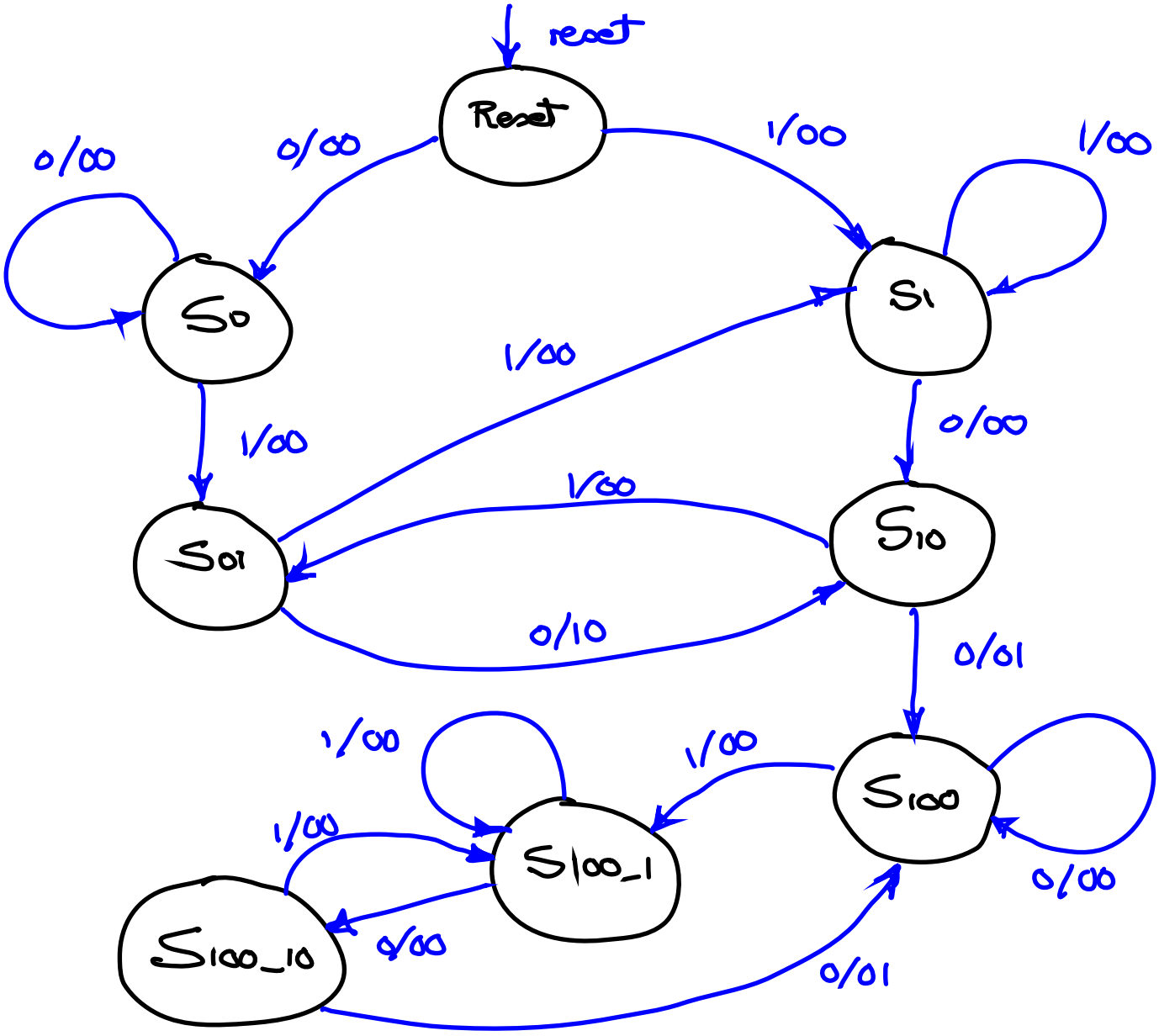
EE 417 – Assignment 3 – Problem 1 – Sequence detector with two outputs

A sequential circuit has one input (X) and two outputs (z1 and z2). An output z1 = 1 occurs every time the input sequence 010 is completed, provided that the sequence 100 has never occurred.

An output z2 = 1 occurs every time the input 100 is completed. Note that once a z2 = 1 output has occurred, z1 = 1 can never occur but not vice versa. Find a Mealy state graph and state table (minimum number of states is eight).

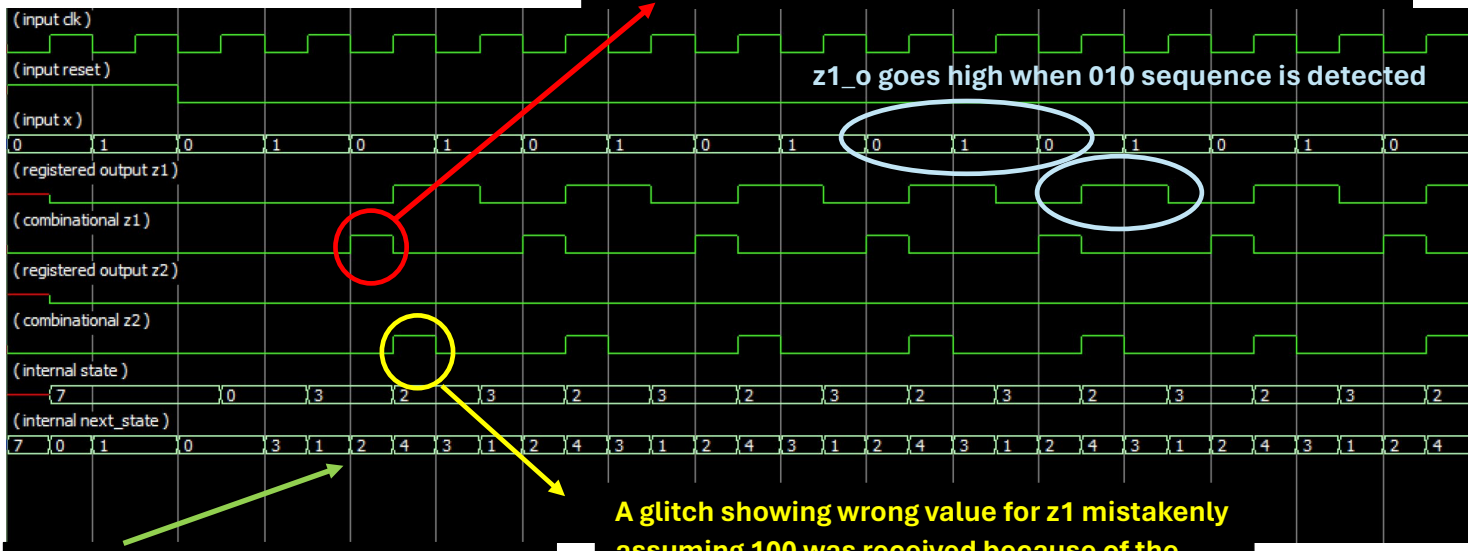
Sketch the state graph, and then create the Verilog code and verify the functionality of the FSM. Use a reset input to initialize the FSM state at the reset state.



```
/* ------------------------------------------------------------------------------------
   A sequential circuit has one input (X) and two outputs (Z1 and Z2).
   An output Z1 = 1 occurs every time the input sequence 010 is completed,
   provided that the sequence 100 has never occurred.
   An output Z2 = 1 occurs every time the input 100 is completed.
   Note that once a Z2 = 1 output has occurred, Z1 = 1 can never occur
   but not vice versa.
   Find a Mealy state graph and state table (minimum number of states is eight).

   Sketch the state graph, and then create the Verilog code and verify the functionality
   of the FSM. Use a reset input to initialize the FSM state at the reset state.
   ------------------------------------------------------------------------------------*/

   module SequenceDetector (clk, reset, x, z1_o, z2_o);

   input          clk, reset, x;
   output   reg   z1_o, z2_o;

   reg [2:0]  state, next_state;
   reg        z1, z2;


   parameter SRst    = 3'b111;   // Rst state - no inputs received  // 7

   //---------------------------------- 100 was never detected
   parameter S0      = 3'b000;   // A 0 was received                // 0
   parameter S1      = 3'b001;   // A 1 was received                // 1
   parameter S01     = 3'b011;   // Sequence 01 was received        // 3
   parameter S10     = 3'b010;   // Sequence 10 was received        // 2

   //---------------------------------- 100 was received
   parameter S100    = 3'b100;   // sequence 100 received           // 4
   parameter S100_1  = 3'b101;   // Received 1 after z2 was 1       // 5
   parameter S100_10 = 3'b110;   // sequence 10 after z2 was 1      // 6


   // assign z1_o = z1;
   // assign z2_o = z2;

   always @ (posedge clk)
   if (reset) begin
              state <= SRst;
              z1_o  <= 1'b0;
              z2_o  <= 1'b0; end
      else    begin
              state <= next_state;
              z1_o  <= z1;
              z2_o  <= z2;   end


    always @ *
       case (state)
           SRst : begin z1 = 1'b0;   z2 = 1'b0;
                        if (x) next_state = S1;
                        else   next_state = S0;   end
           S0   : begin z1 = 1'b0;   z2 = 1'b0;
                        if (x) next_state = S01;
                        else   next_state = S0;   end
           S1   : begin z1 = 1'b0;   z2 = 1'b0;
                        if (x) next_state = S1;
                        else   next_state = S10;   end
           S01  : if (x) begin  next_state = S1;
                                z1 = 1'b0;   z2 = 1'b0;  end
                        else   begin  next_state = S10;
                                z1 = 1'b1;   z2 = 1'b0;  end
           S10  : if (x) begin  next_state = S01;
                                z1 = 1'b0;   z2 = 1'b0;  end
                        else   begin  next_state = S100;
                                z1 = 1'b0;   z2 = 1'b1;  end
          S100  : begin z1 = 1'b0;  z2 = 1'b0;
                        if (x) next_state = S100_1;
                        else   next_state = S100;   end
        S100_1  : begin z1 = 1'b0;  z2 = 1'b0;
                        if (x) next_state = S100_1;
                        else   next_state = S100_10;   end
        S100_10 : if (x) begin  next_state = S100_1;
                                z1 = 1'b0;   z2 = 1'b0;  end
                        else   begin  next_state = S100;
                                z1 = 1'b0;   z2 = 1'b1;  end
         default: begin  z1 = 1'b0; z2 = 1'b0;
                         next_state = SRst;  end
       endcase
   endmodule
```

```
module SequenceDetector_tb ();

wire  z1_o, z2_o;
reg   clk, reset, x;

wire z1, z2;
wire [2:0]  state, next_state;

SequenceDetector UUT (.clk   (clk),
                      .reset (reset),
                      .x     (x),
                      .z1_o  (z1_o),
                      .z2_o  (z2_o) );

assign state      = UUT. state;
assign next_state = UUT.next_state;
assign z1         = UUT.z1;
assign z2         = UUT.z2;

initial
begin clk = 1'b0;
    forever
    #5   clk = ~clk;
    end

initial
begin reset = 1'b1;
   #20  reset = 1'b0;
   #200 reset = 1'b1;
   #20  reset = 1'b0;
   end


initial
begin x = 1'b0;
  repeat (4)          // repeats the code how many times it is assigned
     begin
     #10 x = 1'b1;
     #10 x = 1'b0;  end
     #10
     x = 1'b0;      #50   // testing a long string of 0s
     x = 1'b1;      #50   // testing a long string of 1s
     x = 1'b0;      #20   // two 0s following a 1 - 100 sequence
  repeat (4)               // repeats the sequence 010 but after 100 was received   z2 = 1
     begin                 // z1 is not expected to rize to 1 again
     #10 x = 1'b1;
     #10 x = 1'b0;  end
  repeat (4)               // repeated sequence 100
     begin
     #10 x = 1'b1;
     #10 x = 1'b0;
     #10 x = 1'b0;  end
 end

endmodule
```

Correct value of z1 following a 010 sequence at the input, but z1 will not remain high for a full clock cycle, as x changed within the clock cycle. The registered output was able to hold the correct z1 value for the full clock cycle.

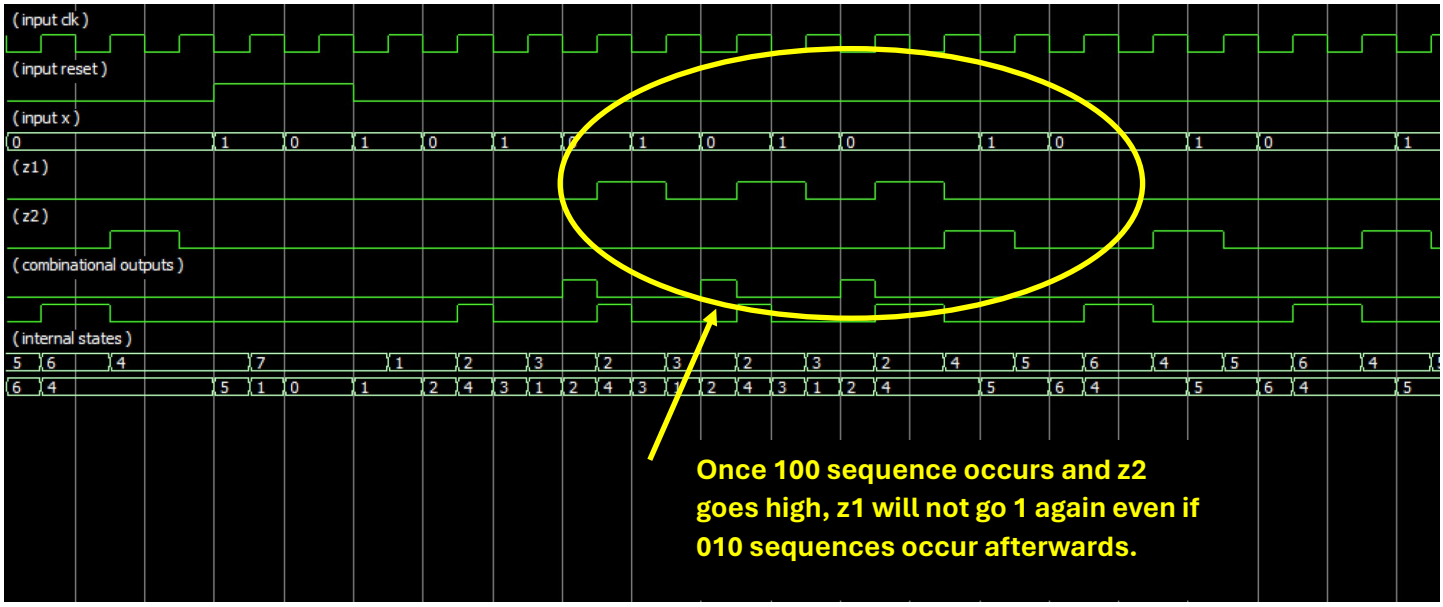


z1_o goes high when 010 sequence is detected

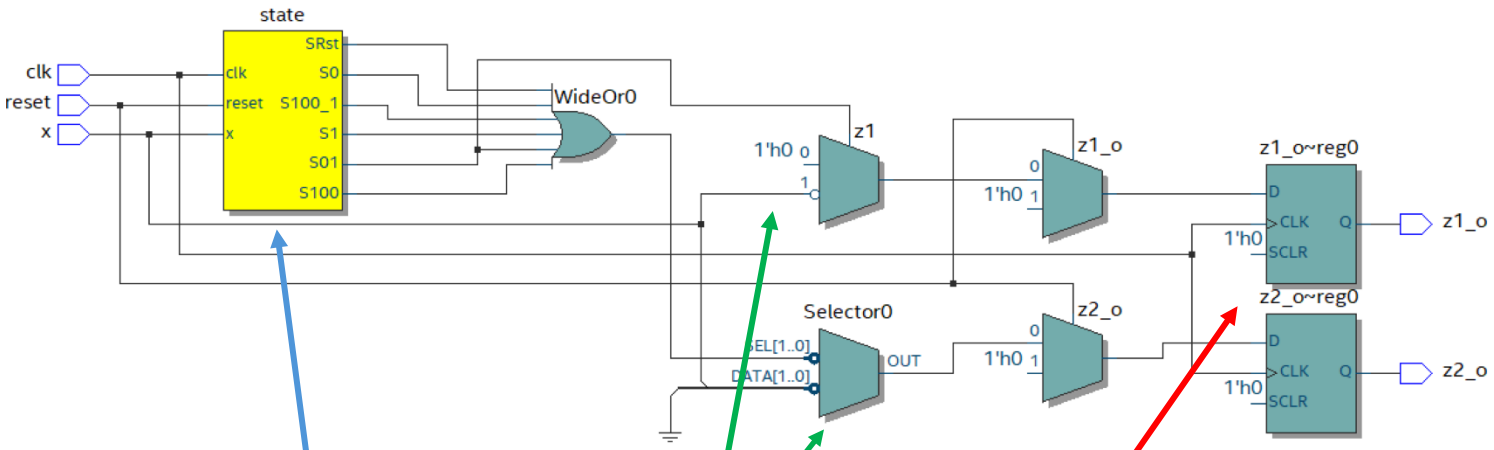
The next_state combinational logic updates with every change in state and x. The state is only updated at the positive clock edge.

Next_state is glitchy because of misalignment of x transitions with the clock positive edge. The registered state follows the state graph based on the sampled x at the posedge clk.

A glitch showing wrong value for z1 mistakenly assuming 100 was received because of the misalignment of x with the change of the state at the positive clock edge. Z2 should remain 0, and this is what we get with the registered output z2_o.



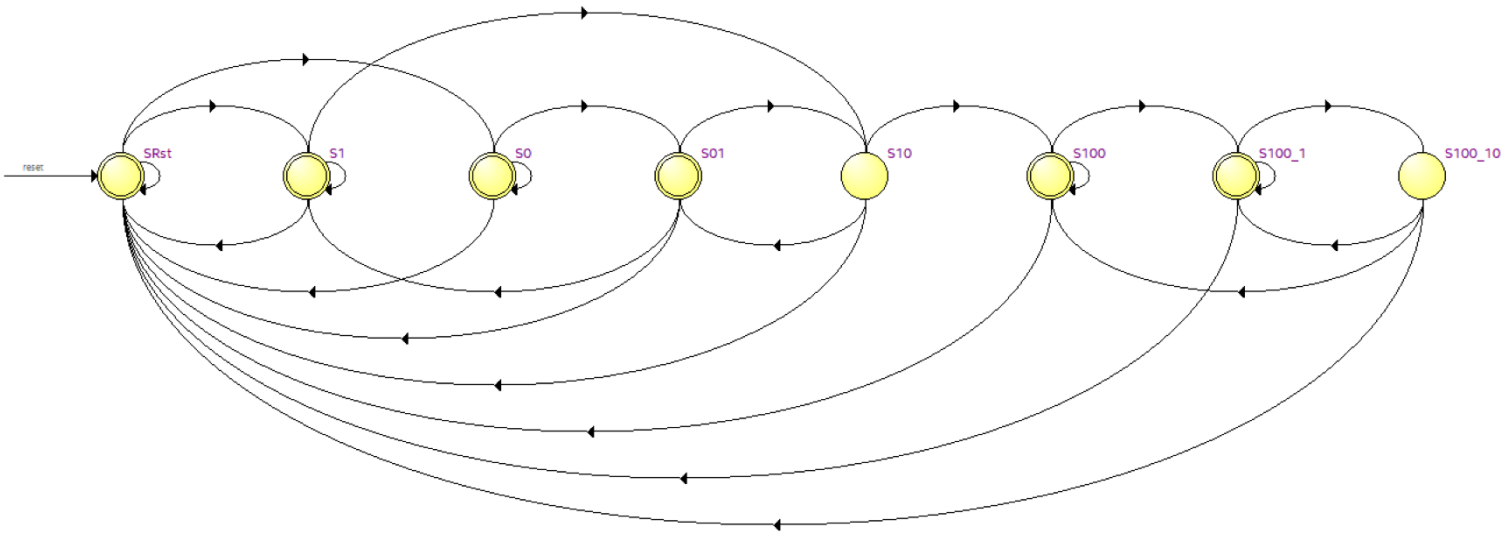
Once 100 sequence occurs and z2 goes high, z1 will not go 1 again even if 010 sequences occur afterwards.



The FF for the state update

Combinational logic (MUX) finding the z1 and z2 internal values

The FF for the outputs z1 and z2



	Source State	Destination State	Condition
1	S0	SRst	(reset)
2	S0	S01	(x).(!reset)
3	S0	S0	(!x).(!reset)
4	S1	SRst	(reset)
5	S1	S10	(!x).(!reset)
6	S1	S1	(x).(!reset)
7	S01	SRst	(reset)
8	S01	S10	(!x).(!reset)
9	S01	S1	(x).(!reset)
10	S10	SRst	(reset)
11	S10	S01	(x).(!reset)
12	S10	S100	(!x).(!reset)
13	S100	SRst	(reset)
14	S100	S100_1	(x).(!reset)
15	S100	S100	(!x).(!reset)
16	S100_1	S100_10	(!x).(!reset)
17	S100_1	SRst	(reset)
18	S100_1	S100_1	(x).(!reset)
19	S100_10	SRst	(reset)
20	S100_10	S100_1	(x).(!reset)
21	S100_10	S100	(!x).(!reset)
22	SRst	SRst	(reset)
23	SRst	S1	(x).(!reset)
24	SRst	S0	(!x).(!reset)