Pedagogy REPORT

Title of the Experiment

Design Verilog program to implement types of De-Multiplexer 4:1 and 8:1

Course

Digital Design and Computer Organization Pedagogy Activity

Objective

To design, implement, and simulate 4:1 and 8:1 Demultiplexer circuits using Verilog HDL, and to verify their functionality through simulation and waveform analysis.

Tools and Software Used

- **Verilog HDL** for hardware description
- **Icarus Verilog** for compilation and simulation
- **GTKWave** for waveform visualization and signal analysis

GTKWave is used to observe the timing behavior of input and output signals and to validate the correctness of the demultiplexer operation.

Prepared a detailed report on the executed program, including:

- Source code
- Explanation of the logic
- Output obtained
- Observations or conclusions

1. 4:1 Demultiplexer

Aim

To design and simulate a **4:1 Demultiplexer** using Verilog HDL and verify its functionality using a testbench.

Introduction

A **demultiplexer (DEMUX)** is a combinational logic circuit that takes **one input signal** and routes it to **one of several output lines** based on the values of select inputs.
In a **4:1 demultiplexer**, one input is routed to **one of four outputs** using **two select lines**.

Source Code:

```
// 4:1 Demultiplexer
module Demux4to1(I, S0, S1, Y0, Y1, Y2, Y3);
    input I, S0, S1;
    output Y0, Y1, Y2, Y3;

    assign Y0 = (~S0) & (~S1) & I;
    assign Y1 = (~S0) & ( S1) & I;
    assign Y2 = ( S0) & (~S1) & I;
    assign Y3 = ( S0) & ( S1) & I;
endmodule


module Demux4to1_TB( );
    reg I, S0, S1;
    wire Y0, Y1, Y2, Y3;

    Demux4to1 DEMUX(I, S0, S1, Y0, Y1, Y2, Y3);

    initial begin
        $dumpfile("demux.vcd");     // output waveform file
        $dumpvars(0, Demux4to1_TB); // dump all signals in testbench

        $monitor("Time=%0t  S1=%b S0=%b  I=%b  |  Y0=%b Y1=%b Y2=%b Y3=%b",
                 $time, S1, S0, I, Y0, Y1, Y2, Y3);

        S0 = 0; S1 = 0; I = 1; #10;
        S0 = 0; S1 = 1; I = 1; #10;
        S0 = 1; S1 = 0; I = 1; #10;
        S0 = 1; S1 = 1; I = 1; #10;

        $finish;
    end
endmodule
```

Logic Explanation

Inputs

- I – Data input
- S0, S1 – Select lines

Outputs

- Y0, Y1, Y2, Y3

Each output corresponds to **one unique combination of select lines**.

Truth Table Logic

S1	S0	Active Output
0	0	Y0 = I
0	1	Y1 = I
1	0	Y2 = I
1	1	Y3 = I

Only **one output is active at a time**, while others remain 0.

Boolean Logic Used

Each output is enabled only when its select condition is satisfied.

Y0 = (~S0) & (~S1) & I;

Y1 = (~S0) & (S1) & I;

Y2 = (S0) & (~S1) & I;

Y3 = (S0) & (S1) & I;

- ~S0 or ~S1 ensures the select line is LOW
- & I ensures output depends on the input
- Logical AND ensures **exclusive activation**

This implementation is **purely combinational** and does not require clocks or memory elements.

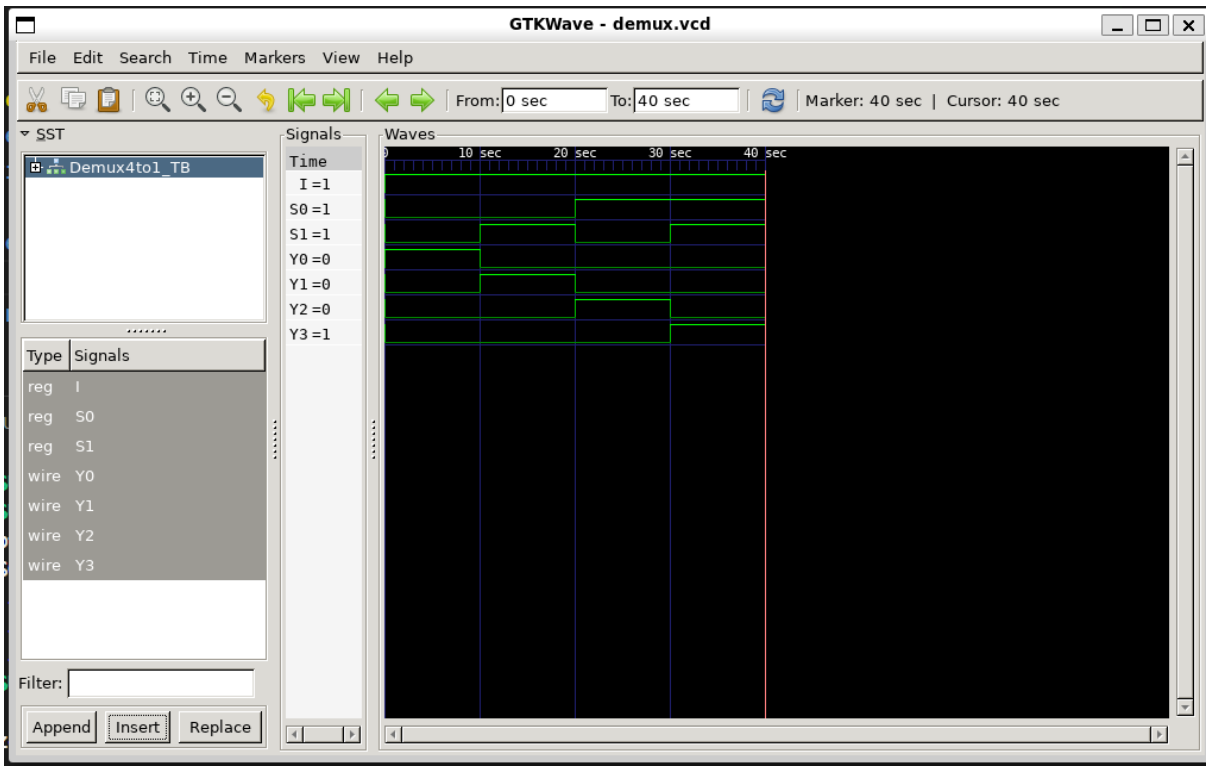
Testbench Explanation

The testbench:

- Generates all possible combinations of select lines
- Keeps input I = 1
- Observes which output becomes HIGH

The $monitor statement prints signal values for each time step, allowing verification of correct routing.

Output:



Observations / Conclusions (4:1 DEMUX)

- For each select combination, **only one output is HIGH**
- All other outputs remain LOW
- The demultiplexer correctly routes the input to the selected output
- The design is free from race conditions and glitches
- Simulation results match the theoretical truth table

2. 8:1 Demultiplexer

Aim

To design and simulate an **8:1 Demultiplexer** using Verilog HDL and verify its operation through simulation.

Introduction

An **8:1 demultiplexer** routes a single input signal to **one of eight outputs** using **three select lines**. It is an extension of the 4:1 demultiplexer and demonstrates scalable digital logic design.

Source Code:

```
// 8:1 Demultiplexer
module Demux8to1(I, S0, S1, S2, Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7);
    input I, S0, S1, S2;
    output Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7;

    assign Y0 = (~S0) & (~S1) & (~S2) & I;
    assign Y1 = (~S0) & (~S1) & ( S2) & I;
    assign Y2 = (~S0) & ( S1) & (~S2) & I;
    assign Y3 = (~S0) & ( S1) & ( S2) & I;
    assign Y4 = ( S0) & (~S1) & (~S2) & I;
    assign Y5 = ( S0) & (~S1) & ( S2) & I;
    assign Y6 = ( S0) & ( S1) & (~S2) & I;
    assign Y7 = ( S0) & ( S1) & ( S2) & I;

endmodule


module Demux8to1_TB();
    reg I, S0, S1, S2;
    wire Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7;

    Demux8to1 DEMUX(
        I, S0, S1, S2,
        Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7
    );

    initial begin
        $dumpfile("demux8.vcd");
        $dumpvars(0, Demux8to1_TB);

        $monitor(
          "Time=%0t  S2=%b S1=%b S0=%b  I=%b  |  Y0=%b Y1=%b Y2=%b Y3=%b Y4=%b Y5=%b Y6=%b Y7=%b",
          $time, S2, S1, S0, I,
          Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7
        );

        S0=0; S1=0; S2=0; I=1; #10;
        S0=0; S1=0; S2=1; I=1; #10;
        S0=0; S1=1; S2=0; I=1; #10;
        S0=0; S1=1; S2=1; I=1; #10;
        S0=1; S1=0; S2=0; I=1; #10;
        S0=1; S1=0; S2=1; I=1; #10;
        S0=1; S1=1; S2=0; I=1; #10;
        S0=1; S1=1; S2=1; I=1; #10;

        $finish;
    end
endmodule
```

Logic Explanation

Inputs

- I – Data input
- S0, S1, S2 – Select lines

Outputs

- Y0 to Y7

Each output corresponds to one **unique 3-bit select combination**.

Truth Table Logic

S2	S1	S0	Active Output
0	0	0	Y0
0	0	1	Y1
0	1	0	Y2
0	1	1	Y3
1	0	0	Y4
1	0	1	Y5
1	1	0	Y6
1	1	1	Y7

Boolean Logic Used

Each output is activated by a **unique AND combination** of select lines and the input.

Y0 = (~S0) & (~S1) & (~S2) & I;

Y1 = (~S0) & (~S1) & (S2) & I;

Y2 = (~S0) & (S1) & (~S2) & I;

Y3 = (~S0) & (S1) & (S2) & I;

Y4 = (S0) & (~S1) & (~S2) & I;

Y5 = (S0) & (~S1) & (S2) & I;

Y6 = (S0) & (S1) & (~S2) & I;

Y7 = (S0) & (S1) & (S2) & I;

Each output:

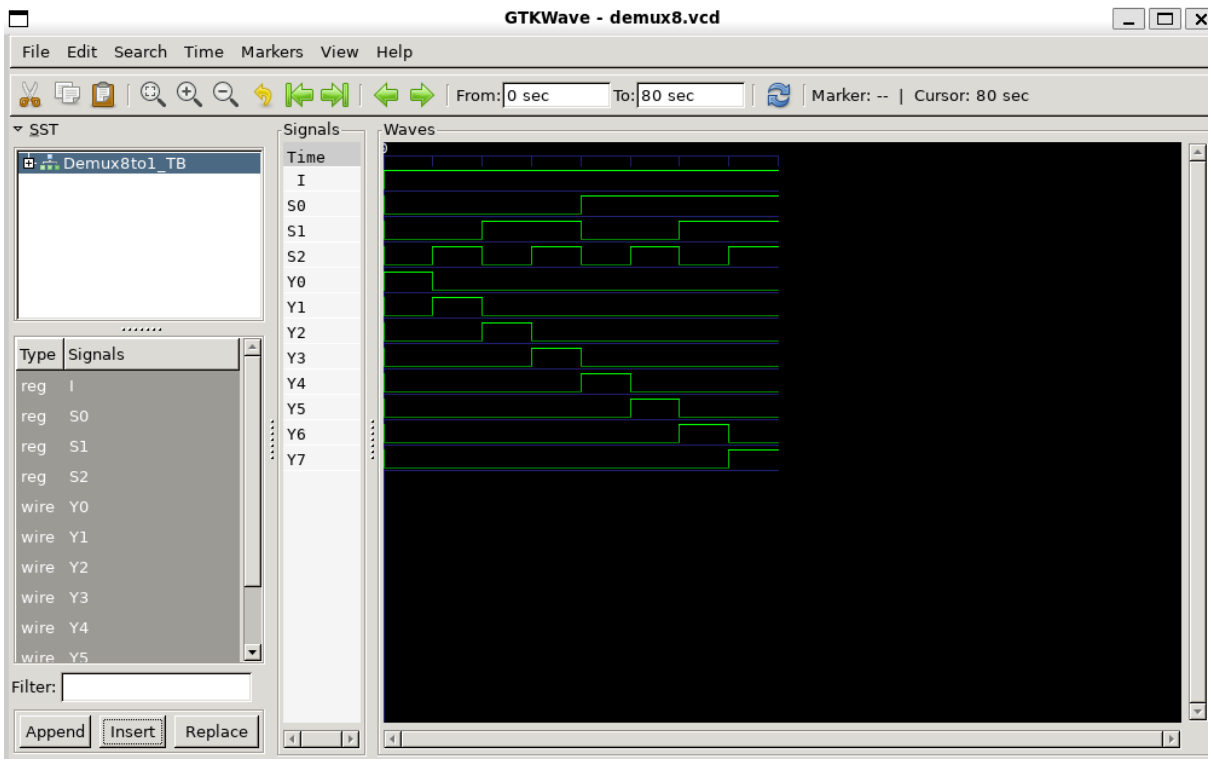
- Is mutually exclusive
- Depends strictly on select lines
- Acts as a **decoded AND gate**

Testbench Explanation

The testbench:

- Cycles through all 8 combinations of select inputs
- Keeps I = 1 to observe output behavior
- Verifies one-hot output behavior

Output:



Observations / Conclusions (8:1 DEMUX)

- Exactly one output is HIGH for each select combination
- No overlap or multiple outputs active
- Output transitions are clean and deterministic
- Logic scales correctly from 4:1 to 8:1
- Simulation confirms theoretical expectations