

Institute of Engineering & Technology (IET)

JK Lakshmipat University

Assignment – 2

For course  
**Computer Organisation & Architecture (CS1134)**

By

*Siddhi Nyati*

(2022BTECH101)  
(Section – A)

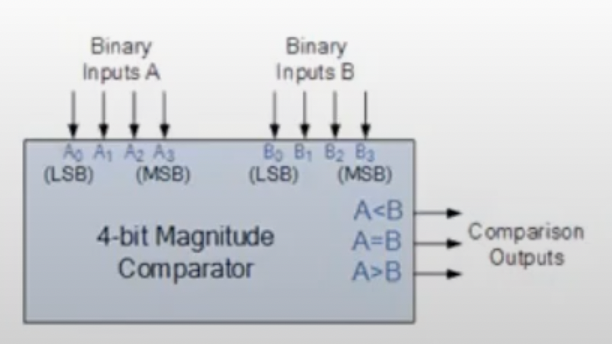
Under the guidance of  
*Dr. Pranab Roy*

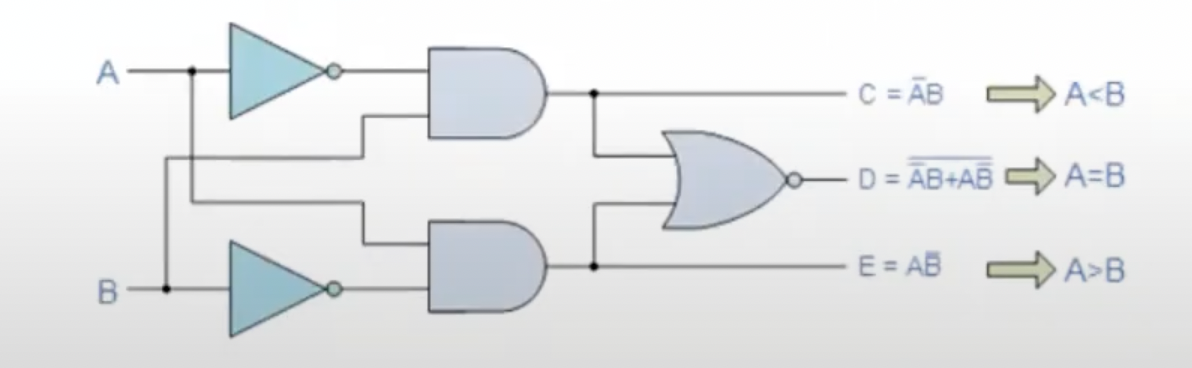
*Dr. Ajai Jain*

**Objective 1:** Design a VHDL model for a hardwired 4-bit comparator using dataflow architecture.

Theory:

A magnitude digital Comparator is a combinational circuit that compares two digital or binary numbers to determine whether one binary number is equal, less than, or greater than the other binary number. We logically design a circuit for which we will have two inputs one for A and the other for B and have three output terminals, one for A > B condition, one for A = B condition, and one for A < B condition



****

**VHDL CODE OF THE 4-BIT COMPARATOR**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Comparator4Bit is

Port ( A : in STD\_LOGIC\_VECTOR (3 downto 0);

B : in STD\_LOGIC\_VECTOR (3 downto 0);

Greater : out STD\_LOGIC;

Equal : out STD\_LOGIC;

Lesser : out STD\_LOGIC);

end Comparator4Bit;

architecture Dataflow of Comparator4Bit is

begin

Greater <= '1' when A > B else '0';

Equal <= '1' when A = B else '0';

Lesser <= '1' when A < B else '0';

end Dataflow;   
  
OUTPUT OF THE CODE

A computer screen shot of a diagram

Description automatically generated

**TESSTBENCH CODE OF THE 4-BIT COMPARATOR**

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

ENTITY Comparator4Bit\_TB IS

END Comparator4Bit\_TB;

ARCHITECTURE behavior OF Comparator4Bit\_TB IS

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

COMPONENT Comparator4Bit

PORT(

A : IN std\_logic\_vector(3 downto 0);

B : IN std\_logic\_vector(3 downto 0);

Greater : OUT std\_logic;

Equal : OUT std\_logic;

Lesser : OUT std\_logic

);

END COMPONENT;

-- Inputs

signal A : std\_logic\_vector(3 downto 0) := (others => '0');

signal B : std\_logic\_vector(3 downto 0) := (others => '0');

-- Outputs

signal Greater : std\_logic;

signal Equal : std\_logic;

signal Lesser : std\_logic;

BEGIN

-- Instantiate the Unit Under Test (UUT)

UUT: Comparator4Bit PORT MAP (

A => A,

B => B,

Greater => Greater,

Equal => Equal,

Lesser => Lesser

);

-- Stimulus process

stim\_proc: process

begin

-- Test Case 1: A = 4, B = 2

A <= "0100"; B <= "0010";

wait for 50 ns;

-- Test Case 2: A = 7, B = 7

A <= "0111"; B <= "0111";

wait for 50 ns;

-- Test Case 3: A = 1, B = 5

A <= "0001"; B <= "0101";

wait for 50 ns;

-- Test Case 4: A = 15, B = 0

A <= "1111"; B <= "0000";

wait for 50 ns;

wait;

end process;

END;

RESULTS:

A screenshot of a computer

Description automatically generated

**Objective 2:** Design a VHDL model for a 3 to 8 Decoder (with an enable) using Dataflow Architecture.

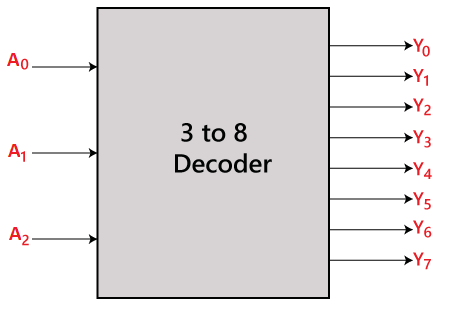
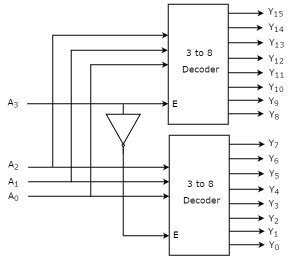
Thereby Design a VHDL model for a 4 to 16 Decoder using 3 to 8 decoders designed earlier

using Structural Architecture.

Theory:

A Decoder is a combinational circuit that converts binary information from the n input lines into a maximum of 2^n unique outputs. A 3-to-8 Decoder takes three inputs and decodes them into one of the eight possible outputs. The additional enable signal ensures that the decoder is active only when required.

The 4-to-16 Decoder is constructed by combining two 3-to-8 Decoders. The selection between the decoders is determined by the most significant input bit, making the entire structure a hierarchical design where one decoder can control others.

**VHDL CODE OF THE 3-8 DECODER**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Decoder3to8 is

Port ( Enable : in STD\_LOGIC;

A : in STD\_LOGIC\_VECTOR(2 downto 0);

Y : out STD\_LOGIC\_VECTOR(7 downto 0));

end Decoder3to8;

architecture Dataflow of Decoder3to8 is

begin

process (A, Enable)

begin

if Enable = '1' then

case A is

when "000" => Y <= "00000001";

when "001" => Y <= "00000010";

when "010" => Y <= "00000100";

when "011" => Y <= "00001000";

when "100" => Y <= "00010000";

when "101" => Y <= "00100000";

when "110" => Y <= "01000000";

when "111" => Y <= "10000000";

when others => Y <= "00000000"; -- Default case

end case;

else

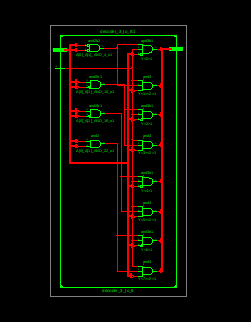
Y <= "00000000"; -- Output low when disabled

end if;

end process;

end Dataflow;

OUTPUT OF THE CODE



**VHDL CODE OF THE 4-16 DECODER**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Decoder4to16 is

Port ( Enable : in STD\_LOGIC;

A : in STD\_LOGIC\_VECTOR(3 downto 0);

Y : out STD\_LOGIC\_VECTOR(15 downto 0));

end Decoder4to16;

architecture Structural of Decoder4to16 is

signal Y\_lower, Y\_upper : STD\_LOGIC\_VECTOR(7 downto 0);

signal Enable\_lower, Enable\_upper : STD\_LOGIC;

component Decoder3to8

Port ( Enable : in STD\_LOGIC;

A : in STD\_LOGIC\_VECTOR(2 downto 0);

Y : out STD\_LOGIC\_VECTOR(7 downto 0));

end component;

begin

-- Enable logic for lower and upper 3-to-8 decoders

Enable\_lower <= Enable and not A(3); -- Lower decoder enabled when A(3) = 0

Enable\_upper <= Enable and A(3); -- Upper decoder enabled when A(3) = 1

-- Instantiate two 3-to-8 decoders

LowerDecoder: Decoder3to8 Port map (Enable => Enable\_lower, A => A(2 downto 0), Y => Y\_lower);

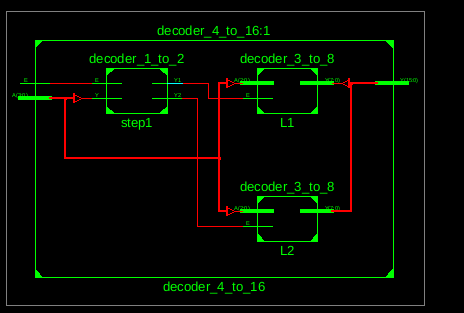
UpperDecoder: Decoder3to8 Port map (Enable => Enable\_upper, A => A(2 downto 0), Y => Y\_upper);

-- Combine outputs of lower and upper decoders

Y <= Y\_lower & Y\_upper;

end Structural;

OUTPUT OF THE CODE



**TESTBENCH CODE OF THE 4-16 DECODER**

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

ENTITY tb\_Decoder4to16 IS

END tb\_Decoder4to16;

ARCHITECTURE behavior OF tb\_Decoder4to16 IS

COMPONENT Decoder4to16

PORT(

Enable : IN std\_logic;

A : IN std\_logic\_vector(3 downto 0);

Y : OUT std\_logic\_vector(15 downto 0)

);

END COMPONENT;

-- Inputs

signal Enable : std\_logic := '0';

signal A : std\_logic\_vector(3 downto 0) := (others => '0');

-- Outputs

signal Y : std\_logic\_vector(15 downto 0);

BEGIN

-- Instantiate the Unit Under Test (UUT)

UUT: Decoder4to16 PORT MAP (

Enable => Enable,

A => A,

Y => Y

);

-- Stimulus process

stim\_proc: process

begin

-- Test Case 1: Enable = '1', A = "0000"

Enable <= '1'; A <= "0000"; wait for 10 ns;

-- Test Case 2: Enable = '1', A = "1000" (activates upper decoder)

A <= "1000"; wait for 10 ns;

-- Test Case 3: Enable = '0', A = "1010" (should output all zeros)

Enable <= '0'; wait for 10 ns;

-- Test Case 4: Enable = '1', A = "1111"

Enable <= '1'; A <= "1111"; wait for 10 ns;

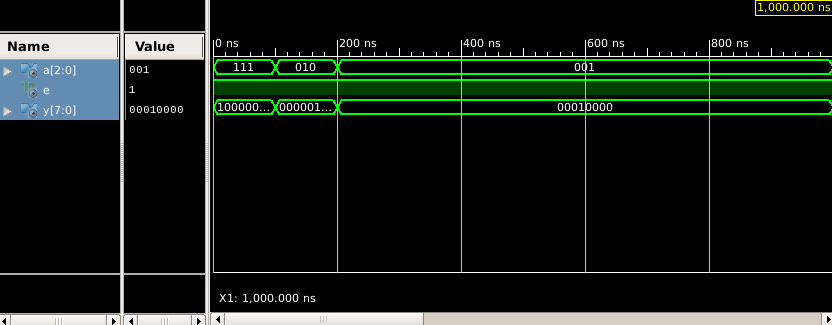
wait;

end process;

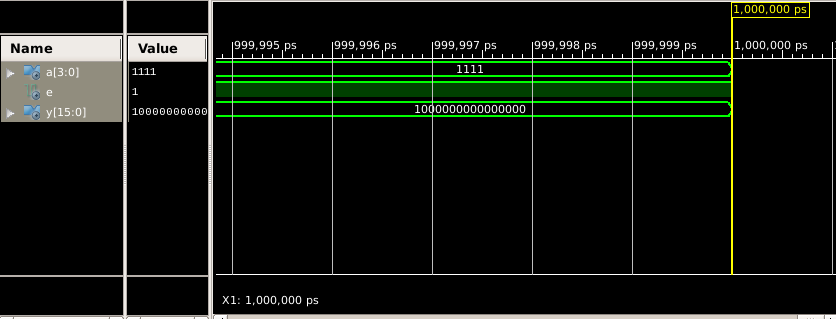
END;

**RESULTS**

3 TO 8 DECODER –



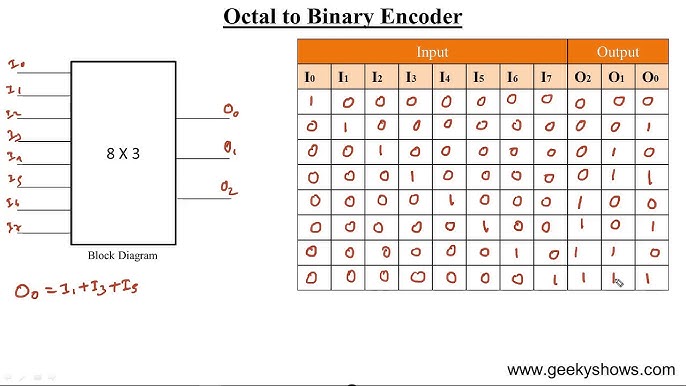
4 TO 16 DECODER –



**Objective 3:** Design a VHDL Model for an Octal to Binary Encoder with 8 input Bits and three output bits. Each input Bit represents one Octal number.

Theory:   
An Encoder is a combinational circuit that converts 2^n input lines into an n-bit binary code. In an Octal to Binary Encoder, there are 8 inputs, each representing one of the octal digits (0 to 7), and 3 output lines that provide the corresponding binary number for the active input. This circuit is useful when encoding information into a more compact binary form.

For this design, the inputs are often labelled as I0 to I7, corresponding to octal numbers, and the outputs are Y2, Y1, and Y0, representing the binary equivalent. If multiple inputs are active, the encoder gives priority to the highest-order input.



If input I6 is active (I6 = 1), the encoder will output Y2Y1Y0 = 110, representing the binary form of octal number 6.

**VHDL CODE OF THE 4-16 DECODER**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity OctalToBinaryEncoder is

Port ( Input : in STD\_LOGIC\_VECTOR(7 downto 0);

BinaryOut : out STD\_LOGIC\_VECTOR(2 downto 0));

end OctalToBinaryEncoder;

architecture Dataflow of OctalToBinaryEncoder is

begin

process(Input)

begin

case Input is

when "00000001" => BinaryOut <= "000"; -- Octal 0

when "00000010" => BinaryOut <= "001"; -- Octal 1

when "00000100" => BinaryOut <= "010"; -- Octal 2

when "00001000" => BinaryOut <= "011"; -- Octal 3

when "00010000" => BinaryOut <= "100"; -- Octal 4

when "00100000" => BinaryOut <= "101"; -- Octal 5

when "01000000" => BinaryOut <= "110"; -- Octal 6

when "10000000" => BinaryOut <= "111"; -- Octal 7

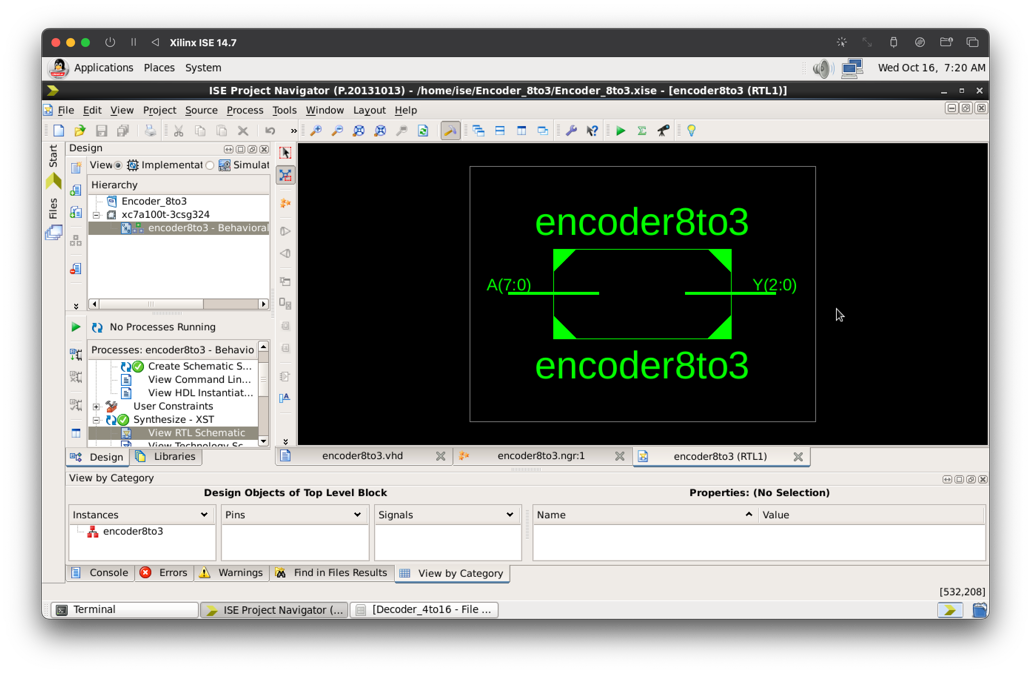
when others => BinaryOut <= "000"; -- Default case (no input)

end case;

end process;

end Dataflow;

OUTPUT OF THE CODE



**TESSTBENCH CODE OF THE 4-BIT COMPARATOR**

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

ENTITY OctalToBinaryEncoder\_TB IS

END OctalToBinaryEncoder\_TB;

ARCHITECTURE behavior OF OctalToBinaryEncoder\_TB IS

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

COMPONENT OctalToBinaryEncoder

PORT(

Input : IN std\_logic\_vector(7 downto 0);

BinaryOut : OUT std\_logic\_vector(2 downto 0)

);

END COMPONENT;

-- Inputs

signal Input : std\_logic\_vector(7 downto 0) := (others => '0');

-- Outputs

signal BinaryOut : std\_logic\_vector(2 downto 0);

BEGIN

-- Instantiate the Unit Under Test (UUT)

UUT: OctalToBinaryEncoder PORT MAP (

Input => Input,

BinaryOut => BinaryOut

);

-- Stimulus process

stim\_proc: process

begin

-- Test case 1: Input = Octal 0

Input <= "00000001";

wait for 20 ns;

-- Test case 2: Input = Octal 2

Input <= "00000100";

wait for 20 ns;

-- Test case 3: Input = Octal 5

Input <= "00100000";

wait for 20 ns;

-- Test case 4: Input = Octal 7

Input <= "10000000";

wait for 20 ns;

-- Test case 5: Default case (no input)

Input <= "00000000";

wait for 20 ns;

wait;

end process;

END;

**RESULTS**

A screenshot of a computer

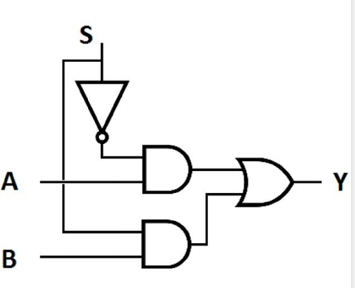
Description automatically generated

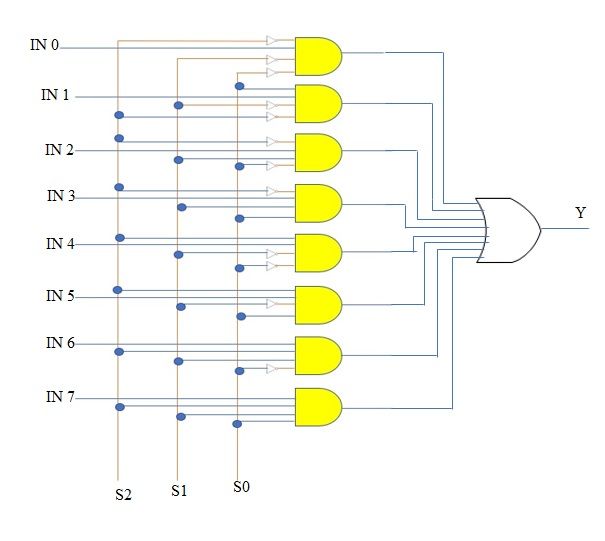
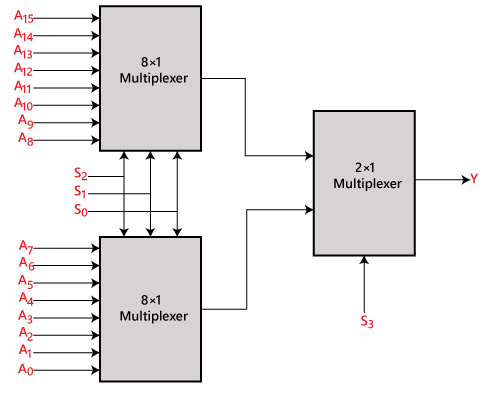
**Objective 4:** Design a VHDL model for 8-1 and 2-1 multiplexer. Using these models develop a VHDL

model for 16 -1 multiplexer (Use structural architecture).

Theory:   
A Multiplexer (MUX) is a combinational circuit that selects one of many input signals and forwards it to a single output. The selection is controlled by select lines. A 2-to-1 multiplexer selects between two inputs using a single select line, while an 8-to-1 multiplexer selects one of eight inputs using three select lines.

The 16-to-1 multiplexer can be constructed hierarchically by combining two 8-to-1 multiplexers. The output of these two 8-to-1 multiplexers is selected by an additional 2-to-1 multiplexer, using the fourth select bit. This design demonstrates the use of structural VHDL to build larger components using smaller, reusable modules.



Operation:

* If S3 = 0, the lower 8-to-1 multiplexer (I0 to I7) is selected.
* If S3 = 1, the upper 8-to-1 multiplexer (I8 to I15) is selected.
* The lower three select bits (S2, S1, S0) determine which of the inputs is passed from the active 8-to-1 multiplexer.

**VHDL CODE OF THE 2-TO-1 MULTIPLEXER**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Mux2to1 is

Port ( A : in STD\_LOGIC;

B : in STD\_LOGIC;

Sel : in STD\_LOGIC;

Y : out STD\_LOGIC);

end Mux2to1;

architecture Behavioral of Mux2to1 is

begin

Y <= A when Sel = '0' else B;

end Behavioral;

OUTPUT OF THE CODE

A computer screen shot of a circuit

Description automatically generated

**VHDL CODE OF THE 8-TO-1 MULTIPLEXER**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Mux8to1 is

Port ( A : in STD\_LOGIC\_VECTOR(7 downto 0);

Sel : in STD\_LOGIC\_VECTOR(2 downto 0);

Y : out STD\_LOGIC);

end Mux8to1;

architecture Behavioral of Mux8to1 is

begin

process(A, Sel)

begin

case Sel is

when "000" => Y <= A(0);

when "001" => Y <= A(1);

when "010" => Y <= A(2);

when "011" => Y <= A(3);

when "100" => Y <= A(4);

when "101" => Y <= A(5);

when "110" => Y <= A(6);

when "111" => Y <= A(7);

when others => Y <= '0';

end case;

end process;

end Behavioral;

OUTPUT OF THE CODE

A computer screen shot of a diagram

Description automatically generated

**VHDL CODE OF THE 16-TO-1 MULTIPLEXER**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Mux16to1 is

Port ( A : in STD\_LOGIC\_VECTOR(15 downto 0);

Sel : in STD\_LOGIC\_VECTOR(3 downto 0);

Y : out STD\_LOGIC);

end Mux16to1;

architecture Structural of Mux16to1 is

signal Y\_lower, Y\_upper : STD\_LOGIC;

component Mux8to1

Port ( A : in STD\_LOGIC\_VECTOR(7 downto 0);

Sel : in STD\_LOGIC\_VECTOR(2 downto 0);

Y : out STD\_LOGIC);

end component;

component Mux2to1

Port ( A : in STD\_LOGIC;

B : in STD\_LOGIC;

Sel : in STD\_LOGIC;

Y : out STD\_LOGIC);

end component;

begin

-- Instantiate two 8-to-1 multiplexers for lower and upper halves

LowerMux: Mux8to1 Port map (A => A(7 downto 0), Sel => Sel(2 downto 0), Y => Y\_lower);

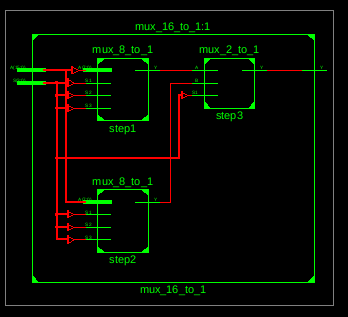
UpperMux: Mux8to1 Port map (A => A(15 downto 8), Sel => Sel(2 downto 0), Y => Y\_upper);

-- Use 2-to-1 mux to select between the two 8-to-1 mux outputs

FinalMux: Mux2to1 Port map (A => Y\_lower, B => Y\_upper, Sel => Sel(3), Y => Y);

end Structural;

OUTPUT OF THE CODE



**TESSTBENCH CODE OF THE 4-BIT COMPARATOR**

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

ENTITY tb\_Mux16to1 IS

END tb\_Mux16to1;

ARCHITECTURE behavior OF tb\_Mux16to1 IS

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

COMPONENT Mux16to1

PORT(

A : IN std\_logic\_vector(15 downto 0);

Sel : IN std\_logic\_vector(3 downto 0);

Y : OUT std\_logic

);

END COMPONENT;

-- Inputs

signal A : std\_logic\_vector(15 downto 0) := (others => '0');

signal Sel : std\_logic\_vector(3 downto 0) := (others => '0');

-- Outputs

signal Y : std\_logic;

BEGIN

-- Instantiate the Unit Under Test (UUT)

UUT: Mux16to1 PORT MAP (

A => A,

Sel => Sel,

Y => Y

);

-- Stimulus process

stim\_proc: process

begin

-- Test case 1: Select input A(0)

A <= "0000000000000001"; Sel <= "0000"; wait for 10 ns;

-- Test case 2: Select input A(7)

A <= "0000000010000000"; Sel <= "0111"; wait for 10 ns;

-- Test case 3: Select input A(8)

A <= "0000000100000000"; Sel <= "1000"; wait for 10 ns;

-- Test case 4: Select input A(15)

A <= "1000000000000000"; Sel <= "1111"; wait for 10 ns;

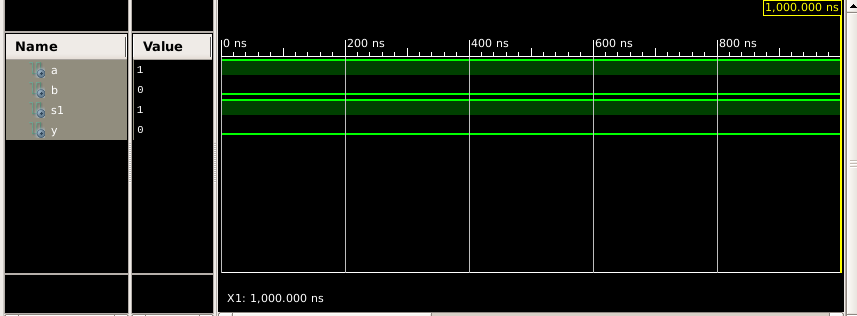
wait;

end process;

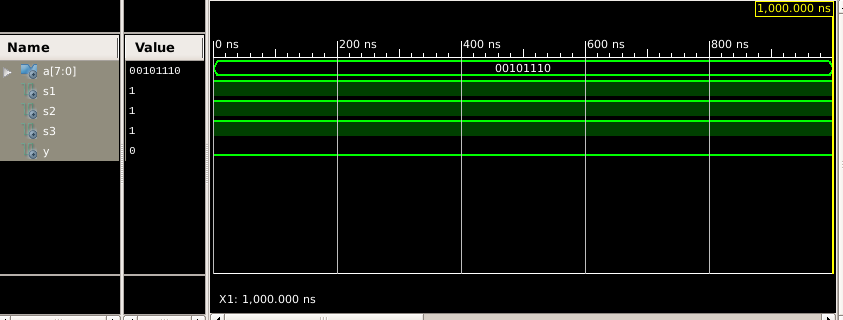
END;

**RESULTS**

2-1 MULTIPLEXER –



8-1 MULTIPLEXER –



16-1 MULTIPLEXER –

