1

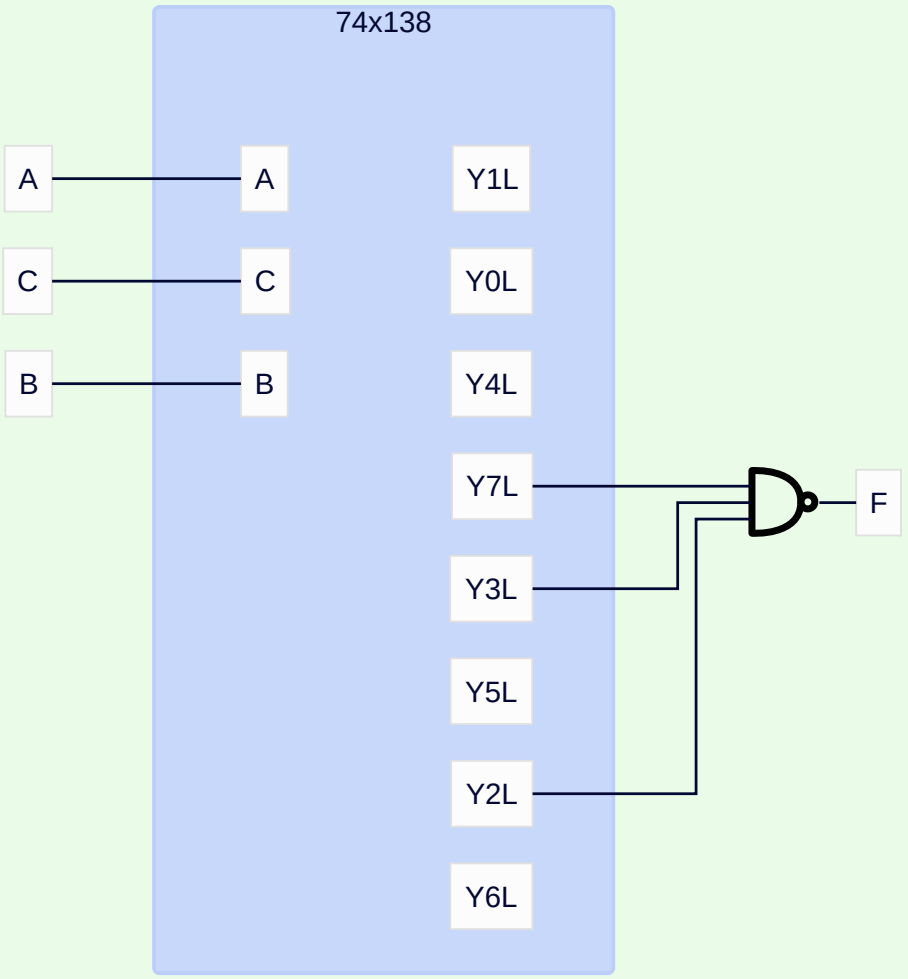
Implement the following functions using only 74x138 binary decoders and NAND gates

a

$F = \prod_{A,B,C} (0, 1, 4, 5, 6)$

✓ Answer

C\AB	00	01	11	10
0	0	0	1	1
1	0	0	1	0

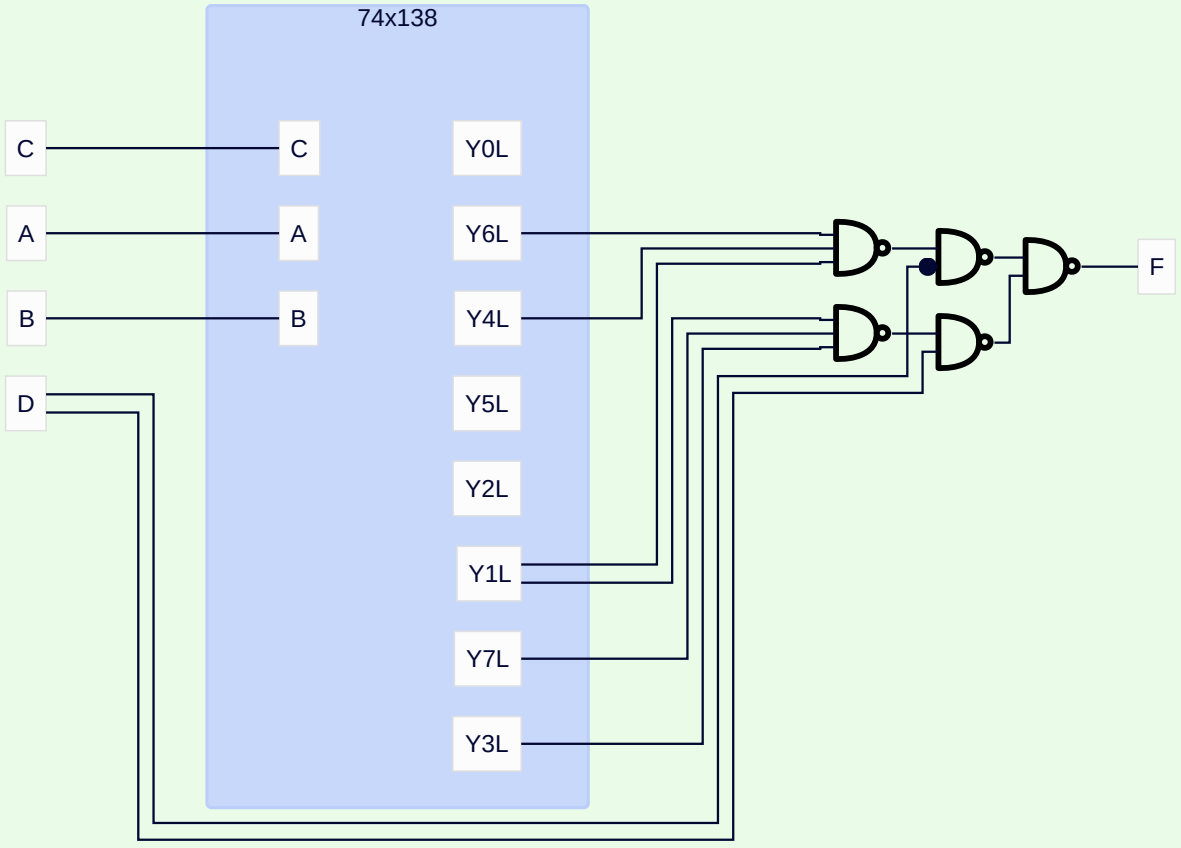


b

$F = \sum_{W,X,Y,Z} (1, 4, 6, 9, 11, 15)$

✓ Answer

CD\AB	00	01	11	10
00	0	1	0	0
01	1	0	0	1
11	0	0	1	0
10	0	1	1	0



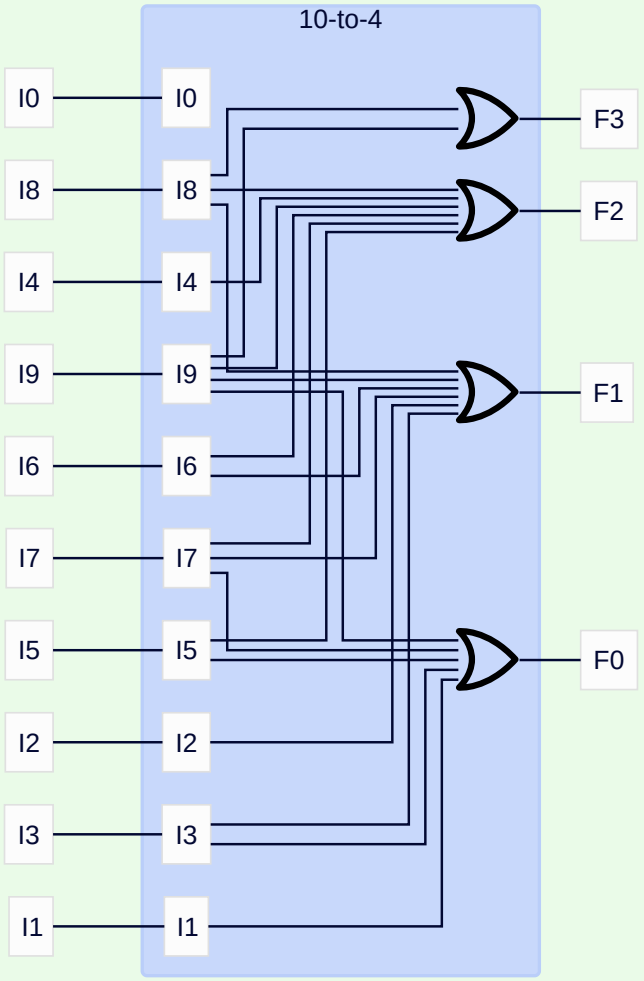
2

Design a 10-to-4 encoder with the inputs 1-out-of-10 code and outputs coded normally for 0 – 7 (binary 0000 - 0111) and 8 is coded as E (1110) and 9 is coded as F (1111). Show the internal circuit.

✓ Answer

I	Y3	Y2	Y1	Y0
0	0	0	0	0

I	Y3	Y2	Y1	Y0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	1	1	0
9	1	1	1	1



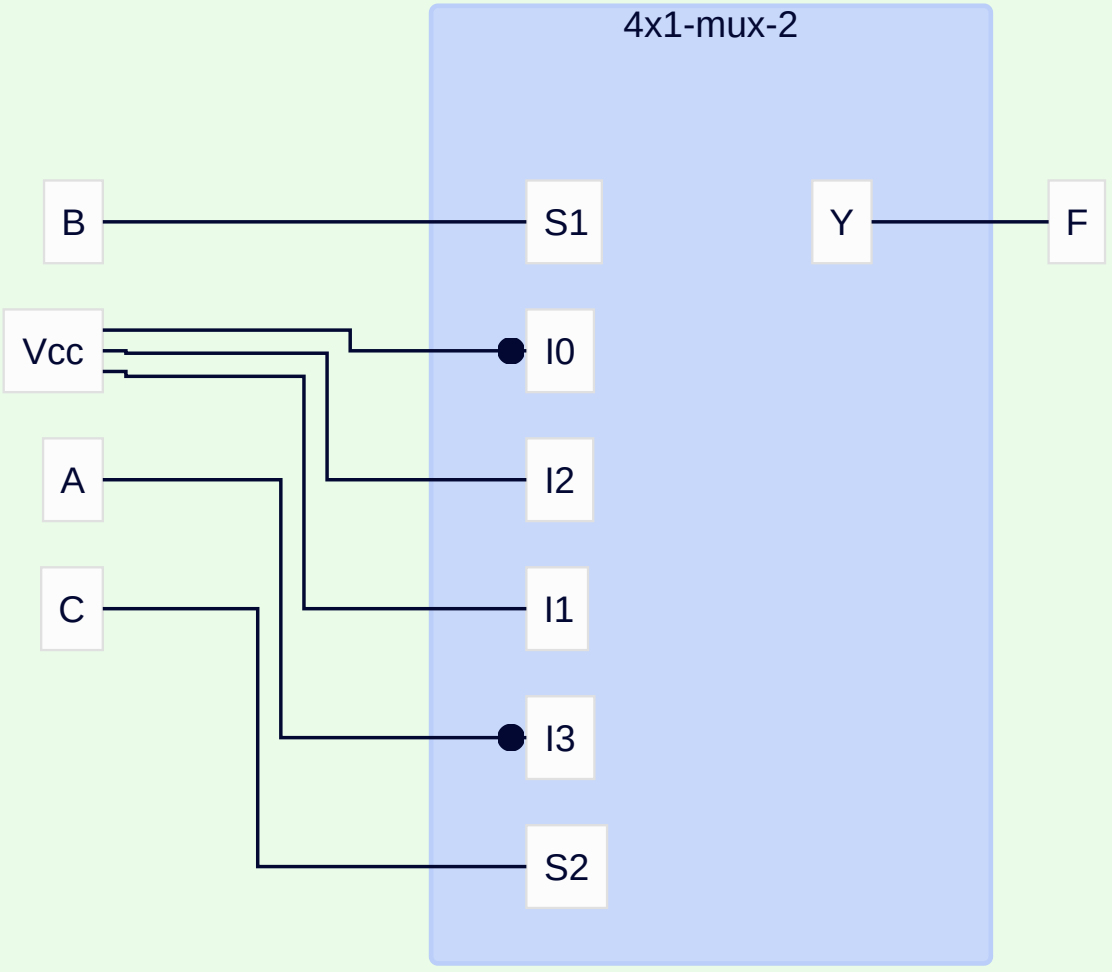
3

Implement the following function using only a single 4×1 multiplexer and inverters

a	b	c	F
0	0	0	0

a	b	c	F
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

✓ Answer



4

For the logic expressions given below, find all of the static hazards and design a hazard-free circuit that realizes the same logic function. Write the functions that are hazard free, you do not need to draw the circuit. (Hint: Use Karnaugh maps to find the timing hazards.)

a

$$F = W \cdot X + W' \cdot Y'$$

✓ **Answer**

Y\WX	00	01	11	10
0	**1**	**1**	*1*	0
1	0	0	*1*	0

$$F = W \cdot X + W' \cdot Y' + X \cdot Y'$$

b

$$F = W \cdot Y + W' \cdot Z' + X \cdot Y' \cdot Z$$

✓ **Answer**

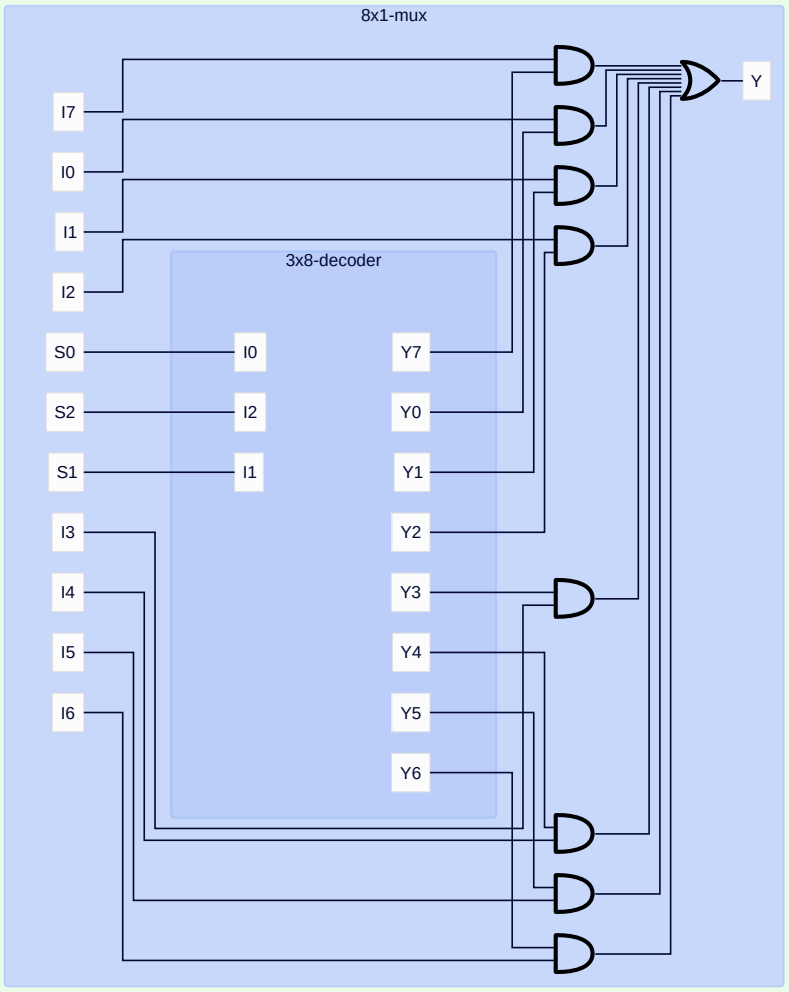
YZ\WX	00	01	11	10
00	0	0	0	0
01	*1*	*1*	1	0
11	*1*	*1*	**1**	**1**
10	0	0	**1**	**1**

$$F = W \cdot Y + W' \cdot Z' + X \cdot Z + Y \cdot Z$$

5

Implement an 8x1 multiplexer using a 3x8 decoder with active high outputs and eight AND gates and one OR gate. Label the inputs and outputs clearly

✓ **Answer**



Code

testbench_hw7.sv

```
module testbench_hw7 ();

    logic clk, en_b, load4_b, load5_b, up, rco4_b, rco5_b;
    logic [3:0] load_in4, q4;
    logic [4:0] load_in5, q5;

    up_down_counter #(
        .N(4)
    ) counter4 (
        .clk    (clk),
        .en_b   (en_b),
        .load_b (load4_b),
        .up     (up),
        .load_in(load_in4),
        .q      (q4),
        .rco_b  (rco4_b)
    );
```

```
    up_down_counter #(
        .N(5)
    ) counter5 (
        .clk    (clk),
        .en_b   (en_b),
        .load_b (load5_b),
        .up     (up),
        .load_in(load_in5),
        .q      (q5),
        .rco_b  (rco5_b)
    );

    initial begin
        clk      = 1'b0;
        en_b     = 1'b1;
        load4_b  = 1'b0;
        load5_b  = 1'b0;
        up       = 1'b1;
        load_in4 = 4'b0000;
        load_in5 = 5'b00000;

        forever #5 clk = ~clk;
    end

    initial begin
        #10;
        load4_b = 1'b0;
        load5_b = 1'b0;
        #10;
        en_b = 1'b0;
        #10;
        load4_b = 1'b1;
        load5_b = 1'b1;
        #320;
        en_b     = 1'b1;
        up       = 1'b0;
        load4_b  = 1'b0;
        load5_b  = 1'b0;
        load_in4 = 4'b1111;
        load_in5 = 5'b11111;
        #10;
        en_b = 1'b0;
        #10;
        load4_b = 1'b1;
        load5_b = 1'b1;
        en_b    = 1'b1;
        #10;
        en_b = 1'b0;
        #320;
```

```
        load4_b  = 1'b0;
        load5_b  = 1'b0;
        load_in4 = 4'b1010;
        load_in5 = 5'b01010;
        #10;
        load4_b = 1'b1;
        load5_b = 1'b1;
        #10;
        up       = 1'b1;
        load4_b  = 1'b0;
        load5_b  = 1'b0;
        load_in4 = 4'b0101;
        load_in5 = 5'b10101;
        #10;
        load4_b = 1'b1;
        load5_b = 1'b1;
        #10;
        $finish();
    end

endmodule
```

up_down_counter.sv

```
module up_down_counter #(
    parameter N = 4
) (
    input  logic         clk,
    input  logic         en_b,
    input  logic         load_b,
    input  logic         up,
    input  logic [N-1:0] load_in,
    output logic [N-1:0] q,
    output logic         rco_b
);
    always_ff @(posedge clk) begin
        if (~en_b) begin
            if (load_b) begin
                q <= up ? q + 1 : q - 1;
            end
            else begin
                q <= load_in;
            end
        end
    end

    assign rco_b = (&q && up) || ~(|q || up);
```

```
endmodule
```

Deliverables