ELEC-4200

Digital System Design

FROM: Jacob Howard

TO: Prof. Ujjwal Guin

DUE DATE: 2/18/21

Lab 5

Introduction

The goal of this lab was to familiarize ourselves with Finite State Machines. There are two main types of FSMs. These are Mealy FSMs, which the output relies on the input and the state register, and a Moore machine, where the output does not necessarily rely on the input but the state register. *Note during this week's lab, I was sick and so I could not perform any physical testing.*

Task 1

In Task 1, we were asked to design a sequence detector implementing a Mealy state machine using three always blocks. I felt like most of these tasks where wuite difficult to complete, but I did end up with a code that I do believe would work in physical testing. The code is shown below in *Code 1*.

```
always @ (posedge clk or negedge reset) begin
if (~reset)
state <= S0;
else
state <= next_state;
end
always @ (*) begin
case (state)
S0:begin
if (a==1)
next_state = S1;
else
next_state = S0;
end
S1:begin
if (a==1)
next_state = S2;
else
next_state = S0;
end
S2:begin
if (a==1)
next_state = S3;
else
next_state = S0;
end
```

```
S3:begin
if (a==1)
next_state = S1;
else
next_state = S0;
end
default: next_state = S0;
endcase
end

always @ (*) begin
case (state)
S0,S1,S2: yout = 1'b0;
S3:begin
if (a==1)
yout = 1'b1;
else
yout = 1'b0;
end

default: yout = 1'b0;
endcase
end

endmodule
```

Code 1

Task 2

In Task 2, we were asked to design a sequence detector implementing a Moore state machine using three always blocks. I felt that task 2 was slightly harder to do than task 1. I never quite figured out how to design this Moore machine and so I do not have any correct code to provide. Having the Tas help during the lab session could have assisted me in solving this problem.

Task 3

In Task 3, we were asked to design a specific counts counter using ROM to develop a Mealy state machine. I believe I solved this task with the design I coded in *Code 2* below.

```
assign out_seq=state;
always@(posedge clk)
begin
if(reset)
state<=s0;
else
state<=nextstate;
end

always@(*)
begin
case(state)
s0: nextstate<=s1;
s1: nextstate<=s2;
s2: nextstate<=s3;
s3: nextstate<=s4;
s4: nextstate<=s5;
s5: nextstate<=s0;
default: nextstate<=s0;
endcase
end
endmodule
```

Code 2

Conclusion

In conclusion, being sick during this lab session made it difficult to receive help for these problems. I do believe I solved most solutions on my own time, but having lab time to physically test and receive help from the TA is very beneficial. I also believe this lab was somewhat difficult in solving all tasks.