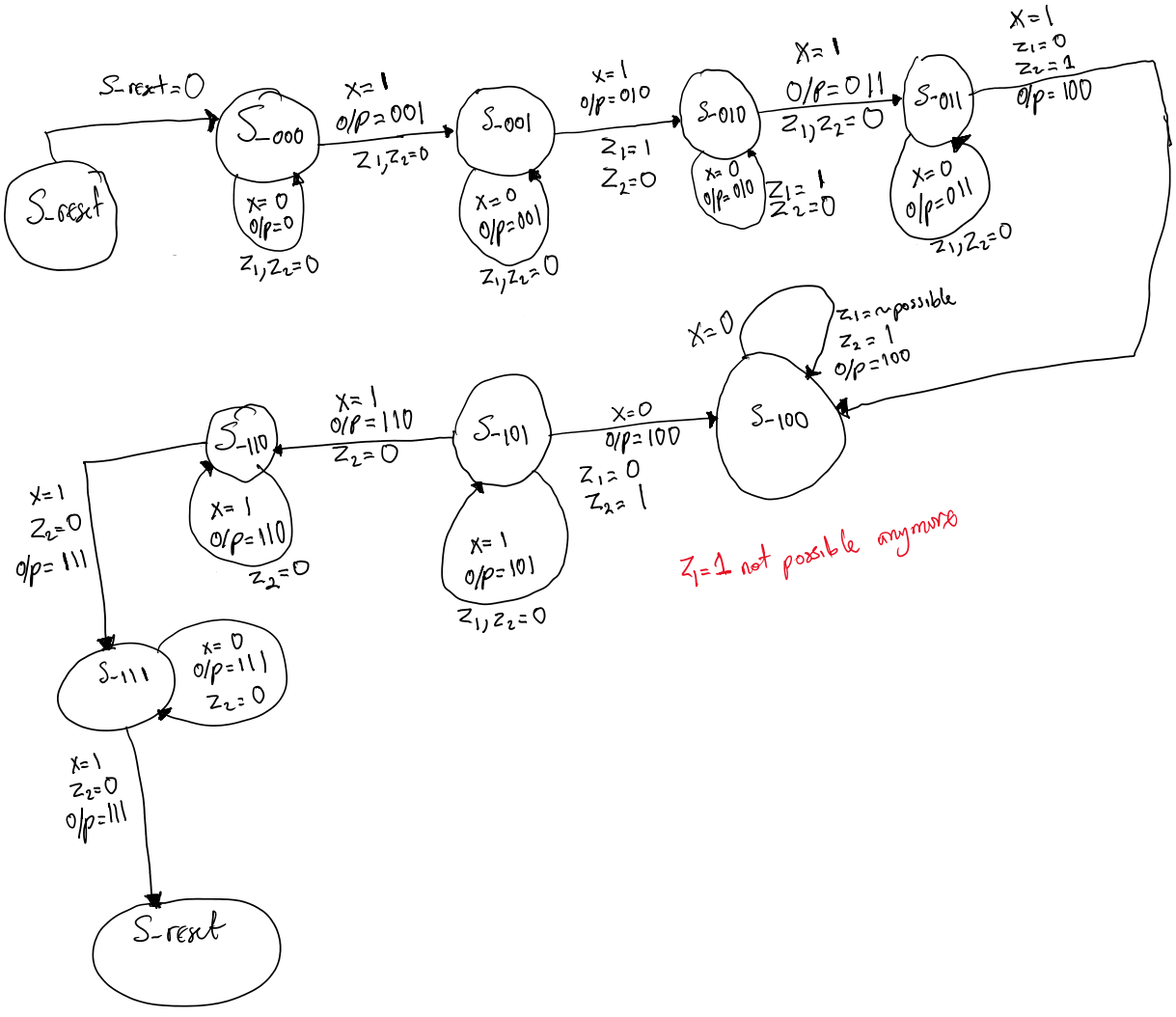
Q1

Saturday, June 8, 2024 8:14 AM

State Table				
	next_state		output	
present_state	x_in=0	x_in=1	z1	z2
S_000	S_000	S_001	0	0
S_001	S_001	S_010	0	0
S_010	S_010	S_011	1	0
S_011	S_011	S_100	0	0
S_100	S_100	S_101	0	1
S_101	S_101	S_110	0	0
S_110	S_110	S_111	0	0
S_111	S_111	S_111	0	0

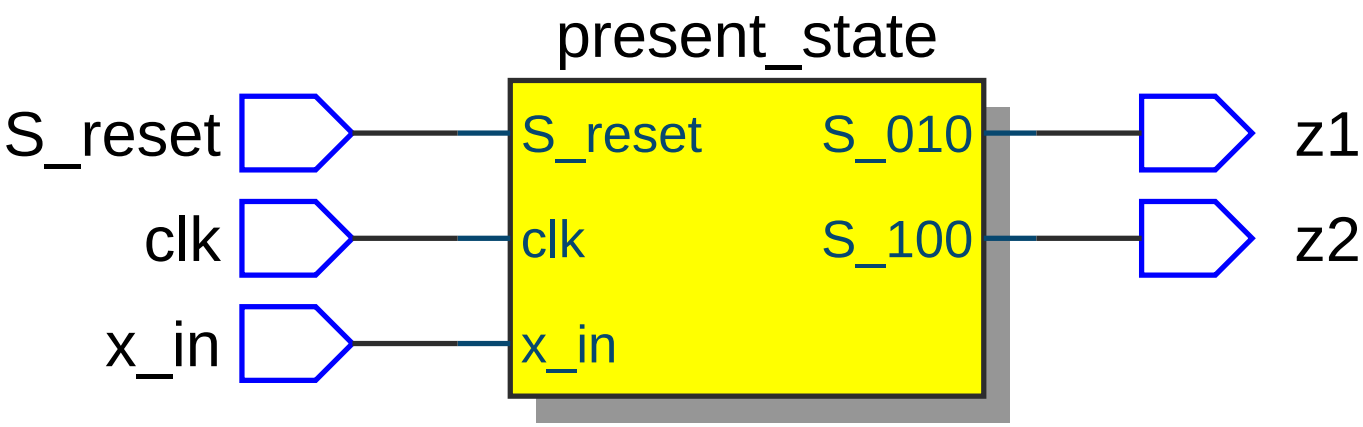
State Table				
	next_state		output	
present_state	x_in=0	x_in=1	z1	z2
S_000	S_000	S_001	0	0
S_001	S_001	S_010	1	0
S_010	S_010	S_011	1	0
S_011	S_011	S_100	0	1
S_100	S_100	S_101	0	1
S_101	S_101	S_110	0	0
S_110	S_110	S_111	0	0
S_111	S_111	S_reset	0	0

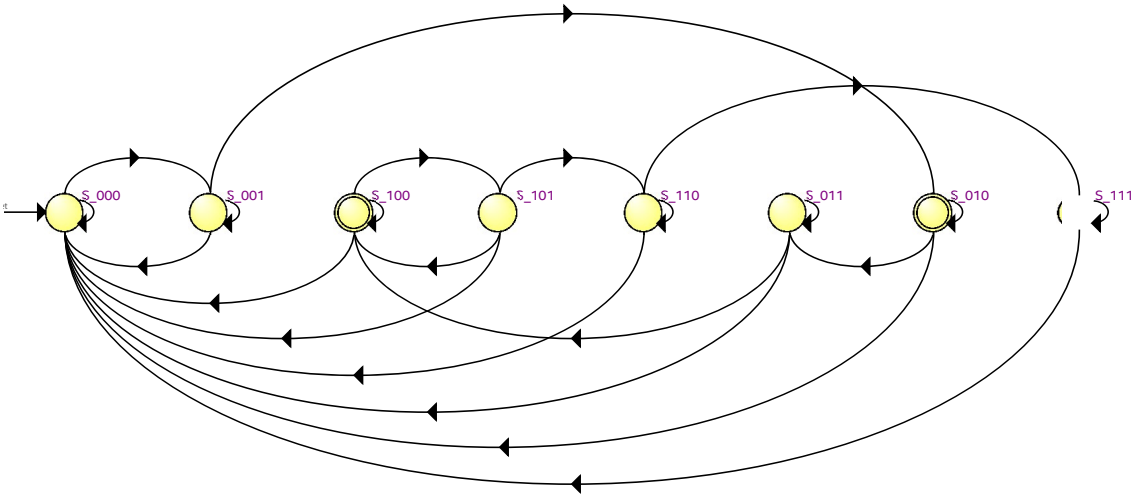
S_reset: Needs to low for the FSM to start
if input sequence = 010 then z1=1 and z2=0
if input sequence = 100 then z1=0 and z2=1
S_000: 3Bit input sequence=000
S_001: 3Bit input sequence=001 received
S_010: 3Bit input sequence=010 received
S_011: 3Bit input sequence=011 received
S_100: 3Bit input sequence=100 received
S_101: 3Bit input sequence=101 received
S_110: 3Bit input sequence=110 received
S_111: 3Bit input sequence=111 received


S_rest=0
S_rest
S_000
x=0
o/p=0
z1,z2=0
x=1
o/p=001
z1,z2=0
S_001
x=0
o/p=001
z1,z2=0
x=1
o/p=010
z1=1
z2=0
S_010
x=0
o/p=010
z1=1
z2=0
x=1
o/p=011
z1,z2=0
S_011
x=0
o/p=011
z1,z2=0
x=1
z1=0
z2=1
o/p=100
S_100
x=0
z1=~possible
z2=1
o/p=100
x=0
o/p=100
z1=0
z2=1
S_101
x=1
o/p=101
z1,z2=0
x=1
o/p=110
z2=0
S_110
x=1
o/p=110
z2=0
x=1
z2=0
o/p=111
S_111
x=0
o/p=111
z2=0
x=1
z2=0
o/p=111
S_rest
z1=1 not possible anymore

```
/*-------------------------------------------------------------------------------
Name Lamin Jammeh
CLass: EE417 Summer 2024
Lesson 04 HW Question 1
Group: Ron Kalin/ Lamin Jammeh
Project Description: This is the main module for a Mealy FSM that receives a
set of bits and checks if they are equa to Z1=010 or Z2=100 and takes adecision
on whether to reset the machine of keeps look receiving data and comparing them
-------------------------------------------------------------------------------*/
 module Sequence_100_Detector_Mealy (z2, z1, clk, S_reset, x_in);

 //define the inputs and outputs of the system
 output reg z2, z1;
 input      clk, S_reset;
 input      x_in;

 //define the states
 reg [2:0] present_state, next_state;

 //define the possible parameter
 parameter  S_000 = 3'b000;
 parameter  S_001 = 3'b001;
 parameter  S_010 = 3'b010;      //z1
 parameter  S_011 = 3'b011;
 parameter  S_100 = 3'b100;      //z2
 parameter  S_101 = 3'b101;
 parameter  S_110 = 3'b110;
 parameter  S_111 = 3'b111;

 //sequenctial logic updating the state register (flip flop)
 always @(posedge clk)
      if (S_reset) present_state <= S_000;
      else       present_state <= next_state;

//combination logic determining the next_state and the output state
always @ *                    //trigger the block when there is any change in the signals
used in the block
      case (present_state)
      S_000    : begin
                   z1 = 1'b0;
                   z2 = 1'b0;
                   if (x_in) next_state = S_001;
                   else      next_state = S_000;
                 end
      S_001    : begin
                   z1 = 1'b0;
                   z2 = 1'b0;
                   if (x_in) next_state = S_010;
                   else      next_state = S_001;
                 end
      S_010    : begin
                   z1 = 1'b1;
                   z2 = 1'b0;
                   if (x_in) next_state = S_011;
                   else      next_state = S_010;
                 end
      S_011    : begin
                   z1 = 1'b0;
                   z2 = 1'b0;
                   if (x_in) next_state = S_100;
                   else      next_state = S_011;
                 end
      S_100    : begin
                   z1 = 1'b0;
                   z2 = 1'b1;
                   if (x_in) next_state = S_101;
                   else      next_state = S_100;
                 end
      S_101    : begin
```

```
69                      z1 = 1'b0;
70                      z2 = 1'b0;
71                      if (x_in) next_state = S_110;
72                      else      next_state = S_100;
73                   end
74           S_110   : begin
75                      z1 = 1'b0;
76                      z2 = 1'b0;
77                      if (x_in) next_state = S_111;
78                      else      next_state = S_110;
79                   end
80           S_111   : begin
81                      z1 = 1'b0;
82                      z2 = 1'b0;
83                      if (x_in) next_state = S_111;
84                      else      next_state = S_111;
85                   end
86           endcase
87   endmodule
```

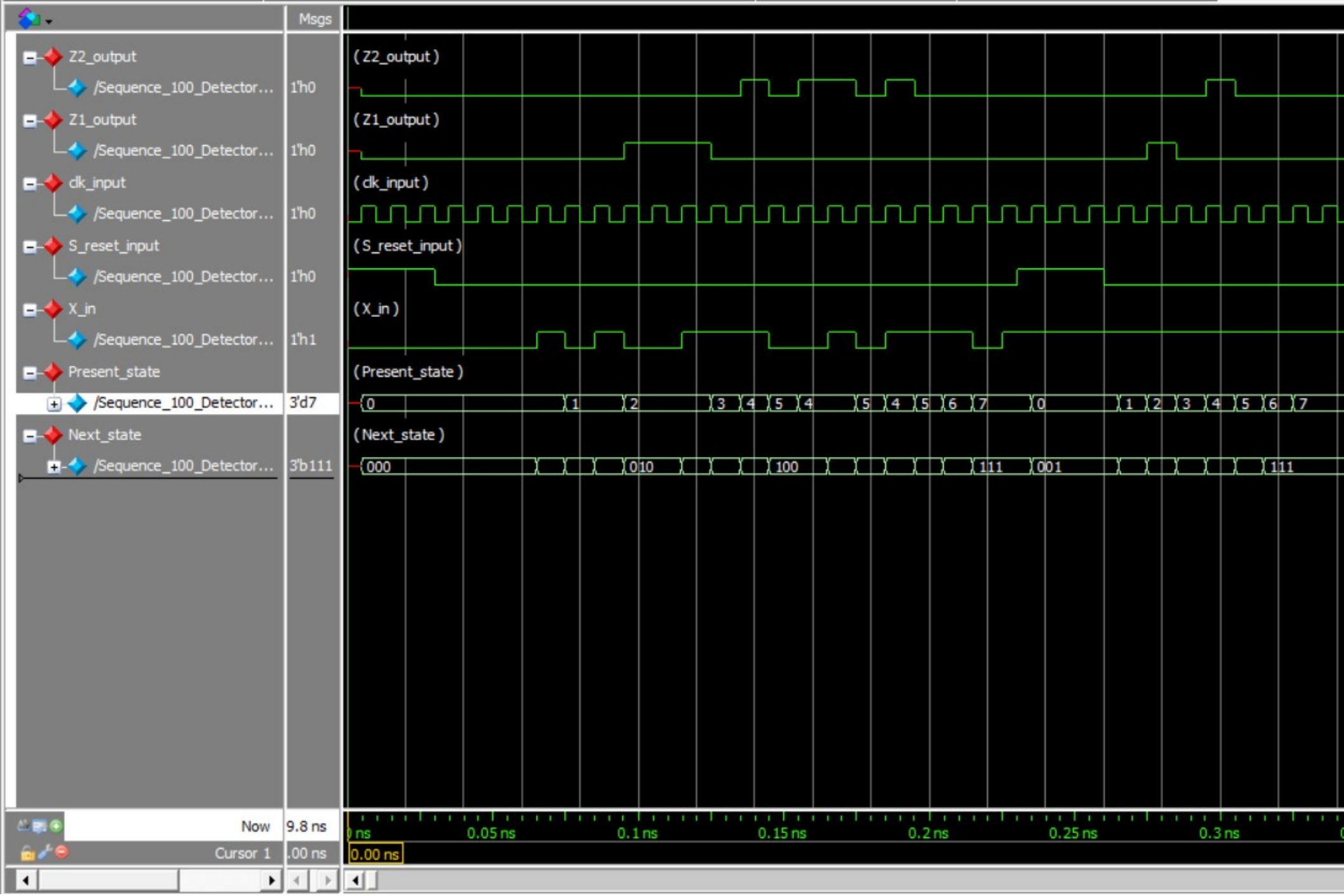

present_state
S_reset
clk
x_in
S_reset
clk
x_in
S_010
S_100
z1
z2


S_000
S_001
S_100
S_101
S_110
S_011
S_010
S_111

```
/*-------------------------------------------------------------
Test Bench for Sequence_100_Detector_Mealy Finite State Machine
CLass: EE417 Summer 2024
Lesson 04 HW Question 1
Group: Ron Kalin/ Lamin Jammeh
-------------------------------------------------------------*/

module Sequence_100_Detector_Mealy_TB ();

//define the input and outputs as wires and registers
wire  z2, z1;
reg   clk, S_reset;
reg   x_in;

//define the internal probes as wires
wire  [2:0] present_state, next_state;

//define the unit under test (UUT)
Sequence_100_Detector_Mealy UUT (z2, z1, clk, S_reset, x_in);

//internal probes to make it easy to track the logic and for troubleshooting
assign present_state = UUT.present_state;
assign next_state = UUT.next_state;

//generate the clk cycle with a period of 5 ns
initial
   begin
      clk = 1'b0;
      forever
      begin
         #5 clk = ~clk;
      end
   end

//initialize the reset cycle
initial
   begin
      S_reset = 1'b1;
      #30  S_reset = 1'b0;
      #200 S_reset = 1'b1;
      #30  S_reset = 1'b0;
   end

//test the possible input sequence combination to form the different states
initial
   begin
      x_in = 1'b0; #15

      begin

         x_in = 1'b0;   #10   x_in = 1'b0;   #10 x_in = 1'b0;  #10;  //S_000  sequence
         x_in = 1'b0;   #10   x_in = 1'b0;   #10 x_in = 1'b1;  #10;  //S_001  sequence
         x_in = 1'b0;   #10   x_in = 1'b1;   #10 x_in = 1'b0;  #10;  //S_010  sequence = z1
         x_in = 1'b0;   #10   x_in = 1'b1;   #10 x_in = 1'b1;  #10;  //S_011  sequence
         x_in = 1'b1;   #10   x_in = 1'b0;   #10 x_in = 1'b0;  #10;  //S_100  sequence = z2
         x_in = 1'b1;   #10   x_in = 1'b0;   #10 x_in = 1'b1;  #10;  //S_101  sequence
         x_in = 1'b1;   #10   x_in = 1'b1;   #10 x_in = 1'b0;  #10;  //S_110  sequence
         x_in = 1'b1;   #10   x_in = 1'b1;   #10 x_in = 1'b1;  #10;  //S_111  sequence
      end
   end

   // Monitor outputs
    initial begin
        $display("x_in____Present_State_____Next_State____Z1____Z1____");
        $monitor("%b,      %b,      %b,      %b,    %b", x_in, present_state, next_state,
z1, z2);
    end
endmodule
```

Transcript

```
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# x_in____Present_State_____Next_State____Z1____Z1____
# 0,            xxx,            xxx,       x,    x
# 0,            000,            000,       0,    0
# 1,            000,            001,       0,    0
# 0,            001,            001,       0,    0
# 1,            001,            010,       0,    0
# 0,            010,            010,       1,    0
# 1,            010,            011,       1,    0
# 1,            011,            100,       0,    0
# 1,            100,            101,       0,    1
# 0,            101,            100,       0,    0
# 0,            100,            100,       0,    1
# 1,            100,            101,       0,    1
# 0,            101,            100,       0,    0
# 1,            100,            101,       0,    1
# 1,            101,            110,       0,    0
# 1,            110,            111,       0,    0
# 0,            111,            111,       0,    0
# 1,            111,            111,       0,    0
# 1,            000,            001,       0,    0
# 1,            001,            010,       0,    0
# 1,            010,            011,       1,    0
# 1,            011,            100,       0,    0
# 1,            100,            101,       0,    1
# 1,            101,            110,       0,    0
# 1,            110,            111,       0,    0
# 1,            111,            111,       0,    0
```


Msgs
Z2_output
/Sequence_100_Detector...
1'h0
Z1_output
/Sequence_100_Detector...
1'h0
clk_input
/Sequence_100_Detector...
1'h0
S_reset_input
/Sequence_100_Detector...
1'h0
X_in
/Sequence_100_Detector...
1'h1
Present_state
/Sequence_100_Detector...
3'd7
Next_state
/Sequence_100_Detector...
3'b111
(Z2_output)
(Z1_output)
(clk_input)
(S_reset_input)
(X_in)
(Present_state)
0 1 2 3 4 5 4 5 4 5 6 7 0 1 2 3 4 5 6 7
(Next_state)
000 010 100 111 001 111
Now
9.8 ns
Cursor 1
.00 ns
0 ns
0.05 ns
0.1 ns
0.15 ns
0.2 ns
0.25 ns
0.3 ns
0.00 ns

The simulation summary shows the following

- Output z1 is high after the S_reset is low, and when the present state=2 or b010
- Output z2 is high after the S_reset is low, and when the present state=4 or b100
    - After z2 is high z1 cannot occur until S_reset gets toggle high and then low