SRI VENKATESWARA COLLEGE OF ENGINEERING AND TECHNOLOGY

(AUTONOMOUS)

R.V.S NAGAR, CHITTOOR-517127

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

LABORATORY MANUAL

LAB CODE : 20AEC29

LABNAME : VHDL Programming

YEAR&SEMESTER : III B.Tech –I SEM

REGULATIONS : R- 20

Roll. No:----------

Name :----------

Class :----------

Syllabus

20AEC29 – VHDL PROGRAMMING

Course Outcomes:

At the end of the course, the students would be:

CO1: understand the fundamentals of design digital systems through VHDL language.

CO2: Analyze and synthesis Sequential circuit using VHDL code.

CO3: Interfacing FPGA with External device

Unit I Introduction to VHDL: (4 Periods)

VHDL Terms- Describing Hardware in VHDL – Entity, Architectures, Concurrent Signal Assignment, Event Scheduling, Statement Concurrency, Structural Designs, Sequential Behavior, Process Statements, Process Declarative Region, Process Statement Part, Process Execution, Sequential Statements, Architecture Selection, Configuration Statements, Power of Configurations.

Unit II Behavioural Modeling: (4 Periods)

Introduction to Behavioral Modeling- Transport Versus Inertial Delay, Inertial Delay, Transport Delay, Simulation Deltas, Drivers- Driver Creation, Bad Multiple Driver Model, Generics - Block Statements, Guarded Blocks.

Unit III Synthesis: (4 Periods)

Register Transfer Level Description, Constraints, Timing Constraints, Clock Constraints, Attributes, Load, Drive, Arrival Time, Technology Libraries, Synthesis, Translation, Boolean Optimization, Flattening, Factoring, Mapping to Gates.

Unit IV High Level Design Flow: (4 Periods)

RTL Simulation, VHDL Synthesis, Functional Gate-Level Verification, Place and Route, Post Layout Timing Simulation, Static Timing.

Unit V Top-Level System Design: (4 Periods)

CPU Design, Top-Level System Operation, Instructions, Sample Instruction Representation,

CPU Top-Level Design, Block Copy Operation.

List of Experiments **(10Periods)**

(Minimum of 10 Experiments to conducted)

1. Basic Logic Gates
2. Half Adder, Full Adder
3. Half Subtractor and Full Subtractor
4. 3-8 Decoder
5. 8-3 Encoder
6. 8:1 Multiplexer
7. 1:4 Demultiplexer
8. 4-Bit Comparator
9. JK Flip-Flop
10. 4-bit Synchronous Binary Counter
11. 4-Bit Universal Shift Register
12. 4-Bit ALU

INDEX

EXPNO:1 **DATE:**

LOGICGATES

AIM: To write and simulate a VHDL Program for AND, OR, NAND, NOR, XOR and NOT gates by using Modelsim.

REQUIREMENTS:

- System with Modelsim Software.

THEORY:

Digital electronics relies on the actions of just seven types of logic gates, called AND, OR,NAND (Not AND), NOR (Not OR), XOR (Exclusive OR) and NOT. In binary logic there are only two states,1 and 0 or, on and off. If something is not 1 it must be 0, if it is not on, it must be off. So NAND (not AND) simply means that a NAND gate performs the opposite function to an AND gate.

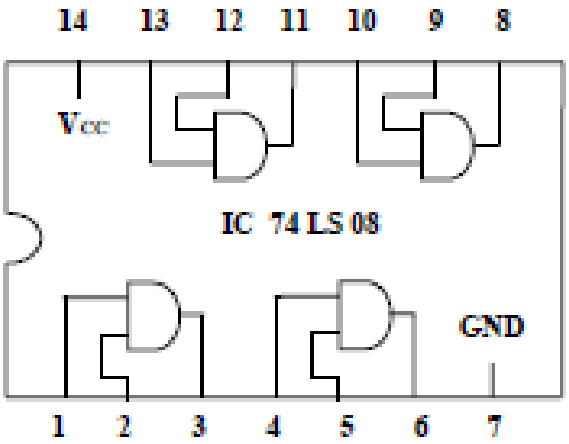
A logic gate is a small transistor circuit, which is implemented in different forms within an integrated circuit. Each type of gate has one or more (most often two) inputs and one output. The principle of operation is that the circuit operates on just two voltage levels, called logic 0 and logic 1.When either of these voltage levels is applied to the inputs, the output of the gate responds by assuming a 1 or a 0 level, depending on the particular logic of the gate. The logic rules for each type of gate can be described in different ways, by a written description of the action, by a truth table, or by a Boolean algebra statement.

PROCEDURE:

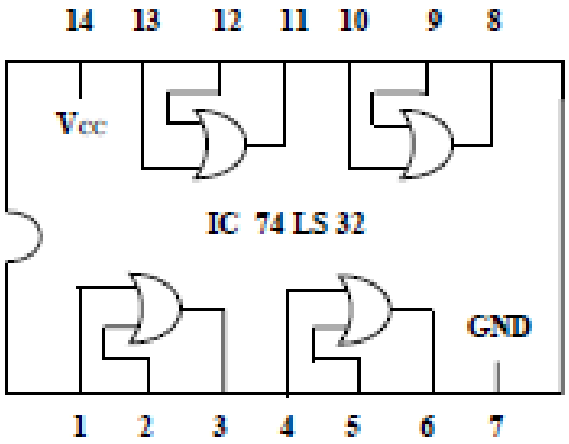
1. Switch on the system and open the Modelsim Window.
2. Open the new editor window through File→New→Source→VHDL.
3. Type the VHDL Program in editor window and save the program with **file_name.vhd**.
4. Compile the program. If there is any error found, rectify the error in a program and a gain compile until getting the error free code.
5. Simulate the program. Add the waveform with input and output signals in the program.
6. Apply different values to the input signals and observe the output signals for corresponding inputs.

PINDIAGRAMS OF LOGIC GATES:

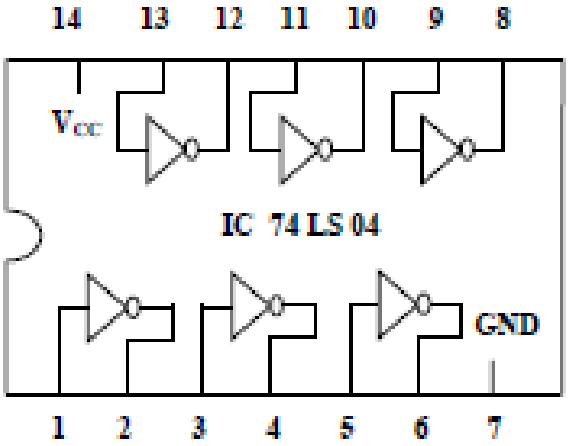
AND GATE



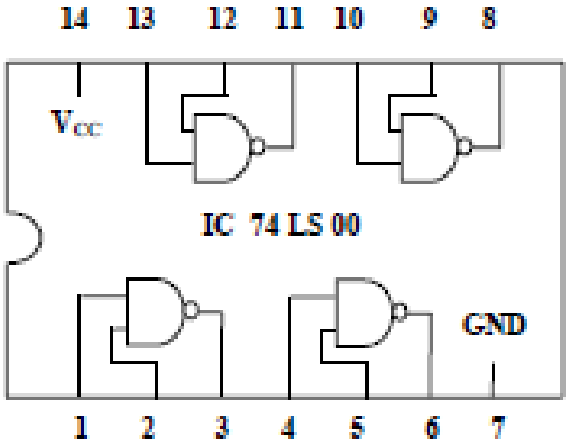
OR GATE



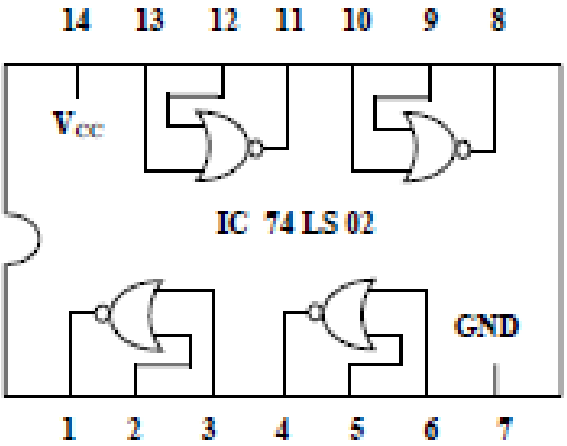
NOT GATE



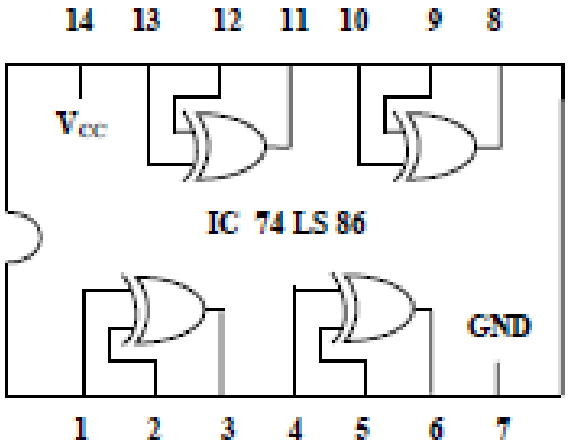
NAND GATE



NOR GATE

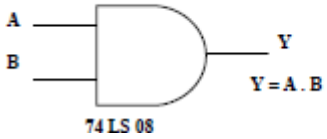


EX - OR GATE



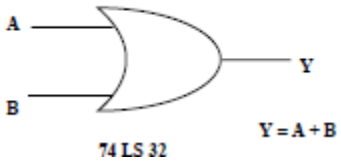
TRUTH TABLES AND LOGIC

SYMBOL:AND GATE:



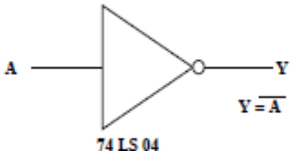
Inputs		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR GATE:



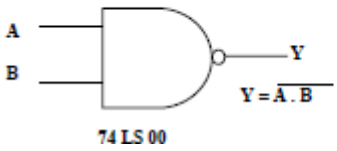
Inputs		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT GATE:



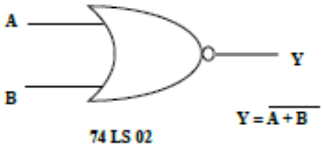
Input	Output
A	Y
0	1
1	0

NAND GATE:



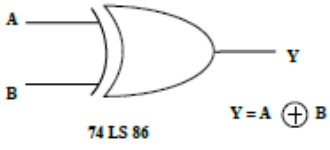
Inputs		Output
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

NOR GATE:



Inputs		Output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

XOR GATE:



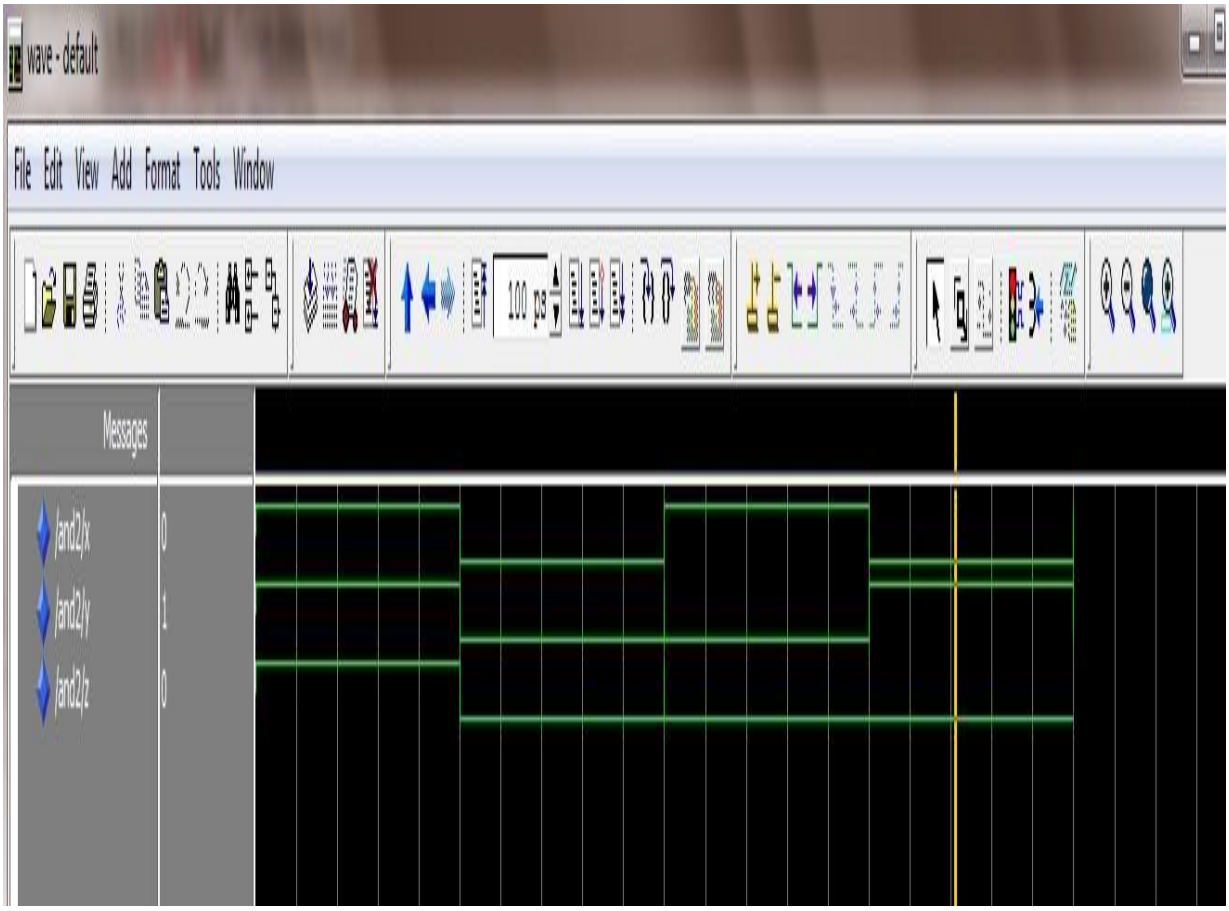
Inputs		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

VHDL PROGRAM FOR LOGIC GATES:

AND GATE:

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

ENTITY AND2 IS
        PORT (X, Y: IN STD_LOGIC;
        Z:OUT STD_LOGIC);
END AND2;
ARCHITECTURE AND2_DATA OF AND2 IS

BEGIN
                Z<=X AND Y;
                END AND2_DATA;
```

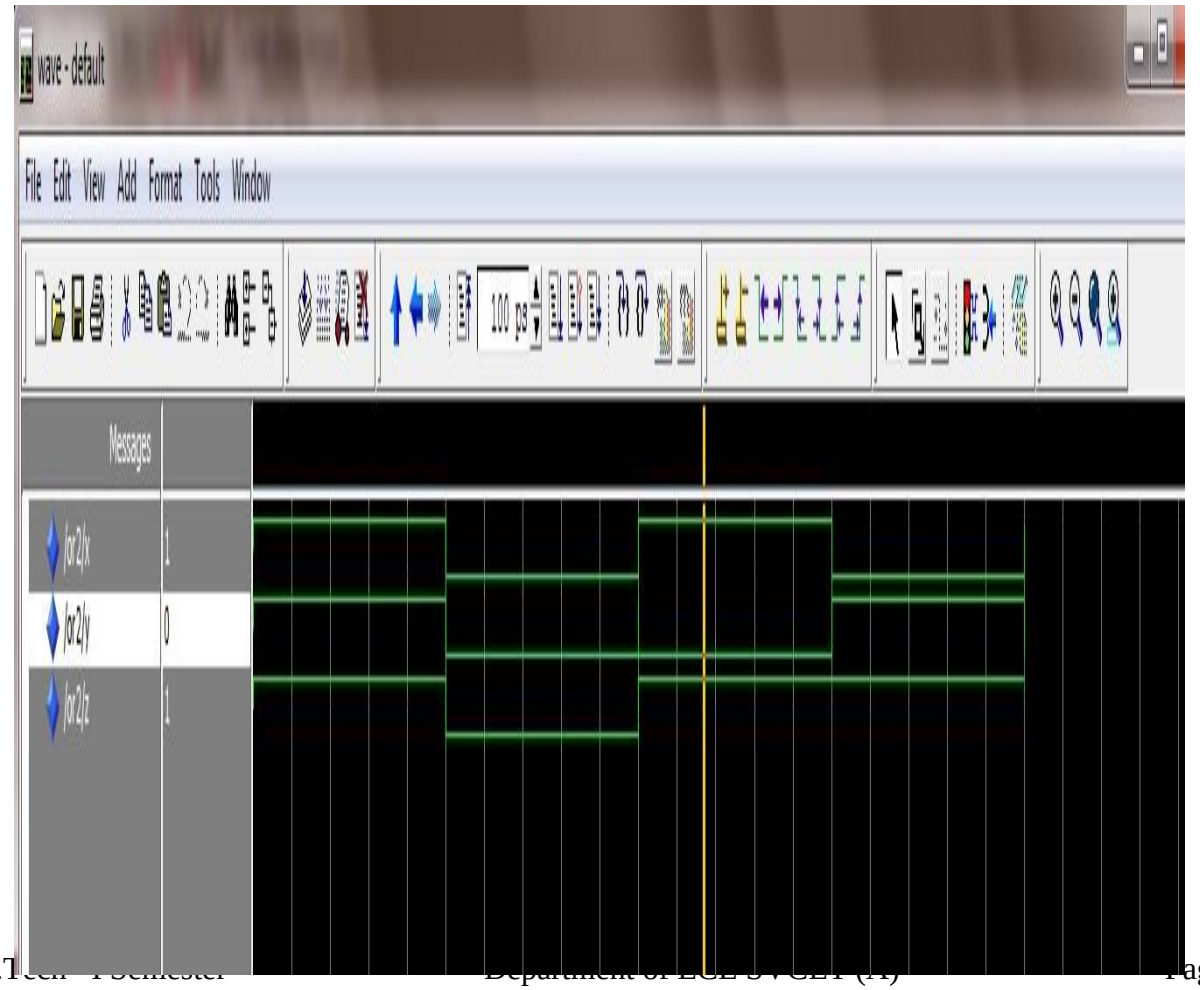
SIMULATIONRESULTS:



OR GATE:

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

ENTITY OR2 IS
        PORT(X, Y: IN STD_LOGIC;
                Z:OUT STD_LOGIC);
END OR2;
ARCHITECTURE OR2_DATA OF OR2 IS

BEGIN
                Z<=X OR Y;
                END OR2_DATA;
```

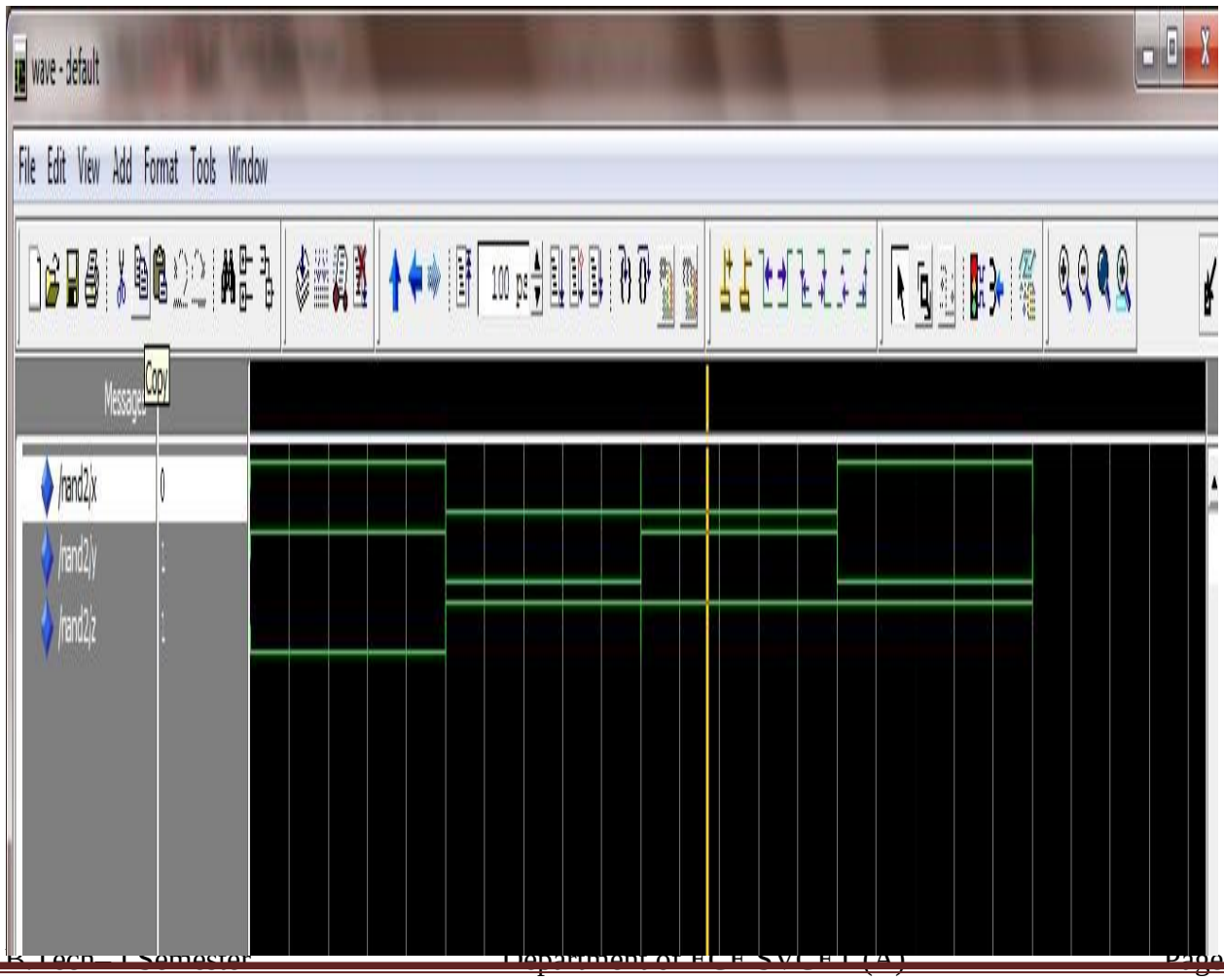
SIMULATIONRESULTS:



NAND GATE:

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

ENTITY NAND2 IS
        PORT (X, Y: IN STD_LOGIC;
        Z:OUT STD_LOGIC);
END NAND2;
ARCHITECTURE NAND2_DATA OF NAND2 IS

BEGIN
            Z<=X NAND Y;
            END NAND2_DATA;
```

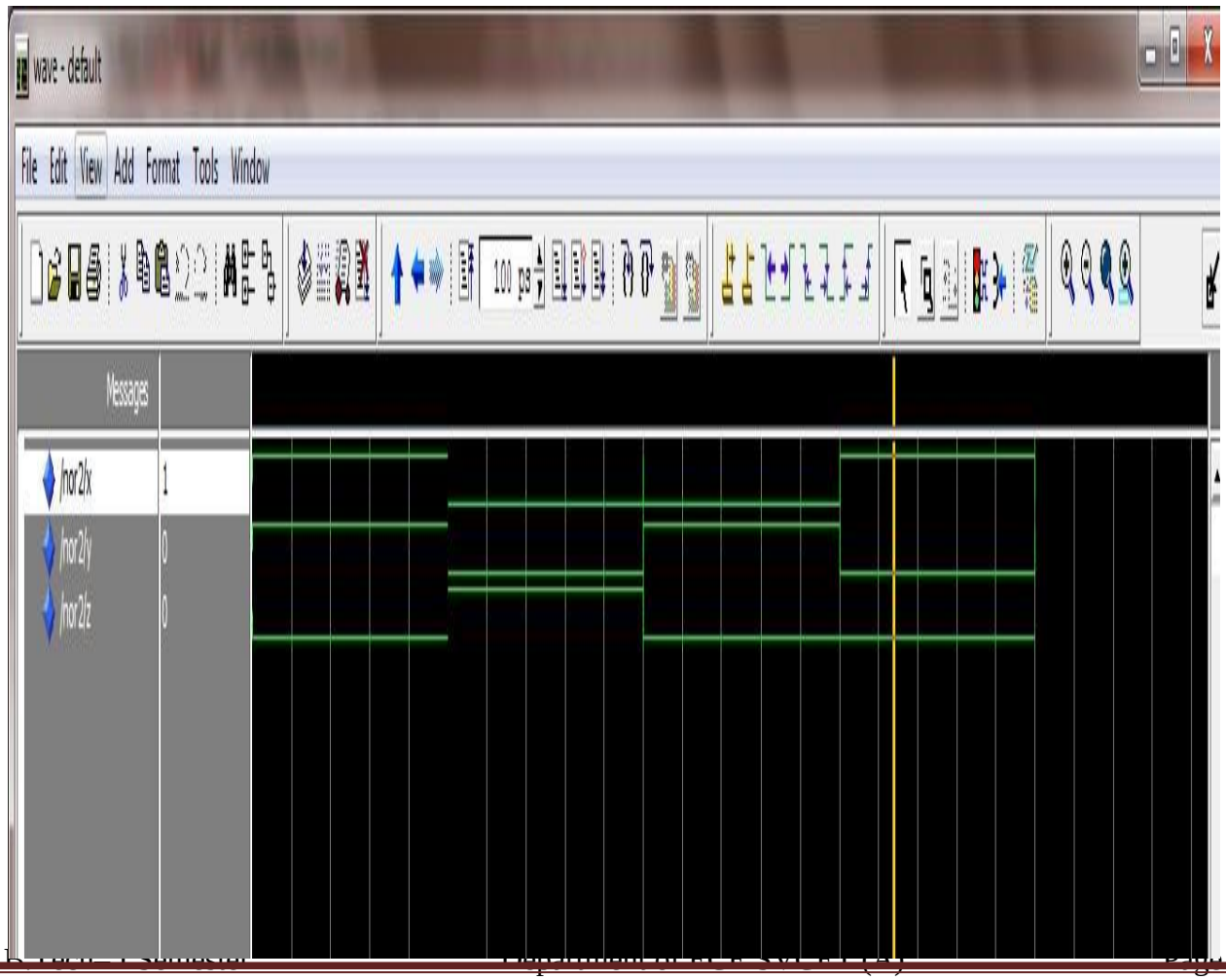
SIMULATIONRESULTS:



NOR GATE:

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

ENTITY NOR2 IS
        PORT(X, Y: IN STD_LOGIC;
                Z:OUT STD_LOGIC);
END NOR2;
ARCHITECTURE NOR2_DATA OF NOR2 IS

BEGIN
        Z<=X NOR Y;
        END NOR2_DATA;
```

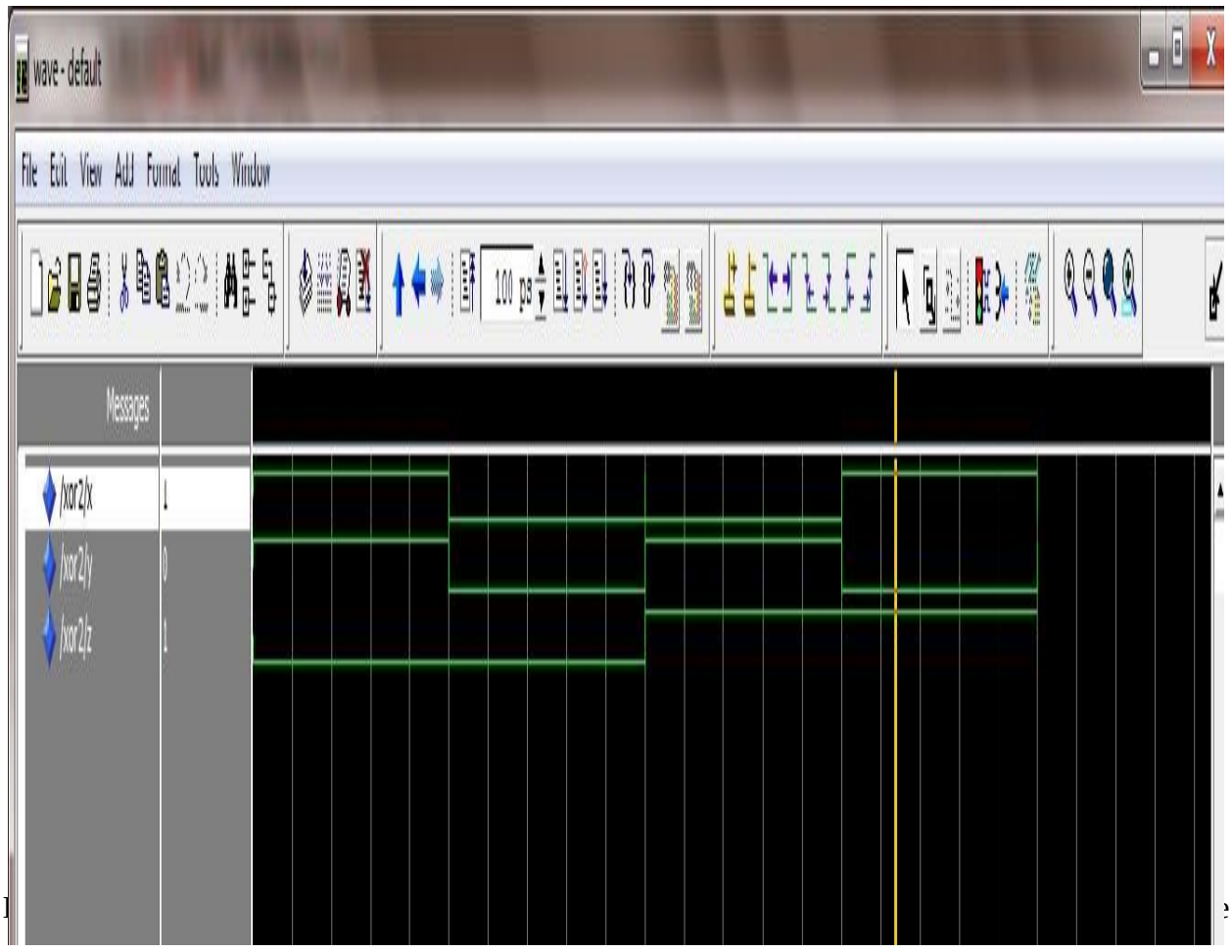
SIMULATION RESULTS:



XOR GATE:

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

ENTITY XOR2 IS
        PORT(X, Y: IN STD_LOGIC;
                Z:OUT STD_LOGIC);
END XOR2;
ARCHITECTURE XOR2_DATA OF XOR2 IS

BEGIN
                Z<=X XOR Y;
                END XOR2_DATA;
```

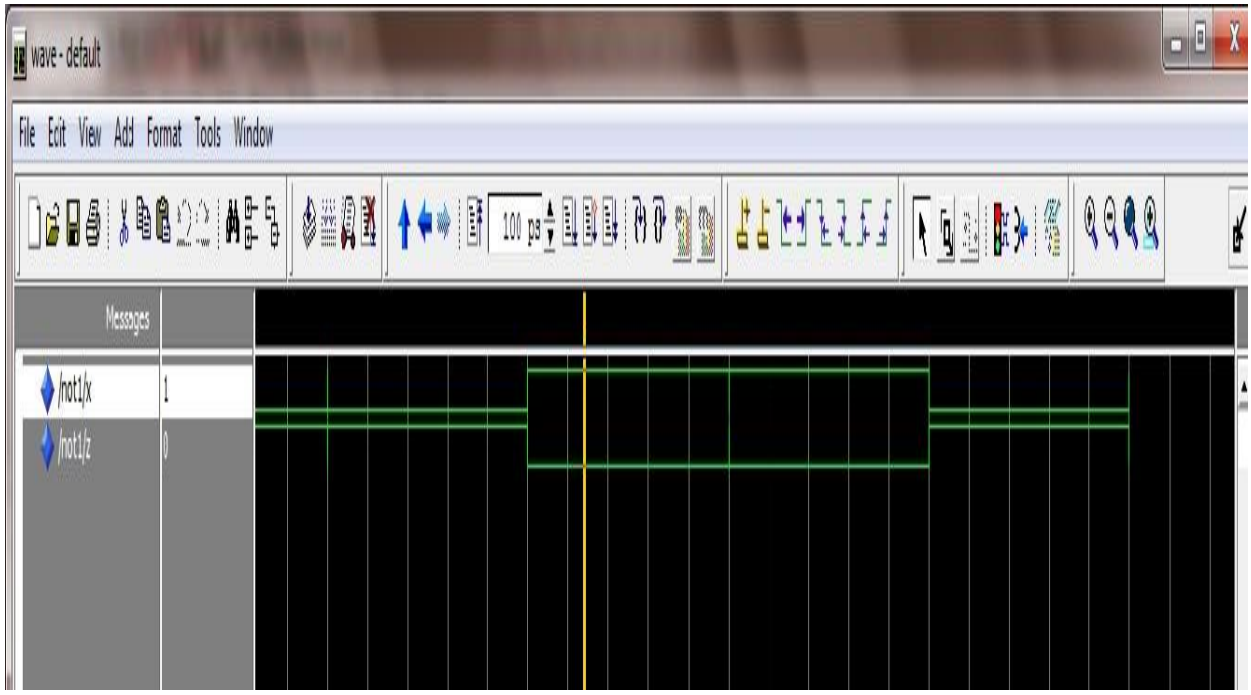
SIMULATIONRESULTS:



NOT GATE:

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

ENTITY NOT1 IS
    PORT(X: IN STD_LOGIC;
         Z:OUT STD_LOGIC);
 END NOT1;
ARCHITECTURE NOT1_DATA OF NOT1 IS

BEGIN
     Z<=NOT X;
END NOT1_DATA;
```

SIMULATIONRESULTS:



RESULT:

EXPNO: 2 **DATE:**

HALF ADDER and FULL ADDER

AIM:

To write and simulate a VHDL Program for Half-Adder, Full-Adder by using Modelsim.

SOFTWARES REQUIRED:

- System with Modelsim 6.3 Version.

THEORY:

An Adder is a device that can add two binary digits. It is a type of digital circuit that performs the operation of additions of two numbers. It is mainly designed for the addition of binary number, but they can be used in various other applications like binary code decimal, address decoding, table index calculation, etc. There are two types of Adder. One is Half Adder, and another one is known as Full Adder. There are two inputs and two outputs in a Half Adder. Inputs are named as A and B, and the outputs are named as Sum and Carry. The Sum is X-OR of the input A and B. Carry is AND of the input A and B. With the help of half adder, one can design a circuit that is capable of performing simple addition with the help of logic gates. The full adder is a little more difficult to implement than a half adder. The main difference between a half adder and a full adder is that the full adder has three inputs and two outputs. The two inputs are A and B, and the third input is a carry input C. The output carry is designated as CARRY, and the normal output is designated as SUM. A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers. A binary adder can be constructed with ful ladders connected in cascade with the output carry form each full adder connected to the input carry of the next full adder in the chain.

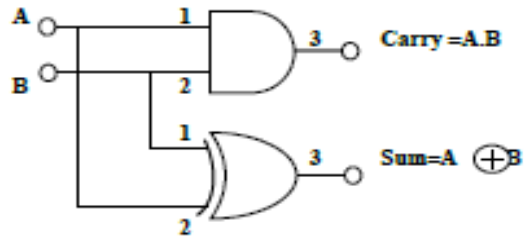
PROCEDURE:

1. Switch on the system and open the Modelsim Window.
2. Open the new editor window through File→New→Source→VHDL.
3. Type the VHDL Program in editor window and save the program with **file_name.vhd**.
4. Compile the program. If there is any error found, rectify the error in a program and again Compile until getting the error free code.
5. Simulate the program. Add the waveform with input and output signals in the program.
6. Apply different values to the input signals and observe the output signals for corresponding inputs.

LOGIC DIAGRAMS AND TRUTH TABLE

HALFADDER:

CIRCUIT DIAGRAM:

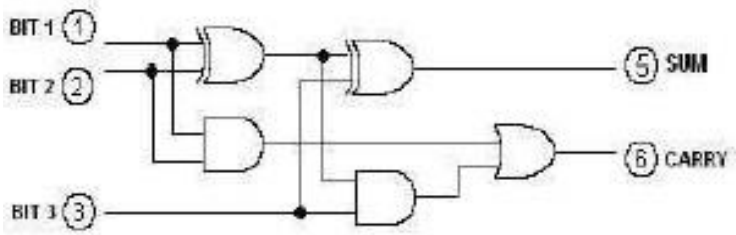


TRUTH TABLES

Inputs		Outputs	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

FULLADDER:

Full adder: A = Bit 1, B = Bit 2, C= Bit 3



BITS				
1	2	3	CARRY	SUM
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Sum = A ⊕ B ⊕ C

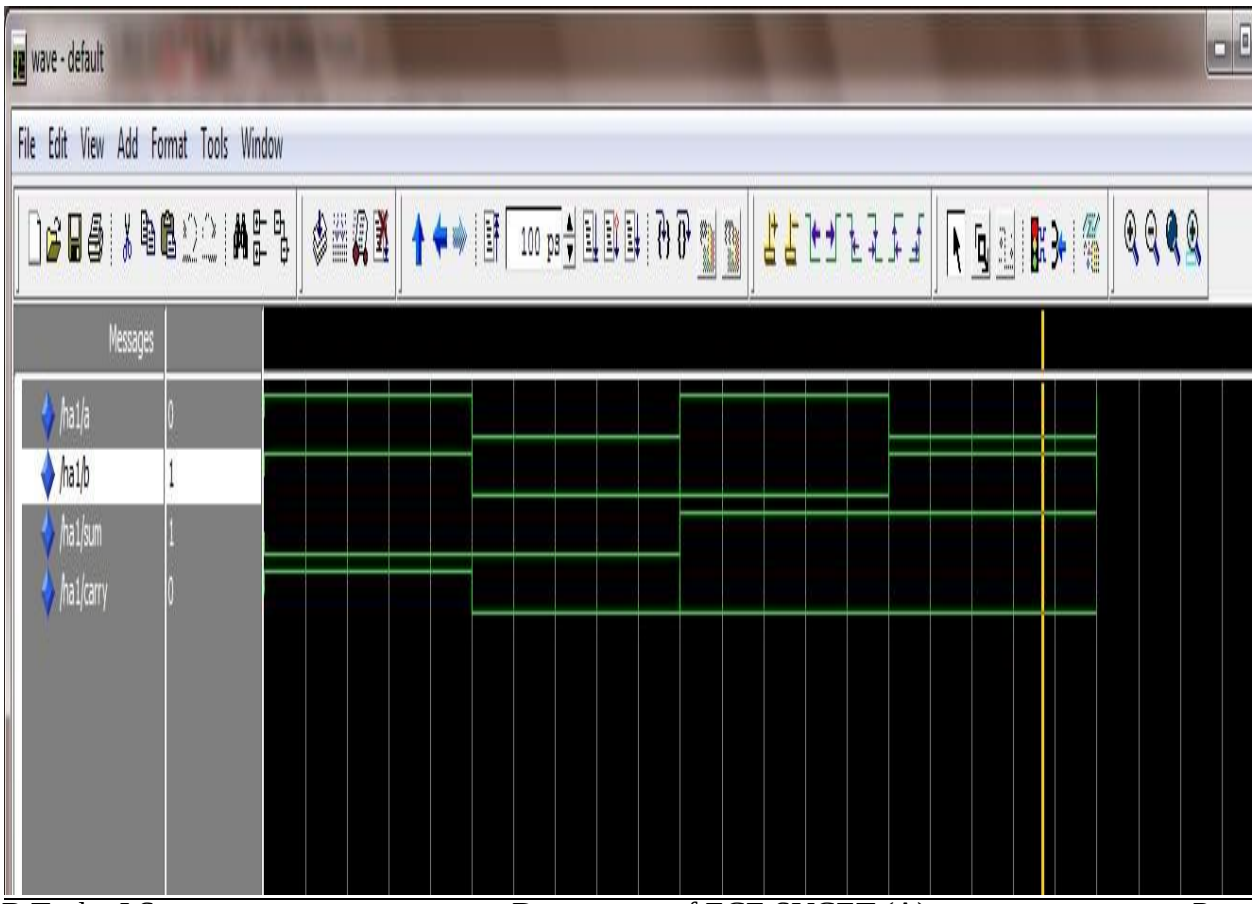
Carry = A . B + C (A⊕B)

VHDL PROGRAMS ADDERS:

HALF-ADDER:

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

ENTITY HALFADDER IS
    PORT(A, B:IN STD_LOGIC;
        SUM, CARRY: OUT STD_LOGIC);

        END HALFADDER;
ARCHITECTURE HALFADDER_DATA OF HALFADDER IS
BEGIN
            SUM<=A XOR B;
            CARRY<=A AND B;
 END HALFADDER_DATA;
```

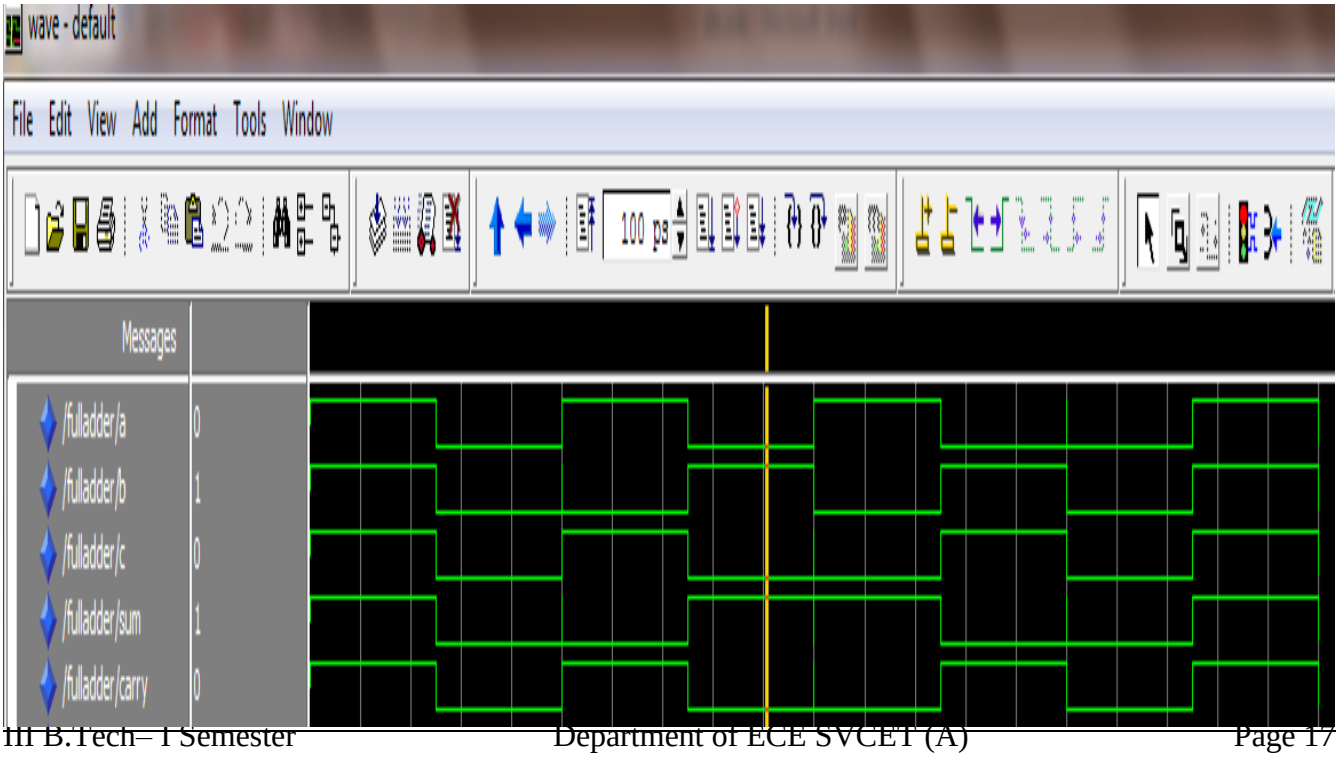
SIMULATIONRESULTS:



FULL-ADDER:

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

ENTITY FULLADDER IS
        PORT(A, B,C:IN STD_LOGIC;
                SUM, CARRY: OUT STD_LOGIC);

                END FULLADDER;
ARCHITECTURE FULLADDER_BEHAVE OF FULLADDER IS
BEGIN
                PROCESS (A, B, C)
                BEGIN
                        SUM<=A XOR B XOR C;
                        CARRY<= (A AND B) OR (B AND C) OR (C AND A);

                        ENDPROCESS;
END FULLADDER_BEHAVE;
```

SIMULATION RESULTS



RESULT:

EXPNO: 3 **DATE:**

HALF SUBTRACTOR AND FULL SUBTRACTOR

AIM: To write and simulate a VHDL Program for Half Subtractor and Full Subtractor by using Modelsim.

SOFTWARESREQUIRED:

- SystemwithModelsim6.3Version.

THEORY:

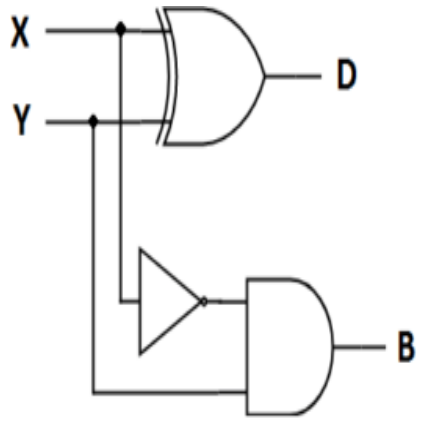
The half subtractor is a combinational circuit which is used to perform subtraction of two bits. It has two inputs, the minuend X and subtrahend Y and two outputs the difference D and borrow out B. The borrow out signal is set when the subtractor needs to borrow from the next digit in a multi-digit subtraction. That is, B=1, when X<Y. Since X and Y are bits Bo=1 if and only if X=0 and Y=1. Thefull subtractor is a combinational circuit which is used to perform subtraction of three input bits: the minuend X, subtrahend Y, and borrow in Z. The full subtractor generates two output bits: the difference D and borrow out B. Z is set when the previous digit is borrowed from X. Thus, Z is also subtracted from X as well as the subtrahend Y.

PROCEDURE:

1. Switch on the system and open the Modelsim Window.
2. Open the new editor window through File→New→Source→VHDL.
3. Type the VHDL Program in editor window and save the program with **file_name.vhd**.
4. Compile the program. If there is any error found, rectify the error in a program and again Compile until getting the error free code.
5. Simulate the program. Add the waveform with input and output signals in the program.
6. Apply different values to the input signals and observe the output signals for corresponding inputs.

LOGICDIAGRAMS ANDTRUTH TABLES:

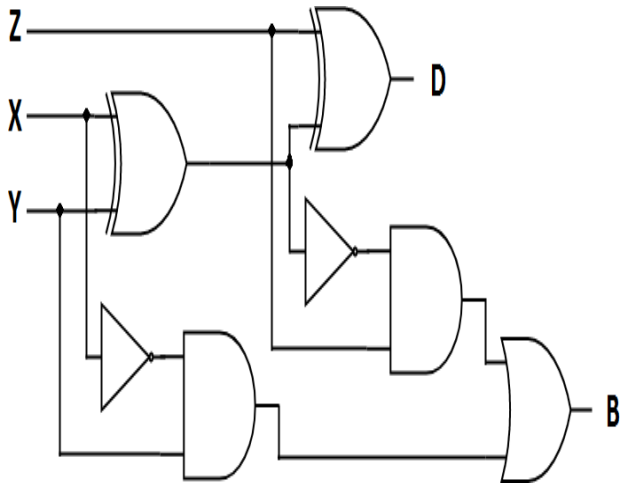
HALF SUBTRACTOR:



Inputs		Outputs	
X	Y	D	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

D=X⊕YB=
X'Y

FULL SUBTRACTOR:



Inputs			Outputs	
X	Y	Z	D	B
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

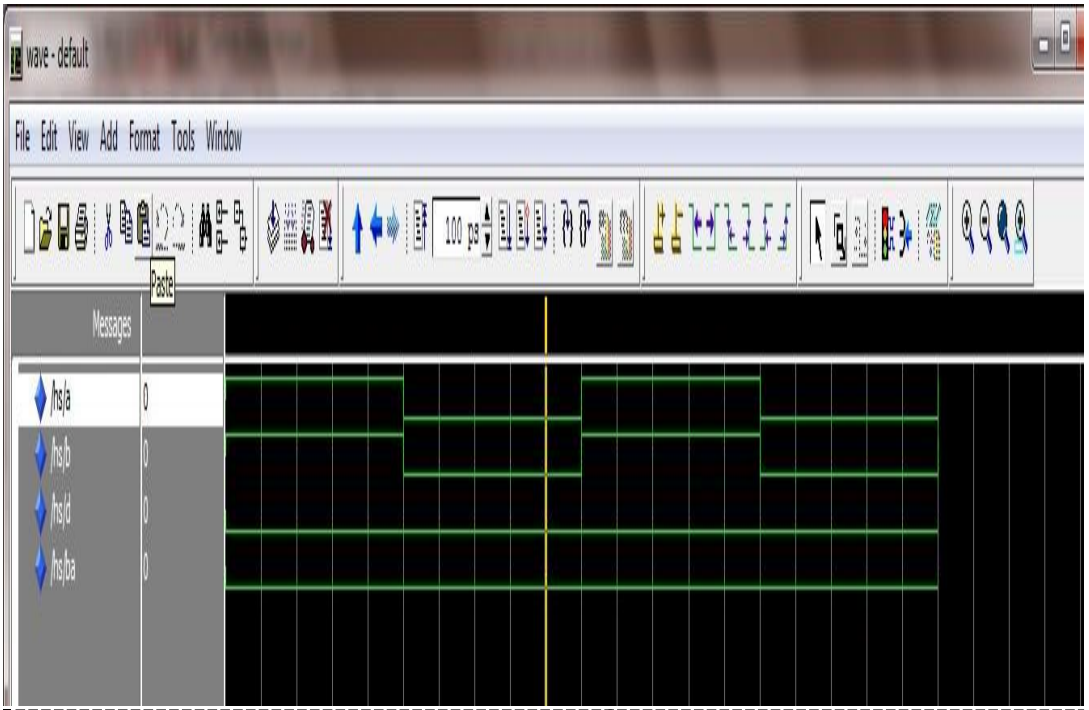
D=X⊕Y⊕Z

B=X'(Y⊕Z)+YZ

VHDL PROGRAMS SUBTRACTORS:HALF-SUBTRACTOR:

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

ENTITY HALFSUBTRACTOR IS
PORT(A,B:IN STD_LOGIC;
        D,BA:OUT STD_LOGIC);
        END HALFSUBTRACTOR;

ARCHITECTURE HALFSUBTRACTOR_DATA OF HALFSUBTRACTOR IS
BEGIN
                D<=A XOR B;
                BA<=(NOT A) AND B;
END HALFSUBTRACTOR_DATA;
```

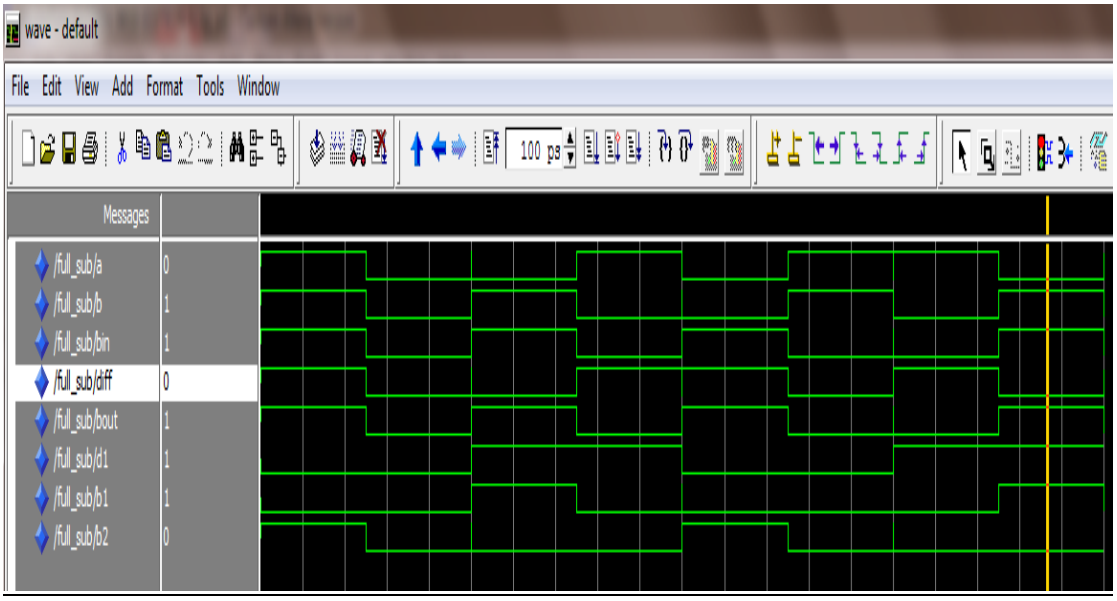
SIMULATION RESULTS:



FULL-SUBTRACTOR:

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

  ENTITY FULL_SUB IS
        PORT (A, B, BIN: IN STD_LOGIC;

         DIFF,BOUT:OUT STD_LOGIC);
  END FULL_SUB;
ARCHITECTURE FULLSUB_BEHAVE OF FULL_SUB IS

BEGIN
          PROCESS(A,B,BIN)
           BEGIN
        DIFF<=A XOR B XOR BIN;
        BOUT<=(NOT A AND B) OR (B AND BIN) OR (BIN AND NOT A);

        END PROCESS;
END FULLSUB_BEHAVE;
```

SIMULATIONRESULTS:



RESULT:

EXPNO:4 **DATE:**

3-8 DECODER

AIM: To write and simulate a VHDL Program for 3-8 decoder by using Modelsim.

SOFTWARESREQUIRED:

- System with Modelsim 6.3 Version.

THEORY:

A decoder is a multiple-input, multiple-output logic circuit that converts coded inputs into coded outputs, where the input and output codes are different. The 74x138 is a commercially available MSI 3 to 8 decoder. It has a 3-bit binary input code and a 1-out-of-2^3 output code. The input code word A, B, C represents an integer in the range 0 –7, the output code word Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7 which are active low outputs has Yi equal to 1 if and only if the input code word is the binary representation of "I" and G1 = 1, G2 A_L = 0, G2 B_L = 0, where G1, G2A_L, G2B_L are three enable inputs. An output is asserted if and only if the decoder is enabled and the output is selected.

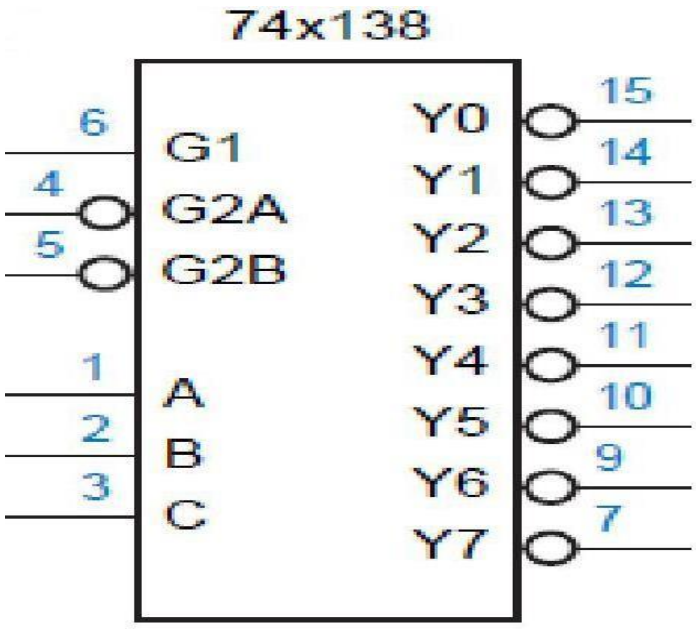
PROCEDURE:

1. Switch on the system and open the Modelsim Window.
2. Open the new editor window through File→New→Source→VHDL.
3. Type the VHDL Program in editor window and save the program with **file_name.vhd**.
4. Compile the program. If there is any error found, rectify the error in a program and again Compile until getting the error free code.
5. Simulate the program. Add the waveform with input and output signals in the program.
6. Apply different values to the input signals and observe the output signals for corresponding inputs.

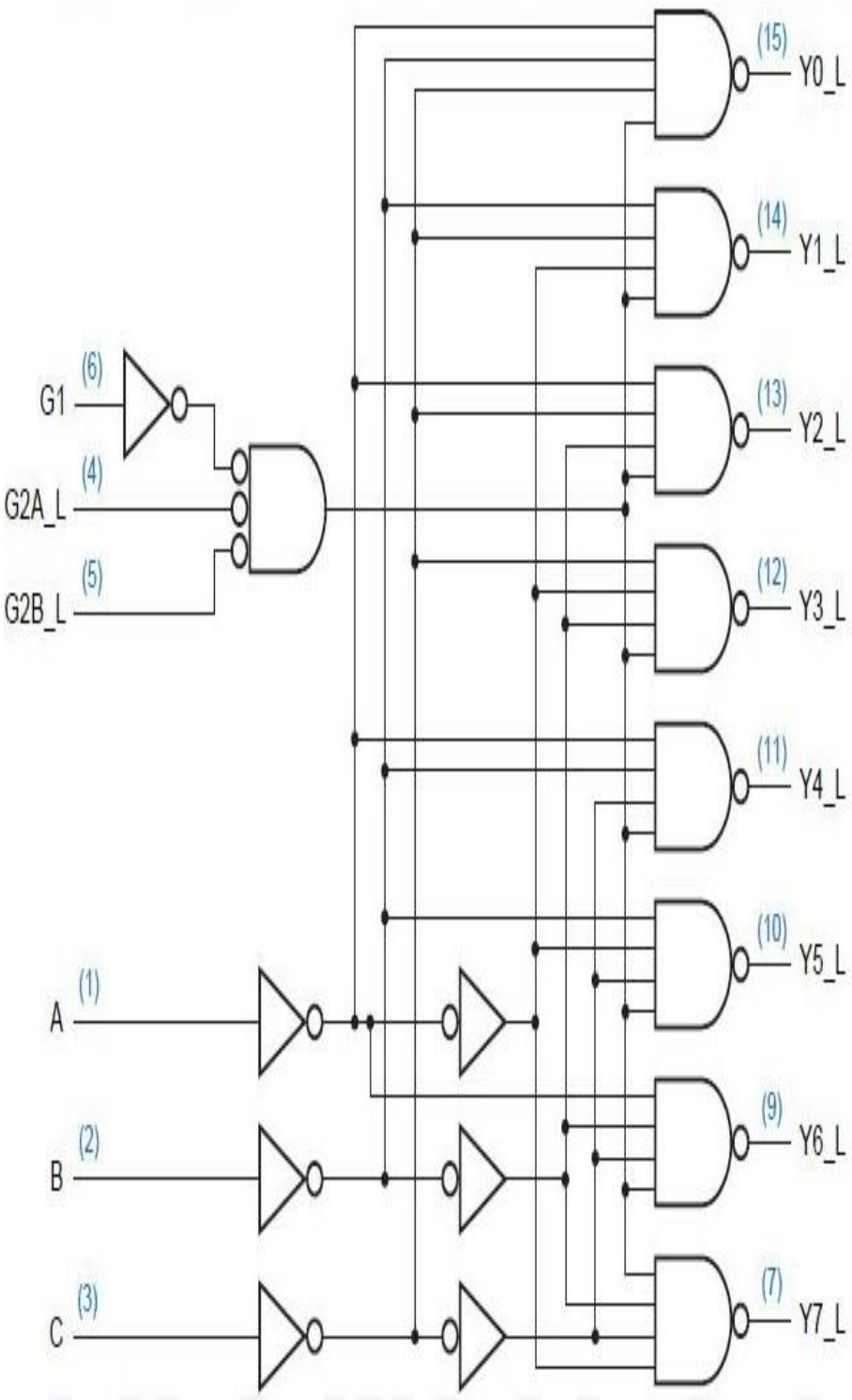
TRUTHTABLE:

Inputs						Outputs							
G1	G2A_L	G2B_L	C	B	A	Y7_L	Y6_L	Y5_L	Y4_L	Y3_L	Y2_L	Y1_L	Y0_L
0	x	x	x	x	x	1	1	1	1	1	1	1	1
x	1	x	x	x	x	1	1	1	1	1	1	1	1
x	x	1	x	x	x	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	1	1	1	1	1	1	0
1	0	0	0	0	1	1	1	1	1	1	1	0	1
1	0	0	0	1	0	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	0	1	1	1
1	0	0	1	0	0	1	1	1	0	1	1	1	1
1	0	0	1	0	1	1	1	0	1	1	1	1	1
1	0	0	1	1	0	1	0	1	1	1	1	1	1
1	0	0	1	1	1	0	1	1	1	1	1	1	1

LOGICSYMBOL:



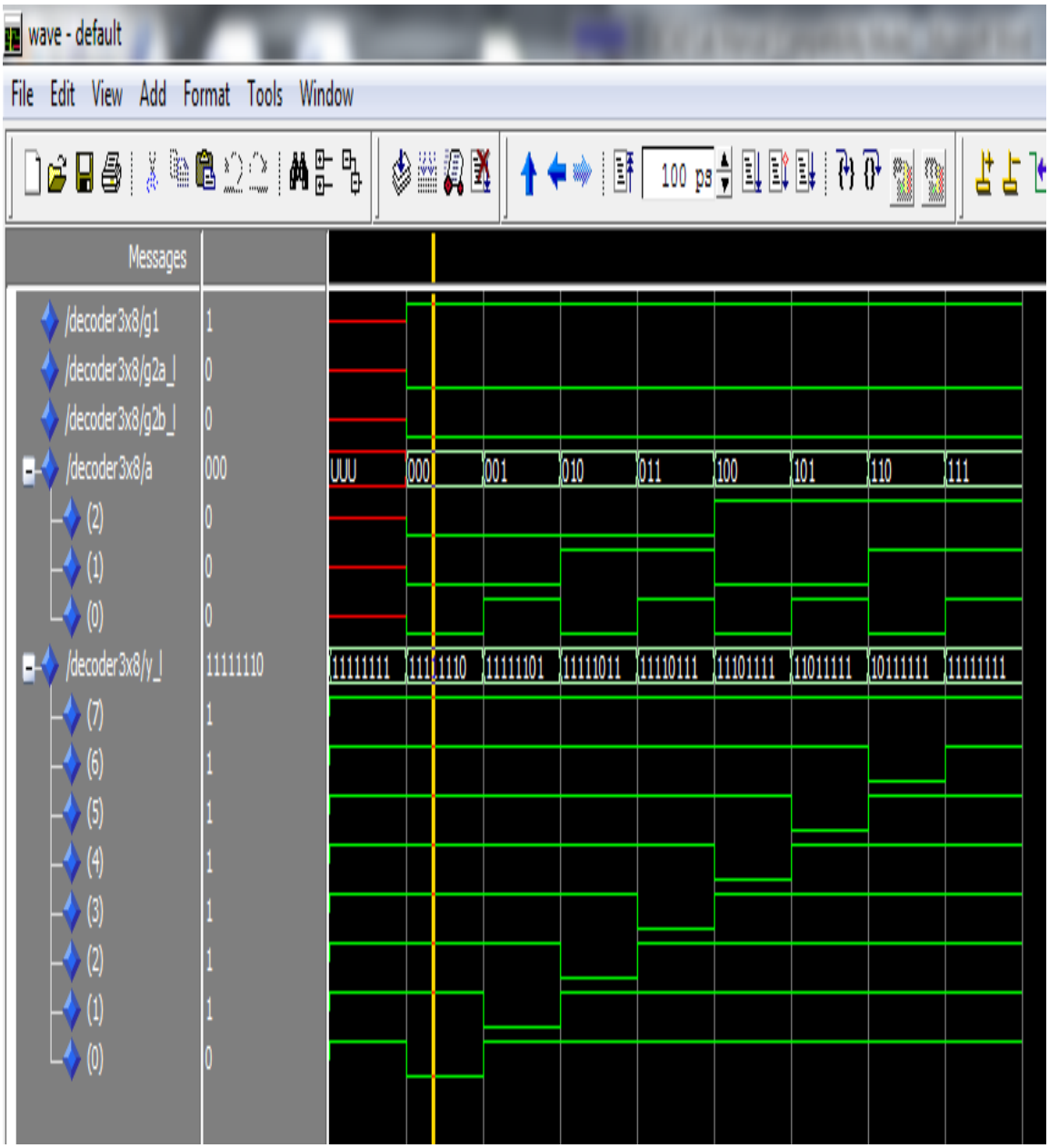
LOGICDIAGRAM:



VHDL PROGRAM FOR 3-8 DECODER:

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

ENTITY DEC3TO8 IS
   PORT(G1,G2A_L, G2B_L:IN STD_LOGIC;
            A:IN STD_LOGIC_VECTOR(2 DOWNTO 0);
            Y_L: OUT STD_LOGIC_VECTOR (0 TO 7));
            END DEC3TO8;

ARCHITECTURE DATAFLOW OF DEC3TO8 IS
 SIGNALY_L_I:STD_LOGIC_VECTOR(0 TO 7);
        BEGIN
        WITH A SELECT Y_L_I<="11111110 000",
                    "11111101" WHEN "001",
                    "11111011" WHEN "010",
                    "11110111" WHEN "011",
                    "11101111" WHEN "100",
                    "11011111" WHEN "101",
                    "10111111" WHEN "110",
                    "01111111" WHEN "111",
                    "01111111" WHENOTHERS;
         Y_L<=Y_L_I WHEN (G1AND(NOT G2A_L) AND (NOT G2B_L))='1' ELSE
                  "11111111";
END DATAFLOW;
```

SIMULATIONRESULTS:



RESULT:

EXP NO:5 **DATE:**

8-3 PRIORITY ENCODER

AIM: To write and simulate a VHDL Program for 8-3 priority encoder by using Modelsim.

SOFTWARESREQUIRED:

- System with Modelsim6.3 Version.

THEORY:

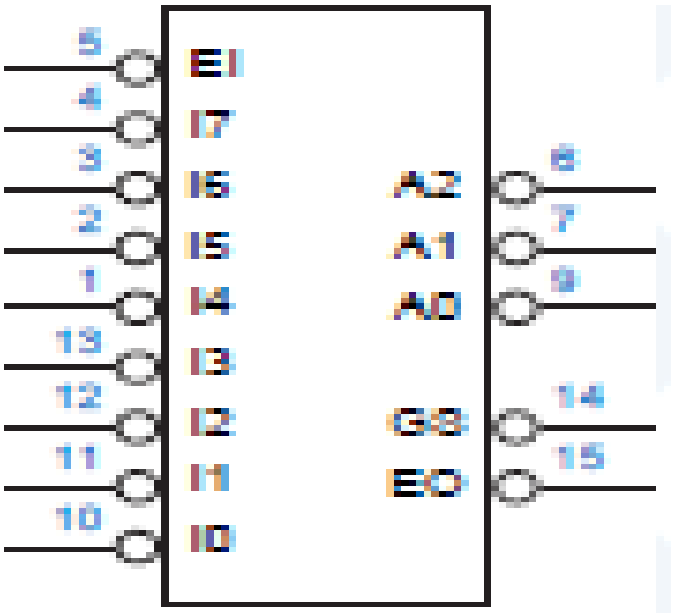
A priority encoder is a circuit or algorithm that compresses multiple binary inputs into a smaller number of outputs. The output of a priority encoder is the binary representation of the original number starting from zero of the most significant input bit. They are often used to control interrupt requests by acting on the highest priority encoder. If two or more inputs are given at the same time, the input having the highest priority will take precedence.

The 74x148 is a commercially available, MSI 8-input priority encoder it has an enable input, EI_L that must be asserted for any of its outputs to be asserted. Instead of an IDLE output, the "148 has a GS_L output that is asserted when the device is enabled and one or more of the request inputs are asserted. The manufacturer calls this "Group Select," but it is easier to remember as "Got Something". The EO_L signal is an enable output designed to be connected to the EI_L input of another "148 that handles lower-priority requests. /EO is asserted if EI_L is asserted but no request input is asserted; thus, a lower-priority"148maybe enabled.

PROCEDURE:

1. Switch on the system and open the Modelsim Window.
2. Open the new editor window through File→New→Source→VHDL.
3. Type the VHDL Program in editor window and save the program with **file_name.vhd**.
4. Compile the program. If there is any error found, rectify the error in a program and again Compile until getting the error free code.
5. Simulate the program. Add the waveform with input and output signals in the program.
6. Apply different values to the input signals and observe the output signals for corresponding inputs.

LOGICSYMBOL:



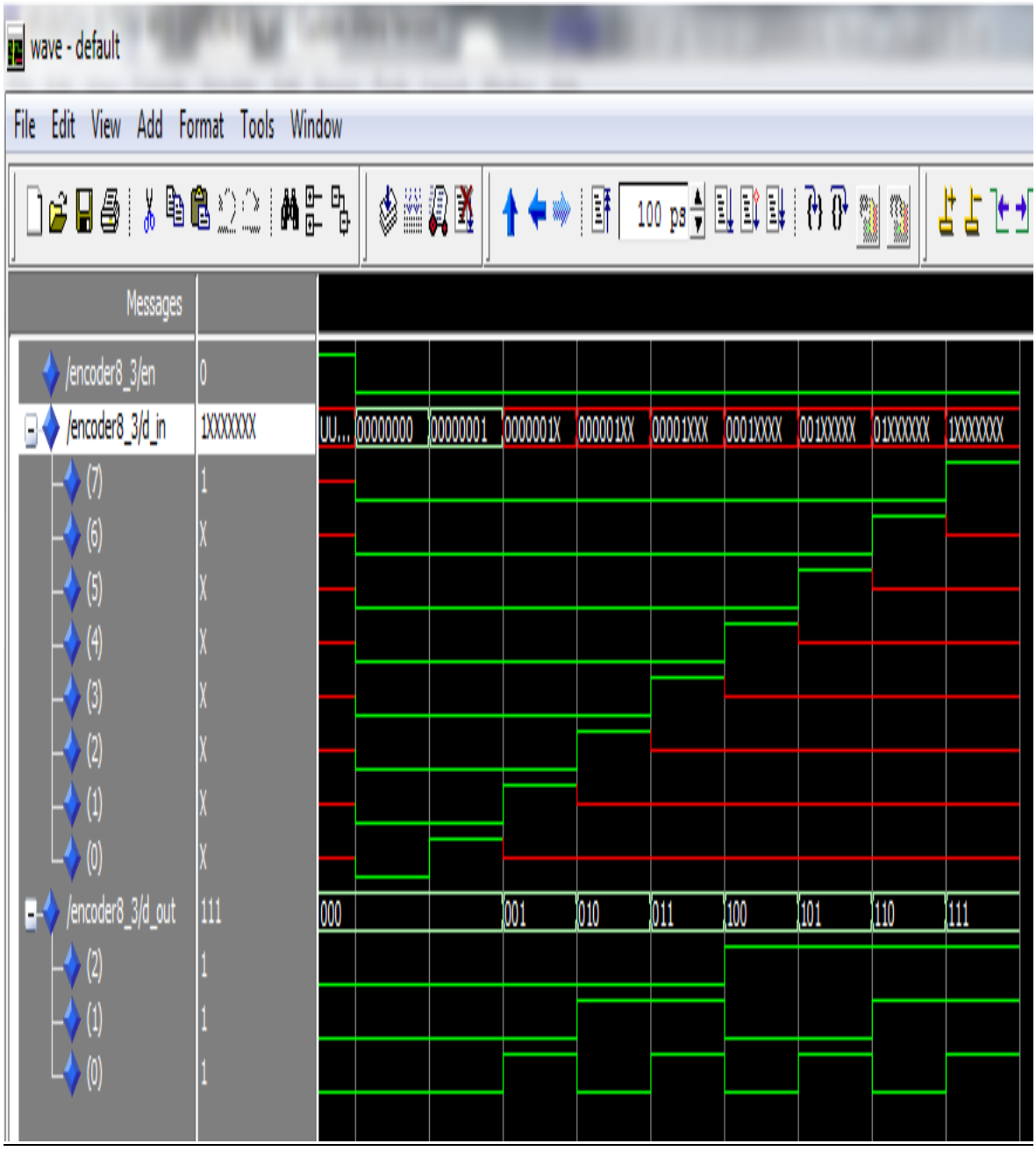
TRUTHTABLE:

Inputs									Outputs				
$\overline{E1}$	$\overline{I_0}$	$\overline{I_1}$	$\overline{I_2}$	$\overline{I_3}$	$\overline{I_4}$	$\overline{I_5}$	$\overline{I_6}$	$\overline{I_7}$	$\overline{GS}$	$\overline{A_0}$	$\overline{A_1}$	$\overline{A_2}$	$\overline{EO}$
H	X	X	X	X	X	X	X	X	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	L
L	X	X	X	X	X	X	X	L	L	L	L	L	H
L	X	X	X	X	X	X	L	H	L	H	L	L	H
L	X	X	X	X	X	L	H	H	L	L	H	L	H
L	X	X	X	X	L	H	H	H	L	H	H	L	H
L	X	X	X	L	H	H	H	H	L	L	L	H	H
L	X	X	L	H	H	H	H	H	L	H	L	H	H
L	X	L	H	H	H	H	H	H	L	L	H	H	H
L	L	H	H	H	H	H	H	H	L	H	H	H	H

VHDLPROGRAMFOR 8-3PRIORITYENCODER:

```
LIBRARYIEEE;
USEIEEE.STD_LOGIC_1164.ALL;

  ENTITY ENCODER8_3 IS
    PORT(EN:IN STD_LOGIC;
              D_IN: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
              D_OUT:OUTSTD_LOGIC_VECTOR(2 DOWNTO 0));
     END ENCODER8_3;
  ARCHITECTURE ENCODER_ARCH OF ENCODER8_3 IS
  BEGIN
       PROCESS (EN, D_IN)
       BEGIN
            IF (EN = '1') THEN
            D_OUT<="000";
            ELSE
              CASE D_IN IS
                WHEN "00000001" => D_OUT <= "000";
                WHEN "0000001X" => D_OUT <= "001";
                WHEN "000001XX" => D_OUT <= "010";
                WHEN "00001XXX" => D_OUT <= "011";
                WHEN "0001XXXX" => D_OUT <= "100";
                WHEN "001XXXXX" => D_OUT <= "101";
                WHEN "01XXXXXX" => D_OUT <= "110";
                WHEN"1XXXXXXX"=>D_OUT<="111";
                WHEN OTHERS=> NULL;
              END CASE;
              END IF;
      END PROCESS;
  END ENCODER_ARCH;
```

SIMULATIONRESULTS:



RESULT:

EXP NO:6 **DATE:**

8:1 MULTIPLEXER

AIM: To write and simulate a VHDL Program for 8X1 Multiplexer by using Modelsim.

SOFTWARES REQUIRED:

- System with Modelsim 6.3 Version.

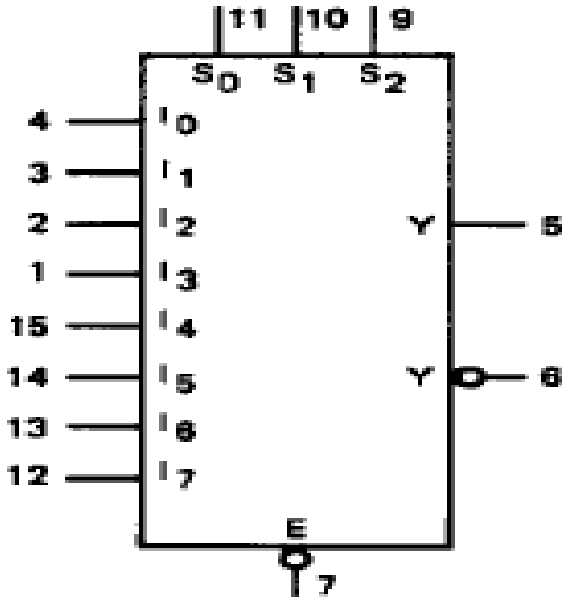
THEORY:

Multiplexing means transmitting a large number of information units over a smaller number of channels or lines. A digital multiplexer is a combinational circuit that selects binary information from one of many inputs lines and directs it to a single output line. Normally there are 2^n input lines and n selection lines whose bit combinations determine which input is selected. The selection depends on set of selection lines also called as selector. In 8 to1 multiplexer, there are 3 select lines and 2^3 minterms by connecting the function variables directly to select inputs, a multiplexer can be made to a select and AND gate that corresponds to the minterms in the function. The IC 74X151 is an 8-1 multiplexer. It has eight inputs. It provides two outputs, one is active high, and the other is active low.

PROCEDURE:

1. Switch on the system and open the Modelsim Window.
2. Open the new editor window through File→New→Source→VHDL.
3. Type the VHDL Program in editor window and save the program with **file_name.vhd**.
4. Compile the program. If there is any error found, rectify the error in a program and again Compile until getting the error free code.
5. Simulate the program. Add the waveform with input and output signals in the program.
6. Apply different values to the input signals and observe the output signals for corresponding inputs.

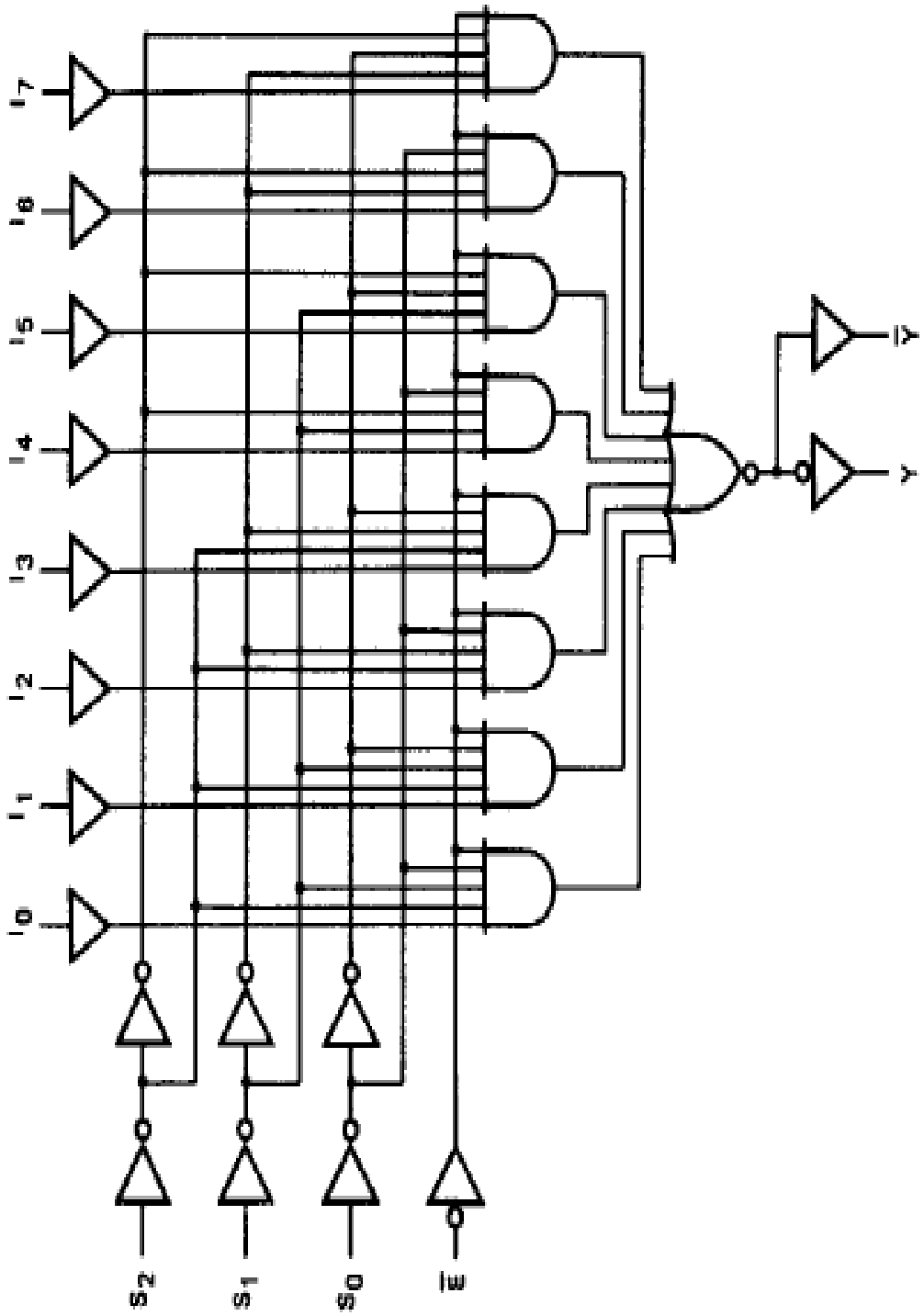
LOGICSYMBOL:



TRUTHTABLE:

INPUTS												OUTPUTS	
$\overline{E}$	S_2	S_1	S_0	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	$\overline{Y}$	Y
H	X	X	X	X	X	X	X	X	X	X	X	H	L
L	L	L	L	L	X	X	X	X	X	X	X	H	L
L	L	L	L	H	X	X	X	X	X	X	X	L	H
L	L	L	H	X	L	X	X	X	X	X	X	H	L
L	L	L	H	X	H	X	X	X	X	X	X	L	H
L	L	H	L	X	X	L	X	X	X	X	X	H	L
L	L	H	L	X	X	H	X	X	X	X	X	L	H
L	L	H	H	X	X	X	L	X	X	X	X	H	L
L	L	H	H	X	X	X	H	X	X	X	X	L	H
L	H	L	L	X	X	X	X	L	X	X	X	H	L
L	H	L	L	X	X	X	X	H	X	X	X	L	H
L	H	L	H	X	X	X	X	X	L	X	X	H	L
L	H	L	H	X	X	X	X	X	H	X	X	L	H
L	H	H	L	X	X	X	X	X	X	L	X	H	L
L	H	H	L	X	X	X	X	X	X	H	X	L	H
L	H	H	H	X	X	X	X	X	X	X	L	H	L
L	H	H	H	X	X	X	X	X	X	X	H	L	H

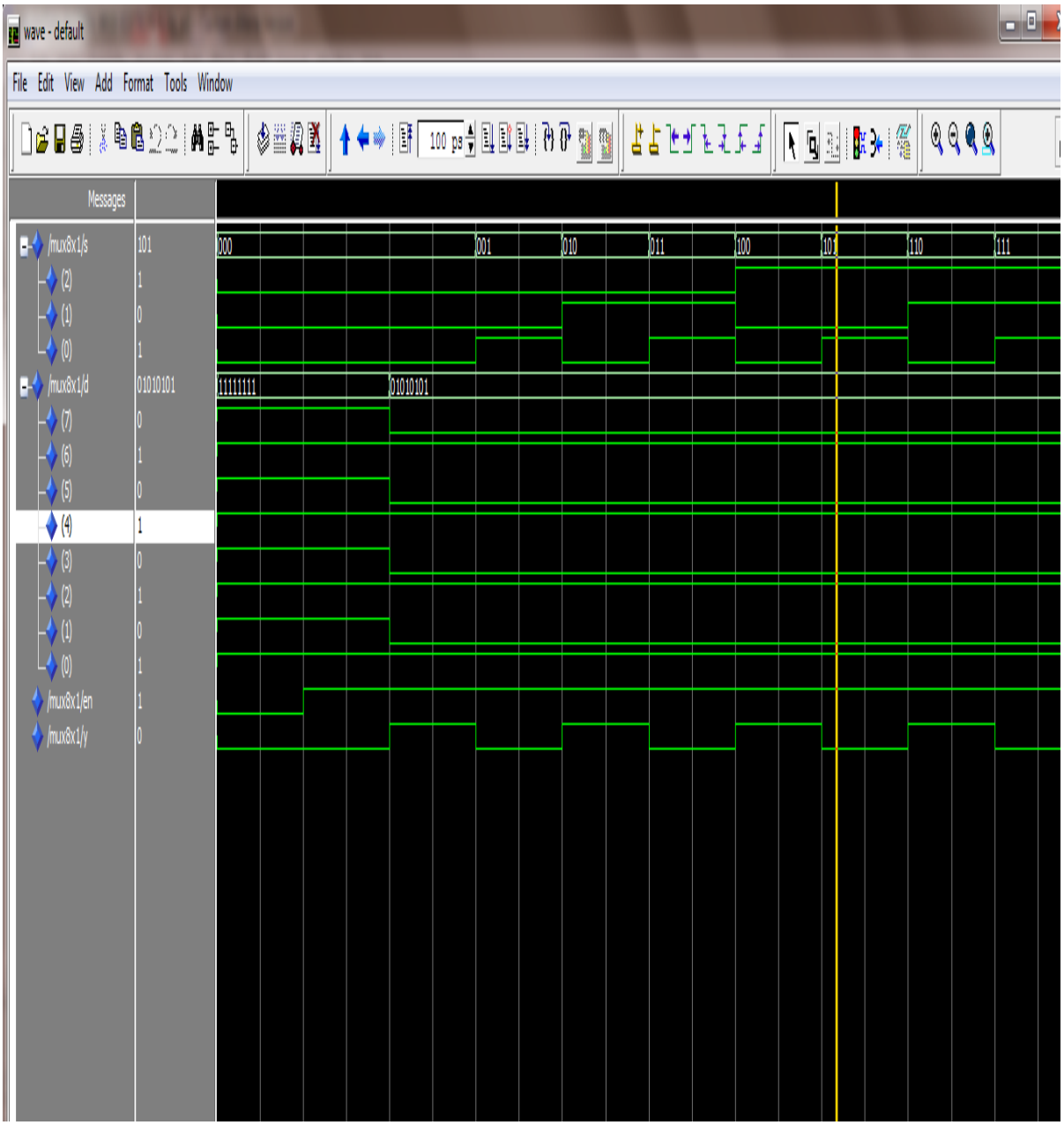
LOGICDIAGRAM:



VHDL PROGRAM FOR 8:1 MULTIPLEXER:

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

  ENTITY MUX8X1 IS
    PORT (S: IN STD_LOGIC_VECTOR (2 DOWNTO 0);
    D: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
    EN:IN STD_LOGIC;
            Y: OUT STD_LOGIC);
            END MUX8X1;
  ARCHITECTURE MUX_8X1 OF MUX8X1 IS
  BEGIN
     PROCESS (S, D, EN)
     BEGIN
    IF (EN='0') THEN Y<='0';
    ELSE
            CASE S IS
                    WHEN "000"=>Y<=D(0);
                    WHEN "001"=>Y<=D(1);
                    WHEN "010"=>Y<=D(2);
                    WHEN "011"=>Y<=D(3);
                    WHEN "100"=>Y<=D(4);
                    WHEN "101"=>Y<=D(5);
                    WHEN "110"=>Y<=D(6);
                    WHEN "111"=>Y<=D (7);
                    WHENOTHERS=>Y<='0';
            END CASE;
            END IF;
    END PROCESS;
    END MUX_8X1;
```

SIMULATIONRESULTS:



RESULT:

EXP NO:7 **DATE:**

1:4 DEMULTIPLEXER

AIM: To write and simulate a VHDL Program for 1:4 Demultiplexer by using Modelsim.

SOFTWARESREQUIRED:

- System with Modelsim6.3 Version.

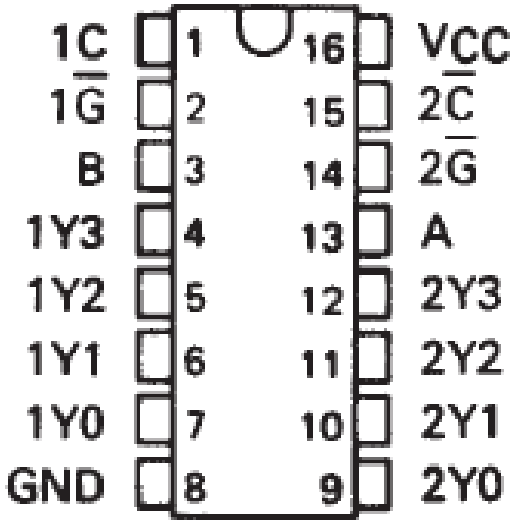
THEORY:

A demultiplexer (or demux) is a device that takes a single input line and routes it to one of several digital output lines. A demultiplexer of 2^n outputs has n select lines, which are used to select which output line to send the input. A demultiplexeris also called a data distributor. Demultiplexers can be used to implement general purpose logic. By setting the input to true, the demux behaves as a decoder.

PROCEDURE:

1. Switch on the system and open the Modelsim Window.
2. Open the new editor window through File→New→Source→VHDL.
3. Type the VHDL Program in editor window and save the program with **file_name.vhd**.
4. Compile the program. If there is any error found, rectify the error in a program and again Compile until getting the error free code.
5. Simulate the program. Add the waveform with input and output signals in the program.
6. Apply different values to the input signals and observe the output signals for corresponding inputs.

PINCONFIGURATION:

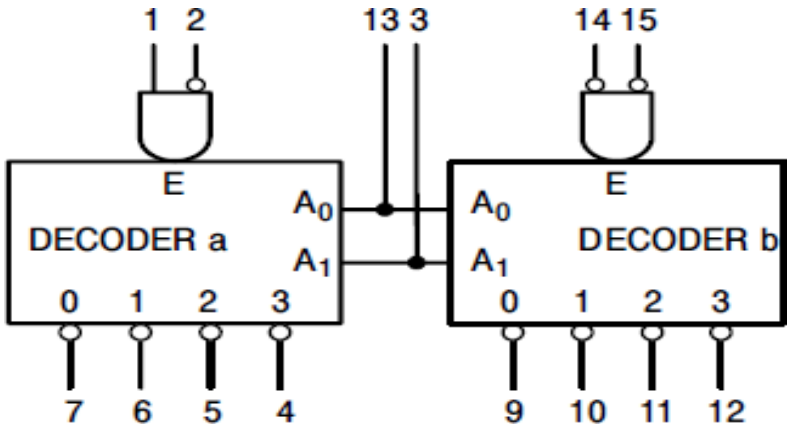


TRUTHTABLE:

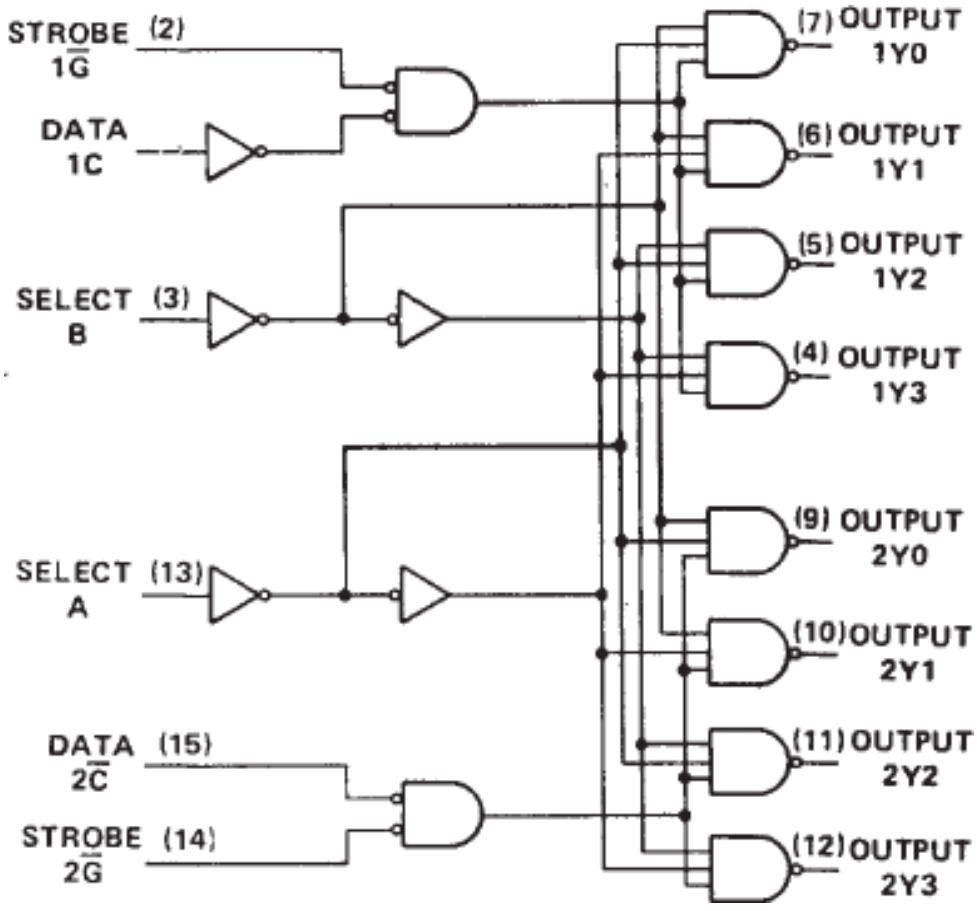
INPUTS				OUTPUTS			
SELECT		STROBE	DATA				
B	A	$1\overline{G}$	1C	1Y0	1Y1	1Y2	1Y3
X	X	H	X	H	H	H	H
L	L	L	H	L	H	H	H
L	H	L	H	H	L	H	H
H	L	L	H	H	H	L	H
H	H	L	H	H	H	H	L
X	X	X	L	H	H	H	H

INPUTS				OUTPUTS			
SELECT		STROBE	DATA				
B	A	$2\overline{G}$	$2\overline{C}$	2Y0	2Y1	2Y2	2Y3
X	X	H	X	H	H	H	H
L	L	L	L	L	H	H	H
L	H	L	L	H	L	H	H
H	L	L	L	H	H	L	H
H	H	L	L	H	H	H	L
X	X	X	H	H	H	H	H

LOGICSYMBOL:



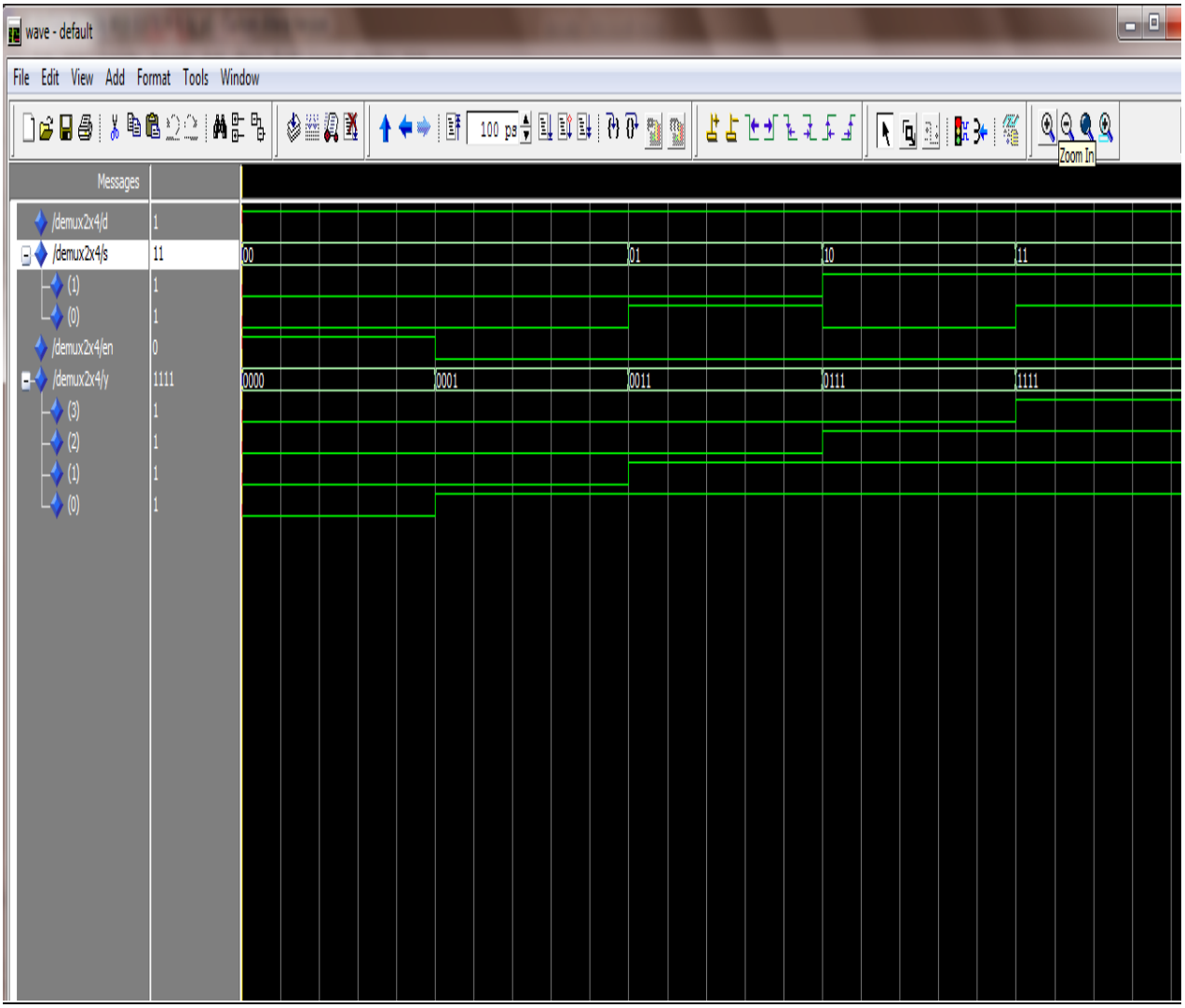
LOGICDIAGRAM:



VHDLPROGRAMFOR1X4MULTIPLEXER:

```
LIBRARY IEEE;
USEIEEE.STD_LOGIC_1164.ALL;

ENTITY DEMUX1to4 IS
    PORT(D,EN:IN STD_LOGIC;
            S:IN STD_LOGIC_VECTOR(1DOWNTO 0);
            Y: OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
            END DEMUX1to4;

  ARCHITECTURE DEMUX_1to4 OF DEMUX1to4 IS
  BEGIN
      PROCESS (S, EN, D)
      BEGIN
          IF(EN='0')THEN
                    Y<="0000";
              ELSE
              CASE S IS
                    WHEN "00"=>Y(0)<=D;
                    WHEN "01"=>Y(1)<=D;
                    WHEN "10"=>Y(2)<=D;
                    WHEN "11"=>Y(3)<=D;
                    WHENOTHERS=>Y<="0000";
       END CASE;
       END IF;
       END PROCESS;
   END DEMUX_2X4;
```

SIMULATIONRESULTS:



RESULT:

EXPNO: 8 **DATE:**

4-BIT COMPARATOR

AIM: To write and simulate a VHDL Program for 4-bit Comparator by using Modelsim.

SOFTWARES REQUIRED:

- System with Modelsim6.3 Version.

THEORY:

The 74X85 is a 4-bit magnitude comparator that can be expanded to almost any length. It compares two 4-bit binary, BCD, or other monotonic codes and presents the three possible magnitude results at the outputs. The 4-bit inputs are weighted (A0–A3) and (B0–B3) where A3 and B3 are the most significant bits. The operation of the 74X85 is described in the Function Table, showing all possible logic conditions. The upper part of the table describes the normal operation under all conditions that will occur in a single device or in a series expansion scheme. In the upper part of the table the three outputs are mutually exclusive. In the lower part of the table, the outputs reflect the feed-forward conditions that exist in the parallel expansion scheme.

The expansion inputs I_A>B, and I_A=B and I_A<B are the least significant bit positions. When used for series expansion, the A>B, A=B and A<B outputs of the lease significant word are connected to the corresponding I_A>B, I_A=B and I_A<B inputs of the next higher stage. Stages can be added in this manner to any length, but a propagation delay penalty of about 15ns is added with each additional stage. For proper operation, the expansion inputs of the least significant word should be tied as follows: I_A>B=Low, I_A=B=High, and I_A<B=Low.

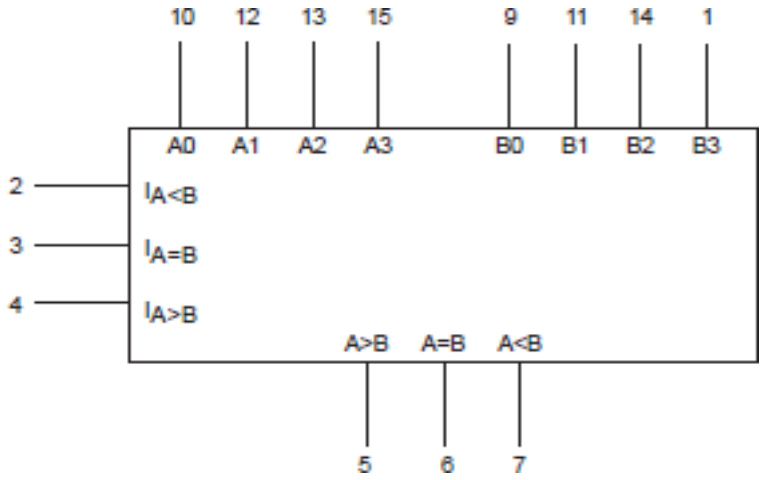
PROCEDURE:

1. Switch on the system and open the Modelsim Window.
2. Open the new editor window through File→New→Source→VHDL.
3. Type the VHDL Program in editor window and save the program with **file_name.vhd**.
4. Compile the program. If there is any error found, rectify the error in a program and again Compile until getting the error free code.
5. Simulate the program. Add the waveform with input and output signals in the program.
6. Apply different values to the input signals and observe the output signals for corresponding inputs.

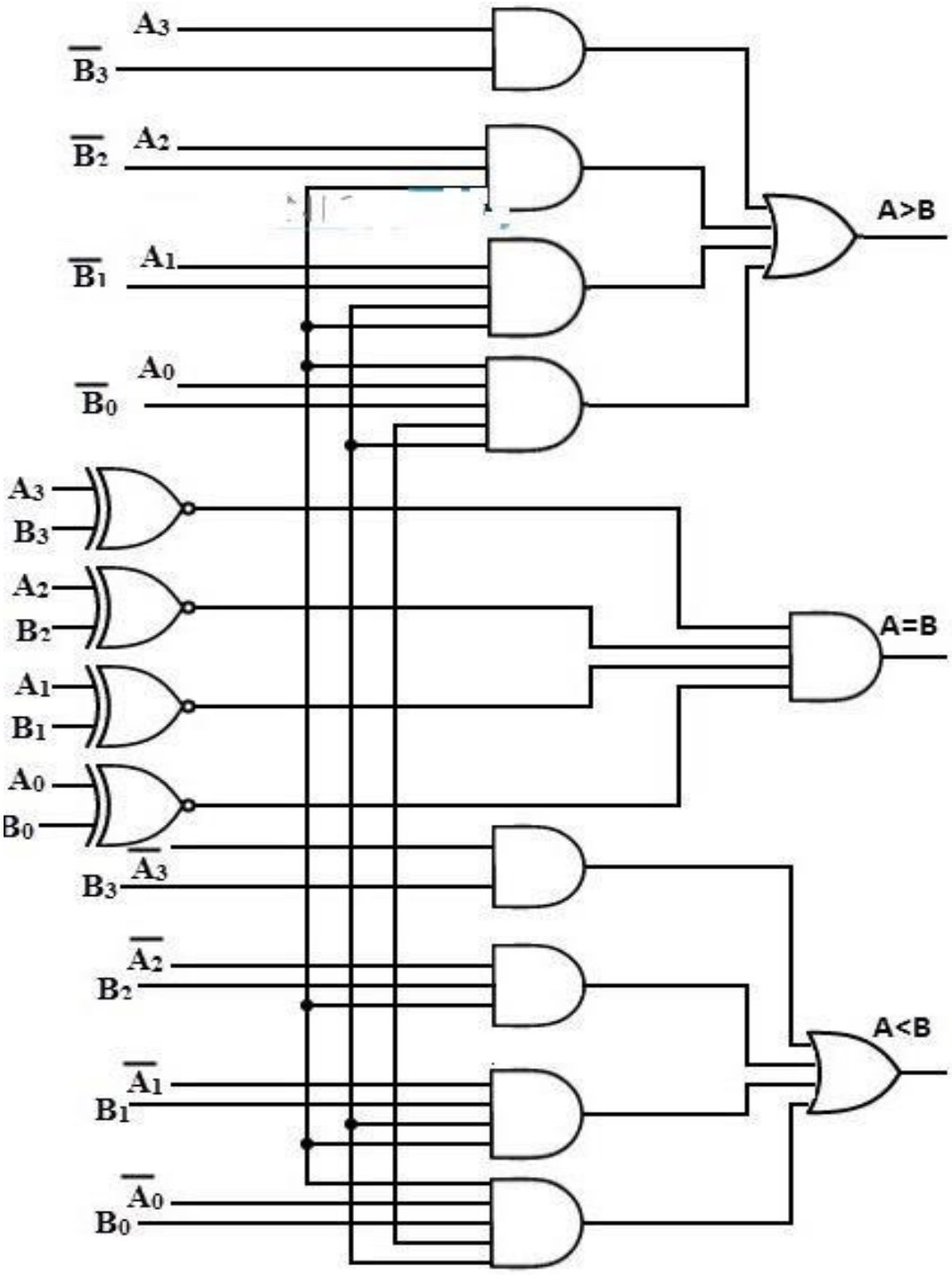
TRUTHTABLE:

COMPARING INPUTS				EXPANSION INPUTS			OUTPUTS		
A3,B3	A2,B2	A1,B1	A0,B0	$I_{A>B}$	$I_{A<B}$	$I_{A=B}$	A>B	A<B	A=B
A3>B3	X	X	X	X	X	X	H	L	L
A3<B3	X	X	X	X	X	X	L	H	L
A3=B3	A2>B2	X	X	X	X	X	H	L	L
A3=B3	A2<B2	X	X	X	X	X	L	H	L
A3=B3	A2=B2	A1>B1	X	X	X	X	H	L	L
A3=B3	A2=B2	A1<B1	X	X	X	X	L	H	L
A3=B3	A2=B2	A1=B1	A0>B0	X	X	X	H	L	L
A3=B3	A2=B2	A1=B1	A0<B0	X	X	X	L	H	L
A3=B3	A2=B2	A1=B1	A0=B0	H	L	L	H	L	L
A3=B3	A2=B2	A1=B1	A0=B0	L	H	L	L	H	L
A3=B3	A2=B2	A1=B1	A0=B0	L	L	H	L	L	H
A3=B3	A2=B2	A1=B1	A0=B0	X	X	H	L	L	H
A3=B3	A2=B2	A1=B1	A0=B0	H	H	L	L	L	L
A3=B3	A2=B2	A1=B1	A0=B0	L	L	L	H	H	L

LOGICSYMBOL:



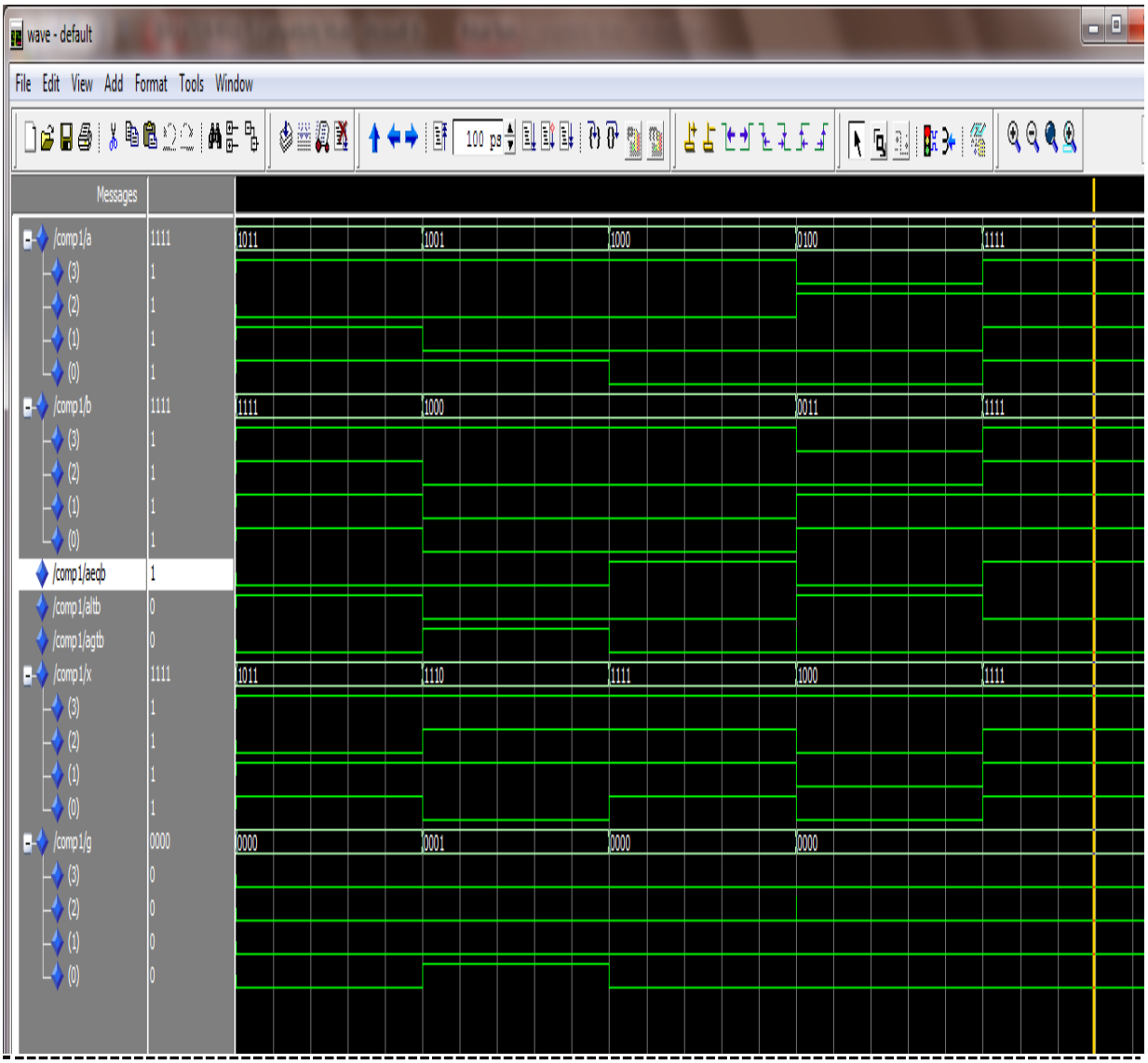
LOGICDIAGRAM:



VHDL PROGRAM FOR 4-BIT COMPARATOR:

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

ENTITY COMP IS
 PORT(A,B:IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        AGTB,ALTB,AEQB: OUT STD_LOGIC);
END COMP;

ARCHITECTURE COMP_BEH OF COMP IS BEGIN
        PROCESS (A, B)
        BEGIN
                IF (A>B) THEN
                        AGTB<= '1';
                        AEQB<='0';
                        ALTB<='0';
                ELSIF (A<B) THEN
                        AGTB<= '0';
                        AEQB<='0';
                        ALTB<='1';
                ELSIF (A=B) THEN
                        AGTB<= '0';
                        AEQB<='1';
                        ALTB<='0';
                END IF;
                END PROCESS;
END COMP_BEH;
```

SIMULATIONRESULTS:



RESULT:

EXPNO: 9 **DATE:**

JK-FLIPFLOP

AIM: To write and simulate a VHDL Program for JK-Flip-Flop by using Modelsim.

SOFTWARESREQUIRED:

- SystemwithModelsim6.3Version.

THEORY:

The name JK flip-flop is termed from the inventor Jack Kilby from Texas instruments. Due to its versatility they are available as IC packages. The major applications of JK flip-flop are Shift registers, storage registers, counters and control circuits. In spite of the simple wiring of D type flip-flop, JK flip-flop has a toggling nature. This has been an added advantage. Hence they are mostly used in counters and PWM generation, etc. Whenever the clock signal is LOW, the input is never going to affect the output state. The clock has to be high for the inputs to get active. Thus, JK flip-flop is a controlled Bi-stable latch where the clock signal is the control signal.

PROCEDURE:

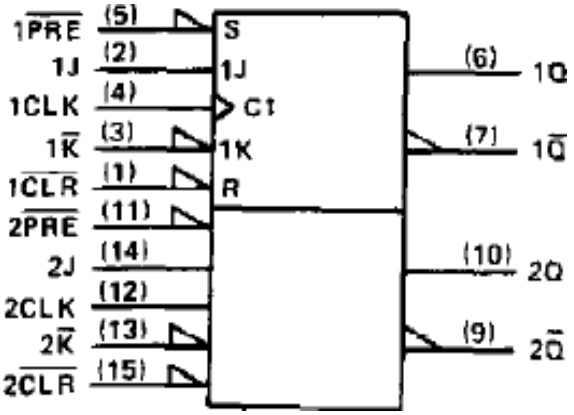
1. Switch on the system and open the Modelsim Window.
2. Open the new editor window through File→New→Source→VHDL.
3. Type the VHDL Program in editor window and save the program with **file_name.vhd**.
4. Compile the program. If there is any error found, rectify the error in a program and again Compile until getting the error free code.
5. Simulate the program. Add the waveform with input and output signals in the program.
6. Apply different values to the input signals and observe the output signals for corresponding inputs.

TRUTHTABLE:

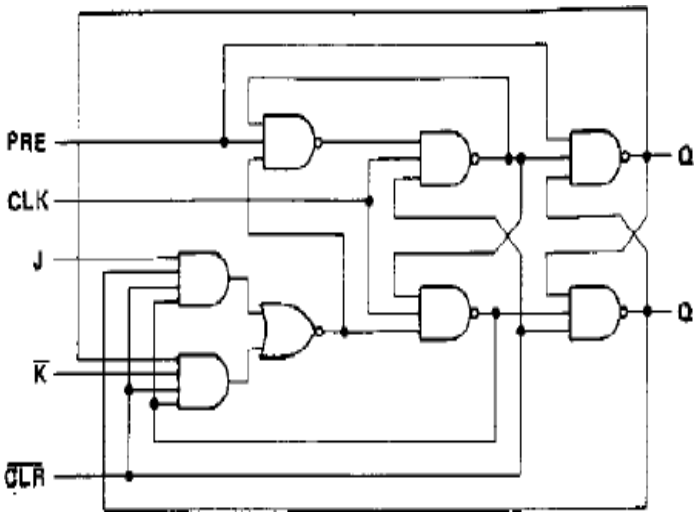
FUNCTION TABLE (each flip-flop)

INPUTS					OUTPUTS	
$\overline{PRE}$	$\overline{CLR}$	CLK	J	K	Q	$\overline{Q}$
L	H	X	X	X	H	L
H	L	X	X	X	L	H
L	L	X	X	X	H†	H†
H	H	↑	L	L	L	H
H	H	↑	H	L	TOGGLE	
H	H	↑	L	H	Q_0	$\overline{Q}_0$
H	H	↑	H	H	H	L
H	H	L	X	X	Q_0	$\overline{Q}_0$

LOGICSYMBOL:



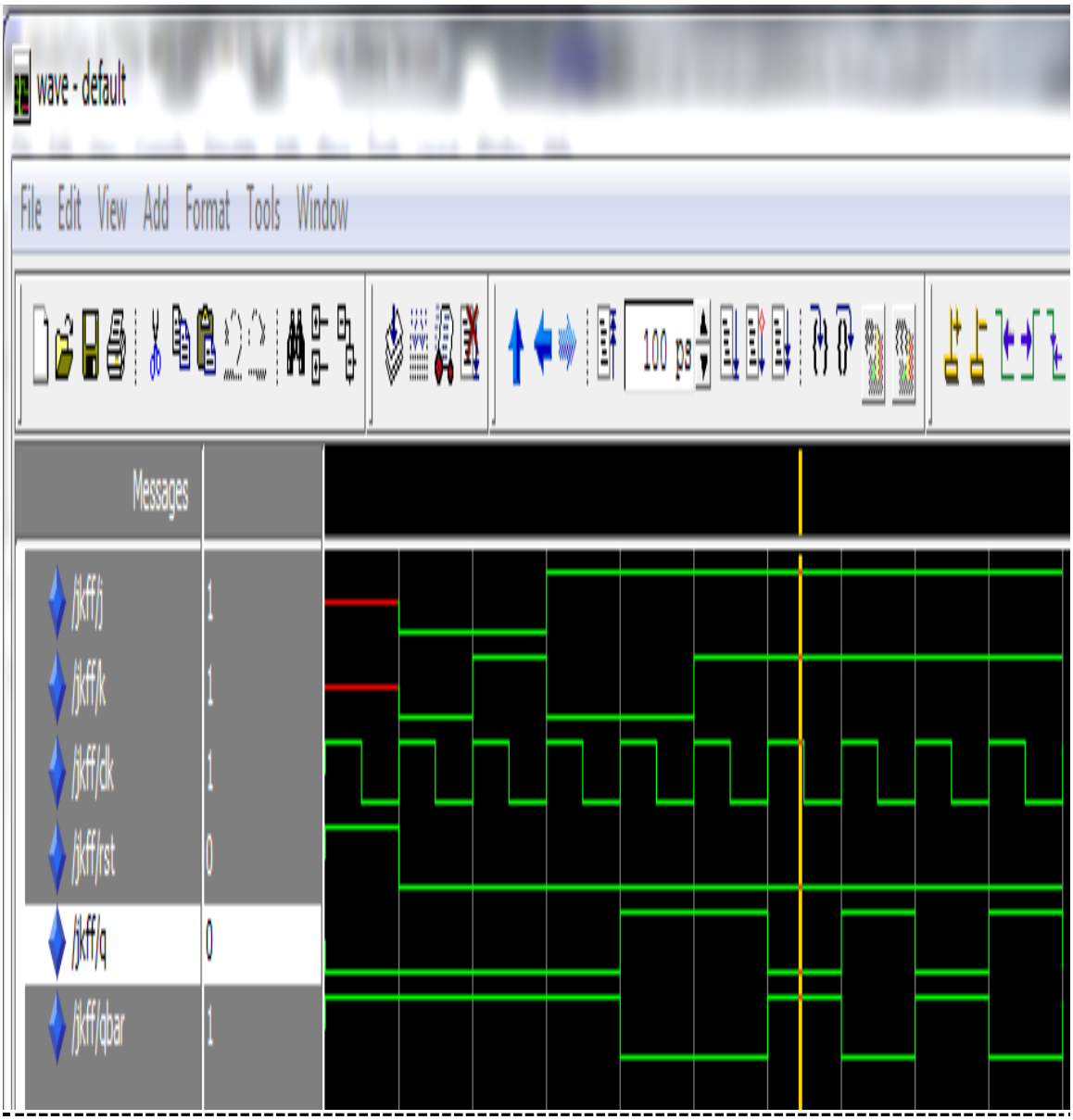
LOGICDIAGRAM:



VHDL PROGRAM FOR JK-FLIP-FLOP:

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

ENTITY JKFF IS
   PORT(J,K, CLK,RST:IN STD_LOGIC;
   Q:INOUT STD_LOGIC);
END JKFF;
ARCHITECTURE JKFF1 OF JKFF IS
SIGNAL QBAR: STD_LOGIC;
BEGIN
      PROCESS (CLK, J, K, RST)
      BEGIN
            IF(CLK='1' AND CLK'EVENT) THEN
            IF(RST='1')THEN
                  Q<='0';
              ELSE
                   Q<=(J AND NOTQ) OR( (NOT K) AND Q));
                   END IF;
            END IF;
            END PROCESS;
            QBAR<=NOT Q;
 END JKFF1;
```

SIMULATIONRESULTS:



RESULT:

EXPNO:10 **DATE:**

4-bit Synchronous Binary Counter

AIM: To write and simulate a VHDL Program for 4–bit synchronous binary counter by using Modelsim.

SOFTWARES REQUIRED:

- System with Modelsim 6.3 Version

THEORY:

This synchronous, pre settable, 4-bitbinary counter has internal carry look-ahead circuitry for use in high-speed counting designs. Synchronous operation is provided by having all flip-flops clocked simultaneously so that the outputs change coincident with each other when so instructed by the count-enable (ENP, ENT) inputs and internal gating. This mode of operation eliminates the output counting spikes that normally are associated with asynchronous (ripple-clock) counters. However, counting spikes can occur on the ripple-carry (RCO) output. A buffered clock (CLK) input triggers the four flip-flops on the rising (positive-going) edge of CLK.

This counter is fully programmable. That is, it can be preset to any number between 0 and 15.Because presetting is synchronous, a low logic level at the load (LOAD) input disables the counter andcausestheoutputstoagreewiththesetupdataafterthenextclockpulse, regardless of the levels of ENP and ENT. The clear function is synchronous, and a low logic level at the clear (CLR) input sets all four of the flip-flop outputs to low after the next low-to-high transition of the clock, regardless of the levels of ENP and ENT.

This synchronous clear allows the count length to be modified easily by decoding the Q outputs for the maximum count desired. The active-low output of the gate used for decoding is connected to the clear input to synchronously clear the counter to 0000(LLLL).

The carry look-ahead circuitry provides for cascading counters for n-bit synchronous applications, without additional gating. This function is implemented by the ENP and ENT inputs and an RCO output. Both ENP and ENT must be high to count, and ENT is fed forward to enable RCO.RCO, thus enabled, produces a high-logic-level pulse while the count is 15(HHHH). The high-logic- level overflow ripple-carry pulse can be used to enable successive cascaded stages. Transitions at ENP or ENT are allowed,regardless of the level of CLK.

The 74163 features a fully independent clock circuit. Changes at ENP, ENT, or LOAD that

modify the operating mode have no effect on the contents of the counter until clocking occurs. The function of the counter (whether enabled, disabled, loading, or counting) is dictated solely by the conditions meeting the setup and hold times.

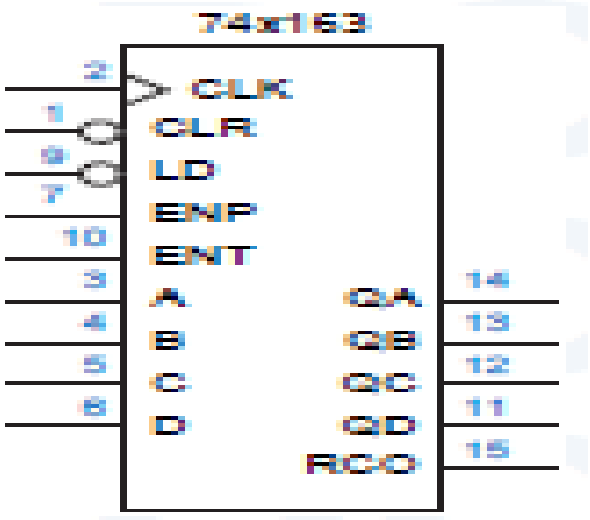
PROCEDURE:

1. Switch on the system and open the Model sim Window.
2. Open the new editor window through File→New→Source→VHDL.
3. Type the VHDL Program in editor window and save the program with **file_name.vhd**.
4. Compile the program. If there is any error found, rectify the error in a program and again. Compile until getting the error free code.
5. Simulate the program. Add the waveform with input and output signals in the program.
6. Apply different values to the input signals and observe the output signals for corresponding inputs.

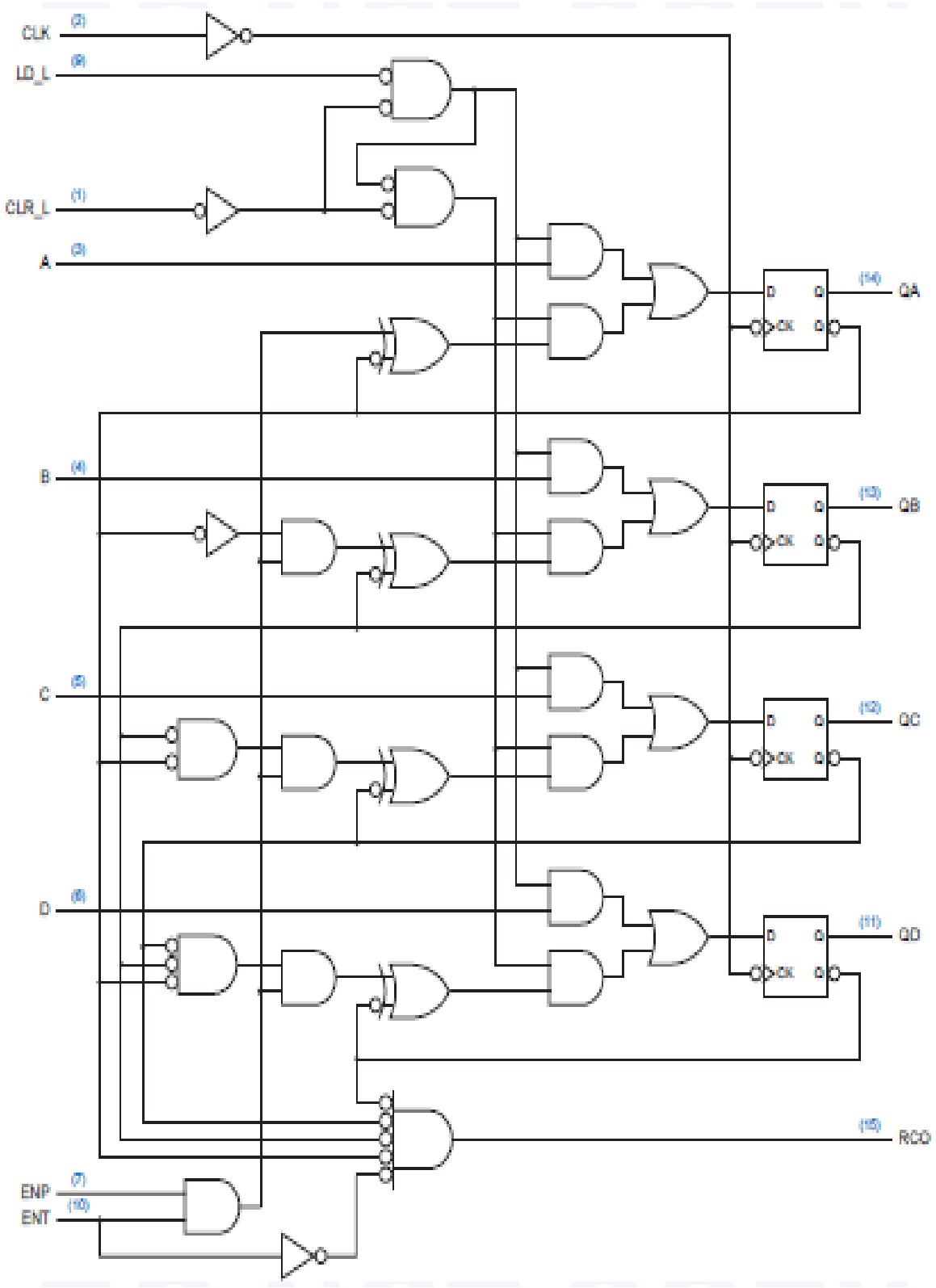
TRUTH TABLE:

Inputs				Current State				Next State			
CLR_L	LD_L	ENT	ENP	QD	QC	QB	QA	QD+	QC+	QB+	QA+
0	x	x	x	x	x	x	x	0	0	0	0
1	0	x	x	x	x	x	x	D	C	B	A
1	1	0	x	x	x	x	x	QD	QC	QB	QA
1	1	x	0	x	x	x	x	QD	QC	QB	QA
1	1	1	1	0	0	0	0	0	0	0	1
1	1	1	1	0	0	0	1	0	0	1	0
1	1	1	1	0	0	1	0	0	0	1	1
1	1	1	1	0	0	1	1	0	1	0	0
1	1	1	1	0	1	0	0	0	1	0	1
1	1	1	1	0	1	0	1	0	1	1	0
1	1	1	1	0	1	1	0	0	1	1	1
1	1	1	1	0	1	1	1	1	0	0	0
1	1	1	1	1	0	0	0	1	0	0	1
1	1	1	1	1	0	0	1	1	0	1	0
1	1	1	1	1	0	1	0	1	0	1	1
1	1	1	1	1	0	1	1	1	1	0	0
1	1	1	1	1	1	0	0	1	1	0	1
1	1	1	1	1	1	0	1	1	1	1	0
1	1	1	1	1	1	1	0	1	1	1	1
1	1	1	1	1	1	1	1	0	0	0	0

LOGIC SYMBOL:



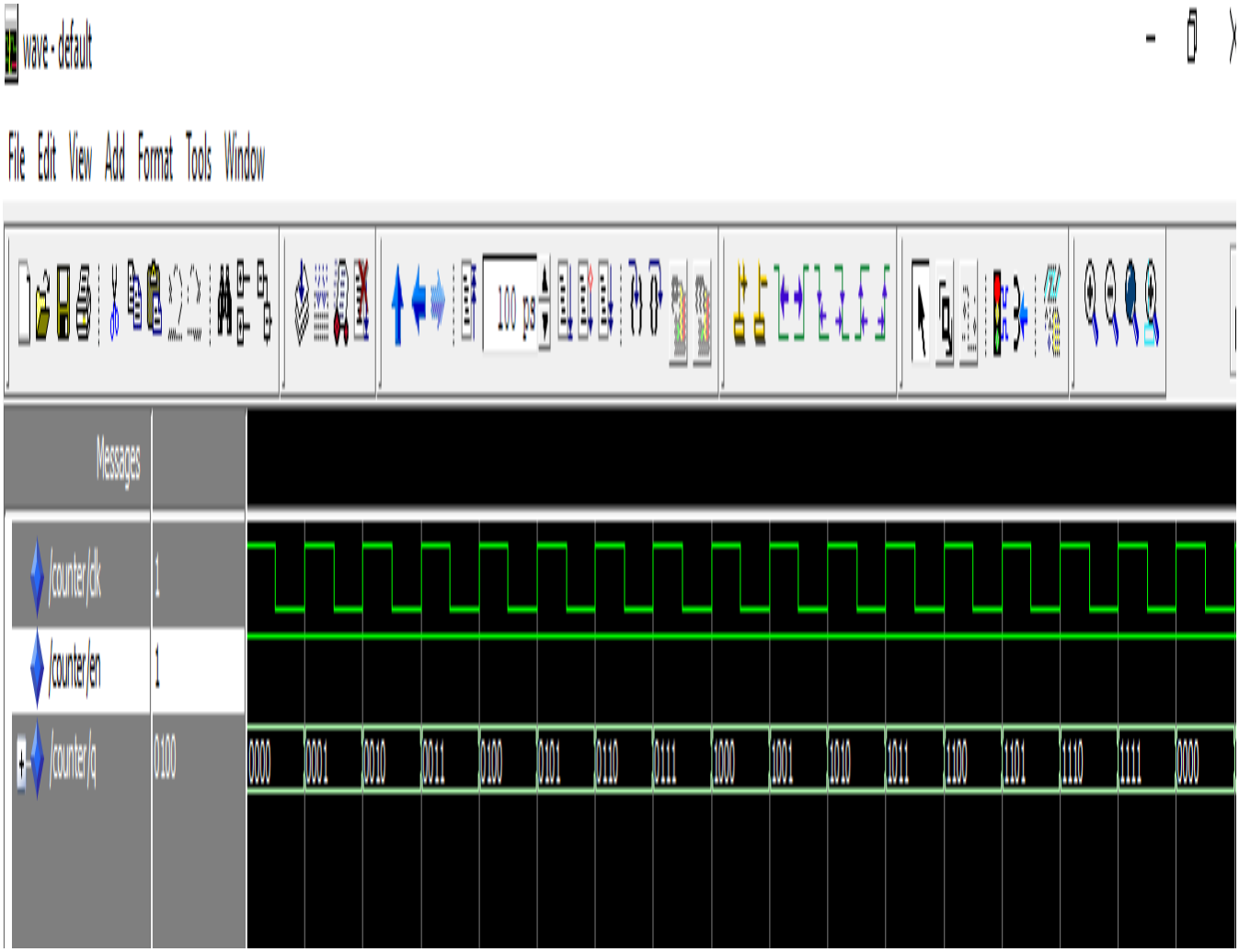
LOGIC DIAGRAM:



VHDL PROGRAM FOR 4-BIT SYNCHRONOUS BINARY COUNTER:

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

ENTITY COUNTER IS
    PORT(CLK,EN:IN STD_LOGIC;
         Q: IN OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
         0END COUNTER;

ARCHITECTURE BEHAVE OF COUNTER IS
    BEGIN
      PROCESS (CLK, EN)
        BEGIN
         IF(CLK'EVENT AND CLK='1')THEN
            IF(EN='0')THEN
                  Q<="0000";
             ELSE
                  Q<=Q+1;
             END IF;
        END IF;
    END PROCESS;
END BEHAVE;
```

SIMULATION RESULTS:



RESULT:

EXPNO:11 **DATE:**

4-BIT UNIVERSAL SHIFT REGISTER

AIM:To write and simulate a VHDL Program for 4-bit universal shift register by using Model sim.

SOFTWARES REQUIRED:

- System with Model sim 6.3Version.

THEORY:

This bidirectional shift register is designed to incorporate virtually all of the features a system designer may want in a shift register; they feature parallel inputs, parallel outputs, right-shift and left-shift serial inputs, operating-mode-control inputs, and a direct overriding clear line. The register has four distinct modes of operation, namely:

- Parallel(broadside)load
- Shift right(in the direction QA toward QD)
- Shift left(in the direction QD toward QA)
- Inhibit clock(do nothing)

Synchronous parallel loading is accomplished by applying the four bits of data and taking both mode control inputs, S0 and S1, HIGH. The data is loaded into the associated flip-flops and appear at the outputs after the positive transition of the clock input. During loading, serial data flow is inhibited. Shift right is accomplished synchronously with the rising edge of the clock pulse when S0 is HIGH andS1 is LOW. Serial data for this mode is entered at the shift-right data input. When S0 is LOW and S1 is HIGH, data shifts left synchronously and new data is entered at the shift-left serial input. Clocking ofthe flip-flop is inhibited when both mode control inputs are LOW.

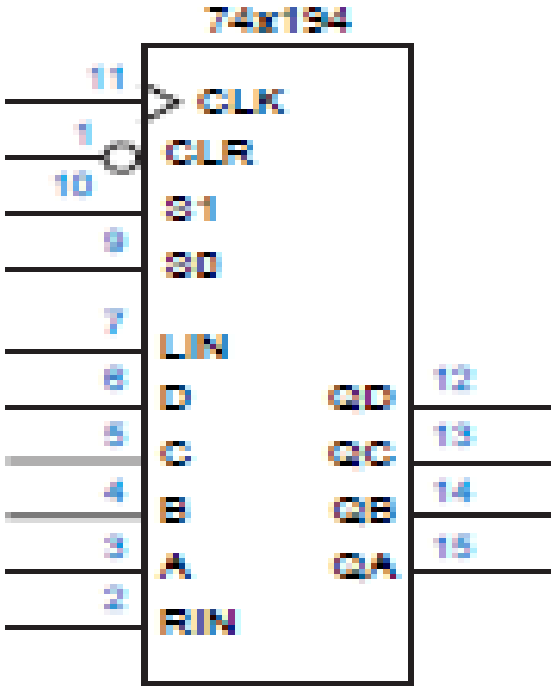
PROCEDURE:

1. Switch on the system and open the Model sim Window.
2. Open the new editor window through File→New→Source→VHDL.
3. Type the VHDL Program in editor window and save the program with **file_name.vhd**.
4. Compile the program. If there is any error found, rectify the error in a program and again. Compile until getting the error free code.
5. Simulate the program. Add the waveform with input and output signals in the program.
6. Apply different values to the input signals and observe the output signals for corresponding inputs.

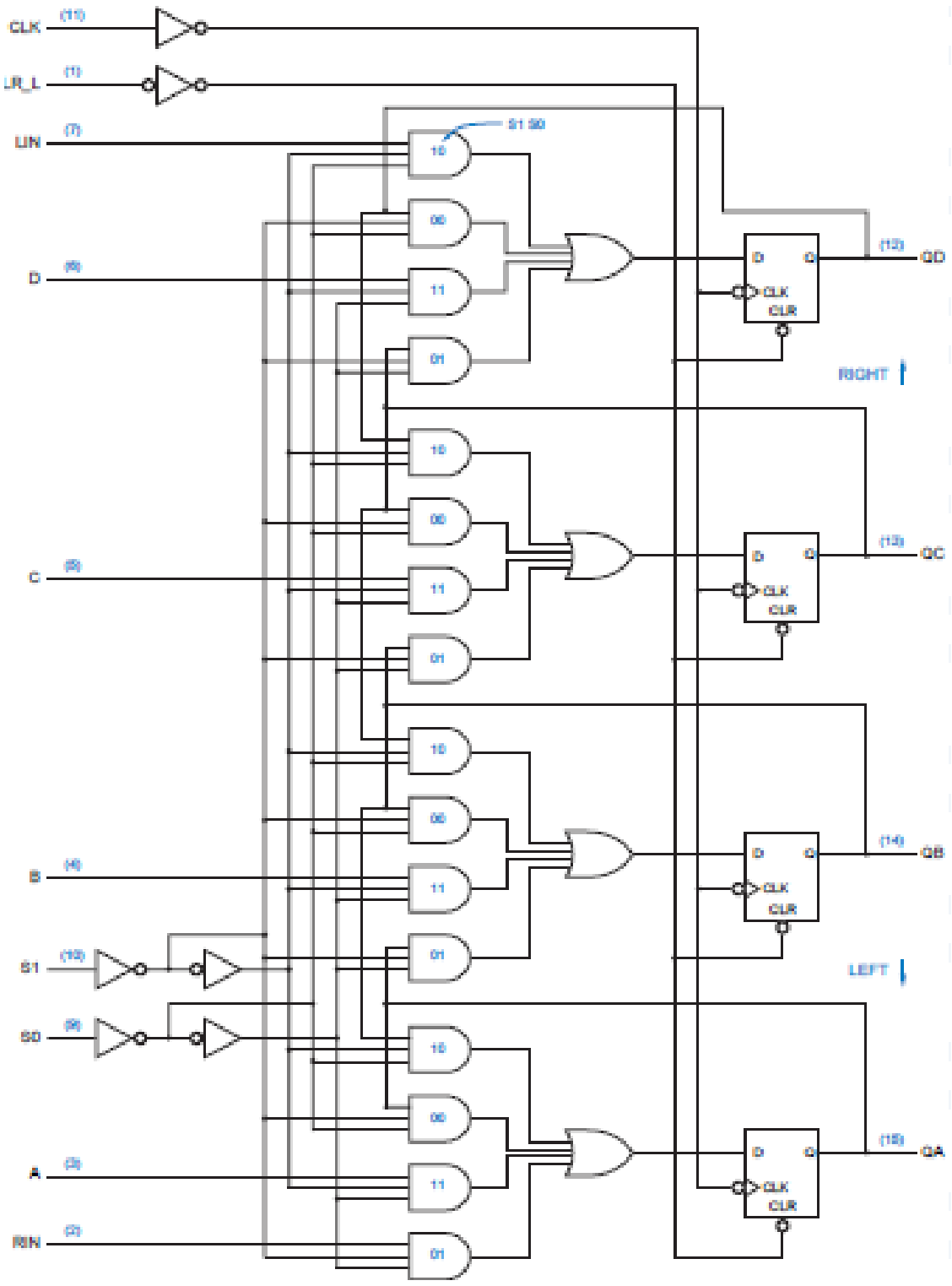
TRUTH TABLE:

Function	Inputs		Next state			
	S1	S0	QA^+	QB^+	QC^+	QD^+
Hold	0	0	QA	QB	QC	QD
Shift right	0	1	RIN	QA	QB	QC
Shift left	1	0	QB	QC	QD	LIN
Load	1	1	A	B	C	D

LOGIC SYMBOL:



LOGIC DIAGRAM:



VHDL PROGRAM FOR 4-BIT UNIVERSAL SHIFT REGISTER:

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

ENTITY  USR IS
        PORT(CLK,RST,SIR,SIL:IN STD_LOGIC;
                D: IN STD_LOGIC_VECTOR (3DOWNTO 0);
                Q: OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
                S: IN STD_LOGIC_VECTOR (1 DOWNTO 0));
END USR;

ARCHITECTURE BEHAVIORAL OF USR IS
        SIGNAL TEMP:STD_LOGIC_VECTOR (3 DOWNTO 0);
        BEGIN
           PROCESS (RST, CLK, S, D, SIR, SIL)
              BEGIN
                IF RST='1'THEN
                        TEMP<="0000";
                         Q<="0000";
                ELSIF (CLK='1' AND CLK'EVENT) THEN
                        CASE S IS
                        WHEN"11"=>
                                TEMP<=D;
                                Q<=TEMP;
                        WHEN"10"=>
                                TEMP<= D;
                                TEMP (3 DOWNTO 1) <= TEMP (2 DOWNTO 0);
                                TEMP(0)<=SIL;
                                Q <= TEMP;
```

```
                WHEN"01"=>
                        TEMP<= D;
                        TEMP(2 DOWNTO 0) <= TEMP(3 DOWNTO 1);
                        TEMP(3)<=SIR;
                        Q <= TEMP;
                WHEN "00"=>
                        TEMP<=TEMP;
                        Q <= TEMP;
                        WHENOTHERS=>NULL;
                END CASE;
        END IF;
END PROCESS;
END BEHAVIORAL;
```

SIMULATION RESULTS:

RESULT:

EXPNO:12 **DATE:**

4-BIT ALU

AIM: To write and simulate a VHDL Program for 4-bit ALU by using Modelsim.

SOFTWARES REQUIRED:

- System with Modelsim 6.3version.

THEORY:

An arithmetic logic unit (ALU) is a digital circuit used to perform arithmetic and logic operations. The 74X382 provide eight different but useful functions. It provides group carry look ahead outputs.

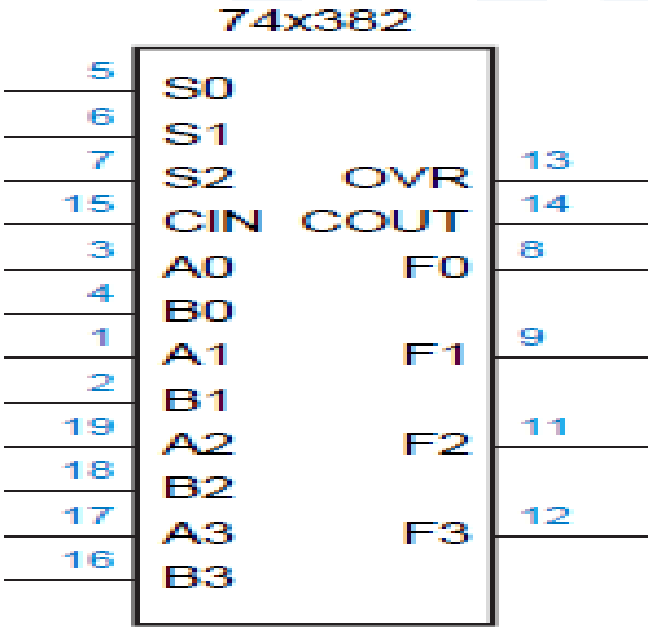
PROCEDURE:

1. Switch on the system and open the Model sim Window.
2. Open the new editor window through File→New→Source→VHDL.
3. Type the VHDL Program in editor window and save the program with **file_name.vhd**.
4. Compile the program. If there is any error found, rectify the error in a program and again. Compile until getting the error free code.
5. Simulate the program. Add the waveform with input and output signals in the program.
6. Apply different values to the input signals and observe the output signals for corresponding inputs.

TRUTH TABLE:

Inputs			
S2	S1	S0	Function
0	0	0	F = 0000
0	0	1	F = B minus A minus 1 plus CIN
0	1	0	F = A minus B minus 1 plus CIN
0	1	1	F = A plus B plus CIN
1	0	0	F = A ⊕ B
1	0	1	F = A + B
1	1	0	F = A · B
1	1	1	F = 1111

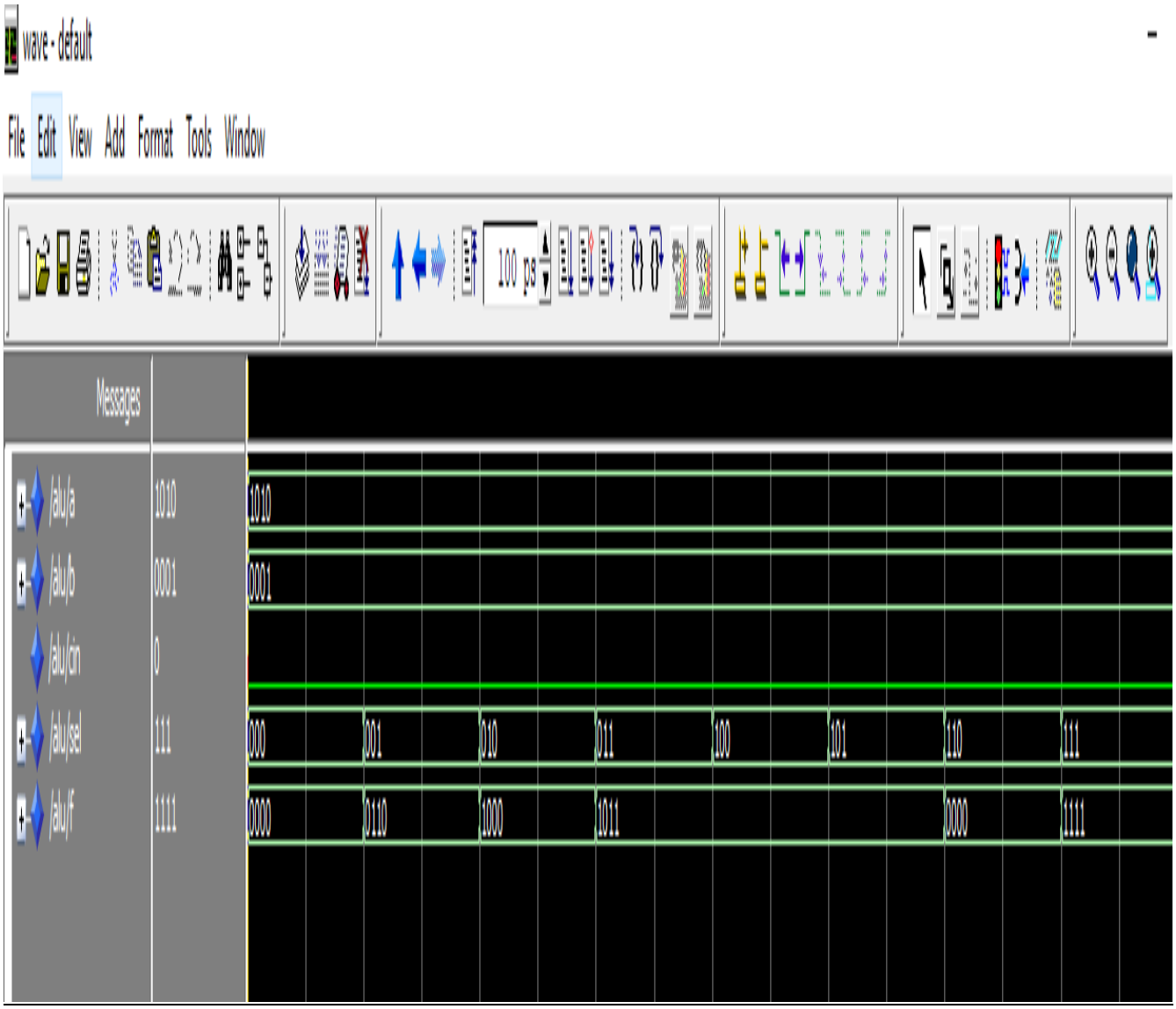
LOGIC SYMBOL:



VHDL PROGRAM FOR 4-BIT ALU

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

ENTITY ALU IS
PORT (A, B: IN STD_LOGIC_VECTOR (3 DOWNTO 0) ;
        CIN:IN STD-LOGIC;
        SEL:IN STD_LOGIC_VECTOR(2  DOWNTO 0);
        F: OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
 END ENTITY ALU;

ARCHITECTURE BEHAVIORAL OF ALU IS
     BEGIN
        PROCESS (A, B, SEL)
        BEGIN
        CASE SEL IS
        WHEN "000" =>F <= “0000”;
        WHEN "001" => F <= B-A-1+CIN;
        WHEN "010" => F<= A-B-1+CIN;
        WHEN "011" => F <= A+B+CIN;
        WHEN "100" => F <= A XOR B;
        WHEN "101" => F <= A OR B;
        WHEN "110" => F <= A and B;
        WHEN "111"=> F<=“1111”;
        WHEN OTHERS=>F<= "0000";
     END CASE;
END PROCESS;
END BEHAVIORAL;
```

SIMULATION RESULTS:



RESULT: