**ELEC 4200 Lab 9 Report**

**Xiaodong Song**

**4/11/2020**

**Goal for the Lab**

For lab 9, students would need to try to write Model Mealy FSMs and Model Moore FSMs. Finite State Machines are sequential circuit used in many digital systems to control the behavior of systems and dataflow paths. This lab introduces Mealy State Machine and Moore State Machine. There are totally three tasks need student to model and develop testbench.

**Design for the Lab**

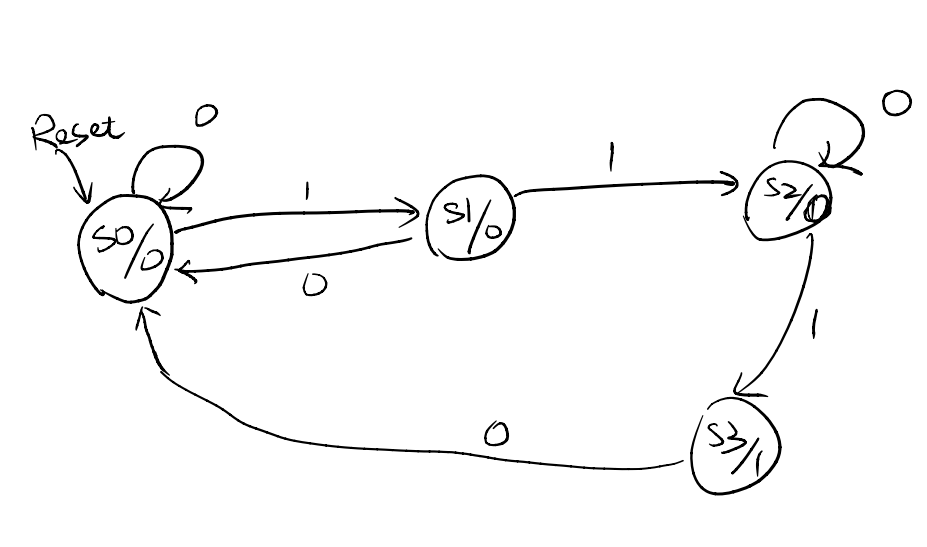
For lab 9-1-1, students would design a sequence detector implementing a Mealy state machine using three always blocks. After finishing the design source, students need to develop testbench for verification.

For lab 9-2-1, students would design a sequence detector implementing a Moore state machine using three always blocks. After finishing the design source, students need to develop testbench for verification.

For lab 9-3-1, students would design a specific counts counter using ROM to develop a Mealy state machine. After finishing the design source, students need to develop testbench for verification.

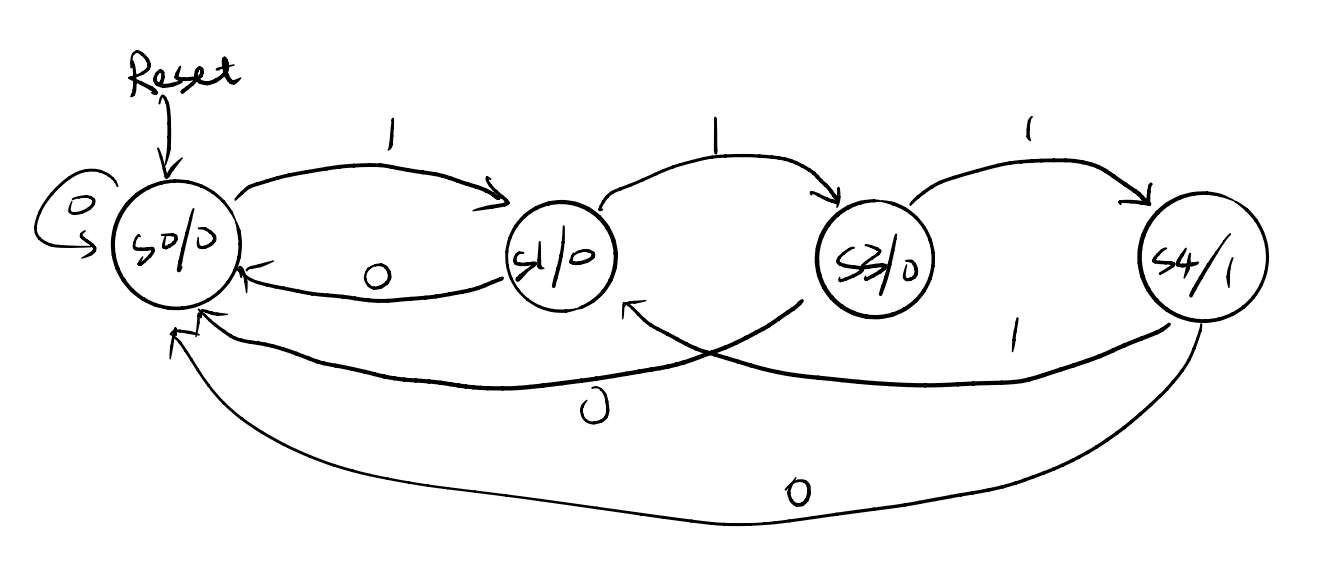
**Detailed Design**

For lab 9-1-1, there were one input “ain” and one output “yout”. The output yout would be 1 if and only if the total number of 1s received is divisible by 3. There should be a clock and a BTNU button which is reset. 4-bit output count represent the number of 1s. And clock controls when output changes as input changing. And reset controls when the count becomes 0. The following is the FSMs model:



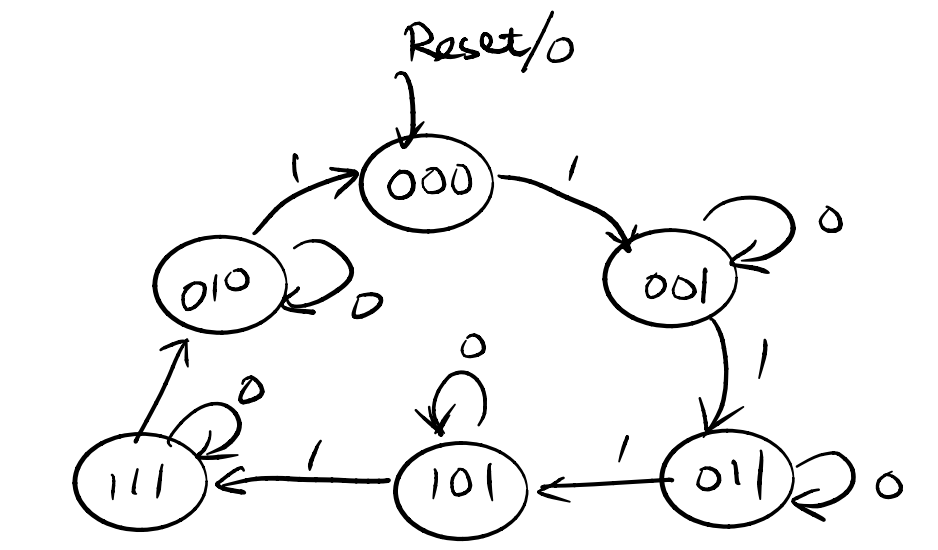
**Figure 1**: 9-1-1’s FSM model

For lab 9-2-1, there were one 2-bit input “ain[1:0]” and one output “yout” for presenting the result. Also, clock and reset still are needed here. The output “yout” would change as input “ain” change. To be specific, if the input sequence ain[1:0] is equal to 01, 00, then the output “yout” should become 0. If the input sequence ain[1:0] is equal to 11, 00, then the output “yout” should become 1. And if the input ain[1:0] is equal to 10, 00, then the output “yout” should toggle. Clock controls when output changes as input changing. And reset controls when the state becomes 0. The following is the FSMs model:



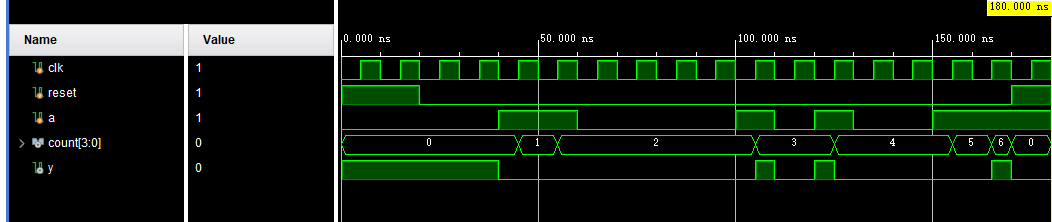
**Figure** **2**: 9-2-1’s FSM model

For lab 9-3-1, there were two inputs “clock”, “reset” and one 3-bit output “count”. The output “count” would present the count output of counter. The counting sequence would be: 000, 001, 011, 101, 111, 010 (repeat) 000, … The sequence should be input as a .txt file for the simulation file. Clock still control when the output “count” go to the next one sequence number. And count would become 3-bit number 0 when the “reset” is high. The following is the FSMs model:



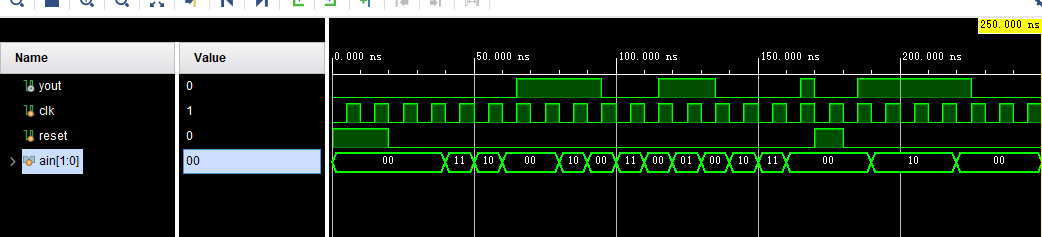
**Figure 3:** 9-3-1’s FSM model

**Design Verification**

For lab 9-1-1’s verification, the following is the simulation result. As the figure showing, at the 45ns, the input “a” and “clk” are high, then the output count became “1”, and when the next moment “clk” becomes high, the counter becomes 2. For reset, at 170ns, the reset is high then the counter goes back to 0. For the output y, it becomes 1 when the count is equal to 0, 3, 6, obviously. 

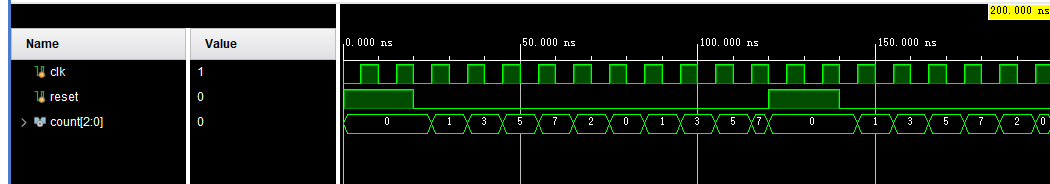
**Figure 4:** 9-1-1’s simulation result

For lab 9-2-1’s verification, the following is the simulation result, as the figure showing, at the 95ns, there were input 10,00 in sequence, then the output “yout” became low by toggling. At 115 ns, there were input 11,00 in sequence, then the output “yout” became high at the 115 ns. And at 135 ns, there were input 01,00 in sequence, then the output “yout” became low by toggling. And 01,00 let the output “yout” keep low at 140ns. Reset let output “yout” go back to 0 at 165 ns.



**Figure 5:** 9-2-1’s simulation result

For lab 9-3-1’s verification, the following is the simulation result, as the figure showing, the output count sequence is 0,1,3,5,7,2,0… which’s binary number is 000,001,011,101,111,010,000… respectively. And reset let the output “count” became low from 7 at 120 ns.



**Conclusion**

After finishing this lab, students would be more familiar with the Mealy State Machines and Moore State Machines. The 9-1-1 is the topic of Mealy State Machines. And the 9-2-1 is the topic of Moore State Machines. And for the 9-3-1, it is the combination between Mealy State Machines and ROM using. Combination practice can be helpful to students’ understanding for Verilog.

**Appendix**

module lab9\_1\_1(

input clk,ain,reset,

output reg yout,

output reg [3:0] count

);

parameter S0 = 0, S1 = 1, S2 = 2, S3 = 3;

reg [1:0] state, nextstate;

initial begin

state = S0;

nextstate = S0;

count = 0;

end

// always block to apply next state

always @(posedge clk or posedge reset) begin

if (reset) begin

count <= 0;

state <= S0;

nextstate <= S0;

end

else begin

count <= count + ain;

state <= nextstate;

end

end

// always block to compute next state

always @(state or ain) begin

if (ain)

case(state)

S0: nextstate <= S1;

S1: nextstate <= S2;

S2: nextstate <= S3;

S3: nextstate <= S1;

endcase

else

nextstate <= state;

end

// always block to compute output

always @(state or ain) begin

case(state)

S0: yout <= ~ain;

S1: yout <= 0;

S2: yout <= 0;

S3: yout <= ain;

endcase

end

endmodule

module lab9\_1\_1\_tb;

reg clk, reset, a;

wire [3:0] count;

wire y;

lab9\_1\_1 DUT (clk, a, reset, y, count);

initial begin

clk = 0; reset = 1; a = 0;

#20 reset = 0;

#20 a = 1;

#20 a = 0;

#40 a = 1;

#10 a = 0;

#10 a = 1;

#10 a = 0;

#20 a = 1;

#10 a = 1;

#10 reset = 1;

#10 $finish();

end

always #5 clk <= ~clk;

endmodule

module lab9\_2\_1(

input clk,reset,

input [1:0]ain ,

output yout

);

parameter S00 = 2'b00, S01 = 2'b01, S10 = 2'b10, S11 = 2'b11;

reg [1:0] state, nextstate;

reg y, nexty;

assign yout = y;

initial begin

y = 0;

nexty = 0;

state = S00;

nextstate = S00;

end

// always block to apply state

always @(posedge clk or posedge reset) begin

if (reset) begin

y <= 0;

state <= S00;

end

else begin

y <= nexty;

state <= nextstate;

end

end

// always block to update state

always @(state or ain) begin

case(ain)

2'b00: nextstate <= S00;

2'b01: nextstate <= S01;

2'b10: nextstate <= S10;

2'b11: nextstate <= S11;

endcase

end

// always block to compute output

always @(state or ain) begin

if (ain == 2'b00)

case(state)

S00: nexty <= y;

S01: nexty <= 0;

S10: nexty <= ~y;

S11: nexty <= 1;

endcase

else

nexty <= y;

end

endmodule

module lab9\_2\_1\_tb;

wire yout;

reg clk, reset;

reg [1:0] ain;

lab9\_2\_1 DUT (clk, reset, ain, yout);

initial begin

clk = 0; reset = 1; ain = 2'b00;

#20 reset = 0;

#20 ain = 2'b11;

#10 ain = 2'b10;

#10 ain = 2'b00;

#20 ain = 2'b10;

#10 ain = 2'b00;

#10 ain = 2'b11;

#10 ain = 2'b00;

#10 ain = 2'b01;

#10 ain = 2'b00;

#10 ain = 2'b10;

#10 ain = 2'b11;

#10 ain = 2'b00;

#10 reset = 1;

#10 reset = 0;

#10 ain = 2'b10;

#30 ain = 2'b00;

#30 $finish();

end

always #5 clk <= ~clk;

endmodule

module lab9\_3\_1(

input clk,reset,

output reg [2:0] cout

);

reg [2:0] state, nextstate;

reg [2:0] NEXT [5:0];

reg [2:0] OUT [5:0];

initial begin

state = 0;

nextstate = 0;

$readmemb("output.txt", OUT, 0, 5);

$readmemh("next.txt", NEXT, 0, 5);

end

always @(posedge clk or posedge reset) begin

if (reset)

state <= 0;

else

state <= nextstate;

end

always @(state) begin

nextstate <= NEXT[state];

cout <= OUT[state];

end

endmodule

module lab9\_3\_1\_tb;

reg clk, reset;

wire [2:0] count;

lab9\_3\_1 DUT (clk, reset, count);

initial begin

clk = 0; reset = 1;

#20 reset = 0;

#100 reset = 1;

#20 reset = 0;

#60 $finish();

end

always #5 clk <= ~clk;

endmodule

next.txt:

1 2 3 4 5 0

output.txt:

000

001

011

101

111

010