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

Practical No. 02

Aim: Design and simulate the VHDL code for Full Adder,2:4 Decoder and 4:1 MUX.

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	**:**

2. AIM: Design and simulate the VHDL code for Full Adder,2:4 Decoder and 4:1 MUX.

OBJECTIVE:

- To verify the functionality of full adder circuit.
- To verify the functionality of 4:1 multiplexer.
- To verify the functionality of 2:4 Decoder

SOFTWARE: - Xilinx ISE14.7.

2.aFull Adder

THEORY:-

A combinational circuit which adds three input bits and produces two output bits is called full adder. The three input bits include two significant bits and a previous carry bit and 2 output bits are sum and carry. A full adder circuit can be implemented with two half adders and one OR gate.

Sum=A xor B xor Cin

Cout=A.B +A.Cin+ B.Cin

OR

Cout=AB+Cin(A⊕B)

BLOCK DIAGRAM

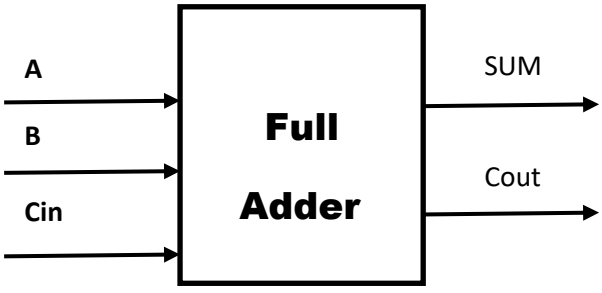


Fig 1. Full Adder

LOGIC DIAGRAM

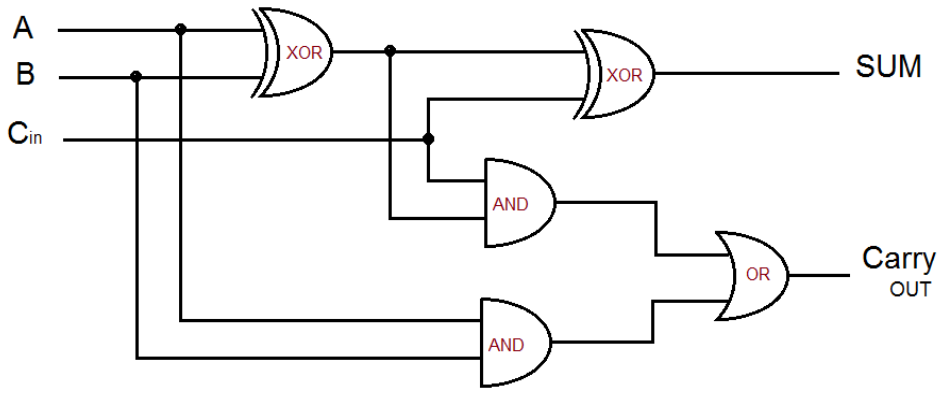


Fig 2.LogicDiagram of Full Adder

TRUTH TABLE

A	B	C_{in}	Sum	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 1: Truth Table of Full Adder

2.b 2 x 4 DECODER

THEORY

A decoder is a combinational logic circuit that converts binary information from 'n' input lines to a maximum of 2^n unique output lines. If the n-bit coded information has unused combinations, the decoder may have fewer than 2^n outputs. A 2 x 4 decoder has 2 inputs and 4 output lines. As can be observed from truth table,depending on the combination of inputs only one output gets activated.

BLOCK DIAGRAM:

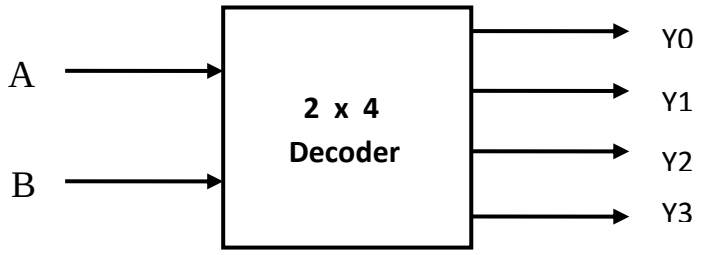


FIGURE 3 2x4 DECODER

LOGIC DIAGRAM

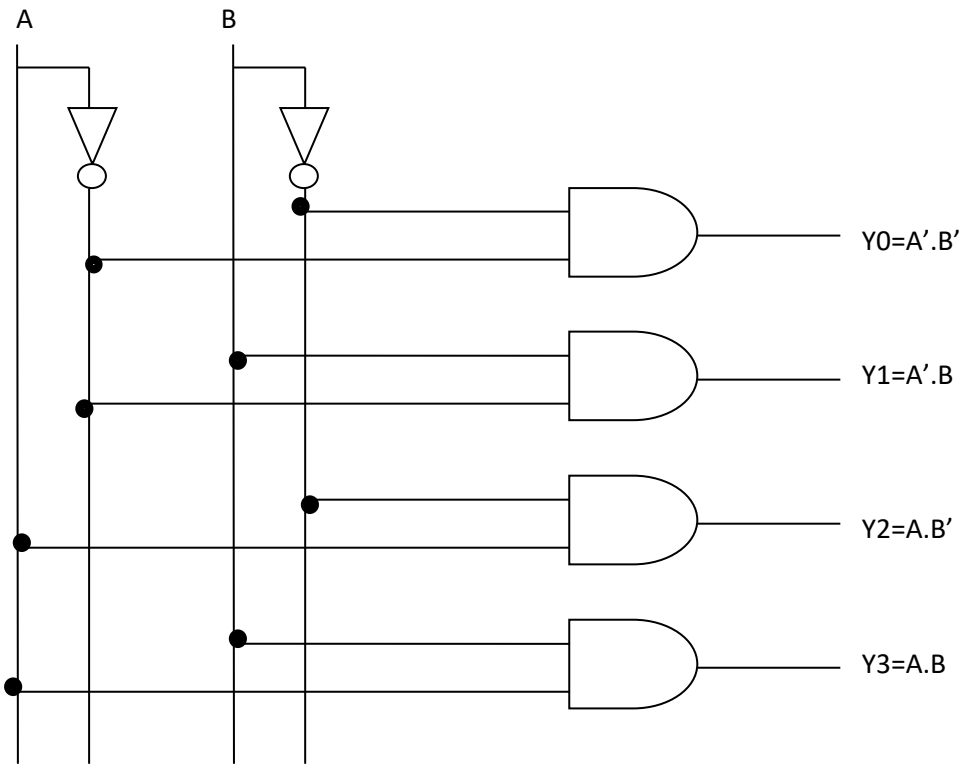


FIGURE 4 2 X 4 DECODER CIRCUIT

TRUTH TABLE

Inputs		Output			
A	**B**	**Y0**	**Y1**	**Y2**	**Y3**
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

TABLE 2 :Truth Table for 2 X 4 DECODER

2.c 4:1 MUX

Theory

An electronic multiplexer can be considered as a multiple-input, single-output Switch. It is used to combines several input information signals to one output Signal. Theinput line to be connected to output line is selected from the status of select lines. The no. of select lines depend upon the no. of input lines to the Multiplexer and are given as

$$N= 2^s$$

Where N =.> NO of inputs of Mux.

S => No of Select lines

BLOCK DIAGRAM:-

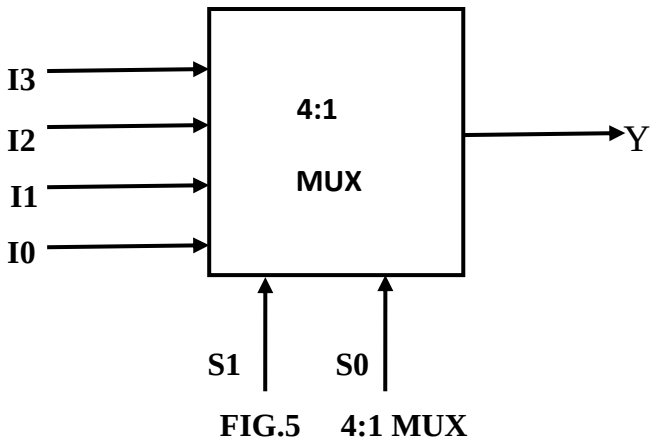


FIG.5 4:1 MUX

LOGIC DIAGRAM

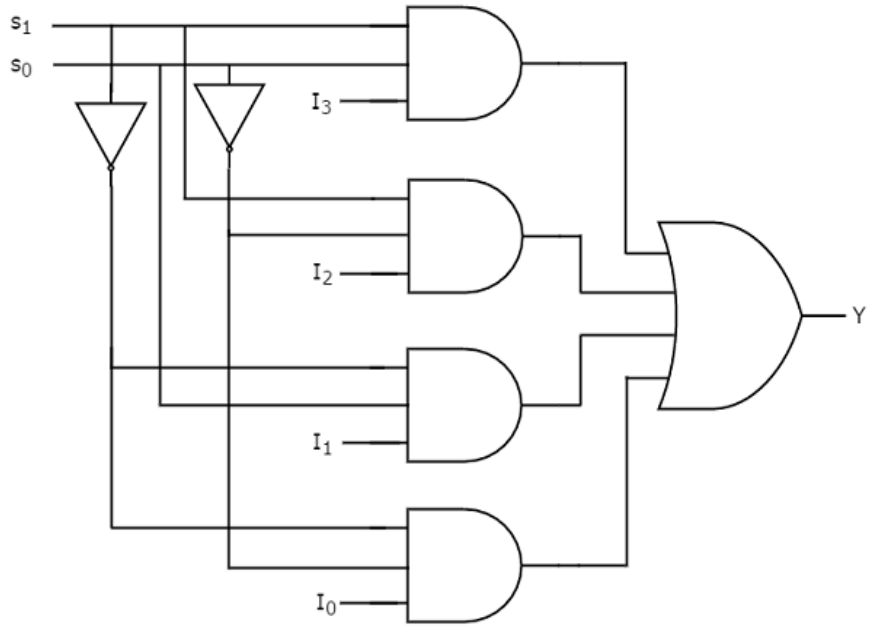


Fig .6 Logic Diagram for 4:1 MUX

TRUTH TABLE

Selection Lines		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

TABLE 3:truth table for 4:1 MUX

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;
```

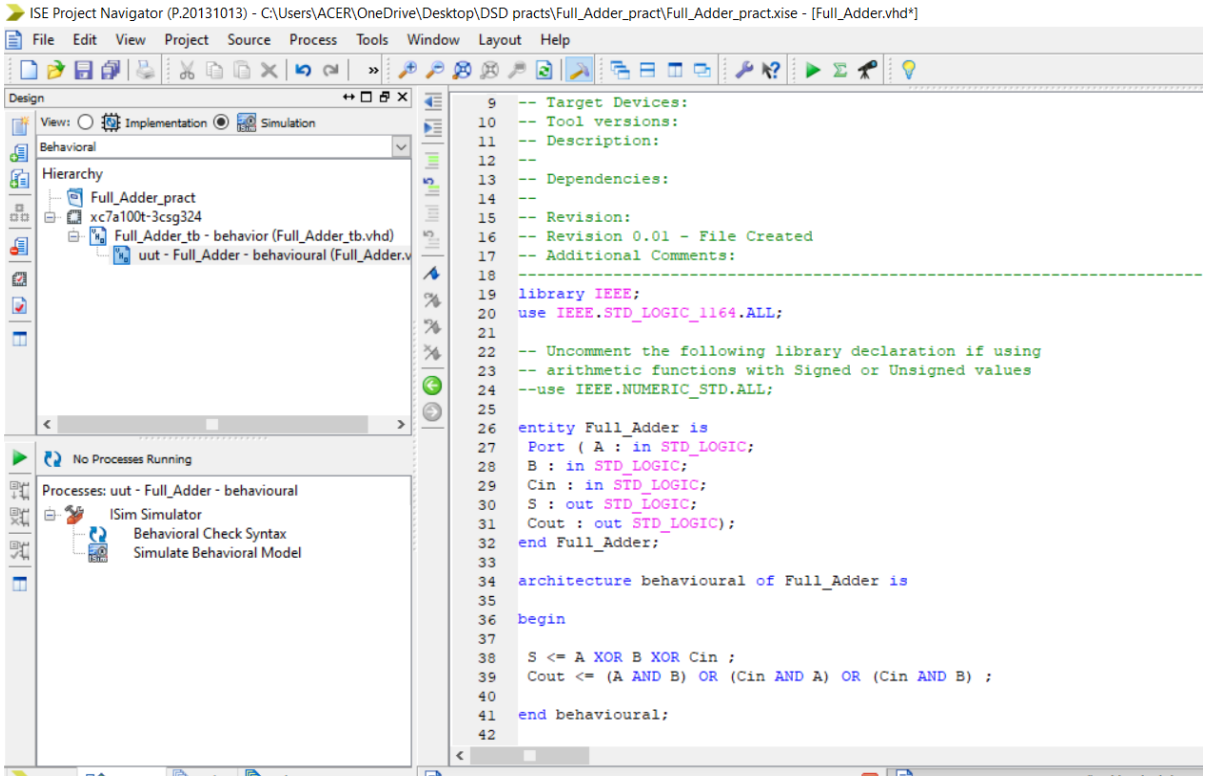
Here we should specify the entity name for which we are writing the architecture body. The architecture statements should be inside the begin and end keyword. Architecture declarative part may contain variables, constants, or component declaration.

Step 4: Simulate the VHDL code and remove the syntax

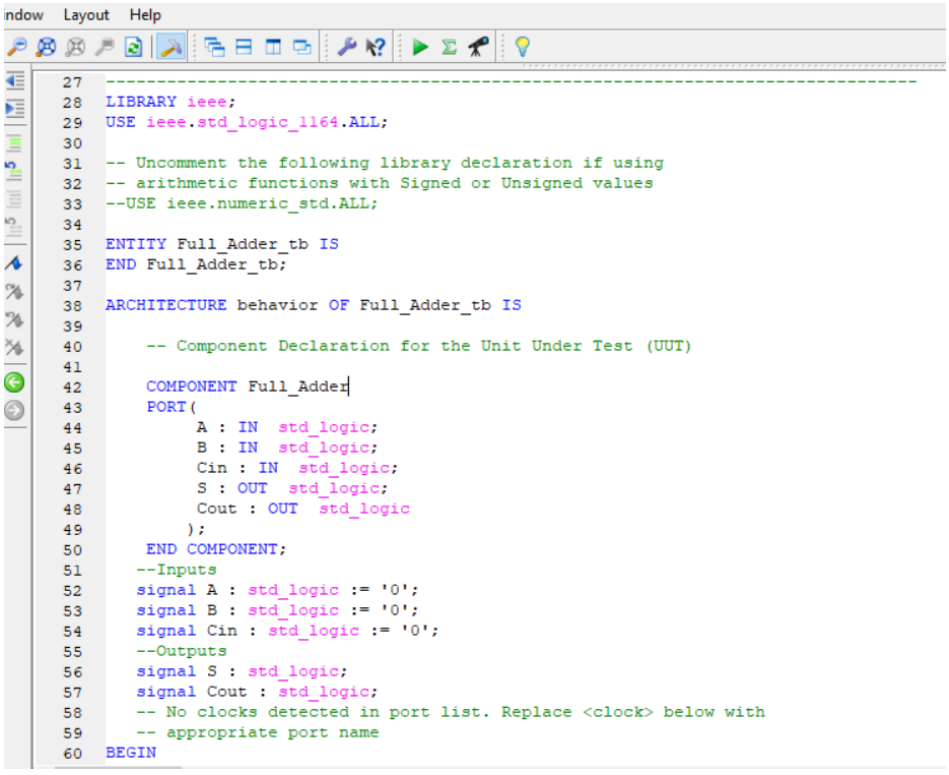
errors if any.

Step 5: Write the testbench and verify the design.

CODE: Full Adder



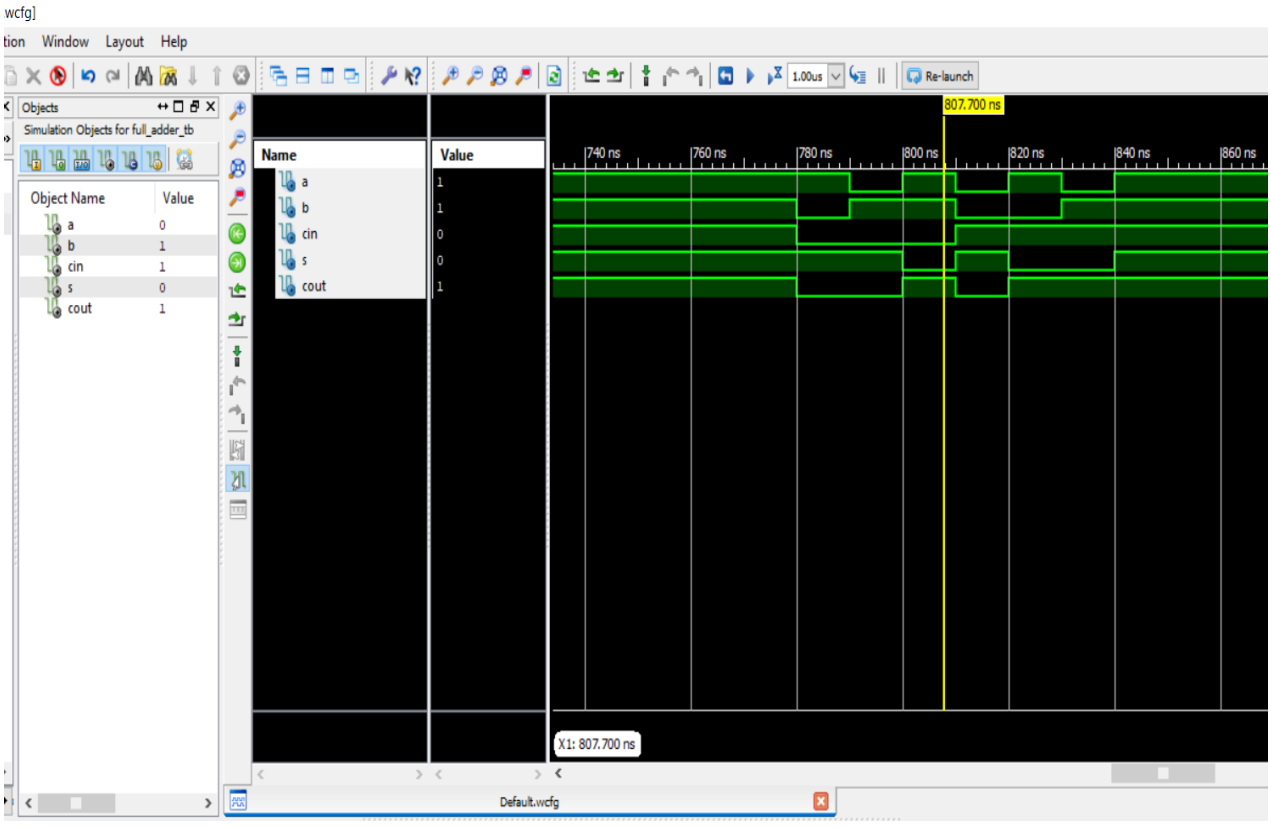
TESTBENCH:



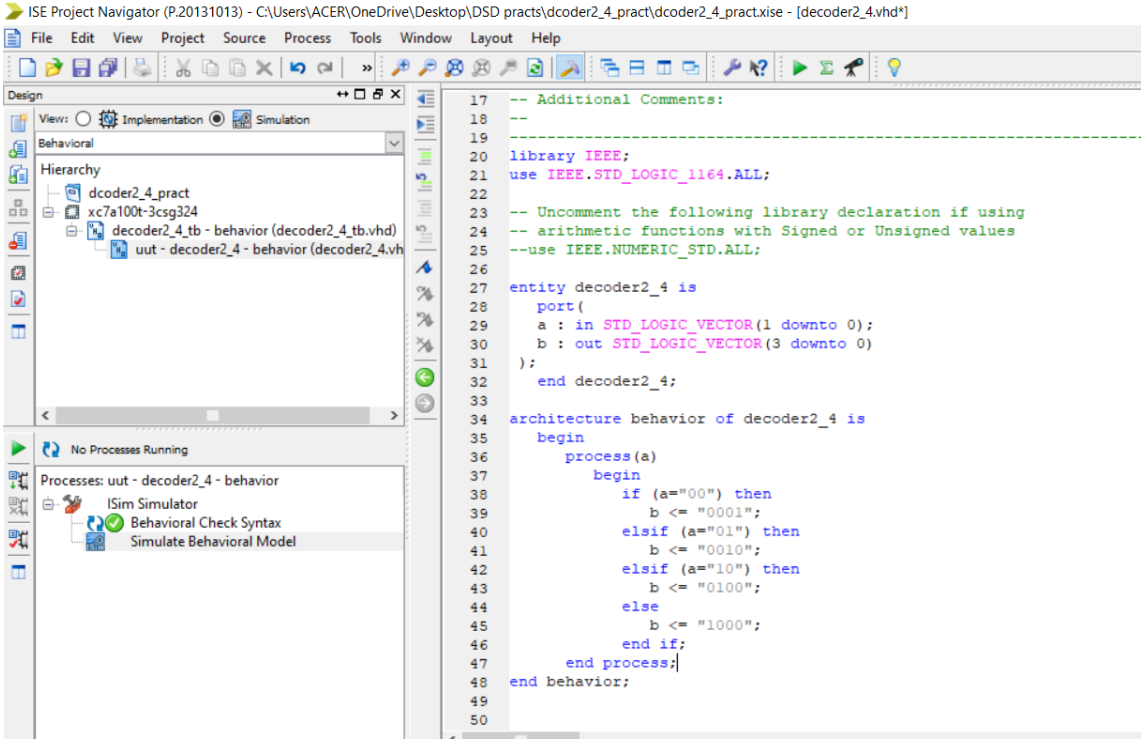
```
61      -- Instantiate the Unit Under Test (UUT)
62      uut: Full_Adder PORT MAP (
63              A => A,
64              B => B,
65              Cin => Cin,
66              S => S,
67              Cout => Cout
68            );
69      -- Stimulus process
70      stim_proc: process
71     begin
72     -- hold reset state for 100 ns.
73     wait for 100 ns;
74       B <= '0';
75       Cin <= '0';
76       wait for 10 ns;
77
78       A <= '0';
79       B <= '1';
80       Cin <= '0';
81       wait for 10 ns;
82
83       A <= '1';
84       B <= '1';
85       Cin <= '0';
86       wait for 10 ns;
87
88       A <= '0';
89       B <= '0';
90       Cin <= '1';
91       wait for 10 ns;
92
93       A <= '1';
94       B <= '0';
```

```
95       Cin <= '1';
96       wait for 10 ns;
97
98       A <= '0';
99       B <= '1';
100      Cin <= '1';
101      wait for 10 ns;
102
103      A <= '1';
104      B <= '1';
105      Cin <= '1';
106      wait for 10 ns;
107      end process;
108   END;
109
```

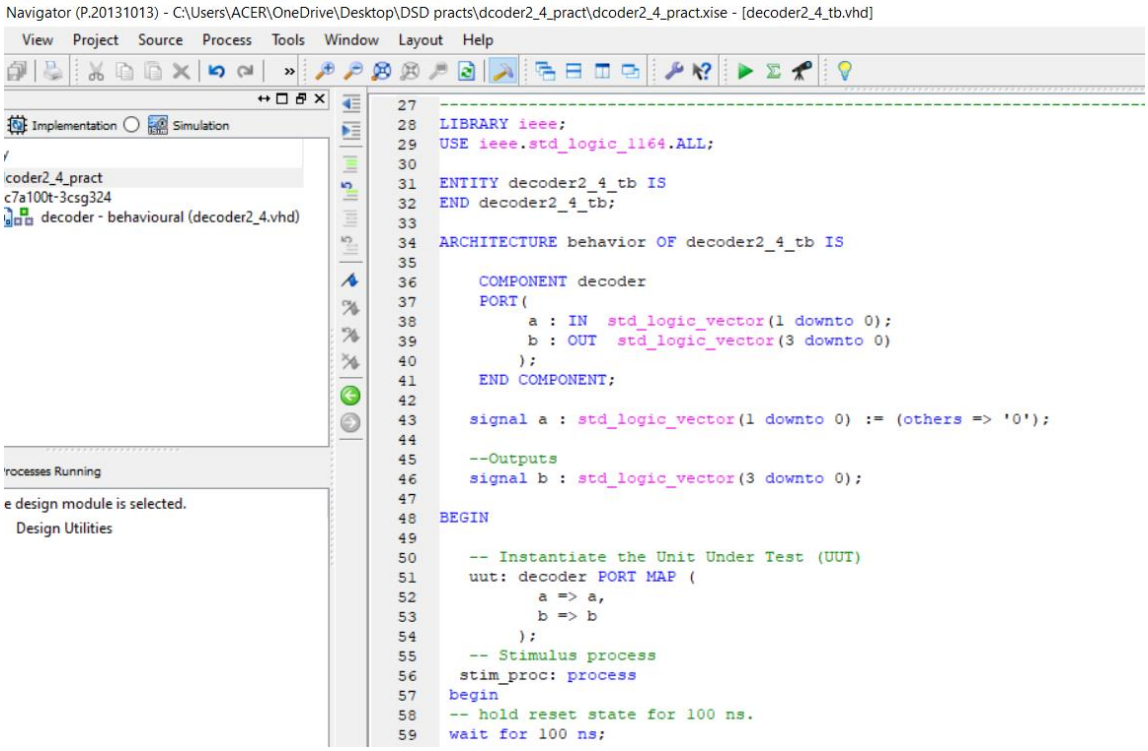
SIMULATION WAVEFORM:-

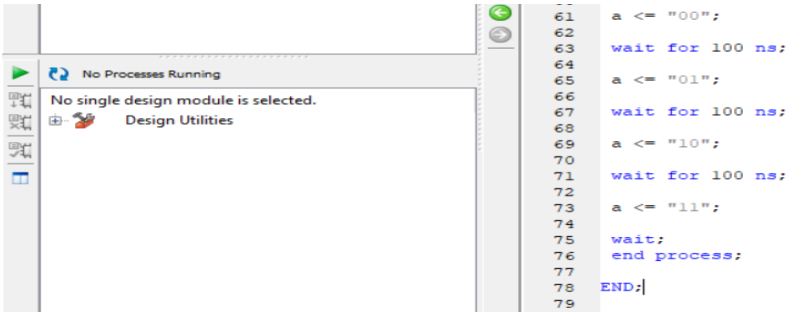


CODE:Decoder 2:4

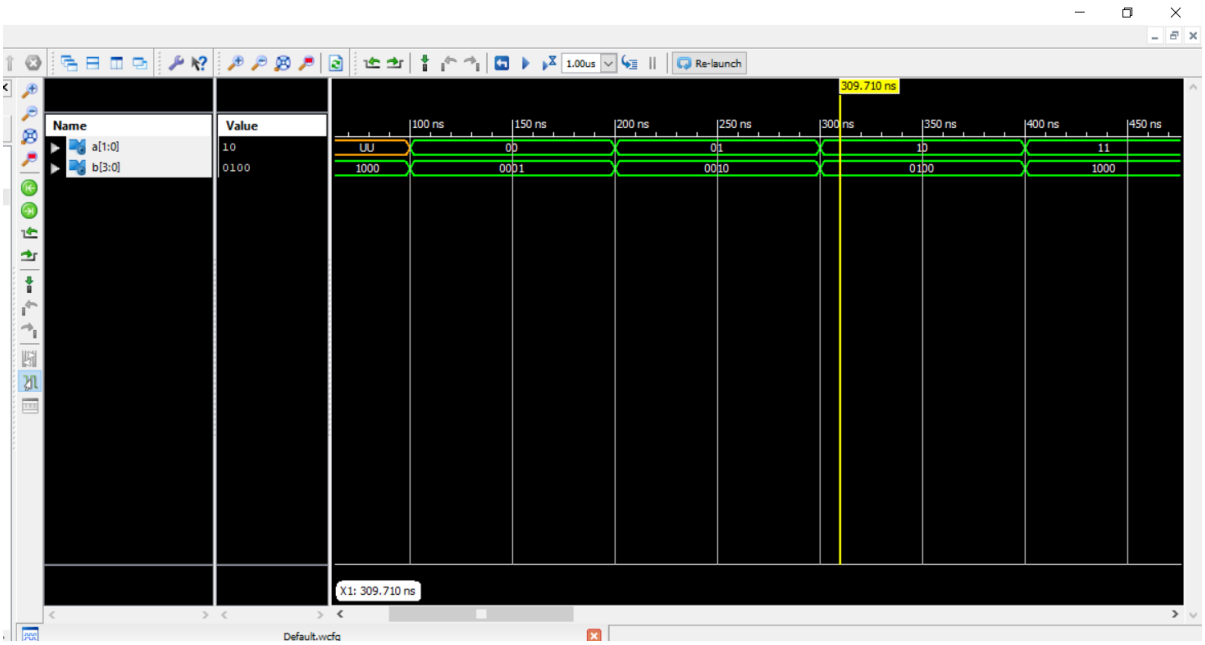


TESTBENCH:

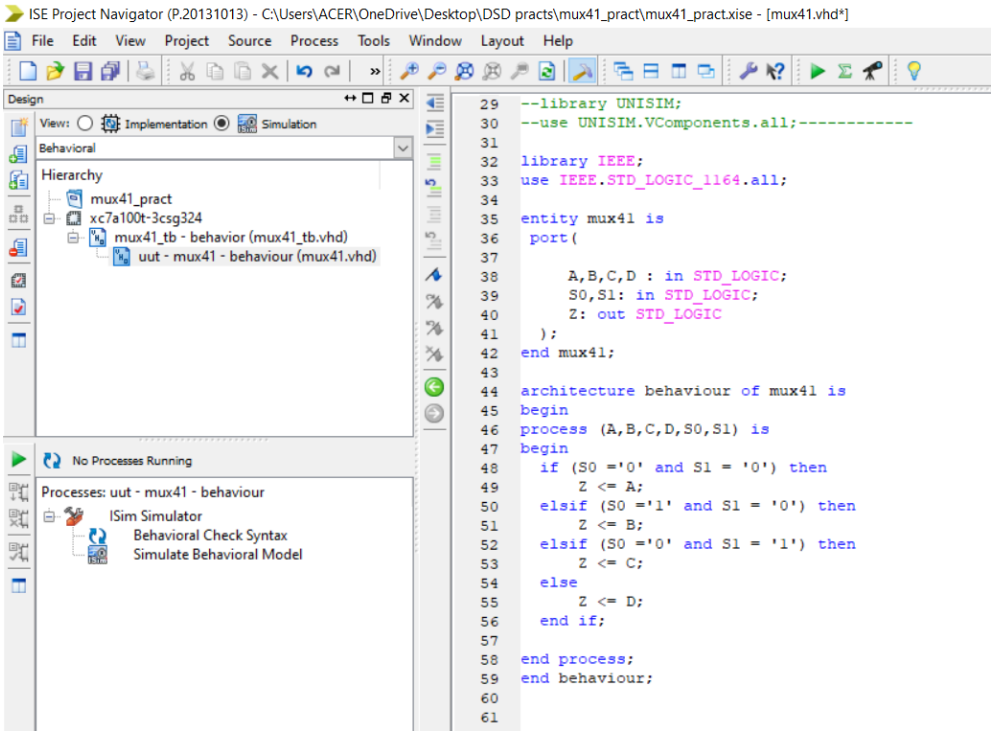




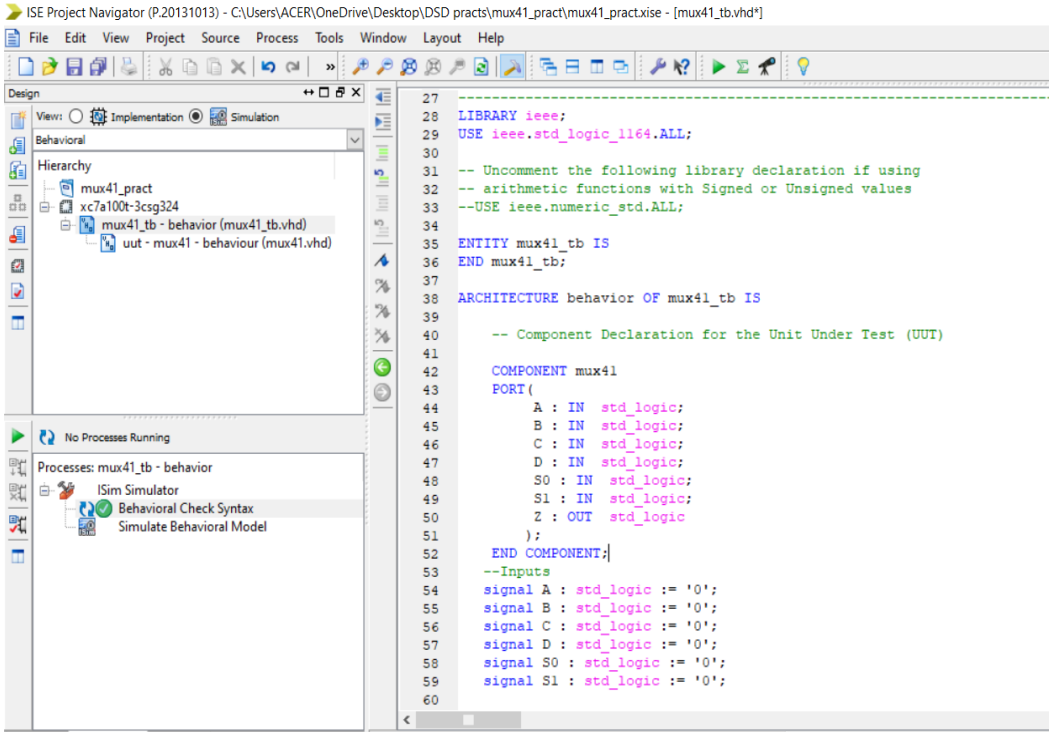
SIMULATION WAVEFORM:-

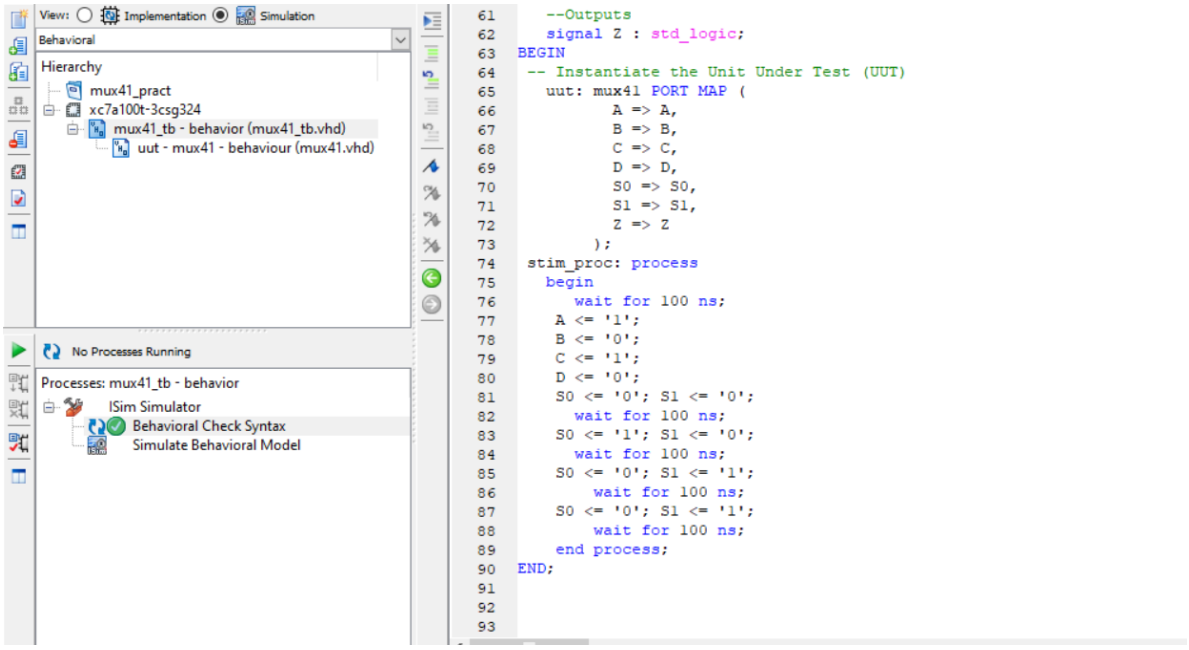


CODE: 4:1 MULTIPLEXER

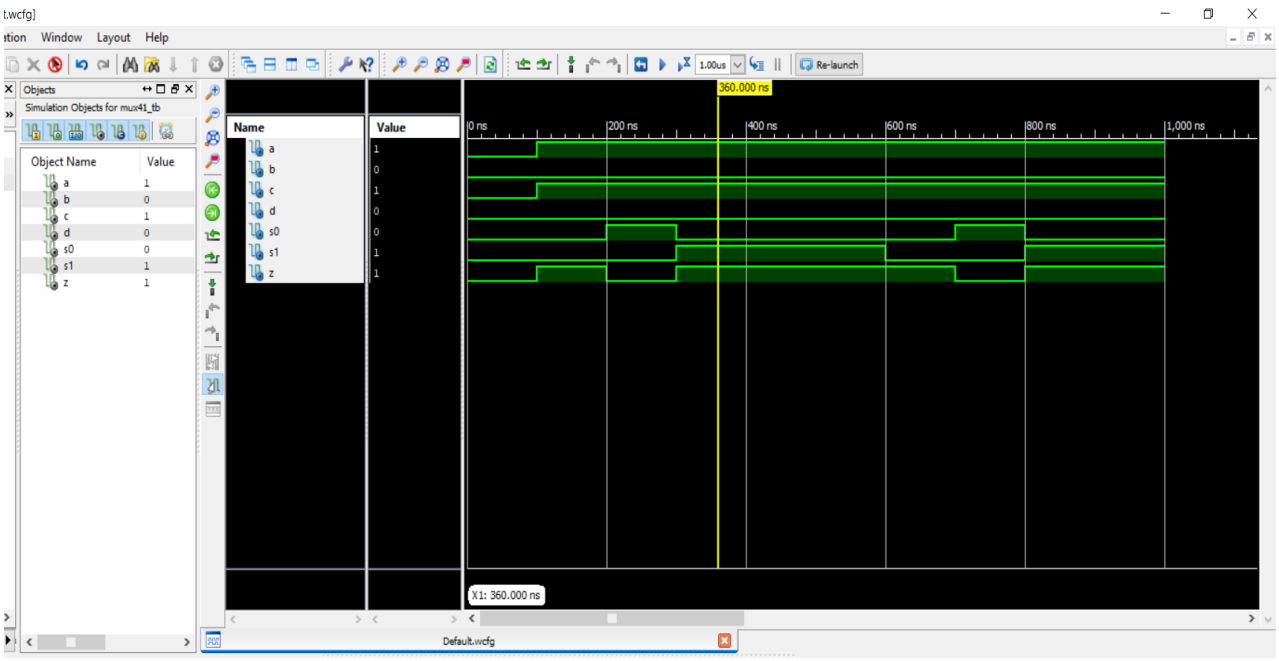


TESTBENCH:


ISE Project Navigator (P.20131013) - C:\Users\ACER\OneDrive\Desktop\DSD practs\mux41_pract\mux41_pract.xise - [mux41_tb.vhd*]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Behavioral
Hierarchy
mux41_pract
xc7a100t-3csg324
mux41_tb - behavior (mux41_tb.vhd)
uut - mux41 - behaviour (mux41.vhd)
No Processes Running
Processes: mux41_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 mux41_tb IS
36 END mux41_tb;
37
38 ARCHITECTURE behavior OF mux41_tb IS
39
40 -- Component Declaration for the Unit Under Test (UUT)
41
42 COMPONENT mux41
43 PORT(
44 A : IN std_logic;
45 B : IN std_logic;
46 C : IN std_logic;
47 D : IN std_logic;
48 S0 : IN std_logic;
49 S1 : IN std_logic;
50 Z : OUT std_logic
51);
52 END COMPONENT;
53 --Inputs
54 signal A : std_logic := '0';
55 signal B : std_logic := '0';
56 signal C : std_logic := '0';
57 signal D : std_logic := '0';
58 signal S0 : std_logic := '0';
59 signal S1 : std_logic := '0';
60


View: Implementation Simulation
Behavioral
Hierarchy
mux41_pract
xc7a100t-3csg324
mux41_tb - behavior (mux41_tb.vhd)
uut - mux41 - behaviour (mux41.vhd)
No Processes Running
Processes: mux41_tb - behavior
ISim Simulator
Behavioral Check Syntax
Simulate Behavioral Model
61 --Outputs
62 signal Z : std_logic;
63 BEGIN
64 -- Instantiate the Unit Under Test (UUT)
65 uut: mux41 PORT MAP (
66 A => A,
67 B => B,
68 C => C,
69 D => D,
70 S0 => S0,
71 S1 => S1,
72 Z => Z
73);
74 stim_proc: process
75 begin
76 wait for 100 ns;
77 A <= '1';
78 B <= '0';
79 C <= '1';
80 D <= '0';
81 S0 <= '0'; S1 <= '0';
82 wait for 100 ns;
83 S0 <= '1'; S1 <= '0';
84 wait for 100 ns;
85 S0 <= '0'; S1 <= '1';
86 wait for 100 ns;
87 S0 <= '0'; S1 <= '1';
88 wait for 100 ns;
89 end process;
90 END;
91
92
93

SIMULATION WAVEFORM:-



RESULT:-

The VHDL code for Full Adder,2:4 Decoder and 4:1 MUX is executed and desired output is obtained.

CONCLUSION:

Here, I successfully designed and test the Full Adder, 2:4 Decoder and 4:1 Multiplexer in VHDL code.

DISCUSSION & VIVA VOCE

1) What is BCD adder?
2) Design a 4 x16 decoder using 3x8 Decoder(s).
3) Difference between MUX and DEMUX.

REFERENCE:

- VHDL Primer–J Bhasker –Pearson Education
- NPTEL Video Lecture link- Lecture 4 - Combinatioal Circuits - YouTube