Lab 4: State Machine

T 2pm-5pm

Zichen Huang and Jack Landers

Introduction

In this lab we designed a state machine for a car's turning signals. This follows the pattern of turning on one direction light and then another if we keep turning that direction and then a 3rd before cycling back to only one light or returning to none if the car straightens. We also had to take into account if the direction changes, then the light's state would switch to the first of the other direction.

Procedure

For our code we defined 7 states for each light output. We also define two 8-bit registers: current state and next state. In an always loop on the positive edge of the clock we initialize the current state to reset or set the current state to the next state so that it changes with each clock cycle. To program our state machine we used a case statement that determines which state we are currently in and then if statements to define which state we would change to next. Our output for each state is described structurally, outside of this always block, where the lights are two buses and each bit is assigned for the relevant states.

Result

Our testbench started with the reset so that our outputs can be declared, and then we tested various patterns of our input direction state changing, to ensure that our states changed to determine our outputs as we wished.

What problems did you encounter while testing your steps yourself?

We had a problem with assigning the output signal of our turning signal module. We used the “case” statementat the beginning, which couldn’t work outside the always block. So we switched to assign each bit of the output signal according to the current state using the “assign” statement. And it worked.

Did any problems arise when demonstrating for the TA? What were they? Explain your thoughts on how/why these testcases escaped your own testing.

We didn’t have two extra straight input after the whole test. It showed that the state won’t change if we do nothing.

Working Source Code for Turning Signal:

```
module TurnSignal (rst, clk, S, L, R);

input rst, clk;
input [1:0]S;
output [2:0]L, R;
reg [7:0]cs, ns;

parameter [7:0] A=8'h01, B=8'h02, C=8'h04, D=8'h08, E=8'h10, F=8'h20, G=8'h40;

// S = 2'b00 is for the straight signal input
// S = 2'b01 is for the left turn signal input
// S = 2'b10 is for the right turn signal input

// Define the sequential block
always @(posedge clk) begin
        if(rst) cs <= A;
        else cs <= ns;
end

// Define the next state combinational circuit
always @(*) begin
        case (cs)
                A:      case (S)
                                2'b00: ns = A;
                                2'b01: ns = B;
                                2'b10: ns = E;
                        endcase
                B:      case (S)
                                2'b00: ns = A;
                                2'b01: ns = C;
                                2'b10: ns = E;
                        endcase
                C:      case (S)
                                2'b00: ns = A;
                                2'b01: ns = D;
                                2'b10: ns = E;
                        endcase
                D:      case (S)
                                2'b00: ns = A;
                                2'b01: ns = B;
                                2'b10: ns = E;
                        endcase
                E:      case (S)
```

```
                    2'b00: ns = A;
                    2'b01: ns = B;
                    2'b10: ns = F;
                endcase
        F:      case (S)
                    2'b00: ns = A;
                    2'b01: ns = B;
                    2'b10: ns = G;
                endcase
        G:      case (S)
                    2'b00: ns = A;
                    2'b01: ns = B;
                    2'b10: ns = E;
                endcase
    endcase
end

// Define the output
assign L[0] = (cs == B || cs == C || cs == D);
assign L[1] = (cs == C || cs == D);
assign L[2] = (cs == D);
assign R[0] = (cs == E || cs == F || cs == G);
assign R[1] = (cs == F || cs == G);
assign R[2] = (cs == G);

endmodule
```

Working Testbench:

```
module TurnSignaltb();

reg clk, rst;
reg [1:0]S;
wire [2:0]L, R;

TurnSignal dut(rst, clk, S, L, R);

always begin
#5 clk = ~clk;
$display("clk = %b, rst = %b, S = %b, L = %b, R = %b, @ %d", clk, rst, S, L, R, $time);
#5 clk = ~clk;
end

initial begin
    $display("Start of the turn signal state machine test");
```

```
clk <= 0;
#1
rst <= 1; S = 2'b00; //beginning
#10
rst <= 0; S = 2'b01; //left turn signal test B-C-D-B
#10
rst <= 0; S = 2'b01;
#10
rst <= 0; S = 2'b01;
#10
rst <= 0; S = 2'b01;
#10
rst <= 0; S = 2'b00; //switch back to straight
#10
rst <= 0; S = 2'b10; //right turn signal test E-F-G-E
#10
rst <= 0; S = 2'b10;
#10
rst <= 0; S = 2'b10;
#10
rst <= 0; S = 2'b10;
#10
rst <= 0; S = 2'b00; //switch back to straight
#10
rst <= 0; S = 2'b01; //left to right test
#10
rst <= 0; S = 2'b10;
#10
rst <= 0; S = 2'b01; //right to left test
#10
rst <= 1; S = 2'b01; //reset test
#10
rst <= 0; S = 2'b00; //reset test
#10
rst <= 0; S = 2'b00;
#10
rst <= 0; S = 2'b00;
#10
        $finish;
end
endmodule
```