := '0';
--Outputs
signal c : std_logic;
-- No clocks detected in port list. Replace <clock> below with
-- approp
BEGIN
-- Instantiate the Unit Under Test (UUT)
uut: OR1 PORT MAP (
a => a,
b => b,
c => c
);
-- Stimulus process
stim_proc: process
begin
-- hold reset state for 100 ns.
wait for 100 ns;
-- insert stimulus here
a <= '0';
b <= '1';
wait for 10 ns;
a <= '1';
b <= '0';
wait for 10 ns;
a <= '1';
b <= '1';
wait for 10 ns;
a <= '0';
b <= '0';
wait for 10 ns;
a <= '0';
b <= '1';
wait for 10 ns;
a <= '1';
b <= '0';
wait for 10 ns;
a <= '1';
b <= '1';
wait for 10 ns;
wait;
end process;
END;

OUTPUT:

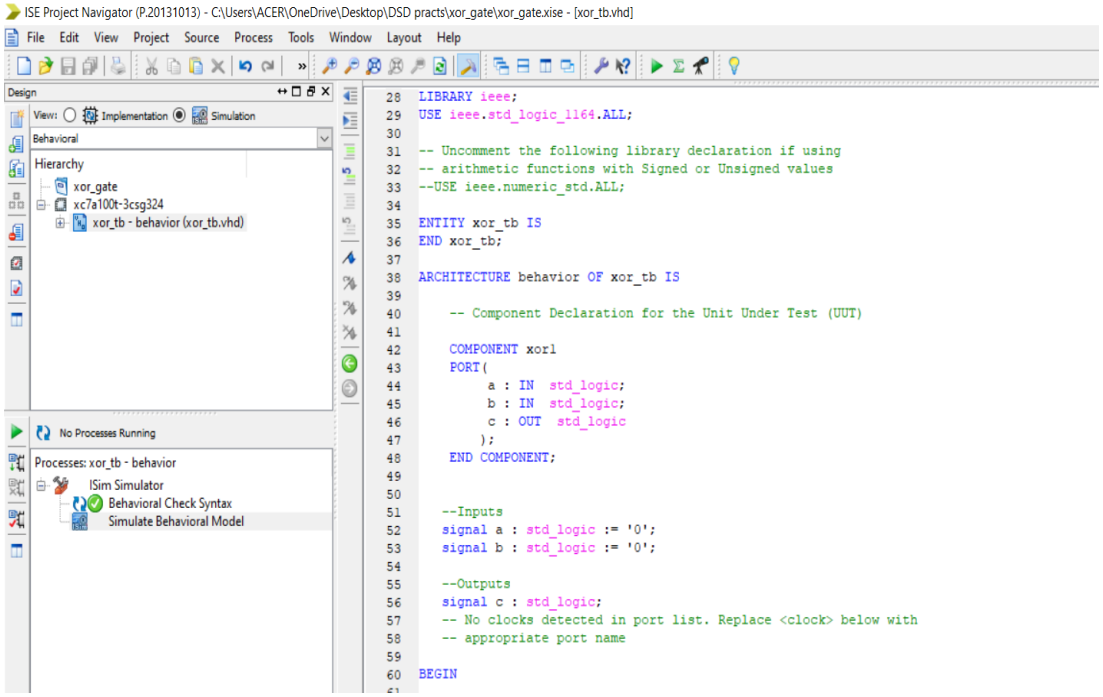


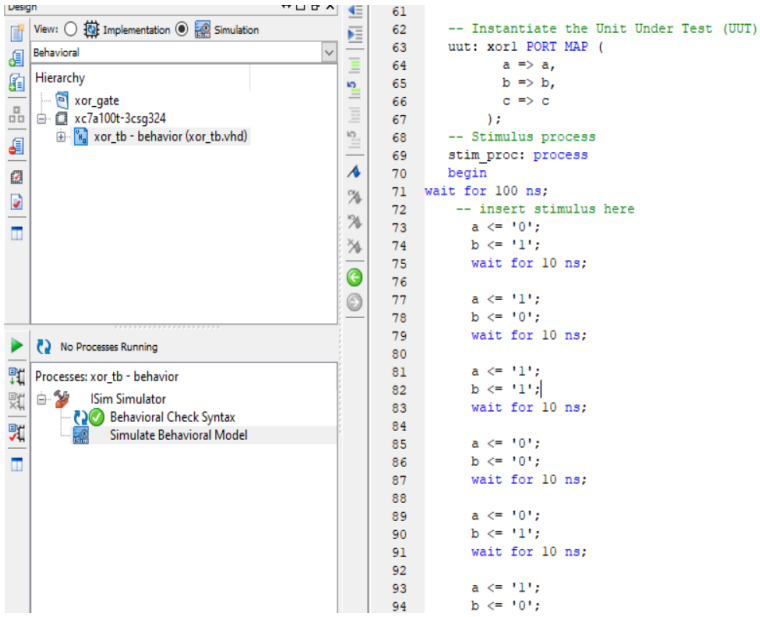
3. XOR Gate:

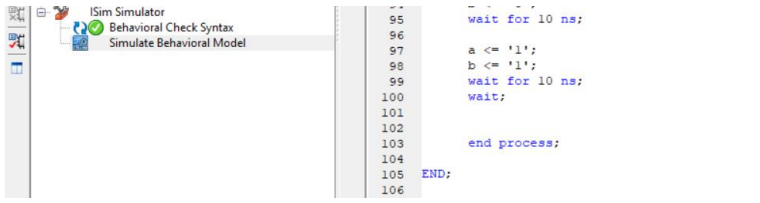
Code:



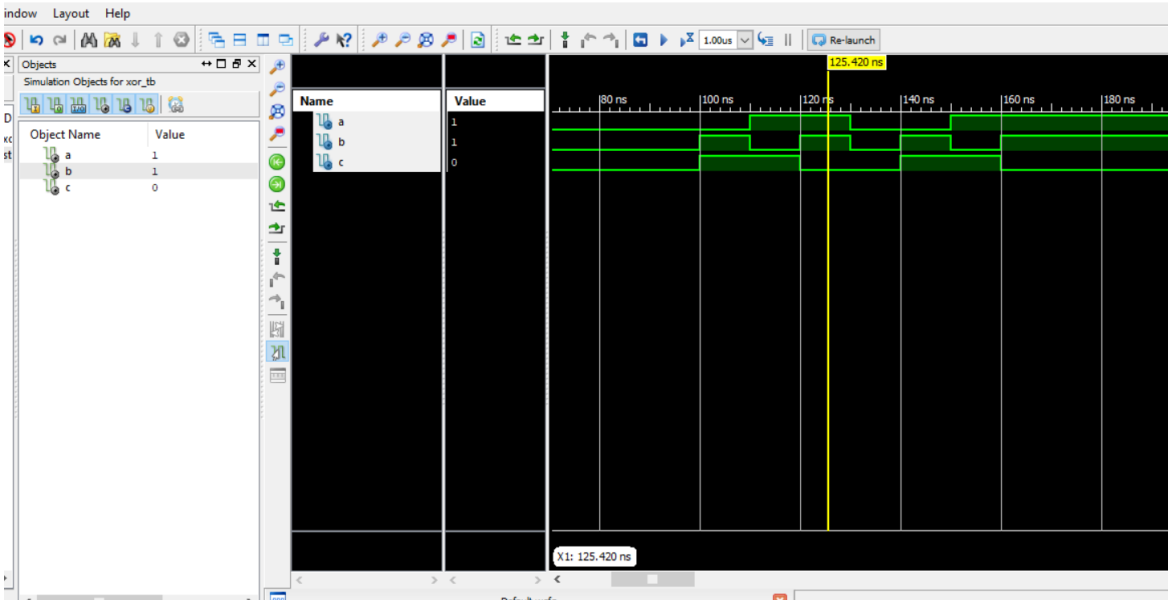
Testbench:


ISE Project Navigator (P.20131013) - C:\Users\ACER\OneDrive\Desktop\DSD practs\xor_gate\xor_gate.xise - [xor_tb.vhd]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Behavioral
Hierarchy
xor_gate
xc7a100t-3csg324
xor_tb - behavior (xor_tb.vhd)
No Processes Running
Processes: xor_tb - behavior
ISim Simulator
Behavioral Check Syntax
Simulate Behavioral Model
LIBRARY ieee;
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 xor_tb IS
END xor_tb;
ARCHITECTURE behavior OF xor_tb IS
-- Component Declaration for the Unit Under Test (UUT)
COMPONENT xor1
PORT(
a : IN std_logic;
b : IN std_logic;
c : OUT std_logic
);
END COMPONENT;
--Inputs
signal a : std_logic := '0';
signal b : std_logic := '0';
--Outputs
signal c : std_logic;
-- No clocks detected in port list. Replace <clock> below with
-- appropriate port name
BEGIN


Design
View: Implementation Simulation
Behavioral
Hierarchy
xor_gate
xc7a100t-3csg324
xor_tb - behavior (xor_tb.vhd)
No Processes Running
Processes: xor_tb - behavior
ISim Simulator
Behavioral Check Syntax
Simulate Behavioral Model
-- Instantiate the Unit Under Test (UUT)
uut: xor1 PORT MAP (
a => a,
b => b,
c => c
);
-- Stimulus process
stim_proc: process
begin
wait for 100 ns;
-- insert stimulus here
a <= '0';
b <= '1';
wait for 10 ns;
a <= '1';
b <= '0';
wait for 10 ns;
a <= '1';
b <= '1';
wait for 10 ns;
a <= '0';
b <= '0';
wait for 10 ns;
a <= '0';
b <= '1';
wait for 10 ns;
a <= '1';
b <= '0';

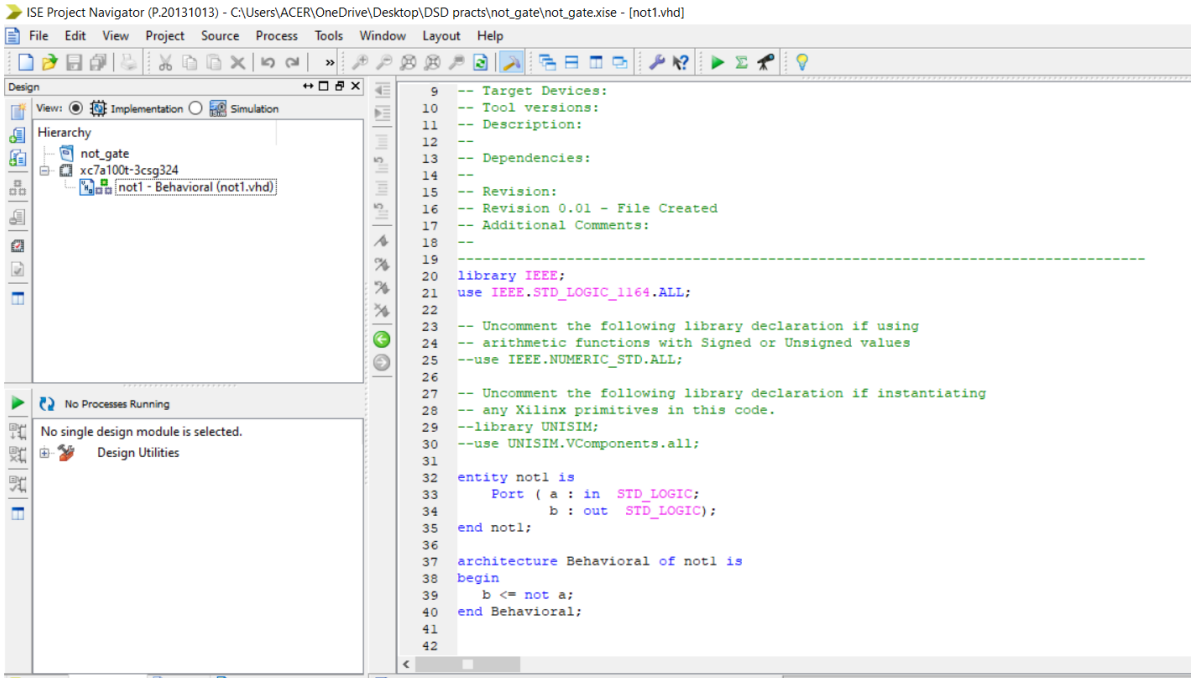

ISim Simulator
Behavioral Check Syntax
Simulate Behavioral Model
wait for 10 ns;
a <= '1';
b <= '1';
wait for 10 ns;
wait;
end process;
END;

Output:

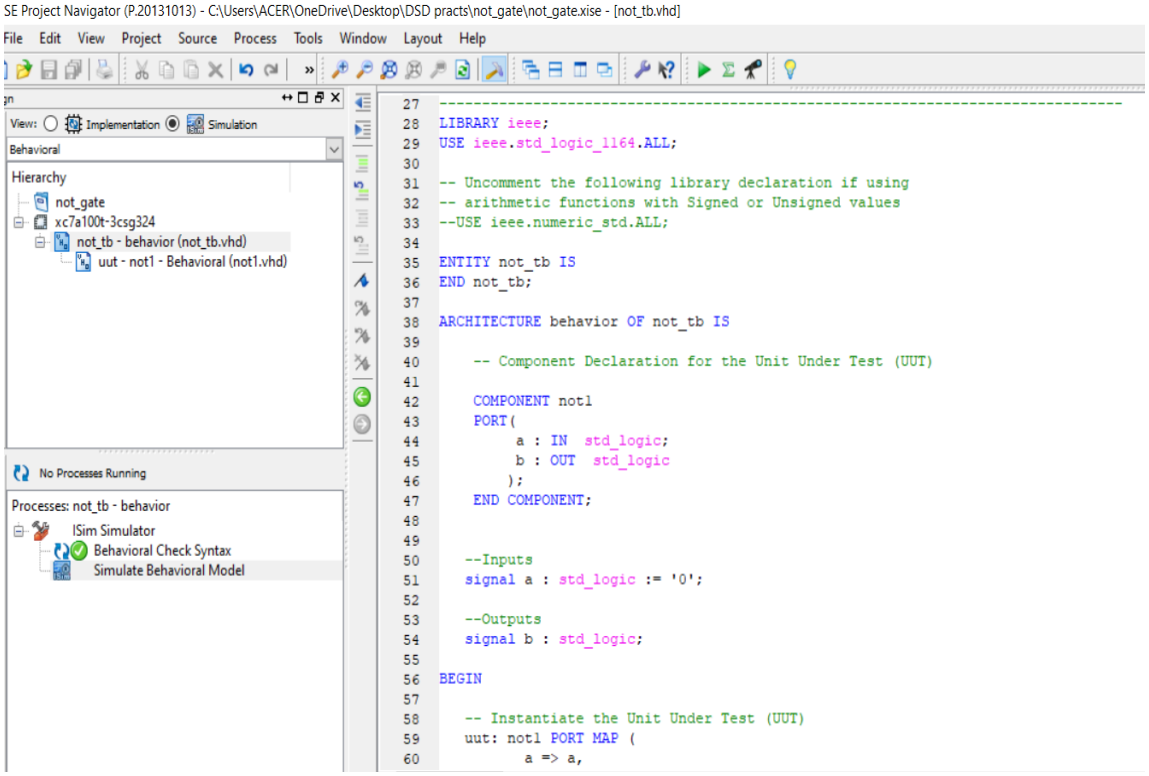


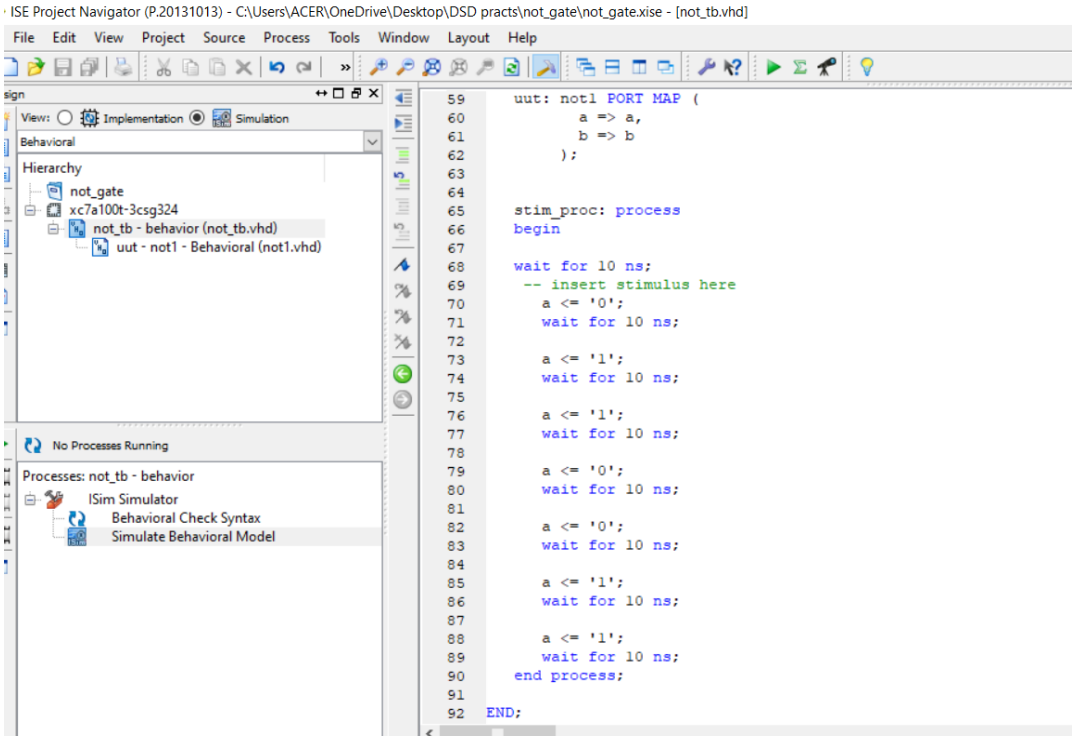
4. NOT Gate:

Code:

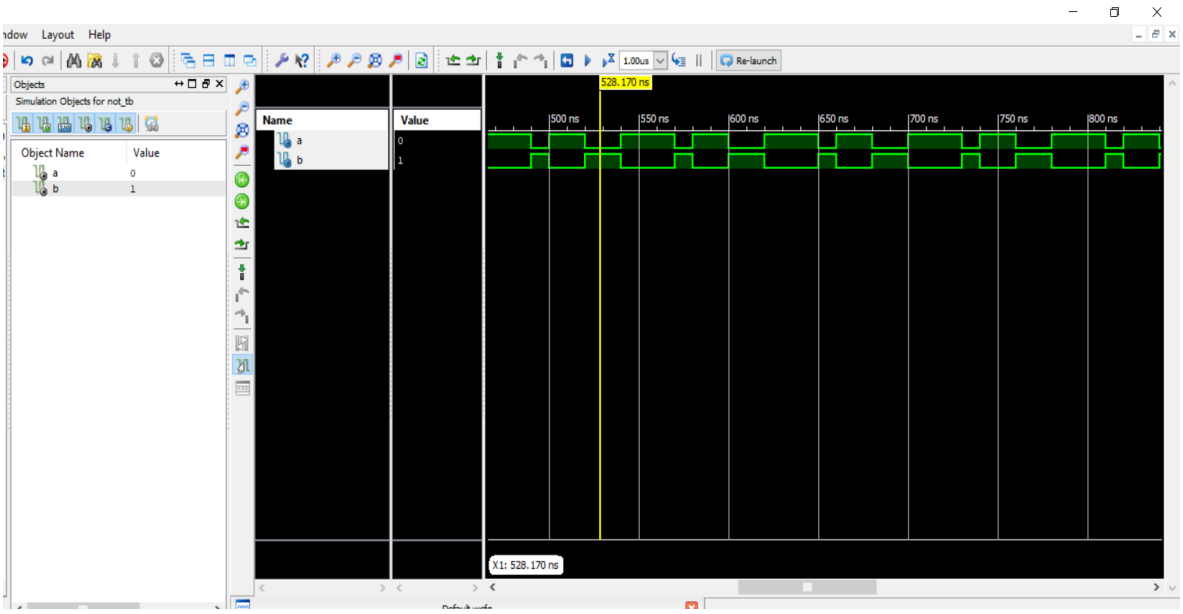


Testbench:


SE Project Navigator (P.20131013) - C:\Users\ACER\OneDrive\Desktop\DSD practs\not_gate\not_gate.xise - [not_tb.vhd]
File Edit View Project Source Process Tools Window Layout Help
View: Implementation Simulation
Behavioral
Hierarchy
not_gate
xc7a100t-3csg324
not_tb - behavior (not_tb.vhd)
uut - not1 - Behavioral (not1.vhd)
No Processes Running
Processes: not_tb - behavior
ISim Simulator
Behavioral Check Syntax
Simulate Behavioral Model
LIBRARY ieee;
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 not_tb IS
END not_tb;
ARCHITECTURE behavior OF not_tb IS
-- Component Declaration for the Unit Under Test (UUT)
COMPONENT not1
PORT(
a : IN std_logic;
b : OUT std_logic
);
END COMPONENT;
--Inputs
signal a : std_logic := '0';
--Outputs
signal b : std_logic;
BEGIN
-- Instantiate the Unit Under Test (UUT)
uut: not1 PORT MAP (
a => a,

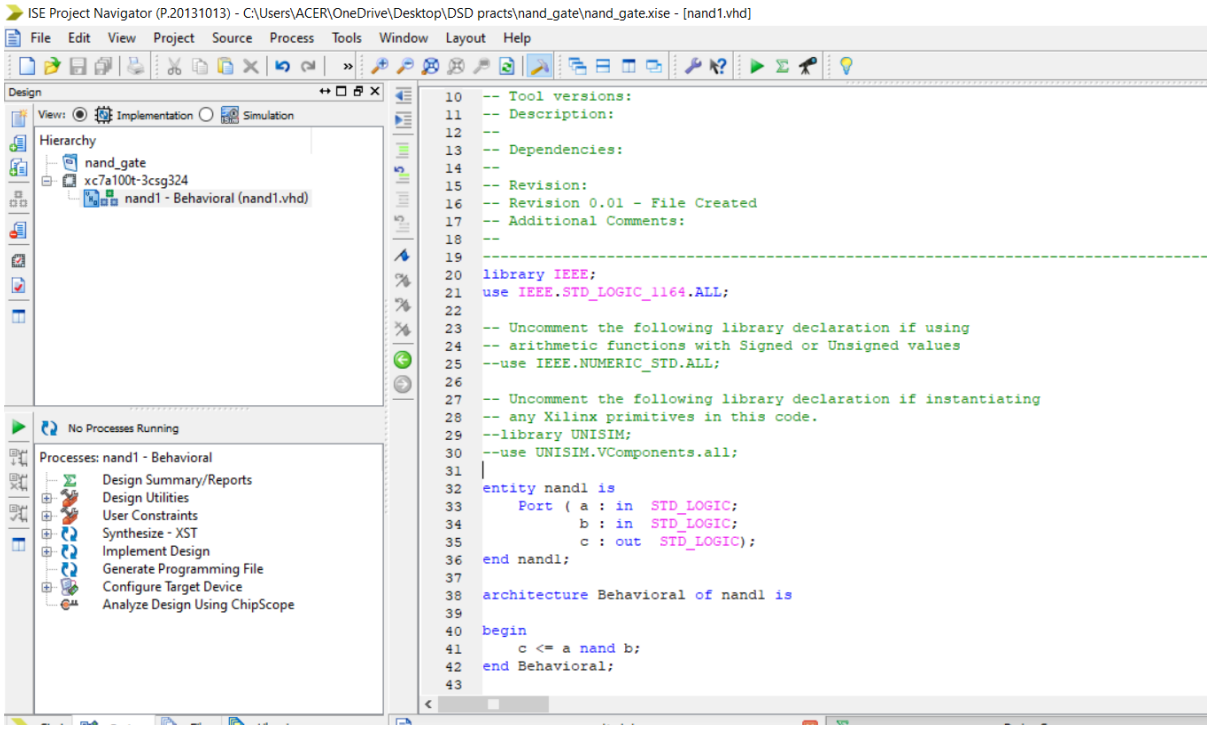

ISE Project Navigator (P.20131013) - C:\Users\ACER\OneDrive\Desktop\DSD practs\not_gate\not_gate.xise - [not_tb.vhd]
File Edit View Project Source Process Tools Window Layout Help
View: Implementation Simulation
Behavioral
Hierarchy
not_gate
xc7a100t-3csg324
not_tb - behavior (not_tb.vhd)
uut - not1 - Behavioral (not1.vhd)
No Processes Running
Processes: not_tb - behavior
ISim Simulator
Behavioral Check Syntax
Simulate Behavioral Model
uut: not1 PORT MAP (
a => a,
b => b
);
stim_proc: process
begin
wait for 10 ns;
-- insert stimulus here
a <= '0';
wait for 10 ns;
a <= '1';
wait for 10 ns;
a <= '1';
wait for 10 ns;
a <= '0';
wait for 10 ns;
a <= '0';
wait for 10 ns;
a <= '1';
wait for 10 ns;
a <= '1';
wait for 10 ns;
end process;
END;

Output:

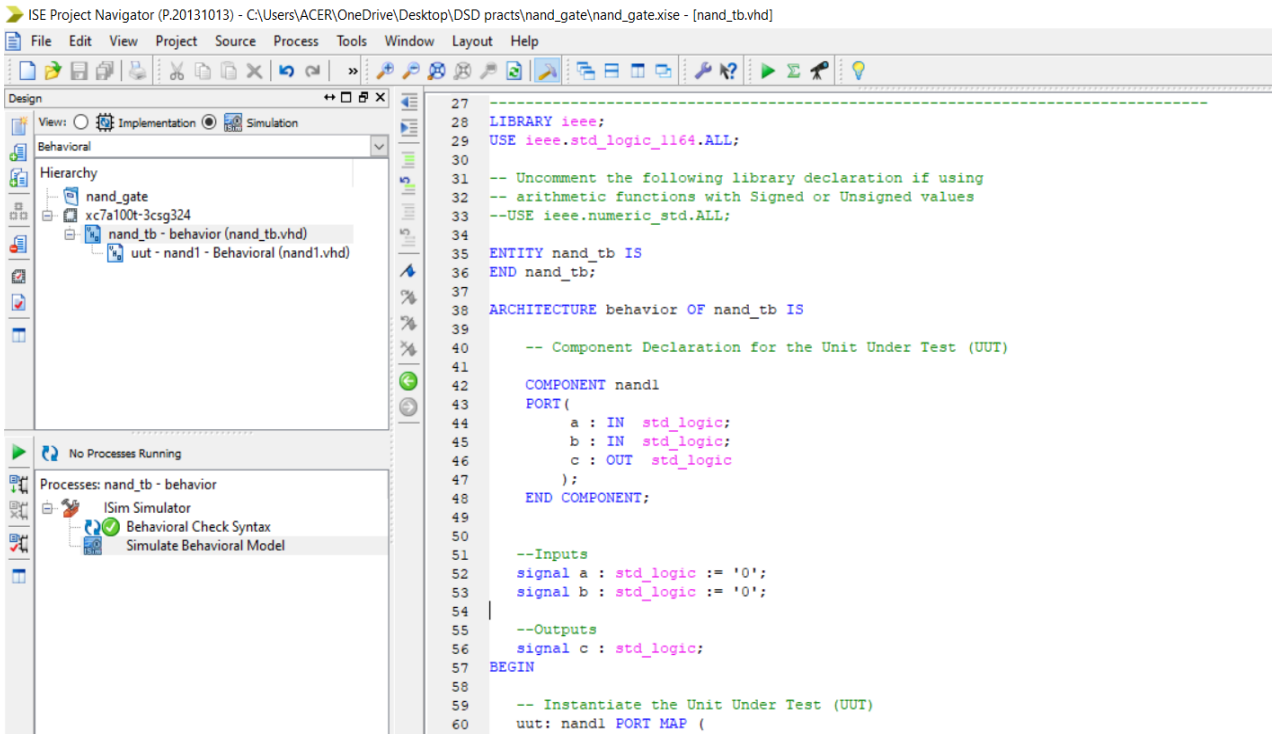


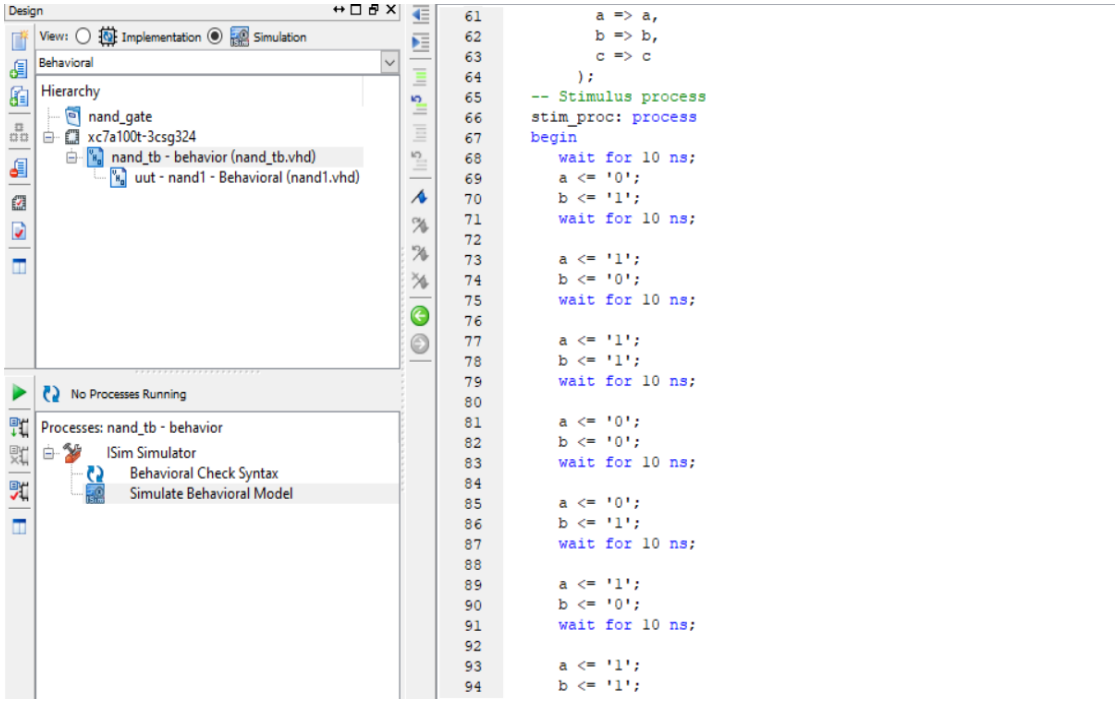
5. NAND Gate:

Code:



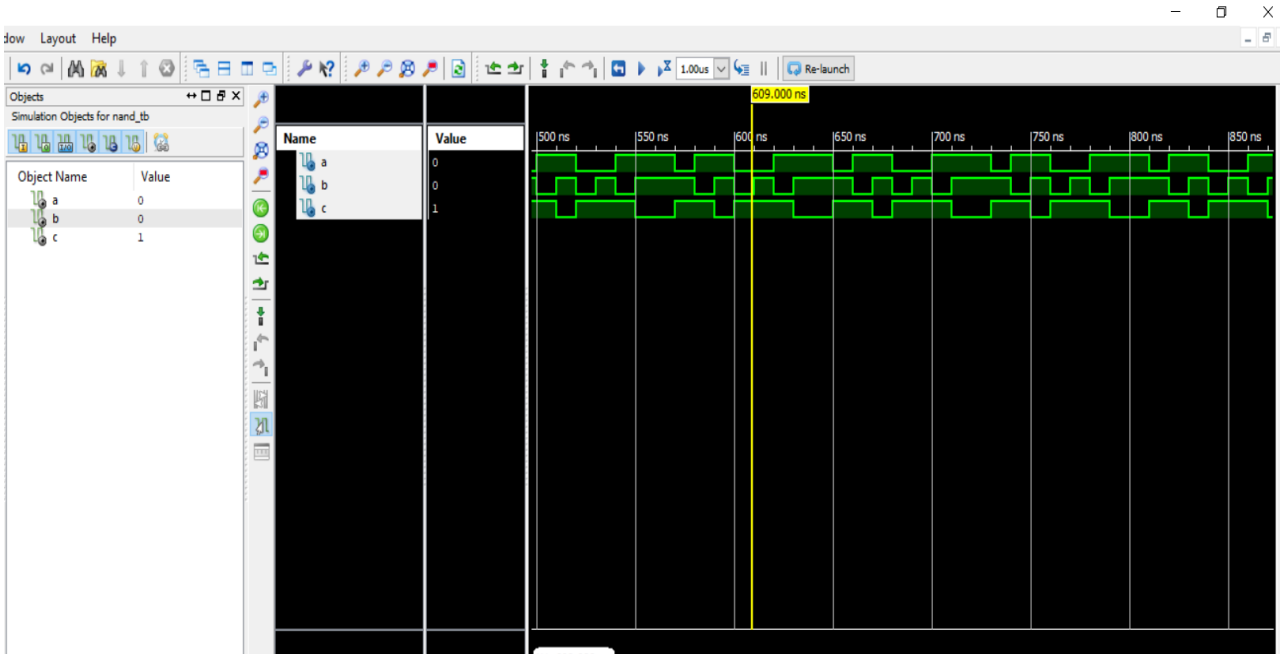
Testbench:


ISE Project Navigator (P.20131013) - C:\Users\ACER\OneDrive\Desktop\DSD practs\nand_gate\nand_gate.xise - [nand_tb.vhd]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Behavioral
Hierarchy
nand_gate
xc7a100t-3csg324
nand_tb - behavior (nand_tb.vhd)
uut - nand1 - Behavioral (nand1.vhd)
No Processes Running
Processes: nand_tb - behavior
ISim Simulator
Behavioral Check Syntax
Simulate Behavioral Model
27 ----------
28 LIBRARY ieee;
29 USE ieee.std_logic_1164.ALL;
30
31 -- Uncomment the following library declaration if using
32 -- arithmetic functions with Signed or Unsigned values
33 --USE ieee.numeric_std.ALL;
34
35 ENTITY nand_tb IS
36 END nand_tb;
37
38 ARCHITECTURE behavior OF nand_tb IS
39
40 -- Component Declaration for the Unit Under Test (UUT)
41
42 COMPONENT nand1
43 PORT(
44 a : IN std_logic;
45 b : IN std_logic;
46 c : OUT std_logic
47);
48 END COMPONENT;
49
50
51 --Inputs
52 signal a : std_logic := '0';
53 signal b : std_logic := '0';
54
55 --Outputs
56 signal c : std_logic;
57 BEGIN
58
59 -- Instantiate the Unit Under Test (UUT)
60 uut: nand1 PORT MAP (


Design
View: Implementation Simulation
Behavioral
Hierarchy
nand_gate
xc7a100t-3csg324
nand_tb - behavior (nand_tb.vhd)
uut - nand1 - Behavioral (nand1.vhd)
No Processes Running
Processes: nand_tb - behavior
ISim Simulator
Behavioral Check Syntax
Simulate Behavioral Model
61 a => a,
62 b => b,
63 c => c
64);
65 -- Stimulus process
66 stim_proc: process
67 begin
68 wait for 10 ns;
69 a <= '0';
70 b <= '1';
71 wait for 10 ns;
72
73 a <= '1';
74 b <= '0';
75 wait for 10 ns;
76
77 a <= '1';
78 b <= '1';
79 wait for 10 ns;
80
81 a <= '0';
82 b <= '0';
83 wait for 10 ns;
84
85 a <= '0';
86 b <= '1';
87 wait for 10 ns;
88
89 a <= '1';
90 b <= '0';
91 wait for 10 ns;
92
93 a <= '1';
94 b <= '1';


95 wait for 10 ns;
96
97
98
99 end process;
100
101 END;
102

Output:



RESULT:-

The VHDL code for all Logic gates are executed and desired output is obtained.

CONCLUSION:

Here I successfully write the codes for Logic Gates and Test Bench for the same.

DISCUSSION & VIVA VOCE

1) Explain the modeling style used to design all Logic gates.
2) What are the applications of all Logic gates.
3) How to design basic gates using Universal gates.

REFERENCE:

- VHDL Primer–J Bhasker –Pearson Education
- NPTEL Video Lecture link- https://www.youtube.com/watch?v=sUutDs7FFeA