# Finite State Machine – Lab 2

by

Jonathan, William and Raashid

March 21, 2015

Design justification:

This FSM can be looked as a stepper motor. The controls take care of which direction to move to and the states take care of the angle of rotation. The reset as the name suggests, resets the machine to a pre-defined initial state which has been selected as 'state_0 i.e. 0101' in our design.

We use five registers in our design. One four bit register to see the output i.e. next state (i.e. curr_state [3:0]). And another one bit register for saving the warning signal (i.e. warning bit).

Following are the design decisions/assumptions taken into consideration before development of the source code:

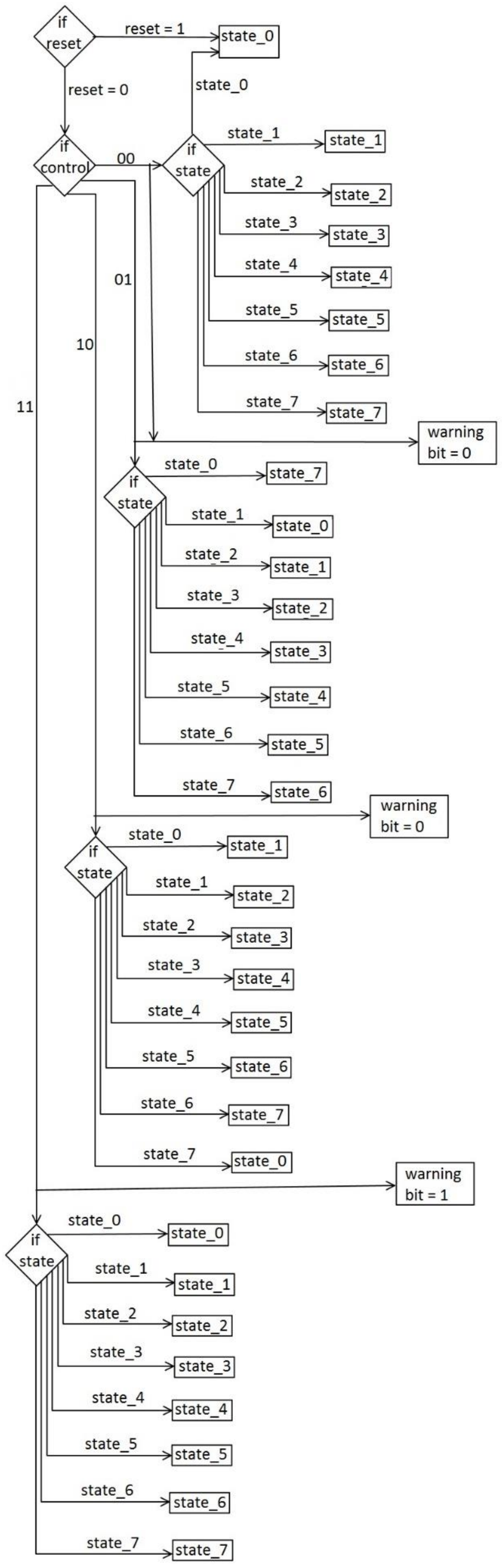
1. A synchronous machine has been developed to change state at the positive edge of the clock.
2. Initial state has been set to state_0 i.e. 0101.
3. To change the states we use a 2-bit control signal in which the 0th bit represent the counter-clockwise (CCW) bit and 1st bit represents clockwise (CW) bit. We have assigned the control signal as follows:
    - Control assignments {CW, CCW}:
        - Stay in same state: 00
        - Move clockwise: 10
        - Move anti-clockwise: 01
        - Warning state: 11
4. When both CW and CCW bits go high, the machine stays in the same state but sends out a warning signal by setting the warning bit high.
5. Since the state assignment have been done using four-bits in the provided problem statement, we decided to use a 4 bit vector (curr_state [3:0]). This represents the current state of the machine.
6. Next state of the machine has been assigned as (state_out [3:0]) in the testbench. Although it is just the next iteration of values of curr_state.
7. To take care of the situations where our FSM goes into an unknown state, we decided to set the machine back to initial state i.e. state_0.
8. State assignments according to the provided problem statement:
    - State 0 = 0101
    - State 1 = 0001
    - State 2 = 1001
    - State 3 = 1000
    - State 4 = 1010
    - State 5 = 0010
    - State 6 = 0110
    - State 7 = 0100

9. Below is the truth table/state transition table for our FSM.

| Present State | Control = {CW,CCW} | Reset (Active high) | Next State | Warning bit |
|---|---|---|---|---|
| State 0 | 00 | 0 | State 0 | 0 |
| State 0 | 00 | 1 | State 0 | 0 |
| State 0 | 01 | 0 | State 7 | 0 |
| State 0 | 01 | 1 | State 0 | 0 |
| State 0 | 10 | 0 | State 1 | 0 |
| State 0 | 10 | 1 | State 0 | 0 |
| State 0 | 11 | 0 | State 0 | 1 |
| State 0 | 11 | 1 | State 0 | 1 |
| State 1 | 00 | 0 | State 1 | 0 |
| State 1 | 00 | 1 | State 0 | 0 |
| State 1 | 01 | 0 | State 0 | 0 |
| State 1 | 01 | 1 | State 0 | 0 |
| State 1 | 10 | 0 | State 2 | 0 |
| State 1 | 10 | 1 | State 0 | 0 |
| State 1 | 11 | 0 | State 1 | 1 |
| State 1 | 11 | 1 | State 0 | 1 |
| State 2 | 00 | 0 | State 2 | 0 |
| State 2 | 00 | 1 | State 0 | 0 |
| State 2 | 01 | 0 | State 1 | 0 |
| State 2 | 01 | 1 | State 0 | 0 |
| State 2 | 10 | 0 | State 3 | 0 |
| State 2 | 10 | 1 | State 0 | 0 |
| State 2 | 11 | 0 | State 2 | 1 |
| State 2 | 11 | 1 | State 0 | 1 |
| State 3 | 00 | 0 | State 3 | 0 |
| State 3 | 00 | 1 | State 0 | 0 |
| State 3 | 01 | 0 | State 2 | 0 |
| State 3 | 01 | 1 | State 0 | 0 |
| State 3 | 10 | 0 | State 4 | 0 |
| State 3 | 10 | 1 | State 0 | 0 |
| State 3 | 11 | 0 | State 3 | 1 |
| State 3 | 11 | 1 | State 0 | 1 |
| State 4 | 00 | 0 | State 4 | 0 |
| State 4 | 00 | 1 | State 0 | 0 |
| State 4 | 01 | 0 | State 3 | 0 |
| State 4 | 01 | 1 | State 0 | 0 |
| State 4 | 10 | 0 | State 5 | 0 |
| State 4 | 10 | 1 | State 0 | 0 |
| State 4 | 11 | 0 | State 4 | 1 |
| State 4 | 11 | 1 | State 0 | 1 |
| State 5 | 00 | 0 | State 5 | 0 |
| State 5 | 00 | 1 | State 0 | 0 |

| | | | | |
|---|---|---|---|---|
| State 5 | 01 | 0 | State 4 | 0 |
| State 5 | 01 | 1 | State 0 | 0 |
| State 5 | 10 | 0 | State 6 | 0 |
| State 5 | 10 | 1 | State 0 | 0 |
| State 5 | 11 | 0 | State 5 | 1 |
| State 5 | 11 | 1 | State 0 | 1 |
| State 6 | 00 | 0 | State 6 | 0 |
| State 6 | 00 | 1 | State 0 | 0 |
| State 6 | 01 | 0 | State 5 | 0 |
| State 6 | 01 | 1 | State 0 | 0 |
| State 6 | 10 | 0 | State 7 | 0 |
| State 6 | 10 | 1 | State 0 | 0 |
| State 6 | 11 | 0 | State 6 | 1 |
| State 6 | 11 | 1 | State 0 | 1 |
| State 7 | 00 | 0 | State 7 | 0 |
| State 7 | 00 | 1 | State 0 | 0 |
| State 7 | 01 | 0 | State 6 | 0 |
| State 7 | 01 | 1 | State 0 | 0 |
| State 7 | 10 | 0 | State 0 | 0 |
| State 7 | 10 | 1 | State 0 | 0 |
| State 7 | 11 | 0 | State 7 | 1 |
| State 7 | 11 | 1 | State 0 | 1 |

The flowchart below gives a better idea about our design:



Source Code Design and Post-Synthesis:

The source code for the finite state machine is constructed primarily utilizing nested case statements, checking the current state of the system and then deciding the proper course of action based on the 2-bit control signal input and 1-bit reset signal input. Before the always statement, each of the possible control inputs and output states are declared as a localparam. On each positive clock edge, the module first checks the state of the reset signal. If high, the current state of the FSM is changed to state_0. If the reset bit is not high, then the control inputs are evaluated. The current state is then evaluated and based on the control inputs, a new state is set relative to the current state. This allows the FSM to easily move clockwise or counterclockwise through its series of states.

In the event that both control bits are high, a warning bit is set to high at the output. Given any other control input on the next clock cycle (not 2’b11), the warning bit is reset to 0 and the proper action is taken based on the new control bit inputs. This module accommodates any possible combination of input signals, allowing the test bench to validate the design effectively. Performing post-synthesis compilation on the module generates the circuit shown below:

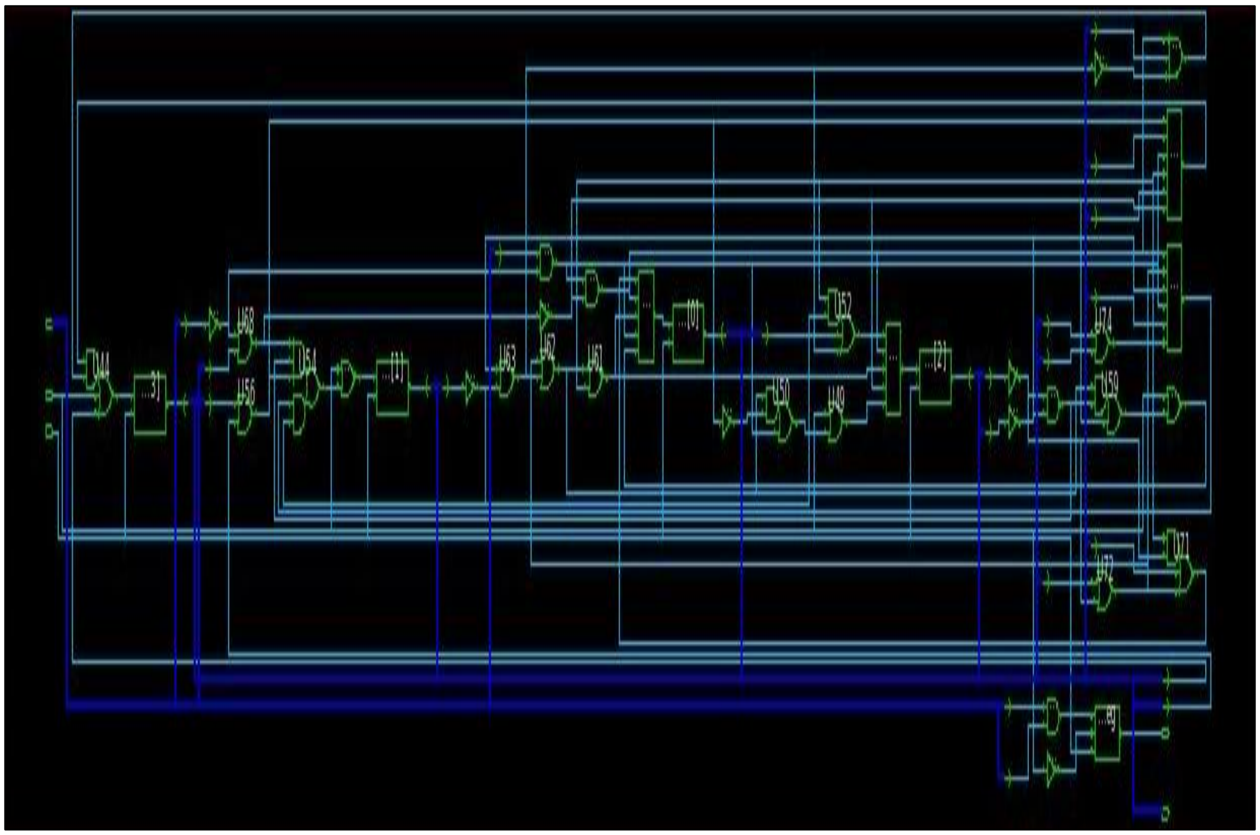


Figure 1: Synthesized Circuit

Module Source Code:

```
`timescale 1ns/1ns
module stepper1(clk, control, reset, warnbit, curr_state);
    input clk, reset;
    input [1:0] control;
    output warnbit;
    output [3:0] curr_state;

    wire reset;
    wire [1:0] control;
    reg warnbit;

    //Control bit inputs (cw, ccw)
    localparam same_state_ctrl = 2'd0;
    localparam clk_wise_ctrl = 2'd2;
    localparam counter_clk_wise_ctrl = 2'd1;
    localparam warn_state_ctrl = 2'd3;

    //State outputs
    localparam state_0 = 4'd5;
    localparam state_1 = 4'd1;
    localparam state_2 = 4'd9;
    localparam state_3 = 4'd8;
    localparam state_4 = 4'd10;
    localparam state_5 = 4'd2;
    localparam state_6 = 4'd6;
    localparam state_7 = 4'd4;

    reg [3:0] curr_state;

    always@(posedge clk)
    begin
        if (reset)
        begin
            curr_state = state_0;
        end
        else
        begin
            case (control)
                same_state_ctrl:
                begin
                    warnbit = 1'b0;
                    case (curr_state)
                        state_0: curr_state = state_0;
                        state_1: curr_state = state_1;
                        state_2: curr_state = state_2;
                        state_3: curr_state = state_3;
                        state_4: curr_state = state_4;
                        state_5: curr_state = state_5;
                        state_6: curr_state = state_6;
                        state_7: curr_state = state_7;
                        default: curr_state = state_0;
                    endcase
                end
                counter_clk_wise_ctrl:
                begin
```

```
                    warnbit = 1'b0;
                    case (curr_state)
                        state_0: curr_state = state_7;
                        state_1: curr_state = state_0;
                        state_2: curr_state = state_1;
                        state_3: curr_state = state_2;
                        state_4: curr_state = state_3;
                        state_5: curr_state = state_4;
                        state_6: curr_state = state_5;
                        state_7: curr_state = state_6;
                        default: curr_state = state_0;
                    endcase
                end
                clk_wise_ctrl:
                begin
                    warnbit = 1'b0;
                    case (curr_state)
                        state_0: curr_state = state_1;
                        state_1: curr_state = state_2;
                        state_2: curr_state = state_3;
                        state_3: curr_state = state_4;
                        state_4: curr_state = state_5;
                        state_5: curr_state = state_6;
                        state_6: curr_state = state_7;
                        state_7: curr_state = state_0;
                        default: curr_state = state_0;
                    endcase
                end
                warn_state_ctrl:
                begin
                    warnbit = 1'b1;
                    case (curr_state)
                        state_0: curr_state = state_0;
                        state_1: curr_state = state_1;
                        state_2: curr_state = state_2;
                        state_3: curr_state = state_3;
                        state_4: curr_state = state_4;
                        state_5: curr_state = state_5;
                        state_6: curr_state = state_6;
                        state_7: curr_state = state_7;
                        default: curr_state = state_0;
                    endcase
                end
            endcase
        end
    end
endmodule
```

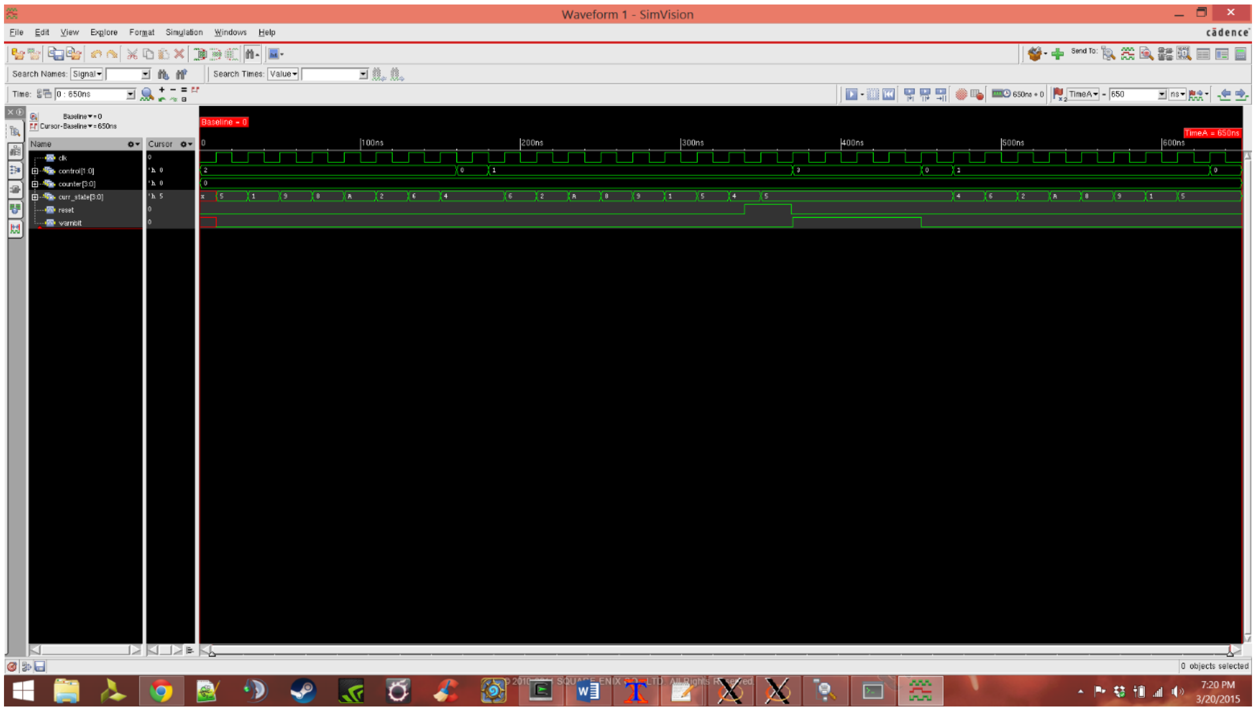
Synthesized Module Code:

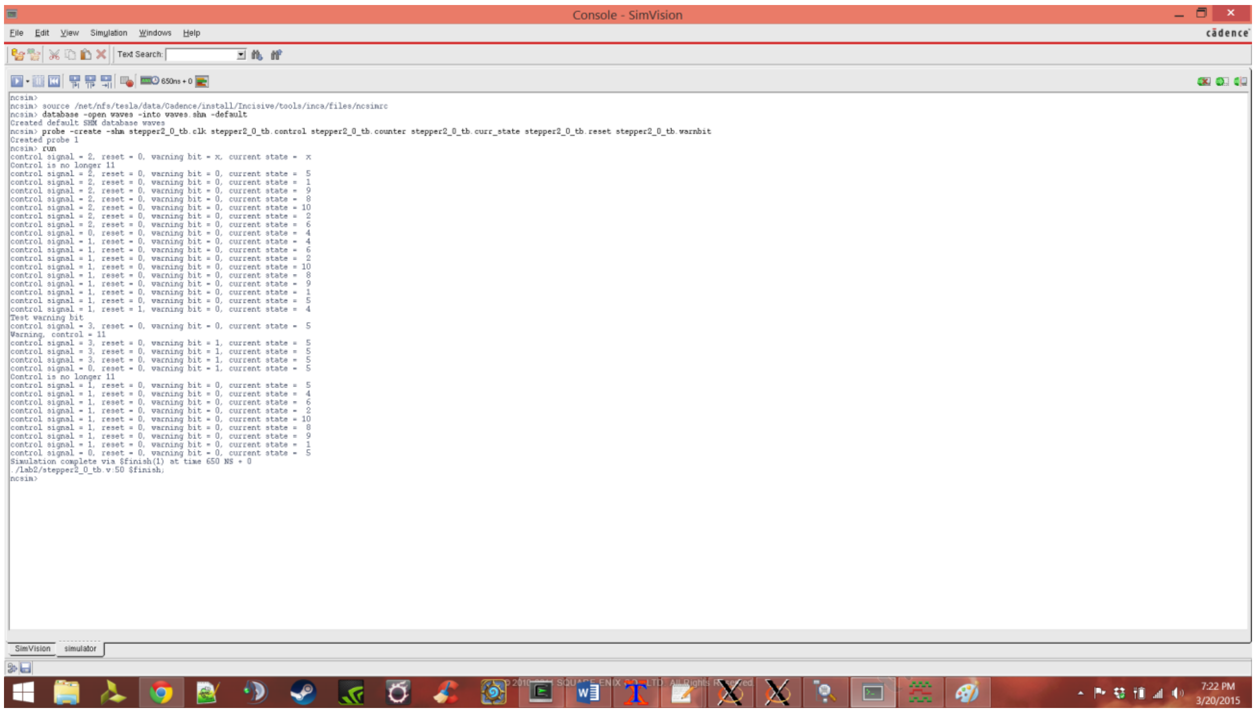
```
module stepper1 ( clk, control, reset, warnbit, curr_state );
  input [1:0] control;
  output [3:0] curr_state;
  input clk, reset;
  output warnbit;
  wire   N31, N46, N47, N48, N49, n12, n36, n37, n38, n39, n40, n41, n42,
n43,
         n44, n45, n46, n47, n48, n49, n50, n51, n52, n53, n54, n55, n56,
n57,
         n58, n59, n60, n61, n62;

  EDFQD1 warnbit_reg ( .D(N31), .E(n12), .CP(clk), .Q(warnbit) );
  DFQD1 \curr_state_reg[0]  ( .D(N46), .CP(clk), .Q(curr_state[0]) );
  DFQD1 \curr_state_reg[1]  ( .D(N47), .CP(clk), .Q(curr_state[1]) );
  DFQD1 \curr_state_reg[2]  ( .D(N48), .CP(clk), .Q(curr_state[2]) );
  DFQD1 \curr_state_reg[3]  ( .D(N49), .CP(clk), .Q(curr_state[3]) );
  CKND0 U43 ( .I(reset), .ZN(n12) );
  AOI211D0 U44 ( .A1(n36), .A2(n37), .B(reset), .C(curr_state[2]), .ZN(N49)
);
  IND3D0 U45 ( .A1(n38), .B1(curr_state[3]), .B2(n39), .ZN(n37) );
  CKND0 U46 ( .I(n40), .ZN(n39) );
  OA33D0 U47 ( .A1(n41), .A2(curr_state[1]), .A3(n42), .B1(n43), .B2(
        curr_state[0]), .B3(n44), .Z(n36) );
  OAI221D0 U48 ( .A1(n45), .A2(n38), .B1(n46), .B2(n44), .C(n47), .ZN(N48) );
  NR2D0 U49 ( .A1(reset), .A2(n48), .ZN(n47) );
  AOI21D0 U50 ( .A1(n49), .A2(n50), .B(n42), .ZN(n48) );
  CKND0 U51 ( .I(n41), .ZN(n50) );
  AOI211D0 U52 ( .A1(n43), .A2(n51), .B(curr_state[2]), .C(n40), .ZN(n45) );
  NR2D0 U53 ( .A1(reset), .A2(n52), .ZN(N47) );
  AOI32D0 U54 ( .A1(n53), .A2(n54), .A3(n41), .B1(n55), .B2(n51), .ZN(n52) );
  OAI33D0 U55 ( .A1(n38), .A2(n56), .A3(n57), .B1(n42), .B2(curr_state[2]),
        .B3(n43), .ZN(n55) );
  NR2D0 U56 ( .A1(curr_state[0]), .A2(curr_state[3]), .ZN(n41) );
  OAI221D0 U57 ( .A1(n58), .A2(n38), .B1(n46), .B2(n42), .C(n59), .ZN(N46) );
  NR2D0 U58 ( .A1(reset), .A2(n60), .ZN(n59) );
  AOI21D0 U59 ( .A1(n49), .A2(n54), .B(n44), .ZN(n60) );
  CKND2D0 U60 ( .A1(n61), .A2(n57), .ZN(n54) );
  INR2D0 U61 ( .A1(n49), .B1(n43), .ZN(n46) );
  NR2D0 U62 ( .A1(n56), .A2(n40), .ZN(n49) );
  NR2D0 U63 ( .A1(n57), .A2(n51), .ZN(n40) );
  CKND0 U64 ( .I(curr_state[0]), .ZN(n51) );
  CKND0 U65 ( .I(curr_state[1]), .ZN(n57) );
  CKND2D0 U66 ( .A1(n42), .A2(n44), .ZN(n38) );
  CKND0 U67 ( .I(n53), .ZN(n44) );
  NR2D0 U68 ( .A1(n62), .A2(control[1]), .ZN(n53) );
  CKND2D0 U69 ( .A1(control[1]), .A2(n62), .ZN(n42) );
  CKND0 U70 ( .I(control[0]), .ZN(n62) );
  AOI211D0 U71 ( .A1(n43), .A2(n61), .B(curr_state[0]), .C(n56), .ZN(n58) );
  INR2D0 U72 ( .A1(curr_state[3]), .B1(n61), .ZN(n56) );
  CKND0 U73 ( .I(curr_state[2]), .ZN(n61) );
  NR2D0 U74 ( .A1(curr_state[1]), .A2(curr_state[3]), .ZN(n43) );
  AN2D0 U75 ( .A1(control[1]), .A2(control[0]), .Z(N31) );
endmodule
```

Testbench justification:

a. We verify the function by observing the current states of the FSM. We start testing the FSM by setting the control signal to binary 10, which sets the CW signal to 1, and CCW to 0 respectively. This essentially makes the FSM move about clockwise. This will make the FSM move throughout the states. We do this until the FSM has made a full circle. We then set the control bit to 01, which will make the FSM move counter-clockwise. We have the FSM move states counter clockwise one full circle. We then set the reset bit to 1, forcing a reset to the first FSM state. We then set the reset bit back to 0, and have then test the control's warning signal. The warning bit goes high when the control is manually set to binary 11. We then set the control signal back to 01 to have the FSM move counter-clockwise again. For one full circle.

b. We believe this is enough cases as it tests all of the possible outcomes.





```
ncsim>
ncsim> source /net/nfs/tesla/data/Cadence/install/Incisive/tools/inca/files/ncsimrc
ncsim> database -open waves -into waves.shm -default
Created default SHM database waves
ncsim> probe -create -shm stepper2_0_tb.clk stepper2_0_tb.control stepper2_0_tb.counter stepper2_0_tb.curr_state stepper2_0_tb.reset stepper2_0_tb.warnbit
Created probe 1
ncsim> run
control signal = 2, reset = 0, warning bit = x, current state =  x
Control is no longer 11
control signal = 2, reset = 0, warning bit = 0, current state =  5
control signal = 2, reset = 0, warning bit = 0, current state =  1
control signal = 2, reset = 0, warning bit = 0, current state =  9
control signal = 2, reset = 0, warning bit = 0, current state =  8
control signal = 2, reset = 0, warning bit = 0, current state = 10
control signal = 2, reset = 0, warning bit = 0, current state =  2
control signal = 2, reset = 0, warning bit = 0, current state =  6
control signal = 0, reset = 0, warning bit = 0, current state =  4
control signal = 1, reset = 0, warning bit = 0, current state =  4
control signal = 1, reset = 0, warning bit = 0, current state =  6
control signal = 1, reset = 0, warning bit = 0, current state =  2
control signal = 1, reset = 0, warning bit = 0, current state = 10
control signal = 1, reset = 0, warning bit = 0, current state =  8
control signal = 1, reset = 0, warning bit = 0, current state =  9
control signal = 1, reset = 0, warning bit = 0, current state =  1
control signal = 1, reset = 0, warning bit = 0, current state =  5
control signal = 1, reset = 1, warning bit = 0, current state =  4
Test warning bit
control signal = 3, reset = 0, warning bit = 0, current state =  5
Warning, control = 11
control signal = 3, reset = 0, warning bit = 1, current state =  5
control signal = 3, reset = 0, warning bit = 1, current state =  5
control signal = 3, reset = 0, warning bit = 1, current state =  5
control signal = 0, reset = 0, warning bit = 1, current state =  5
Control is no longer 11
control signal = 1, reset = 0, warning bit = 0, current state =  5
control signal = 1, reset = 0, warning bit = 0, current state =  4
control signal = 1, reset = 0, warning bit = 0, current state =  6
control signal = 1, reset = 0, warning bit = 0, current state =  2
control signal = 1, reset = 0, warning bit = 0, current state = 10
control signal = 1, reset = 0, warning bit = 0, current state =  8
control signal = 1, reset = 0, warning bit = 0, current state =  9
control signal = 1, reset = 0, warning bit = 0, current state =  1
control signal = 0, reset = 0, warning bit = 0, current state =  5
Simulation complete via $finish(1) at time 650 NS + 0
./lab2/stepper2_0_tb.v:50 $finish;
ncsim>
```

Code:

```
`timescale 1ns/1ns
module stepper2_0_tb;
    reg clk,reset;
    reg[1:0] control;
    wire warnbit;
    wire [3:0] curr_state;
    reg[3:0]  counter;

    stepper1 s1(.clk(clk), .control(control), .reset(reset),
.warnbit(warnbit), .curr_state(curr_state));

    always
        #10 clk= ~clk;

    initial
    begin
        counter = 3'd0;
            clk = 1'b0;
            reset = 1'b0;
        /*
        for(counter = 3'd0; counter < 3'd7; counter = counter + 3'd1)
        begin
        control = 2'h2;
        $display(curr_state);
        end
        */
        control = 2'h2;
        #160
        control = 2'h0;
        #20
        control = 2'h1;

        #160

        reset = 1'b1;
        #29
        reset =1'b0;
        #1
        $display("Test warning bit");
        control = 2'h3;
        #80
        control = 2'h0;
        #20
        control = 2'h1;
        #160
        control = 2'h0;
        #20
        $finish;
    end

    always@(posedge clk)
    begin
     $display("control signal = %d, reset = %b, warning bit = %b, current
state = %d",
              control,reset,warnbit, curr_state);
    end

    always@(warnbit)
    begin
        if(warnbit == 1'b1)
        $display("Warning, control = 11");
        else
        $display("Control is no longer 11");
    end
endmodule
```