UNIVERSITY OF NEVADA LAS VEGAS. DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING LABORATORIES.

Class:	CPE-300L		Semester:	Spring 2020
Points		Document author:	Brysen Kokubun	
		Author's email:	Brysenkokubun96@gmail.com	
		Document topic:	Postlab 5	
Instructor's comments:				

1. Introduction / Theory of Operation

- For this lab experiment, we implemented various FSM machines. The experiments made us create a Digital lock/unlock FSM, Model an FSM from a table, and a serial parity detector.

a. What are two types of FSMs? Explain in detail.

- Two types of FSMs are Mealy and Moore. Mealy Machine's outputs are dependent on present inputs, and current machine state. Moore machine outputs are dependent only on machines current state.

b. Write few sentences: explain how you approach designing a FSM

- The approach to take when designing a FSM is to, 1 - Choose machine type, 2 – Choose encoding style , 3 – Create Combinational logic, 4 – Create Sequential logic, 5 – Create Test bench (optional), 6 – Verify logic (Waveform, etc.)

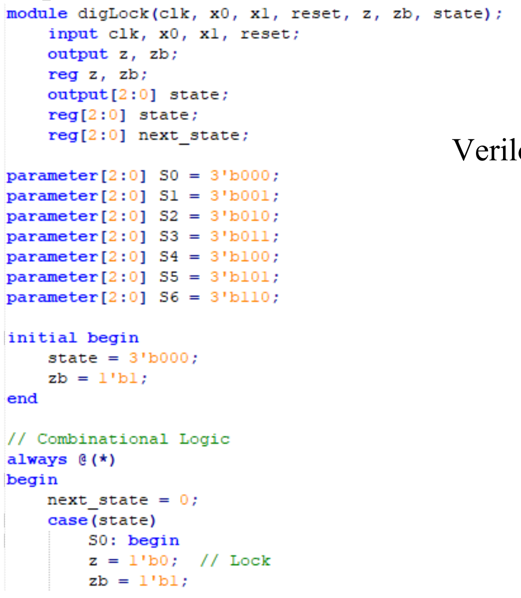
2. Prelab Report

- Attached below

3. Experiment Results

Experiment #1: Write FSM code for digital lock & unlock

'110110' unlocks the lock any other combination will not unlock the lock



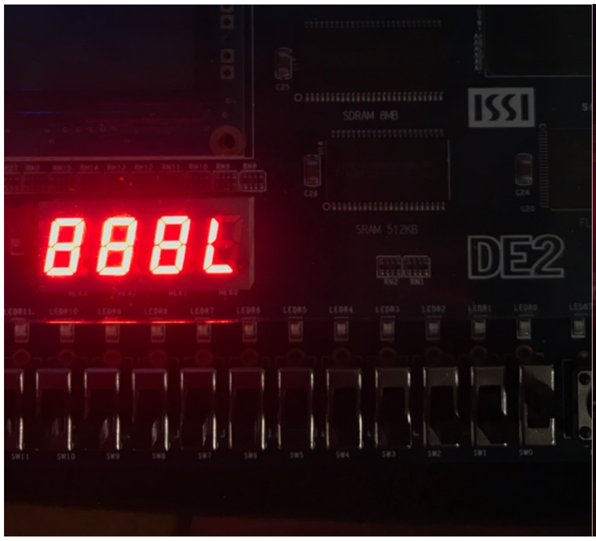
```
module digLock(clk, x0, x1, reset, z, zb, state);
    input clk, x0, x1, reset;
    output z, zb;
    reg z, zb;
    output[2:0] state;
    reg[2:0] state;
    reg[2:0] next_state;

parameter[2:0] S0 = 3'b000;
parameter[2:0] S1 = 3'b001;
parameter[2:0] S2 = 3'b010;
parameter[2:0] S3 = 3'b011;
parameter[2:0] S4 = 3'b100;
parameter[2:0] S5 = 3'b101;
parameter[2:0] S6 = 3'b110;

initial begin
    state = 3'b000;
    zb = 1'b1;
end

// Combinational Logic
always @(*)
begin
    next_state = 0;
    case(state)
        S0: begin
        z = 1'b0;  // Lock
        zb = 1'b1;
```

Verilog Code for Digital Lock

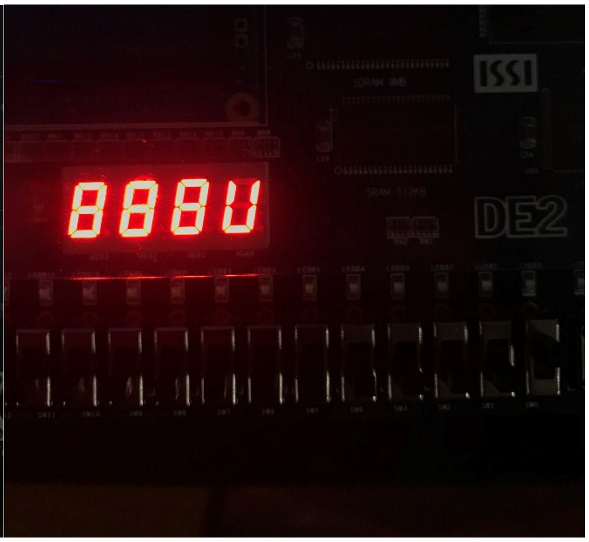
```
        if ((x0 == 1'b1) || (reset == 1'b1)) begin
            next_state = S0;
        end else if (x1 == 1'b1) begin
            next_state = S1;
        end
        else next_state = S0;
        end

        S1: if (x0 == 1'b1 || reset == 1'b1) begin
            next_state = S0;
        end else if (x1 == 1'b1) begin
            next_state = S2;
        end
        else next_state = S1;

        S2: if (x1 == 1'b1 || reset == 1'b1) begin
            next_state = S0;
        end else if (x0 == 1'b1) begin
            next_state = S3;
        end else next_state = S2;

        S3: if (x0 == 1'b1 || reset == 1'b1) begin
            next_state = S0;
        end else if (x1 == 1'b1) begin
            next_state = S4;
        end else next_state = S3;

        S4: if (x0 == 1'b1 || reset == 1'b1) begin
            next_state = S0;
        end else if (x1 == 1'b1) begin
            next_state = S5;
        end else next_state = S4;

        S5: if (x1 == 1'b1 || reset == 1'b1) begin
            next_state = S0;
        end else if (x0 == 1'b1) begin
            next_state = S6;
        end else next_state = S6;

        S6: begin
        z = 1'b1;  // Unlock
        zb = 1'b0;
        if (reset == 1'b1)
        next_state = S0;
        else next_state = S6;
        end

    endcase
end

// Sequential Logic
always @(posedge clk)
begin
state <= #1 next_state;
end

endmodule


module onehertz(clk_50mhz, clk_1hz);
input clk_50mhz;
output clk_1hz;
reg clk_1hz;
reg [24:0] count;
always @ (posedge clk_50mhz)
begin
if(count == 24999999) begin
count <= 0;
 $dumpfile("f.vcd");
 clk_1hz <= ~clk_1hz;
 end
 else begin
 count <= count + 1;
 end
end
endmodule
```

DE2 Board Implementation

L – Signifies Lock



U – Signifies Unlock

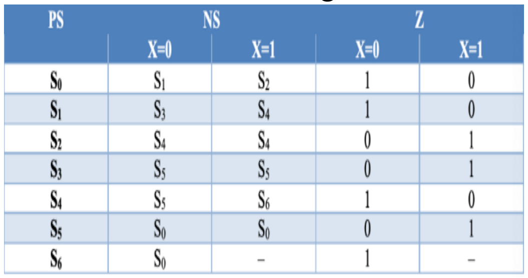


Experiment #2: Model FSM machine:

FSM Verilog Code

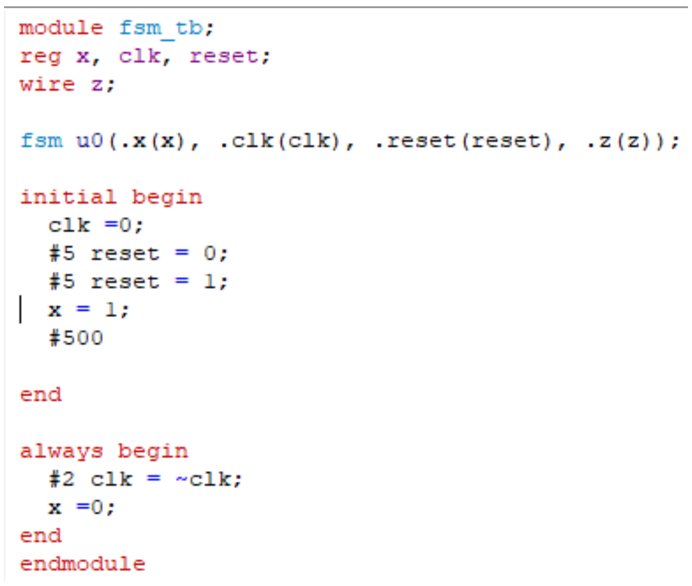
```
module fsm(clk, x, reset, z, state);
    input clk, x, reset;
    output z;
    reg z;
    output[2:0] state;
    reg[2:0] state;
    reg[2:0] next_state;

parameter[2:0] S0 = 3'b000;
parameter[2:0] S1 = 3'b001;
parameter[2:0] S2 = 3'b010;
parameter[2:0] S3 = 3'b011;
parameter[2:0] S4 = 3'b100;
parameter[2:0] S5 = 3'b101;
parameter[2:0] S6 = 3'b110;

initial begin
    state = 3'b000;
end

// Combinational Logic
always @(*)
begin
    next_state = 0;
    case(state)
        S0: begin
        z = 1'b0;  // Lock
        if ((x == 1'b0) || (reset == 1'b1)) begin
            next_state = S1;
            z = 1'b1;
        end else begin
            next_state = S2;
            z = 1'b0;
        end
        end
```

FSM Model's Table's logic



PS	NS		Z	
	X=0	X=1	X=0	X=1
S_0	S_1	S_2	1	0
S_1	S_3	S_4	1	0
S_2	S_4	S_4	0	1
S_3	S_5	S_5	0	1
S_4	S_5	S_6	1	0
S_5	S_0	S_0	0	1
S_6	S_0	-	1	-

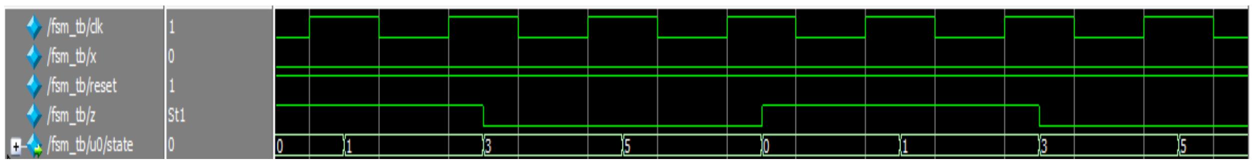
```
        S1: if (x == 1'b0 || reset == 1'b1) begin
            next_state = S3;
            z = 1'b1;
        end else begin
            next_state = S4;
            z = 1'b0;
        end

        S2: if (x == 1'b0 || reset == 1'b1) begin
            next_state = S4;
            z = 1'b0;
        end else begin
            next_state = S4;
            z = 1'b1;
        end

        S3: if (x == 1'b0 || reset == 1'b1) begin
            next_state = S5;
            z = 1'b0;
        end else begin
            next_state = S5;
            z = 1'b1;
        end

        S4: if (x == 1'b0 || reset == 1'b1) begin
            next_state = S5;
            z = 1'b1;
        end else begin
            next_state = S6;
            z = 1'b0;
        end

        S5: if (x == 1'b0 || reset == 1'b1) begin
            next_state = S0;
            z = 1'b0;
        end else begin
            next_state = S0;
            z = 1'b1;
        end

        S6: if (x == 1'b0 || reset == 1'b1) begin
            next_state = S0;
            z = 1'b1;
        end else begin
            next_state = S6;
            z = 1'b1;
        end

    endcase
end

  // Sequential Logic
  always @(posedge clk)
  begin
  state <= #1 next_state;
  end

  endmodule
```



```
module fsm_tb;
reg x, clk, reset;
wire z;

fsm u0(.x(x), .clk(clk), .reset(reset), .z(z));

initial begin
  clk =0;
  #5 reset = 0;
  #5 reset = 1;
  x = 1;
  #500

end

always begin
  #2 clk = ~clk;
  x =0;
end
endmodule
```

FSM Test Bench
Verilog Code

FSM Waveform



Experiment #3: Design Serial parity detector

Parity indicates whether the number of ‘1’s is odd or even. ‘1’ indicates odd, ‘0’ indicates even

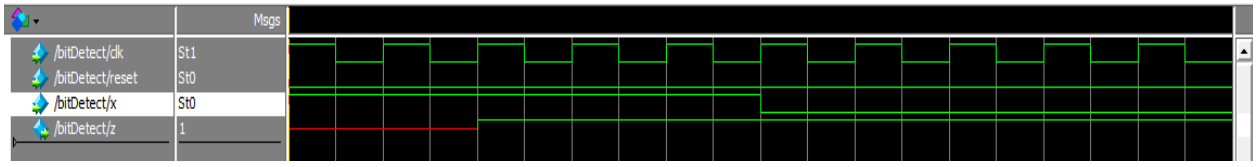
```
module bitDetect(clk, x, reset, z, state);
  input clk, x, reset;
  output z;
  reg z;
  output state;
  reg state;
  reg next_state;

parameter E = 0;
parameter O = 1;

initial begin
  state = 0;
end

// Combinational Logic
always @(*)
begin
  next_state = 0;
  case(state)
    E: begin
    if ((x == 1'b0) || (reset == 1'b1)) begin
      next_state = E;
      z = 1'b0;
    end else if ((x == 1'b1) || (reset == 1'b0)) begin
      next_state = O;
    end
    end

    O: if (x == 1'b0 || reset == 1'b0) begin
      next_state = O;
      z = 1'b1;
    end else if (x == 1'b1 || reset == 1'b1) begin
      next_state = E;
    end

  endcase
end

// Sequential Logic
always @(posedge clk)
begin
state <= #1 next_state;
end

endmodule
```

Serial parity detector Verilog Code

```
module bitDetect_tb;

reg x, clk, reset;
wire z;

initial begin
  clk = 0;
  #5 reset = 1;
  #5 reset = 0;
  x = 1;
end

always begin
  #2 clk = ~clk;
  #500;
  x = 0;
end

bitDetect TB(clk, x, reset, z, state);
endmodule
```

Serial parity detector Test Bench Code

Serial parity detector Waveform



4. Questions

1. What is the difference between mealy and moore machine?

Mealy machine outputs are dependent on present inputs, and current machine state. Moore machine outputs are dependent only on machines current state.

2. What is encoding style in a FSM?

There are many styles to encode a FSM, One hot, Binary, Gray, One cold, etc. Encoding is a way to assign unique patterns of zeros and ones to the defined states of a Finite State Machine.

3. What is a combinational section and sequential section in a FSM design used for?

The combinational section of an FSM is designed for the next state logic of the FSM. Combinational section is modeled using functions, assignment statements or case statements. The sequential section of an FSM is designed for the state registers and output logic. Sequential section is modeled using only edge sensitive logic such as always block w/posedge or negedge of clock.

5. Conclusions

- To conclude this lab experiment, we learned how to properly create a FSM for the given experiments. We learned about the two types of FSMs, Mealy and Moore and how they differ. This Lab allowed us to create the FSM for a digital lock, Modeling a FSM from a table, and create a FSM for a serial parity bit detector. We did not really encounter any problems for this lab, a possible problem could be that we ran out of time in the schedule lab hours. In order to resolve this problem, my partner and I met together outside of the scheduled lab hours to finish any unfinished experiments. Overall this lab taught us the steps to take, necessary logic, and design techniques are needed in order to create a Finite State Machine.

Class:	**CPE-300L**	Semester:	**Spring 2020**
Points		Document author:	**Brysen Kokubun**
		Author's email:	**Brysenkokubun96@gmail.com**
		Document topic:	**Prelab 5**
Instructor's comments:			

Introduction / Theory of operation

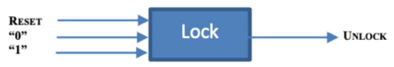
- For this lab experiment we will be learning about Finite State Machines. There are two types of machines that we will be implementing, Mealy and Moore machines. Mealy machines outputs are dependent on present inputs, and current state while Moore machines outputs are dependent only on machines current state. This lab will teach us about FSM components, design and how to implement those designs as waveforms and on DE2 boards.

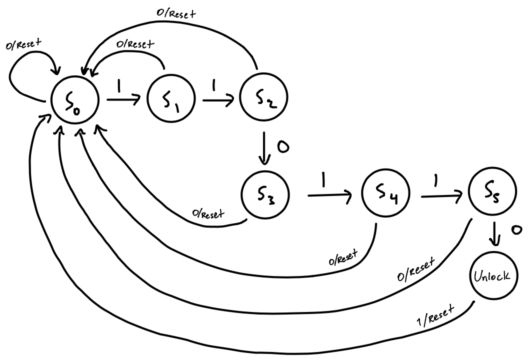
Prelab main content

1. What is a Finite State Machine? What are the components in a FSM?

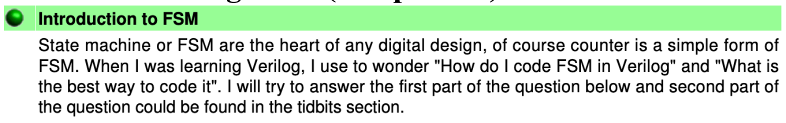
- A Finite State Machine is a computation model (software/hardware) that is used to simulate sequential logic. Also, the FSM can only be in one state at any given time. The components in a FSM are sets of states, transitions between states, and actions associated with each transition (entering, exiting or remaining in a state).

2. Digital lock design





3. Read modeling FSM (Chapter 14)



Introduction to FSM

State machine or FSM are the heart of any digital design, of course counter is a simple form of FSM. When I was learning Verilog, I use to wonder "How do I code FSM in Verilog" and "What is the best way to code it". I will try to answer the first part of the question below and second part of the question could be found in the tidbits section.