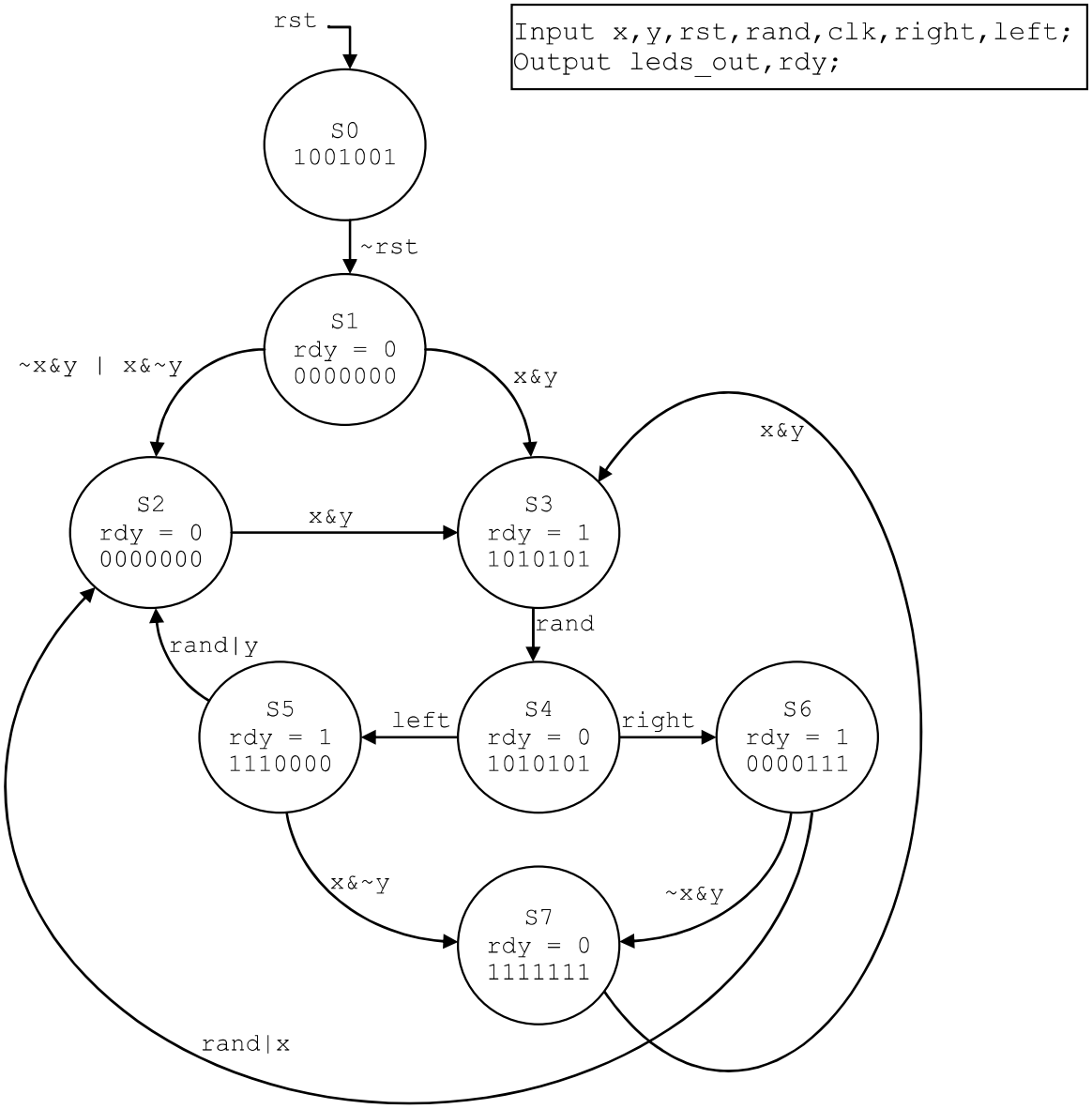


1) What is the purpose of this FSM? Describe (in English) what this FSM does.

This FSM is a game for a single player to test reaction to a signal going left or right. After leaving the reset state (S0), which displays 1001001 on the leds, the prep state (S1) displays 0000000. When both buttons (x&y) are pushed the game enters the ready state (S3) and displays 1010101. Then, after a random amount of time given by the rand signal, it enters the direction decision state (S4). If the left path is indicated (by the left signal), the game enters S5, waits for the left button to be pressed (x), and displays 1110000. If the right path is indicated (by the right signal), the game enters S6, waits for the right button (y) to be pressed, and displays 0000111. If the correct button is pressed before a random amount of time, the game enters the win state (S7) and displays 1111111. If the incorrect button is pressed or a random amount of time passes with no buttons pressed, the game enters the loss state (S2) and displays 0000000. The rdy

signal is generated by various states to indicate the FSM is ready for a rand signal to be generated externally.

2) This finite state machine has 8 states, but, as shown above, has one redundant state. Which state is the redundant one? Would you save any hardware (flip flops or gates) if you used 7 states instead? Why?

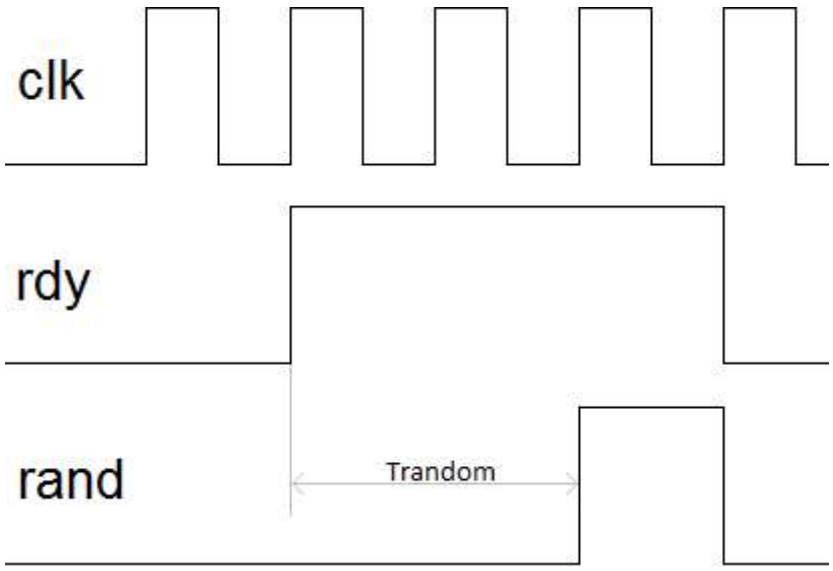
   The state, S2, is redundant as it repeats the behaviour of S1. If you removed S2, the game would operate the same.

   If S2 was removed we would still require 3 flip flops as a 7 state FSM needs 3 bits, the same as an 8 state FSM. We would, however, save some gates, as the logic needed to calculate ~x&y | x&~y would not be needed.

3) What do you think the rdy signal is for? (Hint: It is an output used to control the module that provides the rand signal to the input of this module.)

   The rdy signal is used to tell the module that generates the rand signal to do so. This means that when a state is entered in which rdy=1, a random amount of time later, the next state is entered, as controlled by the rand signal. It could also be used to generate the left/right signal.

4) The rand signal is generated by another module. Define the behaviour of the rand signal (use waveform diagrams; include any signals that control its behaviour, ie the rdy signal).



Note, the information given does not specify the length of rand, but one clock cycle is a reasonable assumption. Also, Trandom is not specified, but is likely many clock cyles.

5) Do we really need a right signal? What would you use instead?

No. A right signal is not needed. ~left can be used instead.

6) Write the verilog module and test bench to implement and test the behaviour of the above diagram (as is, ie: including S2 and right).

```
module FSM_example(clk, rst, x, y, rand, leds_out, rdy, left, right);

//state defines
`define   S0         0
`define   S1         1
`define   S2         2
`define   S3         3
`define   S4         4
`define   S5         5
`define   S6         6
`define   S7         7

input     x,              //Left button
          y,              //Right button
          rand,           //Rand signal externally generated
          clk,            //Clock
          rst,            //Asychronous reset
          left,           //left decision path
          right;          //right decision path
output    leds_out,       //7 output leds
          rdy;            //ready signal for random generator

reg [2:0] state;
reg [2:0] nxtState;
reg rdy;
reg [6:0] leds_out;

//assign next state at every clock; asynchronous reset
always @(posedge clk or posedge rst)
begin
     if (rst) state <= `S0;
     else state <= nxtState;
end

//determines next state
always @(state or x or y or left or right or rand or rst)
begin
```

```
case(state)
    `S0:  if         (!rst)          nxtState = `S1; //Reset
          else                       nxtState = `S0;
    `S1:  if         ((~x&y)|(x&~y)) nxtState = `S2; //Redundant state
          else if    (x&y)           nxtState = `S3;
          else                       nxtState = `S1;
    `S2:  if         (x&y)           nxtState = `S3; //Prep state. Start if
          else                       nxtState = `S2; //both buttons pressed.
    `S3:  if         (rand)          nxtState = `S4; //Wait until rand
          else                       nxtState = `S3;
    `S4:  if         (left)          nxtState = `S5; //Go left
          else if    (right)         nxtState = `S6; //or right
          else                       nxtState = `S4;
    `S5:  if         (rand|y)        nxtState = `S2; //Time up/wrong button
          else if    (x&~y)          nxtState = `S7; //Goto win
          else                       nxtState = `S5;
    `S6:  if         (rand|x)        nxtState = `S2; //Time up/wrong button
          else if    (~x&y)          nxtState = `S7; //Goto win
          else                       nxtState = `S6;
    `S7:  if         (x&y)           nxtState = `S3; //Restart
          else                       nxtState = `S7;
    default:                        nxtState = `S0;
endcase
end

//set outputs depending on state
always @(state)
    case(state)
    `S0:       begin leds_out = 7'b1001001;     rdy = 0; end //Reset
    `S1:       begin leds_out = 7'b0000000;     rdy = 0; end //Prep state
    `S2:       begin leds_out = 7'b0000000;     rdy = 0; end //Lose state
    `S3:       begin leds_out = 7'b1010101;     rdy = 1; end //Ready state
    `S4:       begin leds_out = 7'b1010101;     rdy = 0; end //Decision
    `S5:       begin leds_out = 7'b1110000;     rdy = 1; end //Left state
    `S6:       begin leds_out = 7'b0000111;     rdy = 1; end //Right state
    `S7:       begin leds_out = 7'b1111111;     rdy = 0; end //Win state
    default:   begin leds_out = 7'b1100110;     rdy = 0; end
    endcase

endmodule
```

```
`timescale 1us / 1ns

module FSM_example_tb;

    // Inputs
    reg clk;
    reg rst;
    reg x;
    reg y;
    reg rand;
    reg left;
    reg right;

    // Outputs
    wire [6:0] leds_out;
    wire rdy;

    // Instantiate the Unit Under Test (UUT)
    FSM_example uut (
         .clk(clk),
         .rst(rst),
         .x(x),
         .y(y),
         .rand(rand),
         .leds_out(leds_out),
         .rdy(rdy),
         .left(left),
         .right(right)
    );

    //setup clock
    //T/2 = 1000 us -> f = 500 Hz
    always #1000 clk <= ~clk;

    //setup dynamic data monitors for outputs
    // Label DOUT
    always @(rdy or leds_out)
         $display ("%t - DOUT: rdy = %b leds_out = %b",$time, rdy,leds_out);

    //setup dynamic data monitors for inputs
    //Label D_IN
    always @(x or y or rand or left or right)
         $display ("%t - D_IN: x = %b y = %b rand = %b left = %b right =
         %b",$time, x,y,rand,left,right);
    always @(rst) $display ("%t - D_IN: rst = %b",$time, rst);

    initial begin
```

```
// Initialize Inputs
clk = 0;
rst = 0;
x = 0;
y = 0;
rand = 0;
left = 0;
right = 0;

//format the way time is displayed
//$timeformat(base,decimal places,"display unit",min width);
$timeformat(-3,3,"ms",10);

//Small delay to allow initialization to complete
#5

//Title for data log
$display ("");
$display ("FSM_example testbench data log begin.");
$display ("================================");
$display ("");

//Asynch reset circuit
#20;
$display ("%t - Performing asynchronous reset",$time);
rst = 1; #5;
//Testing reset
// Display expected output values Label CHEK
$display ("%t - CHEK: rdy = 0 leds_out = 1001001",$time);
if (leds_out == 7'b1001001 && rdy == 0)
     $display ("%t - Reset PASS",$time);
else $display ("%t - Reset Fail",$time);
$display ("");
repeat(2) @(posedge clk); #33;


//test S1 leaving reset
$display ("%t - Leaving reset to S1",$time);
rst = 0;
@(posedge clk); #1;
$display ("%t - CHEK: rdy = 0 leds_out = 0000000",$time);
if (leds_out == 7'b0000000 && rdy == 0)
     $display ("%t - S0->S1 PASS",$time);
else $display ("%t - S0->S1 Fail",$time);
$display ("");

//In S1
```

```
//Test pressing only x
@(posedge clk); #1;
$display ("%t - Pressing x. Going to S2.",$time);
$display ("%t - No DOUT due to outputs not changing.",$time);
x = 1;
@(posedge clk); #5;
$display ("%t - CHEK: rdy = 0 leds_out = 0000000",$time);
if (leds_out == 7'b0000000 && rdy == 0)
     $display ("%t - S1->S2 PASS",$time);
else $display ("%t - S1->S2 Fail",$time);
$display ("");
@(posedge clk); #1;
x = 0;
#1; $display ("");

//In S2
repeat(2) @(posedge clk); #21;
//pressing both buttons asynchronously
$display ("%t - Pressing x&y. Going to S3.",$time);
x = 1; y = 1;
@(posedge clk); #5;
$display ("%t - CHEK: rdy = 1 leds_out = 1010101",$time);
if (leds_out == 7'b1010101 && rdy == 1)
     $display ("%t - S2->S3 PASS",$time);
else $display ("%t - S2->S3 Fail",$time);
$display ("");
@(posedge clk); #1;
x = 0; y = 0;
#1; $display ("");

//In S3
//waiting for rand (3 clock cycles arbitrarily choosen)
repeat(3) @(posedge clk); #1;
$display ("%t - rand occuing in 3 clock cyles. Going to S4.",$time);
rand = 1;
@(posedge clk); #5;
$display ("%t - CHEK: rdy = 0 leds_out = 1010101",$time);
if (leds_out == 7'b1010101 && rdy == 0)
     $display ("%t - S3->S4 PASS",$time);
else $display ("%t - S3->S4 Fail",$time);
$display ("");
@(posedge clk); #1;
rand =0;
#1; $display ("");

//In S4
//waiting for left
```

```
repeat(2) @(posedge clk); #1;
$display ("%t - left occuring in 2 clock cycles. Going to
S5.",$time);
left = 1;
@(posedge clk); #5;
$display ("%t - CHEK: rdy = 1 leds_out = 1110000",$time);
if (leds_out == 7'b1110000 && rdy == 1)
     $display ("%t - S4->S5 PASS",$time);
else $display ("%t - S4->S5 Fail",$time);
$display ("");
@(posedge clk); #1;
left = 0;
#1; $display ("");

//In S5
//x pressing button
repeat(1) @(posedge clk); #1;
$display ("%t - x pressing button. Going to S7.",$time);
x = 1;
@(posedge clk); #5;
$display ("%t - CHEK: rdy = 0 leds_out = 1111111",$time);
if (leds_out == 7'b1111111 && rdy == 0)
     $display ("%t - S5->S7 PASS",$time);
else $display ("%t - S5->S7 Fail",$time);
$display ("");
@(posedge clk); #1;
x = 0;
#1; $display ("");

//In S7
//Wait 4 clocks, press x then y asynchronously
repeat(4) @(posedge clk); #15;
$display ("%t - Pressing x then y. Going to S3.",$time);
x = 1; @(posedge clk); #34; y = 1;
@(posedge clk); #5;
$display ("%t - CHEK: rdy = 1 leds_out = 1010101",$time);
if (leds_out == 7'b1010101 && rdy == 1)
     $display ("%t - S7->S3 PASS",$time);
else $display ("%t - S7->S3 Fail",$time);
$display ("");
@(posedge clk); #1;
x = 0; y = 0;
#1; $display ("");

//Testing not complete. More test cases needed here.
//IE. Test all states and transitions.
#1;
```

```
        $display ("More test cases testing the remaining states and
        transistions go here.");
        #1; $display ("");

        repeat(2) @(posedge clk);
        $finish;
    end

endmodule
```

```
--------------------------------------------------------------------
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
0.000ms - D_IN: rst = 0
0.000ms - D_IN: x = 0 y = 0 rand = 0 left = 0 right = 0
Finished circuit initialization process.

FSM_example testbench data log begin.
==============================
0.025ms - Performing asynchronous reset
0.025ms - D_IN: rst = 1
0.025ms - DOUT: rdy = 0 leds_out = 1001001
0.030ms - CHEK: rdy = 0 leds_out = 1001001
0.030ms - Reset PASS

3.033ms - Leaving reset to S1
3.033ms - D_IN: rst = 0
5.000ms - DOUT: rdy = 0 leds_out = 0000000
5.001ms - CHEK: rdy = 0 leds_out = 0000000
5.001ms - S0->S1 PASS

7.001ms - Pressing x. Going to S2.
7.001ms - No DOUT due to outputs not changing.
7.001ms - D_IN: x = 1 y = 0 rand = 0 left = 0 right = 0
9.005ms - CHEK: rdy = 0 leds_out = 0000000
9.005ms - S1->S2 PASS

11.001ms - D_IN: x = 0 y = 0 rand = 0 left = 0 right = 0

15.021ms - Pressing x&y. Going to S3.
15.021ms - D_IN: x = 1 y = 1 rand = 0 left = 0 right = 0
17.000ms - DOUT: rdy = 1 leds_out = 1010101
17.005ms - CHEK: rdy = 1 leds_out = 1010101
17.005ms - S2->S3 PASS
```

19.001ms - D_IN: x = 0 y = 0 rand = 0 left = 0 right = 0

25.001ms - rand occuing in 3 clock cyles. Going to S4.
25.001ms - D_IN: x = 0 y = 0 rand = 1 left = 0 right = 0
27.000ms - DOUT: rdy = 0 leds_out = 1010101
27.005ms - CHEK: rdy = 0 leds_out = 1010101
27.005ms - S3->S4 PASS

29.001ms - D_IN: x = 0 y = 0 rand = 0 left = 0 right = 0

33.001ms - left occuring in 2 clock cycles. Going to S5.
33.001ms - D_IN: x = 0 y = 0 rand = 0 left = 1 right = 0
35.000ms - DOUT: rdy = 1 leds_out = 1110000
35.005ms - CHEK: rdy = 1 leds_out = 1110000
35.005ms - S4->S5 PASS

37.001ms - D_IN: x = 0 y = 0 rand = 0 left = 0 right = 0

39.001ms - x pressing button. Going to S7.
39.001ms - D_IN: x = 1 y = 0 rand = 0 left = 0 right = 0
41.000ms - DOUT: rdy = 0 leds_out = 1111111
41.005ms - CHEK: rdy = 0 leds_out = 1111111
41.005ms - S5->S7 PASS

43.001ms - D_IN: x = 0 y = 0 rand = 0 left = 0 right = 0

51.015ms - Pressing x then y. Going to S3.
51.015ms - D_IN: x = 1 y = 0 rand = 0 left = 0 right = 0
53.034ms - D_IN: x = 1 y = 1 rand = 0 left = 0 right = 0
55.000ms - DOUT: rdy = 1 leds_out = 1010101
55.005ms - CHEK: rdy = 1 leds_out = 1010101
55.005ms - S7->S3 PASS

55.001ms - D_IN: x = 0 y = 0 rand = 0 left = 0 right = 0

More test cases testing the remaining states and transistions go here.