Lab # 07

Design and Implementation of Multiplexers using ICs and Verilog HDL

Objectives:

- **Apply** the knowledge of digital logic to explain the working principle and functionality of multiplexers.
- **Construct** and verify multiplexer circuits using logic gates and truth tables.
- **Apply** digital design techniques to implement and test multiplexers using IC hardware.
- **Construct** Verilog HDL models of multiplexers and simulate their performance using software tools (e.g., Vivado).
- **Apply** the concepts to compare hardware and HDL-based multiplexer implementations.
- **Collaborate** in performing hardware–software co-design and verification of digital circuits.

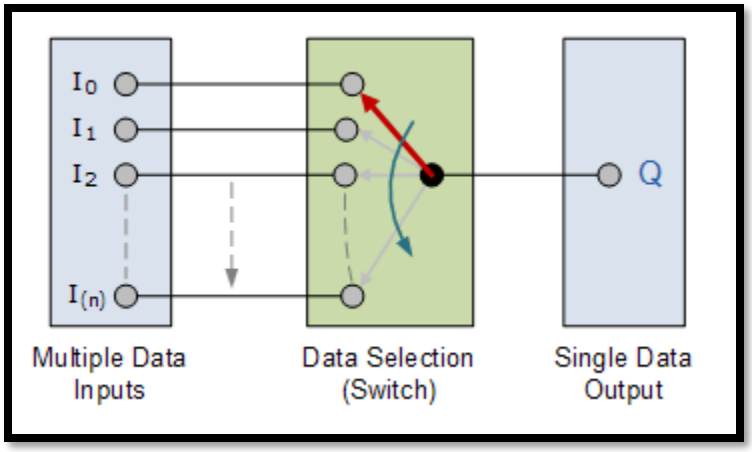
Multiplexers (MUX):

Multiplexing is the generic term used to describe the operation of sending one or more analogue or digital signals over a common transmission line at different times or speeds, and as such, the device we use to do just that is called a **Multiplexer**.

The *multiplexer*, shortened to "MUX" or "MPX", is a combinational logic circuit designed to ***switch one of several input lines through to a single common output line by the application of a control signal.*** Multiplexers operate like very fast-acting, multiple-position rotary switches, connecting or controlling multiple input lines, called "channels," one at a time to the output.

Multiplexers, or MUXs, can be either digital circuits made from high-speed logic gates used to switch digital or binary data, or they can be analogue types using transistors, MOSFET's or relays to switch one of the voltage or current inputs through to a single output.

The most basic type of multiplexer device is that of a one-way rotary switch, as shown.



Basic Multiplexing Switch

The rotary switch, also called a wafer switch, as each layer of the switch is known as a wafer, is a mechanical device whose input is selected by rotating a shaft. In other words, the rotary switch is a manual switch that you can use to select individual data or signal lines simply by turning its inputs "ON" or "OFF". So, how can we select each data input automatically using a digital device?

In digital electronics, **multiplexers are also known as data selectors because they can "select" each input line**. are constructed from individual Analogue Switches encased in a single IC package as opposed to the "mechanical" type selectors, such as normal conventional switches and relays.

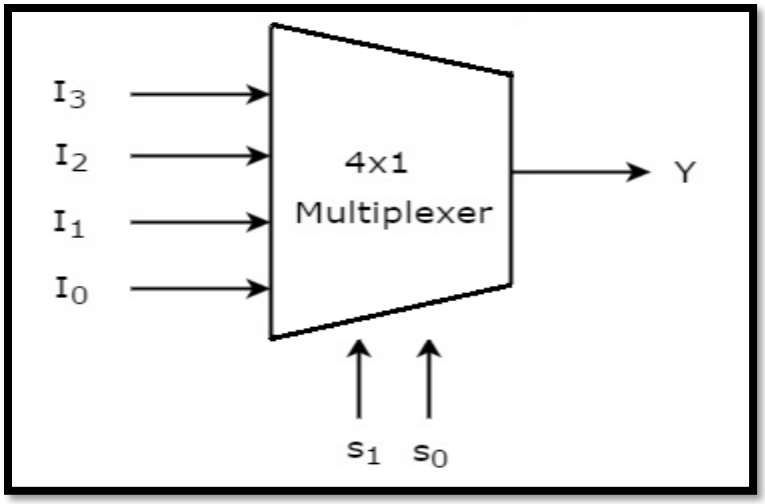
Generally, the selection of each input line in a multiplexer is controlled by an additional set of inputs called *control lines*, and according to the binary condition of these control inputs, either "HIGH" or "LOW," the appropriate data input is connected directly to the output. Normally, a multiplexer has an even number of 2^n data input lines and some "control" inputs that correspond with the number of data inputs.

Note that multiplexers are different in operation than Encoders. Encoders can switch an n-bit input pattern to multiple output lines that represent the binary coded (BCD) output equivalent to the active input.

4-to-1 Channel Multiplexer (4x1 MUX):

4x1 Multiplexer has four data inputs I_3, I_2, I_1 & I_0, two selection lines s_1 & s_0 and one output Y. One of these 4 inputs will be connected to the output based on the combination of inputs present at these two selection lines.

The **block diagram** of a 4x1 Multiplexer is shown in the following figure.



4 x 1 Multiplexer

❖ Task-1:

- ✓ ***Design a circuit for a 4x1 MUX on the trainer board.***
- ✓ ***Verify the given truth table and the Boolean expression.***
- ✓ ***Draw the Logic Diagram for a 4x1 MUX circuit.***

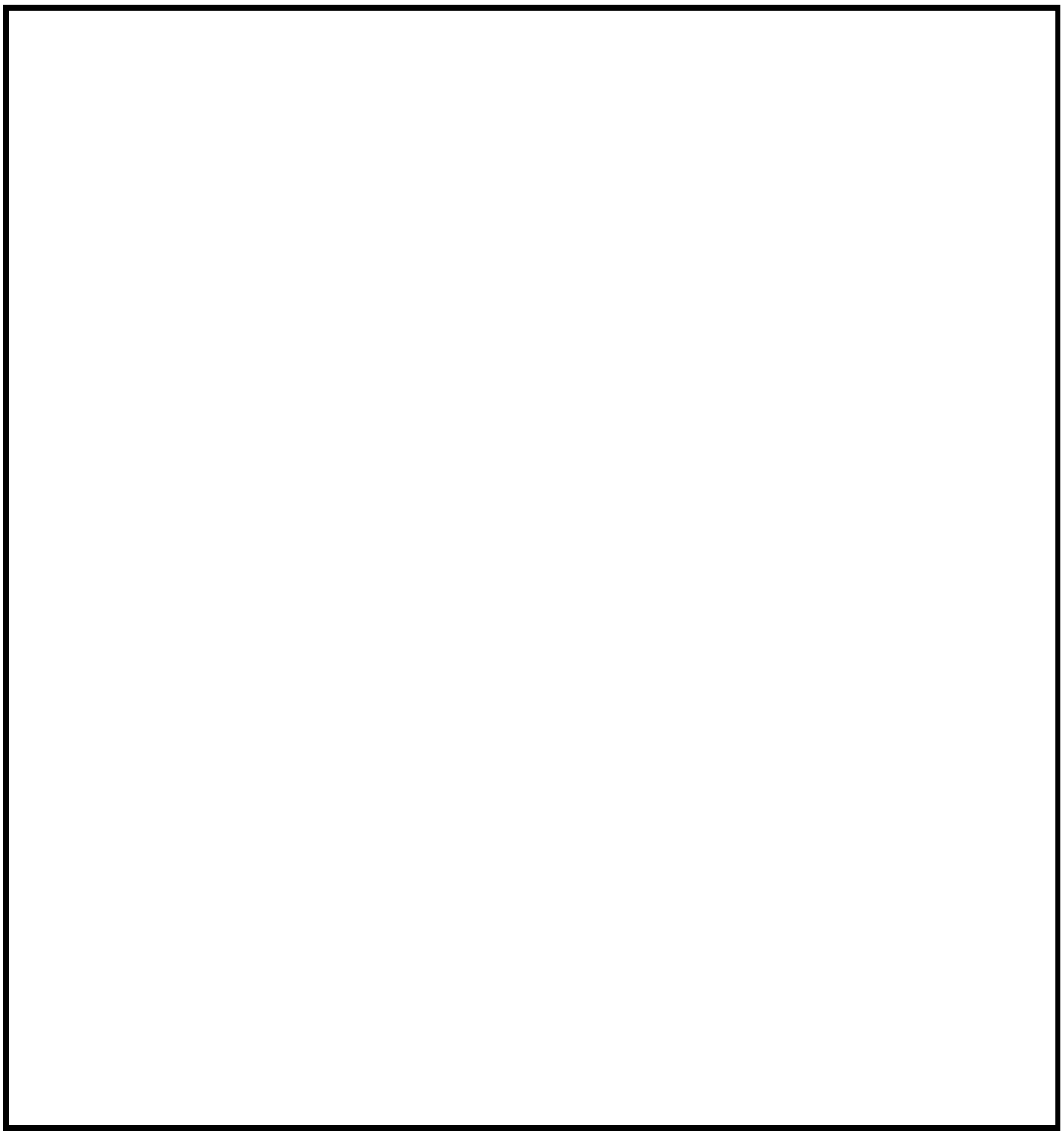
- **Truth Table:**

Inputs				Select Lines		Output
I_3	I_2	I_1	I_0	S_1	S_0	Y
0	0	0	1	0	0	I_0
0	0	1	0	0	1	I_1
0	1	0	0	1	0	I_2
1	0	0	0	1	1	I_3

- **Boolean Expression:**

Y =

- **Logic Diagram:**

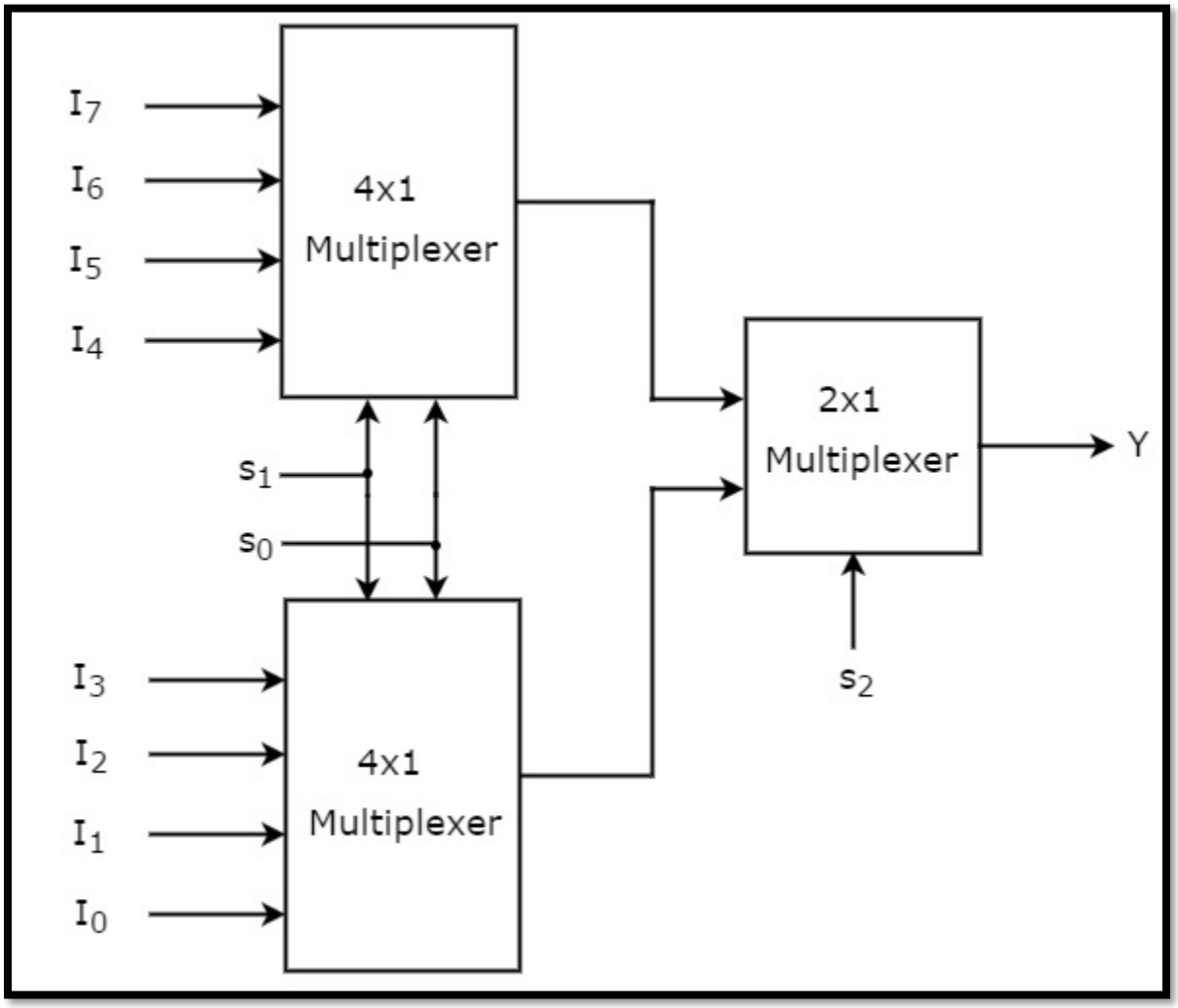


8-to-1 Channel Multiplexer (8x1 MUX):

In this section, let us implement an 8x1 Multiplexer using 4x1 Multiplexers and 2x1 Multiplexers. We know that a 4x1 Multiplexer has 4 data inputs, 2 selection lines, and one output. Whereas an 8x1 Multiplexer has 8 data inputs, 3 selection lines, and one output.

So, we require two **4x1 Multiplexers** in the first stage in order to get the 8 data inputs. Since each 4x1 Multiplexer produces one output, we require a **2x1 Multiplexer** in the second stage by considering the outputs of the first stage as inputs to produce the final output.

Let the 8x1 Multiplexer have eight data inputs I_7 to I_0, three selection lines s_2, s_1 & s0, and one output Y. The **block diagram** of a 4x1 Multiplexer is shown in the following figure.



8 x 1 Multiplexer

The same **selection lines, s_1 & s_0**, are applied to 4x1 Multiplexers. The data inputs of the upper 4x1 Multiplexer are I_7 to I_4, and the data inputs of the lower 4x1 Multiplexer are I_3 to I_0.

Therefore, each 4x1 Multiplexer produces an output based on the values of selection lines, s_1 & s_0.

The outputs of the first stage 4x1 Multiplexers are applied as inputs of the 2x1 Multiplexer that is present in the second stage. The other **selection line, S_2**, is applied to a 2x1 Multiplexer.

- If s_2 is zero, then the output of 2x1 Multiplexer will be one of the 4 inputs I_3 to I_0 based on the values of selection lines s_1 & s_0.
- If s_2 is one, then the output of 2x1 Multiplexer will be one of the 4 inputs I_7 to I_4 based on the values of selection lines s_1 & s_0.

Therefore, the overall combination of two 4x1 Multiplexers and one 2x1 Multiplexer performs as one 8x1 Multiplexer.

Similarly, we can implement a **16x1 Multiplexer** using 8x1 Multiplexers and 2x1 Multiplexers. We know that an 8x1 Multiplexer has 8 data inputs, 3 selection lines, and one output. Whereas a 16x1 Multiplexer has 16 data inputs, 4 selection lines, and one output.

So, we require two 8x1 Multiplexers in the first stage in order to get 16 data inputs. Since each 8x1 Multiplexer produces one output, we require a 2x1 Multiplexer in the second stage, by considering the outputs of the first stage as inputs, to produce the final output.

Instantiation in Verilog

Instantiation in Verilog is the process of creating instances (copies) of existing modules to build hierarchical designs, allowing designers to reuse pre-defined components and create complex systems from simpler building blocks. It follows a "bottom-up" design methodology where smaller modules are first created and then instantiated within larger modules. Each instantiation creates a unique copy of the module with its own set of ports and internal signals. The general syntax involves specifying the module name, instance name, and connecting the ports either by ordered list or by name. This approach promotes code reusability, modular design, and easier debugging by breaking down complex circuits into manageable components.

TASK # 02 Verilog HDL code of 2:1 MUX

Data Flow Modeling

Code

```
module mux2_1(in1, in2, select, out);
input in1, in2, select;
output out;
assign out = select ? in2 : in1;
// assign out =(~select&in1)|(select&in2);
endmodule
```

Test Bench

```
module tb_mux2_1;
   reg in1, in2, select;
   wire out;
   mux2_1 uut (
      .in1(in1),
      .in2(in2),
      .select(select),
      .out(out));
   initial begin
       in1 = 0; in2 = 0; select = 0; #10;
      in1 = 0; in2 = 1; select = 0; #10;
      in1 = 0; in2 = 1; select = 1; #10;
      in1 = 1; in2 = 0; select = 0; #10;
      in1 = 1; in2 = 0; select = 1; #10;
      in1 = 1; in2 = 1; select = 0; #10;
    in1 = 1; in2 = 1; select = 1; #10;
      $finish;
   end
endmodule
```

Task # 03: ***Design and implement a 4×1 Multiplexer using gate-level modeling in Verilog HDL. Write a Verilog code that describes the functionality of the 4×1 MUX using basic logic gates (and, or, not) and develop a testbench to verify its correct operation for all possible input combinations.***
Simulate the design using a Verilog simulator (e.g., Vivado) and observe the output waveforms.?

Task # 04: ***Design and implement a 4×1 Multiplexer using gate-level modeling in Verilog HDL.***

Code:

```
module mux_4x1(op,i0,i1,i2,i3,s0,s1);
input i0,i1,i2,i3;
input s1,s0;
output op;
wire s1bar,s0bar;
wire w1,w2,w3,w4;
not n1(s1bar,s1);
not n2 (s0bar,s0);
and a1(w1,i0,s1bar,s0bar);
and a2 (w2,i1,s1bar,s0);
and a3(w3,i2,s1,s0bar);
and a4(w4,i3,s1,s0);
or o1 (op,w1,w2,w3,w4);
endmodule
```

Test Bench

```
module mux_4x1_tb;
   // Testbench signals
   reg i0, i1, i2, i3;
   reg s0, s1;
   wire op;
   mux_4x1 uut (
      .op(op),
```

```
        .i0(i0),
        .i1(i1),
        .i2(i2),
        .i3(i3),
        .s0(s0),
        .s1(s1)
    );
    initial begin
       i0 = 0; i1 = 1; i2 = 0; i3 = 1;
        s1 = 0; s0 = 0; #10;
        s1 = 0; s0 = 1; #10;
        s1 = 1; s0 = 0; #10;
       s1 = 1; s0 = 1; #10;
        // another input combination
        i0 = 1; i1 = 0; i2 = 1; i3 = 0;
        s1 = 0; s0 = 0; #10;
        s1 = 0; s0 = 1; #10;
        s1 = 1; s0 = 0; #10;
        s1 = 1; s0 = 1; #10;
        $finish;
    end
endmodule
```

Task #05: In a digital system, it is required to select one signal out of eight input lines and send it to a single output line based on a 3-bit select input. Design and implement an 8×1 Multiplexer using basic logic gates (and, or, not) in Verilog HDL. Write a testbench to verify the operation of the multiplexer for all possible select inputs. Simulate the design using a Verilog simulator (such as Vivado) and analyze the output waveforms.

Example

Design and implement a 4-to-1 multiplexer using gate-level modeling by creating a basic 2×1 multiplexer with fundamental logic gates (NOT, AND, OR) and then constructing the 4×1 multiplexer through hierarchical instantiation of three 2×1 multiplexers in a two-stage tree structure, with proper port connections and verification of all selection cases.

```
module mux2x1_gate (output Y, input A, B, S);
    wire S_bar, and1_out, and2_out;
    not (S_bar, S);
    and (and1_out, A, S_bar);
    and (and2_out, B, S);
    or (Y, and1_out, and2_out);
endmodule
```

// 4x1 Multiplexer using 2x1 Multiplexers

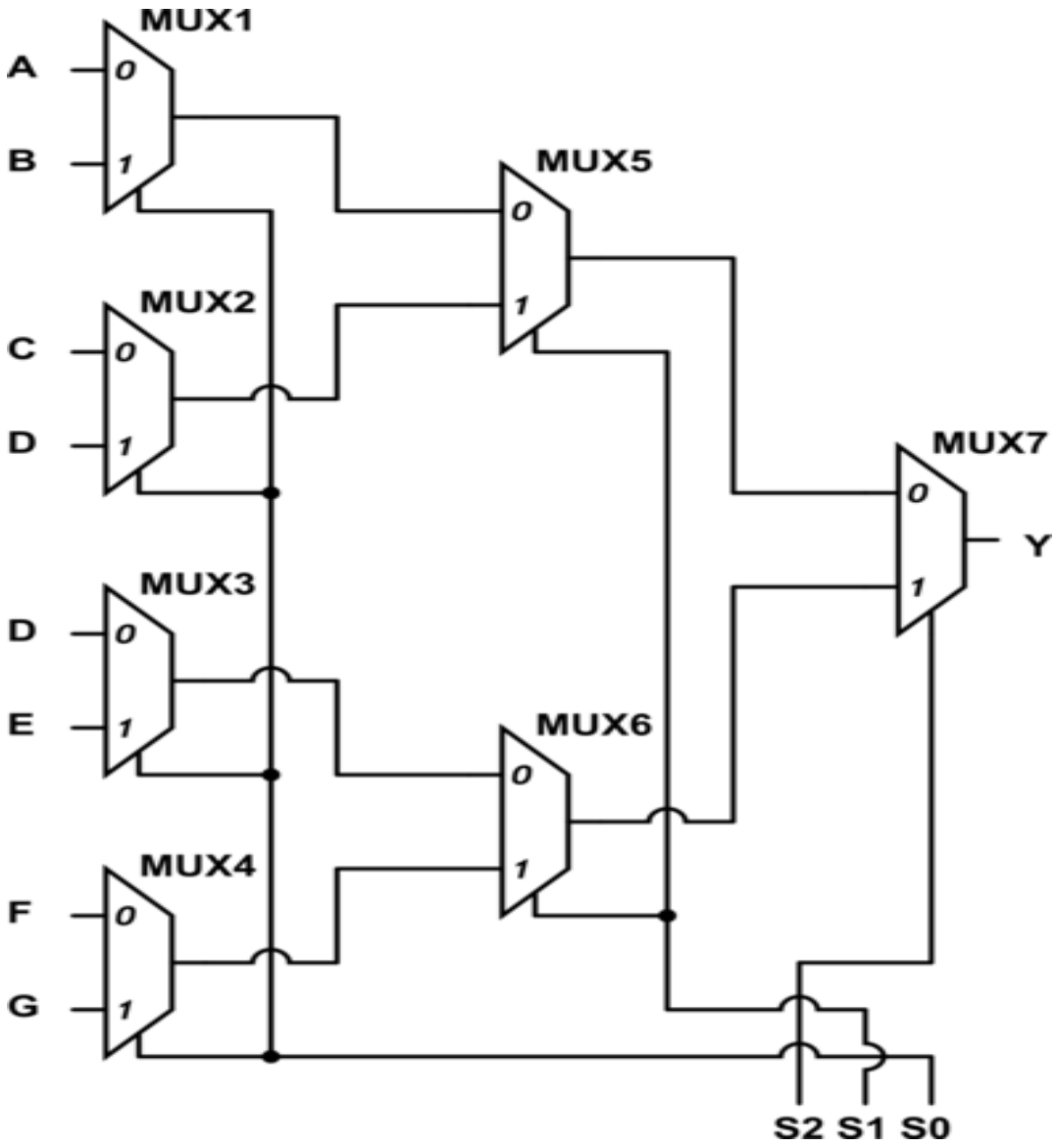
```
module mux4x1_using_2x1 (output Y, input [3:0] I, input [1:0] S);
    wire [1:0] stage_out;
    mux2x1_gate mux0 (stage_out[0], I[0], I[1], S[0]);
    mux2x1_gate mux1 (stage_out[1], I[2], I[3], S[0]);
    mux2x1_gate mux2 (Y, stage_out[0], stage_out[1], S[1]);
endmodule
```

Test Bench

```
module tb_mux4x1;
    reg [3:0] I;
    reg [1:0] S;
    wire Y;
    // Instantiate the 4x1 mux
    mux4x1_using_2x1 dut (Y, I, S); //pos assoc
// mux4x1_using_2x1 dut (.Y(Y), .I(I), .S(S)); //named assoc
    initial begin
        // Test all input selections
        I = 4'b0001; S = 2'b00; #10;
        I = 4'b0010; S = 2'b01; #10;
        I = 4'b0100; S = 2'b10; #10;
        I = 4'b1000; S = 2'b11; #10;
        // Test with multiple inputs high
        I = 4'b1101; S = 2'b00; #10;
        I = 4'b1101; S = 2'b01; #10;
        I = 4'b1101; S = 2'b10; #10;
        I = 4'b1101; S = 2'b11; #10;
        // Test all zeros
        I = 4'b0000; S = 2'b00; #10;
        I = 4'b0000; S = 2'b01; #10;
        I = 4'b0000; S = 2'b10; #10;
        I = 4'b0000; S = 2'b11; #10;
        // Test all ones
        I = 4'b1111; S = 2'b00; #10;
        I = 4'b1111; S = 2'b01; #10;
        I = 4'b1111; S = 2'b10; #10;
        I = 4'b1111; S = 2'b11; #10;
```

```
    $finish;
  end
endmodule
```

Task # 06: Design and implement an 8×1 multiplexer using hierarchical instantiation of 2×1 multiplexers in Verilog. Create a gate-level model of a 2×1 multiplexer using basic logic gates (NOT, AND, OR) and then construct the 8×1 multiplexer by instantiating multiple 2×1 multiplexers in a three-stage tree structure (4,2,1). Verify the design functionality through simulation by testing all input combinations.



CE221L-Digital Logic Design Lab

RUBRIC SHEET

Criteria	Excellent (Full Marks)	Good (Partial Marks)	Needs Improvement (Low Marks)	Max Marks
Apparatus Handling *(Apply)* *(3 marks)*	Applies proper techniques in handling ICs, breadboards, and power supplies with precision and safety.	Apply correct techniques with minor errors or guidance needed.	Fails to apply correct handling techniques, leading to unsafe practices or component damage.	**/3**
Hardware Designing *(Construct)* *(4 marks)*	Constructs accurate and fully functional digital circuits using logic gates and ICs without assistance.	Constructs circuits with minor mistakes, requiring some corrections or support.	Unable to construct a working circuit, with major flaws in design or connections.	**/4**
Objectives *(Apply & Construct)* *(5 marks)*	Apply theoretical concepts effectively to construct the required circuit and fully achieve the experiment's objectives.	Applies concepts partially and constructs a circuit that partially meets objectives.	Unable to apply theory correctly or construct a circuit to meet objectives.	**/5**
Viva *(Collaborate)* *(3 marks)*	Collaborate effectively by communicating ideas clearly, answering questions confidently, and accurately.	Communicates and collaborates with some hesitation or minor conceptual gaps.	Struggles to collaborate or communicate concepts; unable to answer correctly.	**/3**
Total				**/15**

Name: _____ Instructor Signature: _____

Reg #: _____ Date: _____