CS 223 Lab Assignment III
Preliminary Work
Sıla Bozkurt
22401775
Section: 3
Instructor: Sinem Sav
November 23, 2025

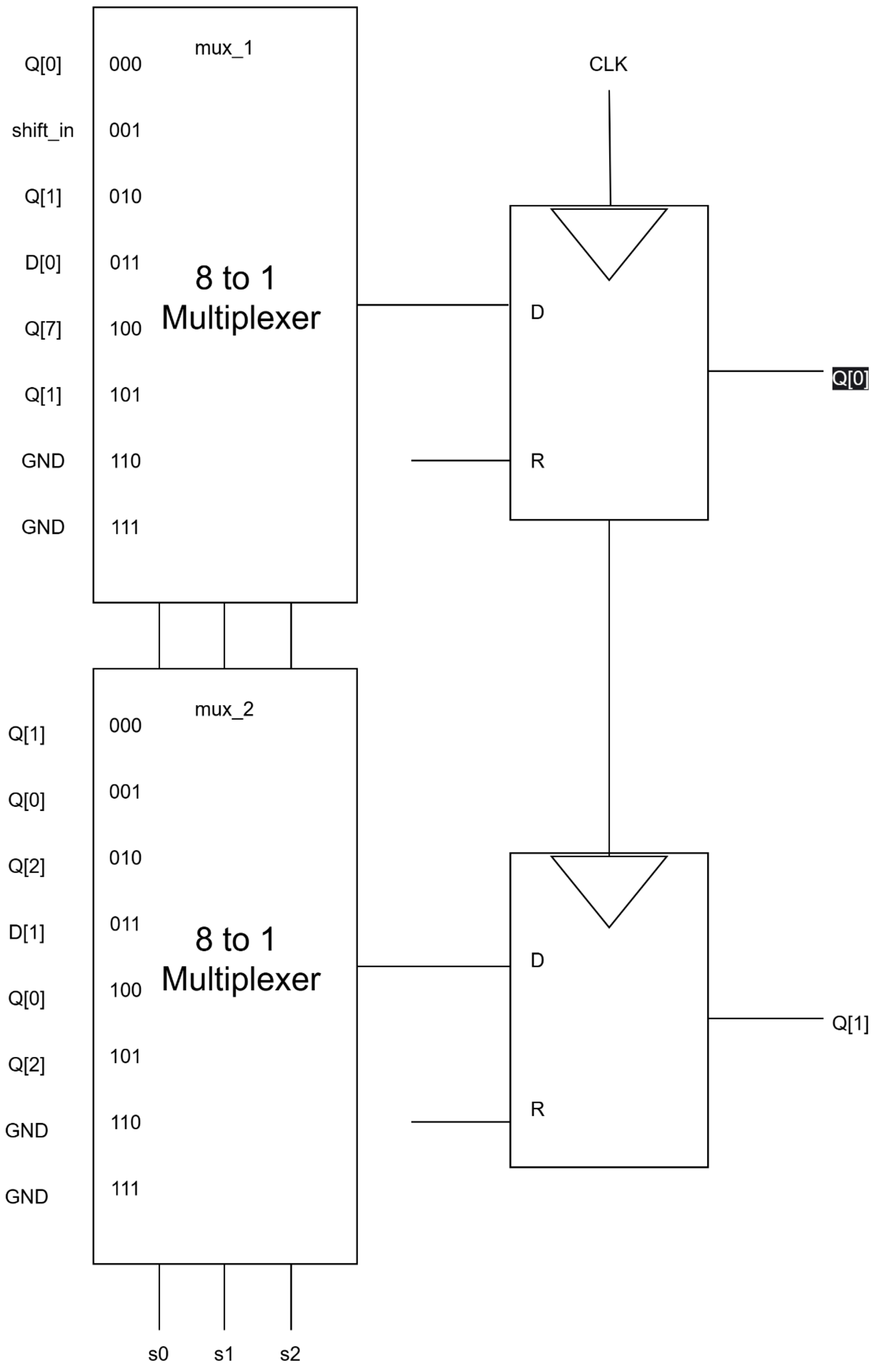
1) Multifunctional Register:

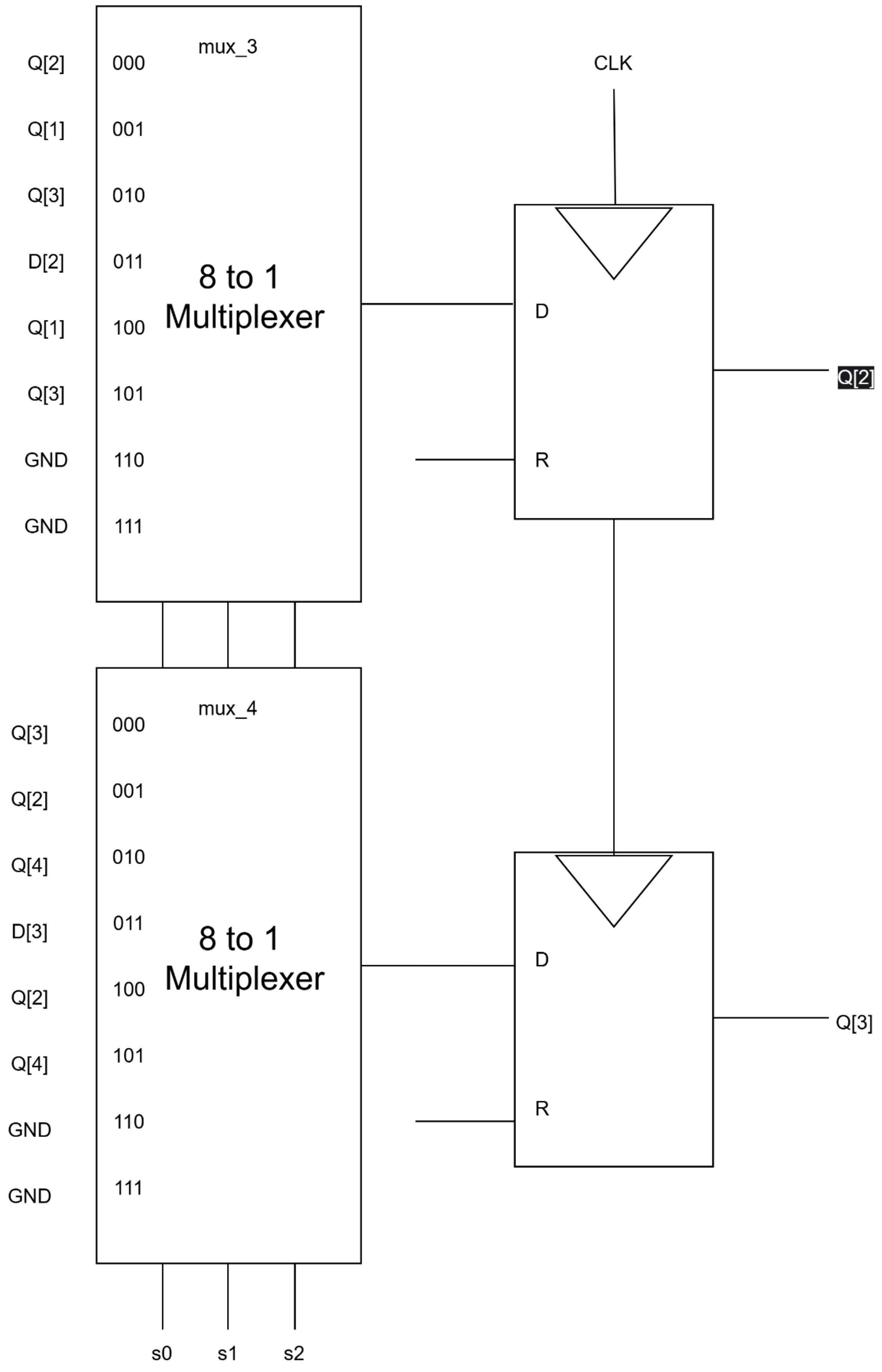
In this experiment, eight resettable D flip-flops with inputs and outputs will be used. All flip-flops will be loaded on every clock cycle. To meet the criteria, I used eight separate 8:1 multiplexers to determine how each output and d and q values of each D flip-flop would change according to the values transferred by the select inputs.
For clearance purposes, I drew each of the two 8-multiplexers separately, although they share common select lines and a common clock that has been slowed in accordance with the lab manual's specifications.

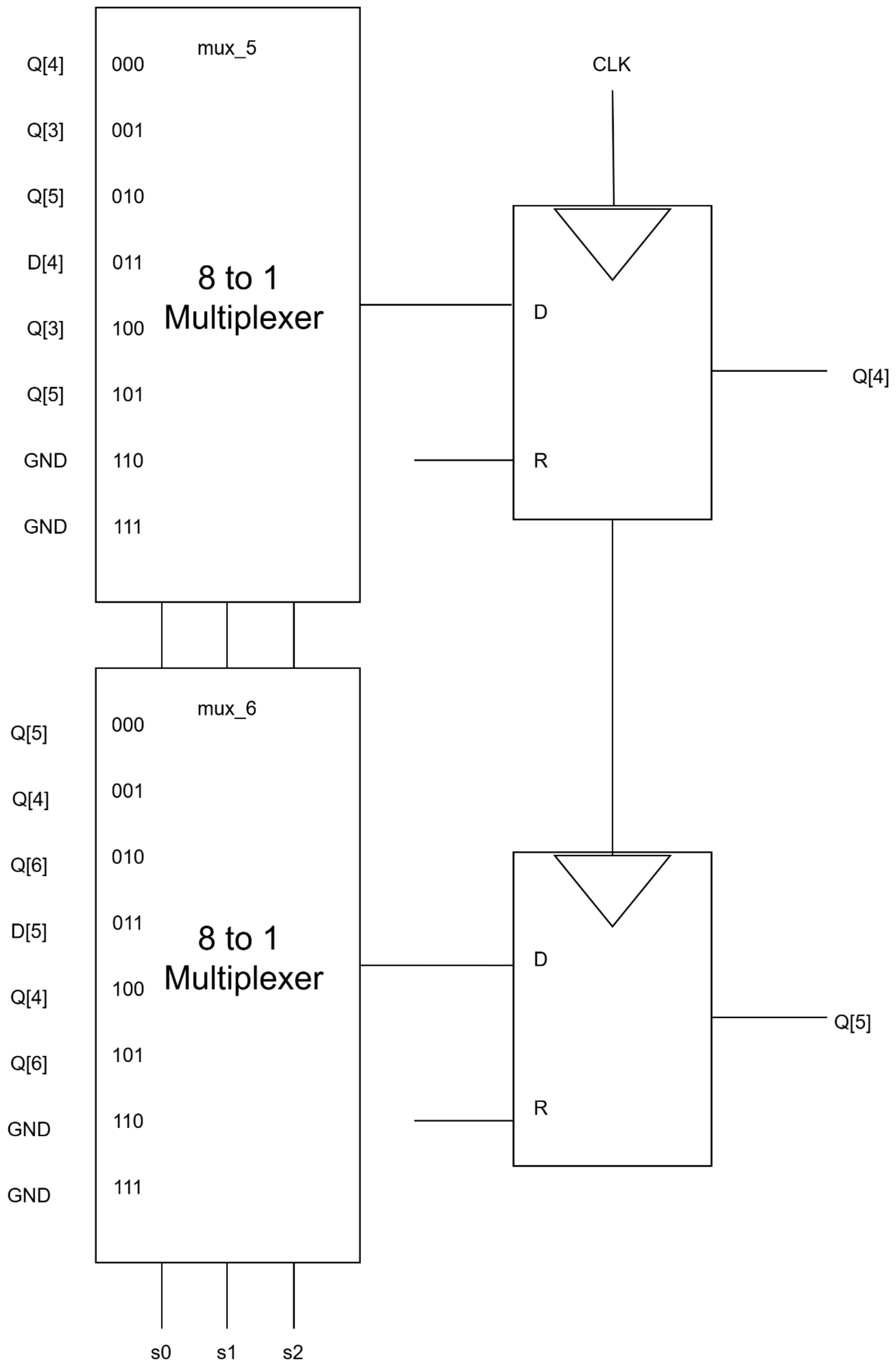
1.1 Module for D Flip-Flop

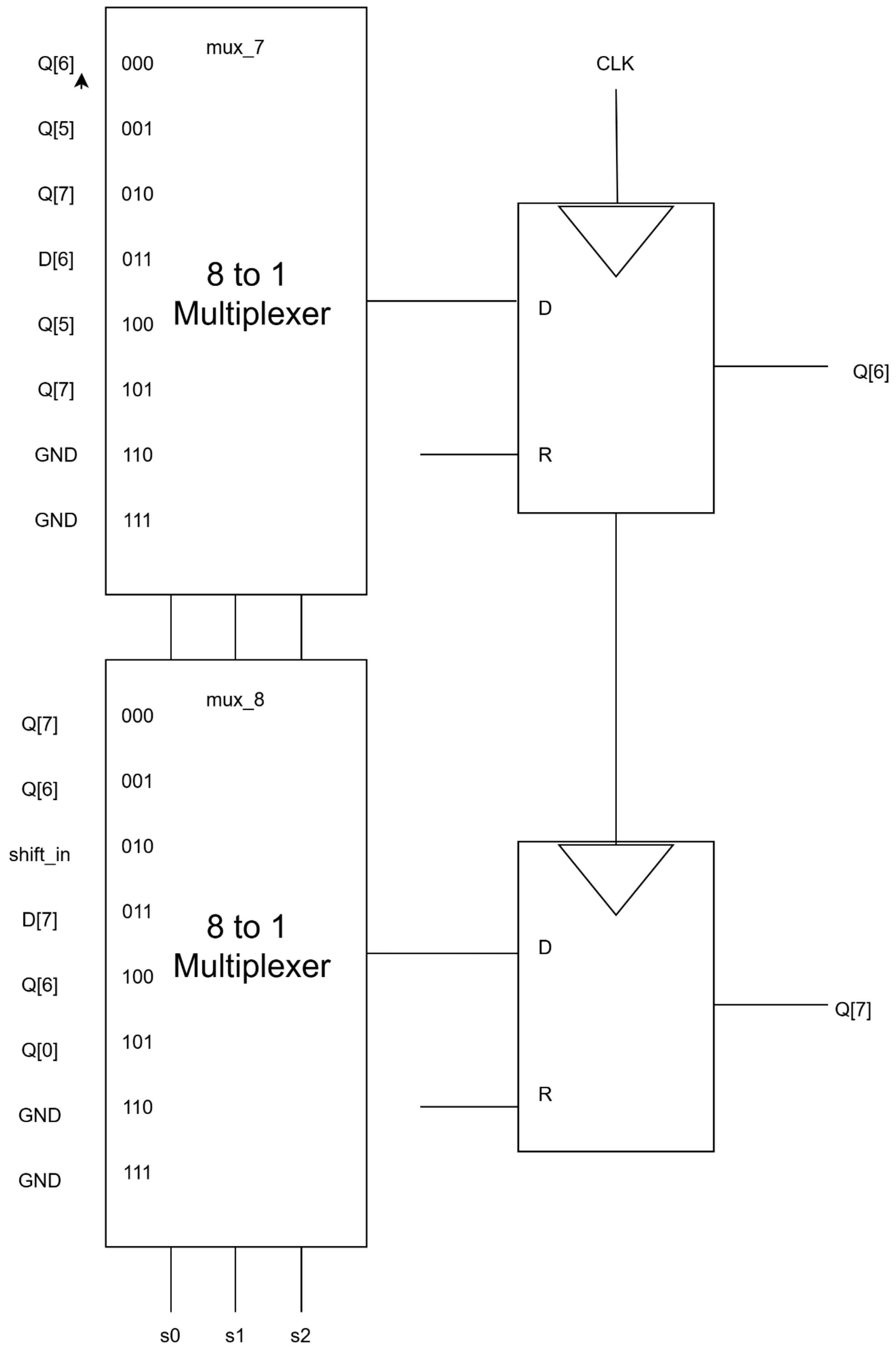
```
module d_flip_flop(
    input logic clk,
    input logic reset,
    input logic d,
    output logic q
    );
    always_ff @(posedge clk) begin
        if (reset) q <= 1'b0;
        else       q <= d;
    end
endmodule
```

1.2 Circuit Schematics:




mux_3
Q[2] 000
Q[1] 001
Q[3] 010
D[2] 011
Q[1] 100
Q[3] 101
GND 110
GND 111
8 to 1 Multiplexer
CLK
D
R
Q[2]
mux_4
Q[3] 000
Q[2] 001
Q[4] 010
D[3] 011
Q[2] 100
Q[4] 101
GND 110
GND 111
8 to 1 Multiplexer
D
R
Q[3]
s0
s1
s2


mux_5
Q[4]
000
Q[3]
001
Q[5]
010
D[4]
011
Q[3]
100
Q[5]
101
GND
110
GND
111
8 to 1 Multiplexer
CLK
D
R
Q[4]
mux_6
Q[5]
000
Q[4]
001
Q[6]
010
D[5]
011
Q[4]
100
Q[6]
101
GND
110
GND
111
8 to 1 Multiplexer
D
R
Q[5]
s0
s1
s2


mux_7
Q[6] 000
Q[5] 001
Q[7] 010
D[6] 011
Q[5] 100
Q[7] 101
GND 110
GND 111
8 to 1 Multiplexer
CLK
D
R
Q[6]
mux_8
Q[7] 000
Q[6] 001
shift_in 010
D[7] 011
Q[6] 100
Q[0] 101
GND 110
GND 111
8 to 1 Multiplexer
D
R
Q[7]
s0
s1
s2

1.3 SystemVerilog Code and Testbenches

1.2.1 Module for multifunctional Register and its Top Module with Adapted Clock

```
module multifunction_register(
    input logic clkQ,
    input logic reset,
    input logic [2:0] select,
    input logic [7:0] D,      // for parallel input
    input logic shift_in,
    output logic [7:0] Q
    );

    logic [7:0] next_state;

    mux8to1 mux_0 (
        .sel(select),
        .i0(Q[0]),
        .i1(shift_in),
        .i2(Q[1]),
        .i3(D[0]),
        .i4(Q[7]),
        .i5(Q[1]),
        .i6(1'b0),
        .i7(1'b0),
        .y(next_state[0])
    );

    mux8to1 mux_1 (
        .sel(select),
        .i0(Q[1]),
        .i1(Q[0]),
        .i2(Q[2]),
        .i3(D[1]),
        .i4(Q[0]),
        .i5(Q[2]),
        .i6(1'b0),
        .i7(1'b0),
        .y(next_state[1])
);

mux8to1 mux_2 (
    .sel(select),
    .i0(Q[2]),
    .i1(Q[1]),
    .i2(Q[3]),
    .i3(D[2]),
    .i4(Q[1]),
    .i5(Q[3]),
    .i6(1'b0),
    .i7(1'b0),
    .y(next_state[2])
);

mux8to1 mux_3 (
    .sel(select),
    .i0(Q[3]),
    .i1(Q[2]),
    .i2(Q[4]),
    .i3(D[3]),
    .i4(Q[2]),
    .i5(Q[4]),
    .i6(1'b0),
    .i7(1'b0),
    .y(next_state[3])
);

mux8to1 mux_4 (
    .sel(select),
    .i0(Q[4]),
    .i1(Q[3]),
    .i2(Q[5]),
    .i3(D[4]),
    .i4(Q[3]),
    .i5(Q[5]),
    .i6(1'b0),
    .i7(1'b0),
    .y(next_state[4])
);
```

```
mux8to1 mux_5 (
    .sel(select),
    .i0(Q[5]),
    .i1(Q[4]),
    .i2(Q[6]),
    .i3(D[5]),
    .i4(Q[4]),
    .i5(Q[6]),
    .i6(1'b0),
    .i7(1'b0),
    .y(next_state[5])
);

mux8to1 mux_6 (
    .sel(select),
    .i0(Q[6]),
    .i1(Q[5]),
    .i2(Q[7]),
    .i3(D[6]),
    .i4(Q[5]),
    .i5(Q[7]),
    .i6(1'b0),
    .i7(1'b0),
    .y(next_state[6])
);

mux8to1 mux_7 (
    .sel(select),
    .i0(Q[7]),
    .i1(Q[6]),
    .i2(shift_in),
    .i3(D[7]),
    .i4(Q[6]),
    .i5(Q[0]),
    .i6(1'b0),
    .i7(1'b0),
    .y(next_state[7])
);
```

```
    d_flipflop ff_0 (.clk(clk), .reset(reset), .d(next_state[0]), .q(Q[0]));
    d_flipflop ff_1 (.clk(clk), .reset(reset), .d(next_state[1]), .q(Q[1]));
    d_flipflop ff_2 (.clk(clk), .reset(reset), .d(next_state[2]), .q(Q[2]));
    d_flipflop ff_3 (.clk(clk), .reset(reset), .d(next_state[3]), .q(Q[3]));
    d_flipflop ff_4 (.clk(clk), .reset(reset), .d(next_state[4]), .q(Q[4]));
    d_flipflop ff_5 (.clk(clk), .reset(reset), .d(next_state[5]), .q(Q[5]));
    d_flipflop ff_6 (.clk(clk), .reset(reset), .d(next_state[6]), .q(Q[6]));
    d_flipflop ff_7 (.clk(clk), .reset(reset), .d(next_state[7]), .q(Q[7]));
endmodule

module multifunc_register( //top module
    input logic clk,
    input logic [15:0] sw,
    output logic [15:0] led
    );

    // Per Lab Manual Part 3 suggestions:
    // SW[15:13] -> Select Lines
    // SW[12]    -> Reset
    // SW[7:0]   -> Data Inputs (D)
    // SW[8]     -> Shift In

    logic [2:0] select_wire;
    logic reset_wire;
    logic [7:0] D_wire;
    logic shift_in_wire;
    logic [7:0] Q_wire;
    logic slow_clk;

    assign select_wire   = sw[15:13]; // Switches 15, 14, 13
    assign reset_wire    = sw[12];    // Switch 12
    assign shift_in_wire = sw[8];     // Switch 8
    assign D_wire        = sw[7:0];   // Switches 7 down to 0

    // Connect output Q to the rightmost LEDs
    assign led[7:0] = Q_wire;

    assign led[15:8] = 8'b00000000;
    // The Basys3 clock is 100 MHz, in order  to slow it down:
    clock_dividerclk_div_inst (
        .clk_in(clk),
        .reset(reset_wire),
        .clk_out(slow_clk)
    );


    multifunction_register my_reg_inst (
        .clk(slow_clk),
        .reset(reset_wire),
        .select(select_wire),
        .D(D_wire),
        .shift_in(shift_in_wire),
        .Q(Q_wire)
    );

endmodule
```

1.2.2 Test bench

```
module tb_multifunction_register();

    logic clk;
    logic reset;
    logic [2:0] select;
    logic [7:0] D;
    logic shift_in;
    logic [7:0] Q;

    multifunction_register dut (
        .clk(clk),
        .reset(reset),
        .select(select),
        .D(D),
        .shift_in(shift_in),
        .Q(Q)
    );

    always begin
```

```
        clk = 0; #5;
        clk = 1; #5;
    end

    initial begin
        reset = 1; select = 0; D = 0; shift_in = 0; #10;
        reset = 0; #10;
        select = 3'b000; D = 8'hFF; shift_in = 0; #10;

        select = 3'b011; D = 8'b10101010; #10;

        select = 3'b000; D = 8'h00; #10;

        select = 3'b001; shift_in = 0; #10;
        select = 3'b001; shift_in = 1; #10;
        select = 3'b001; shift_in = 1; #10;
        select = 3'b010; shift_in = 0; #10;
        select = 3'b010; shift_in = 1; #10;

        select = 3'b011; D = 8'b10000001; #10;
        select = 3'b100; #10;
        select = 3'b100; #10;
        select = 3'b101; #10;
        select = 3'b101; #10;
        select = 3'b110; #10;

        select = 3'b011; D = 8'hFF; #10;
        select = 3'b111; #10;

        $finish;
    end

endmodule
```

2) Hexadecimal to 7-Segment Decoder

To convert a hexadecimal value to seven-segment, four input bits for the digit are taken, and the given truth table is used to determine which of the seven segments should be on or off. In this lab, the truth table is reversed, meaning a 0 turns a segment on and a 1 turns it off. The outputs a through g then activate the correct segments to display the corresponding digit.

2.1 Module and Testbench

```
module hexadecimal_7segment(
   input  logic [3:0] sw,                 // 4-bit Input
   output logic a, b, c, d, e, f, g       // 7 segment outputs
   );

   always_comb begin
      case (sw)
         // Logic: 0 = Segment ON, 1 = Segment OFF
         // Pattern order is {a, b, c, d, e, f, g}

         4'b0000: {a,b,c,d,e,f,g} = 7'b0000001; // 0 (Only g is OFF)
         4'b0001: {a,b,c,d,e,f,g} = 7'b1001111; // 1 (Only b,c are ON)
         4'b0010: {a,b,c,d,e,f,g} = 7'b0010010; // 2
         4'b0011: {a,b,c,d,e,f,g} = 7'b0000110; // 3

         4'b0100: {a,b,c,d,e,f,g} = 7'b1001100; // 4
         4'b0101: {a,b,c,d,e,f,g} = 7'b0100100; // 5
         4'b0110: {a,b,c,d,e,f,g} = 7'b0100000; // 6
         4'b0111: {a,b,c,d,e,f,g} = 7'b0001111; // 7 (Only a,b,c are ON)

         4'b1000: {a,b,c,d,e,f,g} = 7'b0000000; // 8 (All ON)
         4'b1001: {a,b,c,d,e,f,g} = 7'b0000100; // 9
         4'b1010: {a,b,c,d,e,f,g} = 7'b0001000; // A
         4'b1011: {a,b,c,d,e,f,g} = 7'b1100000; // b

         4'b1100: {a,b,c,d,e,f,g} = 7'b0110001; // C
         4'b1101: {a,b,c,d,e,f,g} = 7'b1000010; // d
         4'b1110: {a,b,c,d,e,f,g} = 7'b0110000; // E
         4'b1111: {a,b,c,d,e,f,g} = 7'b0111000; // F
```

```
            default: {a,b,c,d,e,f,g} = 7'b1111111; // Turn all OFF if error
        endcase
    end

endmodule


module tb_sevensegmentdec();

    logic [3:0] hex_in;
    logic [6:0] seg;

    hex_to_7seg uut (
        .hex_in(hex_in),
        .seg(seg)
    );

    initial begin
        integer i;
        for (i = 0; i < 16; i = i + 1) begin
            hex_in = i;
            #10;
        end

        $finish;
    end

endmodule
```

2.2 Questions:

What are the advantages/disadvantages of using hex values to represent numbers? List two or three points and explain them with a few short sentences. Provide at least one disadvantage.

Advantages:

1. Compactness & Readability: This is the biggest one. Binary strings can become very long very quickly. A 16-bit number, such as 1011110011000000, is difficult to read. In Hex, it is just BCC0. It groups bits into "nibbles" (4 bits), making it much easier for humans to spot patterns and errors.

2. Direct Mapping: Unlike decimal (base-10), Hex (base-16) maps perfectly to binary. One Hex digit *always* equals exactly four binary bits. You can convert back and forth instantly without needing to perform complex math.

Disadvantages:

1. The "Translation" Requirement

A standard 7-segment display is just a collection of 7 separate LEDs (labeled a–g). It has no built-in intelligence. If you send the binary value for the number 5 (0101) directly to the display's pins, it will not display a "5". Instead, it would light up segments e and g (based on wiring), resulting in a meaningless shape.

To make the display useful, you cannot simply wire your data directly to it. You are required to design an intermediate piece of hardware, such as a Decoder.

2. The Cost of the Decoder

This decoder incurs additional hardware overhead.

- Combinational Logic: It requires a complex web of logic gates (AND, OR, NOT) to map every 4-bit input combination (0000–1111) to a unique 7-bit output pattern.
- Wiring: It increases the number of connections. We go from 4 wires (the data) to 7 wires (the display control).

3. Comparison to Alternatives

Compare this to a simple Binary LED Bar, where 1 turns a light on and 0 turns it off.

- Binary Display: 4 wires -> 4 LEDs. No logic needed. Zero hardware complexity.
- Hex Display: 4 wires -> Complex Logic Block -> 7 wires -> LEDs.

Research how the full four-digit seven-segment display on the Basys3 board works. Think about which sequential components you need to drive it. Explain with a few short sentences.

The full four-digit seven-segment display on the Basys3 board relies on a shared-bus architecture, where all four digits share the same seven-segment lines (A-G). This means it is physically impossible to drive different numbers to all digits simultaneously using static logic. To display a full four-digit hexadecimal value, the system must utilize time-multiplexing, where the controller activates one digit at a time in a rapid sequence (exploiting the human eye's persistence of vision) to create the illusion of a steady display. This requires specific sequential components: a clock divider to generate a refresh rate (typically around 1 kHz) that is fast enough to prevent flickering but slow enough for the transistors to switch, a 2-bit counter to cyclically select which anode is active, and a multiplexer to route the corresponding 4-bit data chunk to the active digit.