University of Nevada Las Vegas. Department of Electrical and Computer Engineering Laboratories.

Class:	CPE300L	Semester:	Fall 2020
Points		Document author:	Jerrod Batu
		Author's email:	batuj1@unlv.nevada.edu
		Document topic:	Postlab 5
Instructor's comments:			

1. Introduction / Theory of Operation

Throughout this lab I will continue my knowledge on finite state machines (FSMs) and their implementation in FPGA.

1.
   The two types of FSMs are Moore and Mealy. The key difference between these two types of FSMs is how the states lead to the output. **Moore Fsms** have the output dependent only on its states rather than on the inputs; therefore, the output only changes when the state goes to the next or previous state. **Mealy Fsms** have the output dependent on both the state and the inputs; therefore, the outputs are asynchronous throughout each state change. When using a Moore FSM, the best way to model the state diagram is to use an always and case statement where the always statement would initialize the clock on every positive edge, and the case statement would dictate how the states would change and the output logic for each state. When using a Mealy FSM, the best way to model the state diagram is to, in fact, use two always cases where one always case would describe the state transitions, and the other always case were to describe the combinational logic between the next state and the output.
2. **When I approach designing an FSM,** I always think about how the states relate to the desired output. When the problem prompts for a specific locking mechanism, I know it would be best to use a Mealy FSM because the output depends on the input and where the state is at for this specific input. When the problem prompts for a specific sequence of events, such as a traffic light, I prefer Moore FSMs because the output would only be dependent on the state. FSMs have always been easier to understand to me because they model real life examples.

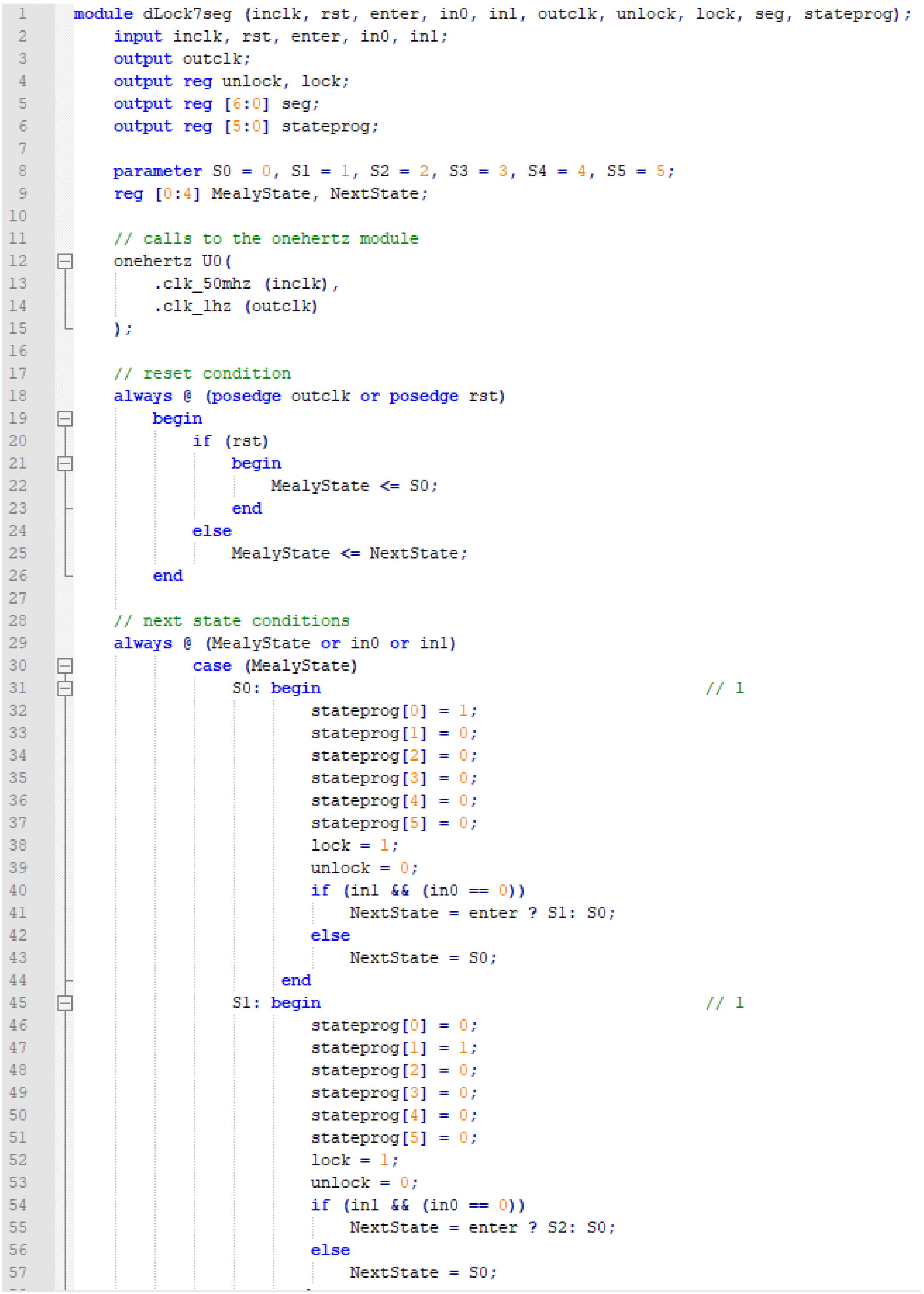
2. Prelab

https://docs.google.com/document/d/1UAKqNVVO6aogDywWX3oxZrXacp6nB52l/edit?usp=sharing&ouid=102808507017671072128&rtpof=true&sd=true

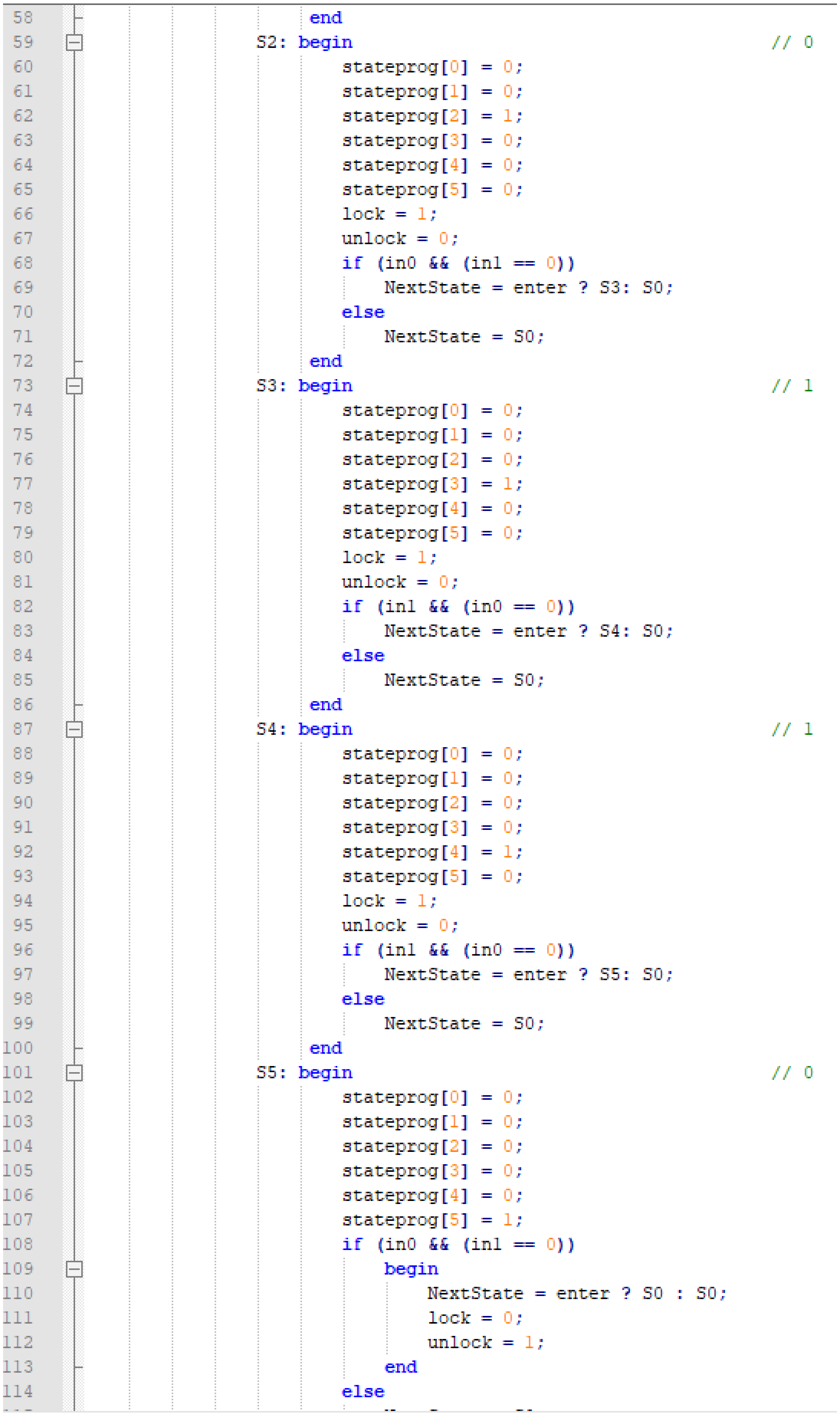
This is the link to my prelab5.

3. Results of Experiments

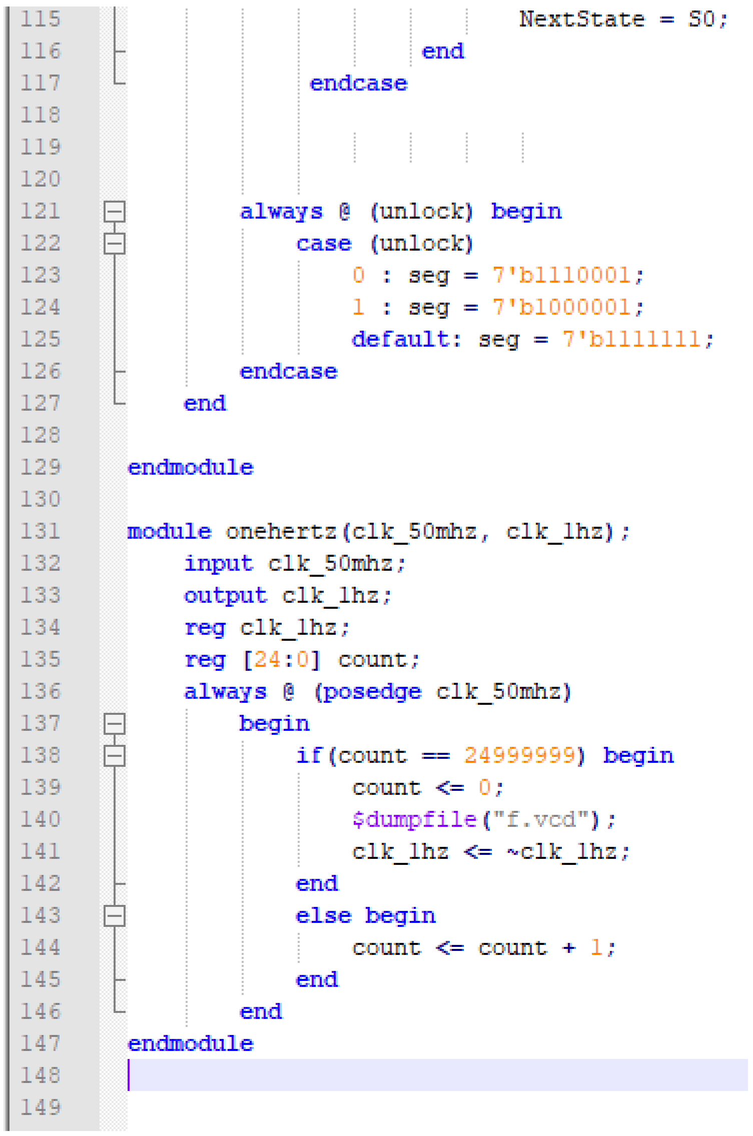
Experiment 1



```verilog
module dLock7seg (inclk, rst, enter, in0, in1, outclk, unlock, lock, seg, stateprog);
    input inclk, rst, enter, in0, in1;
    output outclk;
    output reg unlock, lock;
    output reg [6:0] seg;
    output reg [5:0] stateprog;

    parameter S0 = 0, S1 = 1, S2 = 2, S3 = 3, S4 = 4, S5 = 5;
    reg [0:4] MealyState, NextState;

    // calls to the onehertz module
    onehertz U0(
        .clk_50mhz (inclk),
        .clk_1hz (outclk)
    );

    // reset condition
    always @ (posedge outclk or posedge rst)
        begin
            if (rst)
                begin
                    MealyState <= S0;
                end
            else
                MealyState <= NextState;
        end

    // next state conditions
    always @ (MealyState or in0 or in1)
            case (MealyState)
                S0: begin                                // 1
                        stateprog[0] = 1;
                        stateprog[1] = 0;
                        stateprog[2] = 0;
                        stateprog[3] = 0;
                        stateprog[4] = 0;
                        stateprog[5] = 0;
                        lock = 1;
                        unlock = 0;
                        if (in1 && (in0 == 0))
                            NextState = enter ? S1: S0;
                        else
                            NextState = S0;
                     end
                S1: begin                                // 1
                        stateprog[0] = 0;
                        stateprog[1] = 1;
                        stateprog[2] = 0;
                        stateprog[3] = 0;
                        stateprog[4] = 0;
                        stateprog[5] = 0;
                        lock = 1;
                        unlock = 0;
                        if (in1 && (in0 == 0))
                            NextState = enter ? S2: S0;
                        else
                            NextState = S0;
```

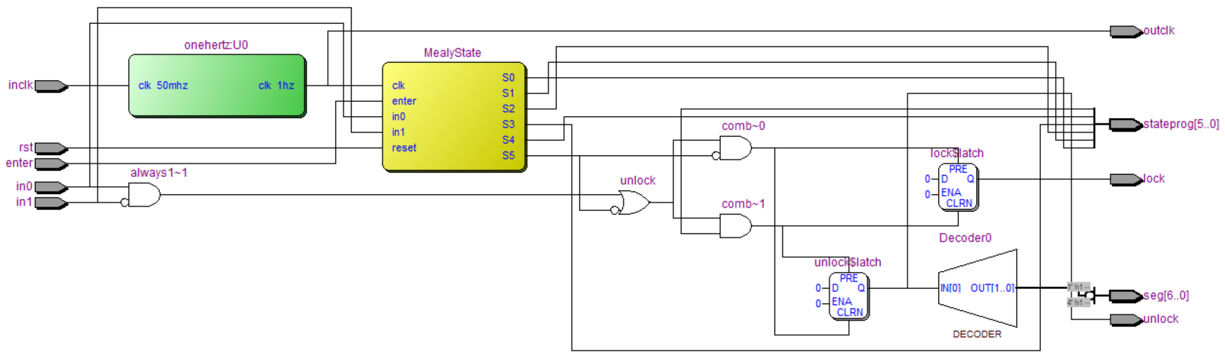


```
                    end
                S2: begin                                           // 0
                        stateprog[0] = 0;
                        stateprog[1] = 0;
                        stateprog[2] = 1;
                        stateprog[3] = 0;
                        stateprog[4] = 0;
                        stateprog[5] = 0;
                        lock = 1;
                        unlock = 0;
                        if (in0 && (in1 == 0))
                            NextState = enter ? S3: S0;
                        else
                            NextState = S0;
                    end
                S3: begin                                           // 1
                        stateprog[0] = 0;
                        stateprog[1] = 0;
                        stateprog[2] = 0;
                        stateprog[3] = 1;
                        stateprog[4] = 0;
                        stateprog[5] = 0;
                        lock = 1;
                        unlock = 0;
                        if (in1 && (in0 == 0))
                            NextState = enter ? S4: S0;
                        else
                            NextState = S0;
                    end
                S4: begin                                           // 1
                        stateprog[0] = 0;
                        stateprog[1] = 0;
                        stateprog[2] = 0;
                        stateprog[3] = 0;
                        stateprog[4] = 1;
                        stateprog[5] = 0;
                        lock = 1;
                        unlock = 0;
                        if (in1 && (in0 == 0))
                            NextState = enter ? S5: S0;
                        else
                            NextState = S0;
                    end
                S5: begin                                           // 0
                        stateprog[0] = 0;
                        stateprog[1] = 0;
                        stateprog[2] = 0;
                        stateprog[3] = 0;
                        stateprog[4] = 0;
                        stateprog[5] = 1;
                        if (in0 && (in1 == 0))
                            begin
                                NextState = enter ? S0 : S0;
                                lock = 0;
                                unlock = 1;
                            end
                        else
```

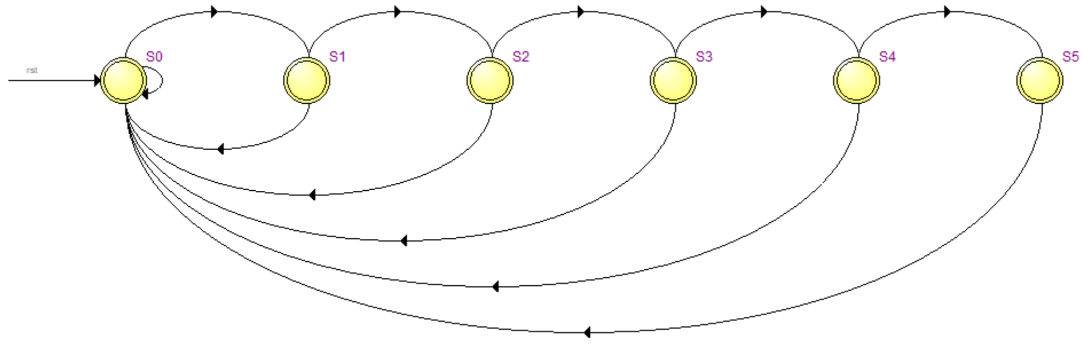


```
                            NextState = S0;
                    end
            endcase



        always @ (unlock) begin
            case (unlock)
                0 : seg = 7'b1110001;
                1 : seg = 7'b1000001;
                default: seg = 7'b1111111;
        endcase
    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
```

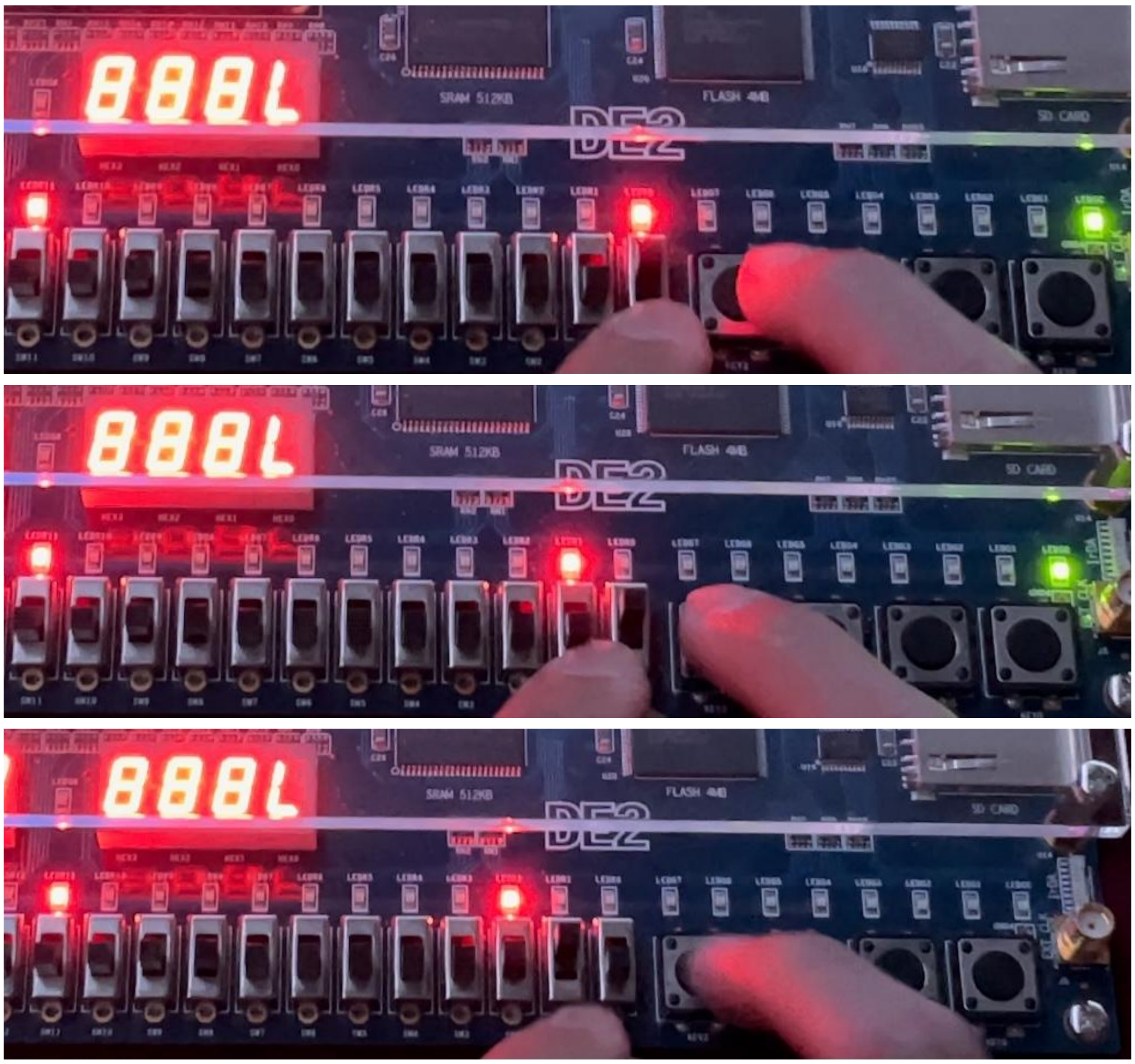
This is my FSM Verilog Code for the digital lock.

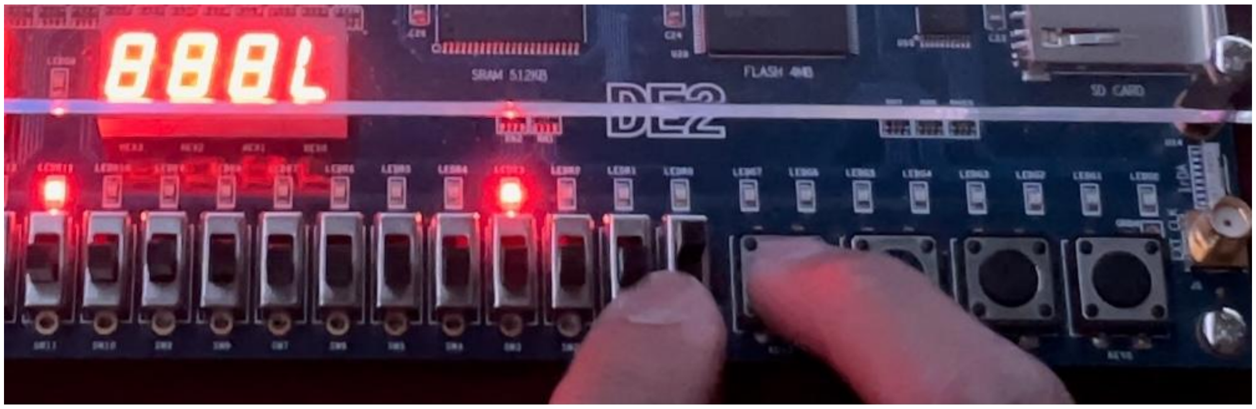


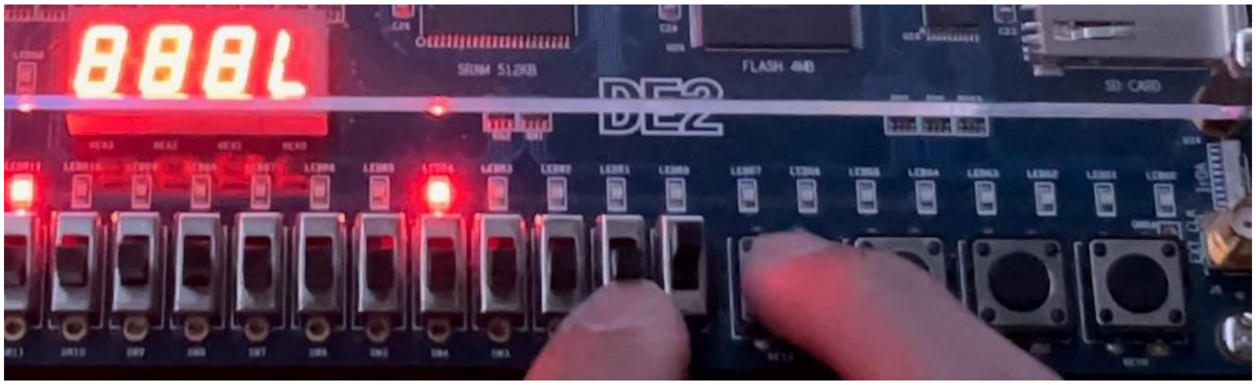
This is my RTL View for the FSM Verilog Code for the digital lock.

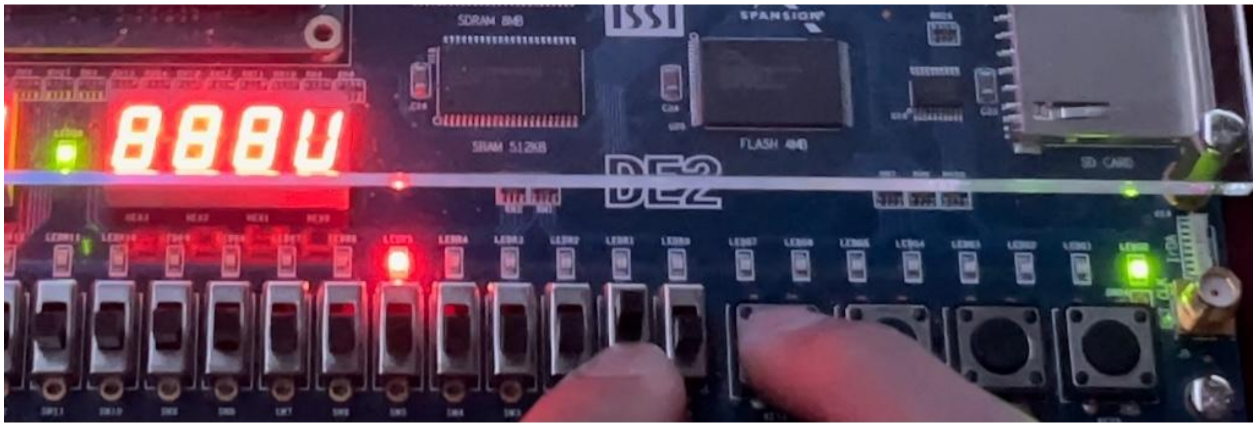


This is my State Machine Diagram View for the FSM Verilog Code for the digital lock.





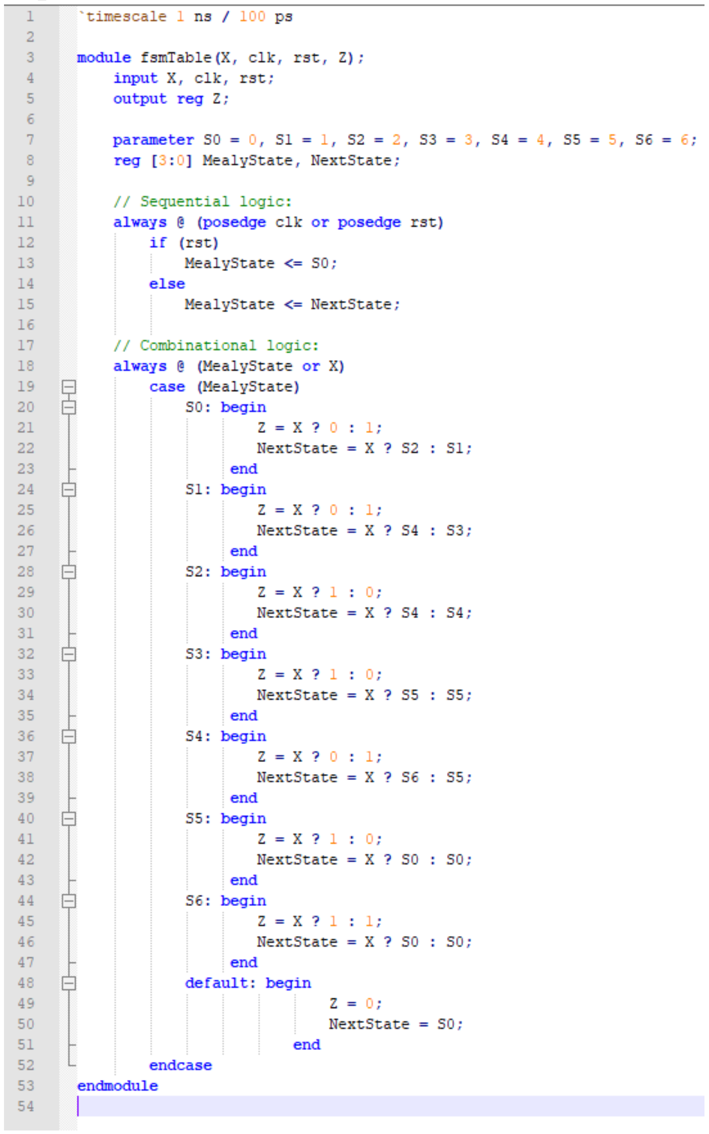




These are my images of my digital combinational lock implemented onto the DE2 board. The combination is 110110, and LED11 will be red as well as the 7-segment display showing “L” when the combinational lock is “locked.” The first two images are when A = 1. The third image is when A = 0. The fourth and fifth images are when A = 1 again. Lastly, the sixth image is when A = 0; thus, the output being shown with a green LED and a U displayed for “unlock.”

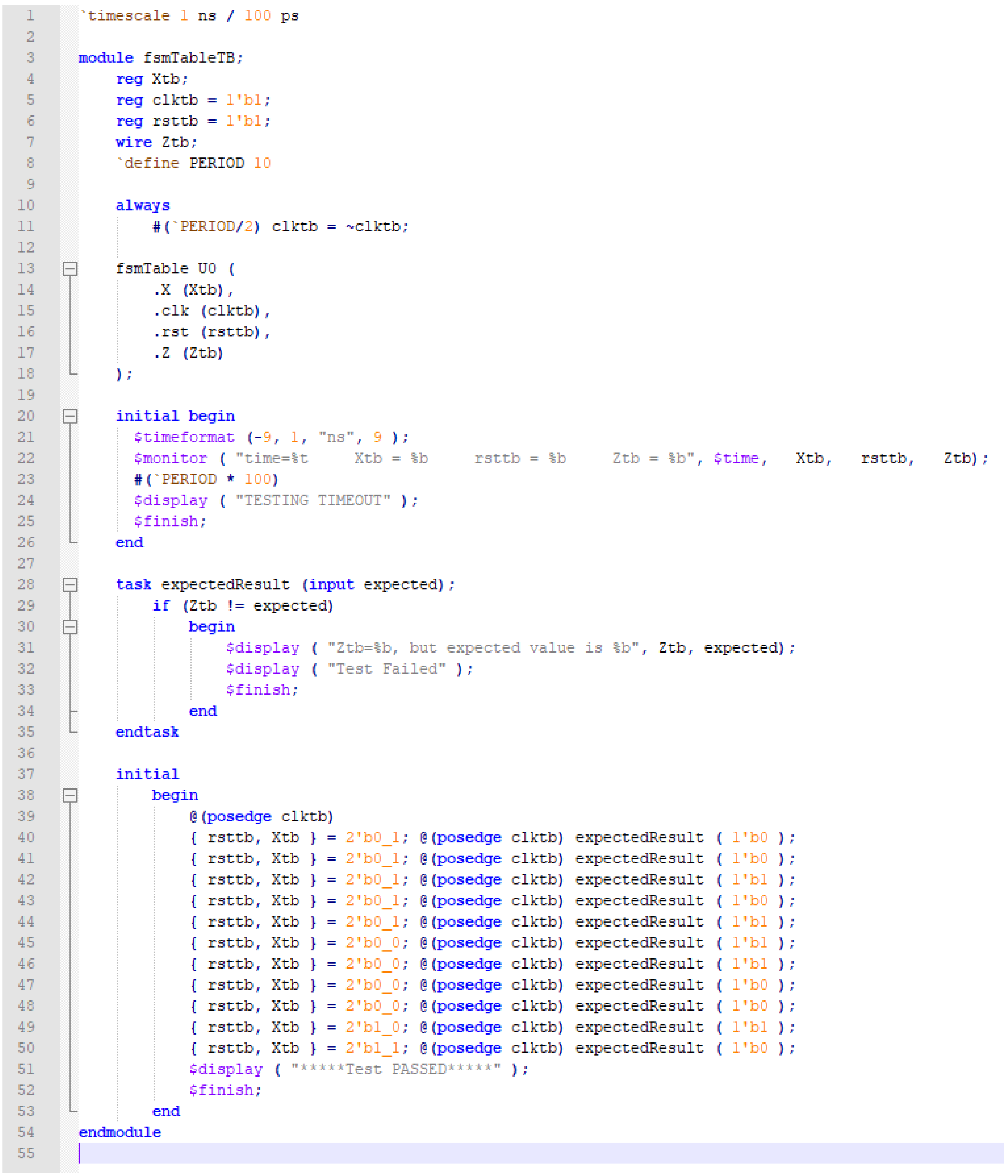
https://drive.google.com/file/d/1cRAC0FEcE2Qc9kZp06RF-yyTNQbC-wVg/view?usp=sharing
This is the link to my video describing the digital combinational lock. Included within the zip file will also be the same video.

Experiment 2



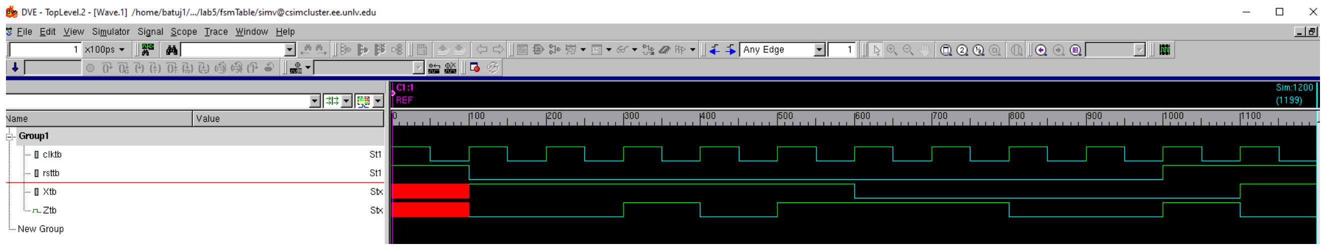
```verilog
`timescale 1 ns / 100 ps

module fsmTable(X, clk, rst, Z);
    input X, clk, rst;
    output reg Z;

    parameter S0 = 0, S1 = 1, S2 = 2, S3 = 3, S4 = 4, S5 = 5, S6 = 6;
    reg [3:0] MealyState, NextState;

    // Sequential logic:
    always @ (posedge clk or posedge rst)
        if (rst)
            MealyState <= S0;
        else
            MealyState <= NextState;

    // Combinational logic:
    always @ (MealyState or X)
        case (MealyState)
            S0: begin
                    Z = X ? 0 : 1;
                    NextState = X ? S2 : S1;
                end
            S1: begin
                    Z = X ? 0 : 1;
                    NextState = X ? S4 : S3;
                end
            S2: begin
                    Z = X ? 1 : 0;
                    NextState = X ? S4 : S4;
                end
            S3: begin
                    Z = X ? 1 : 0;
                    NextState = X ? S5 : S5;
                end
            S4: begin
                    Z = X ? 0 : 1;
                    NextState = X ? S6 : S5;
                end
            S5: begin
                    Z = X ? 1 : 0;
                    NextState = X ? S0 : S0;
                end
            S6: begin
                    Z = X ? 1 : 1;
                    NextState = X ? S0 : S0;
                end
            default: begin
                        Z = 0;
                        NextState = S0;
                    end
        endcase
endmodule
```

This is my Verilog code for the FSM machine table.

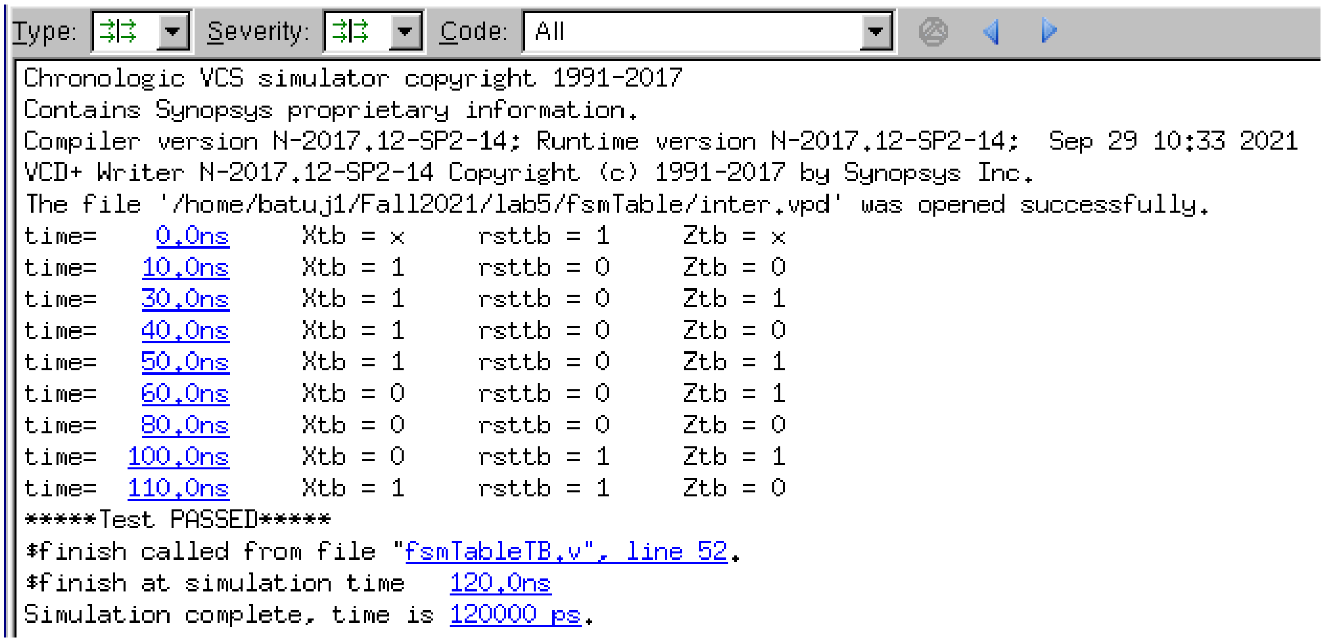


```verilog
`timescale 1 ns / 100 ps

module fsmTableTB;
    reg Xtb;
    reg clktb = 1'b1;
    reg rsttb = 1'b1;
    wire Ztb;
    `define PERIOD 10

    always
        #(`PERIOD/2) clktb = ~clktb;

    fsmTable U0 (
        .X (Xtb),
        .clk (clktb),
        .rst (rsttb),
        .Z (Ztb)
    );

    initial begin
      $timeformat (-9, 1, "ns", 9 );
      $monitor ( "time=%t     Xtb = %b     rsttb = %b     Ztb = %b", $time,   Xtb,   rsttb,   Ztb);
      #(`PERIOD * 100)
      $display ( "TESTING TIMEOUT" );
      $finish;
    end

    task expectedResult (input expected);
        if (Ztb != expected)
            begin
                $display ( "Ztb=%b, but expected value is %b", Ztb, expected);
                $display ( "Test Failed" );
                $finish;
            end
    endtask

    initial
        begin
            @(posedge clktb)
            { rsttb, Xtb } = 2'b0_1; @(posedge clktb) expectedResult ( 1'b0 );
            { rsttb, Xtb } = 2'b0_1; @(posedge clktb) expectedResult ( 1'b0 );
            { rsttb, Xtb } = 2'b0_1; @(posedge clktb) expectedResult ( 1'b1 );
            { rsttb, Xtb } = 2'b0_1; @(posedge clktb) expectedResult ( 1'b0 );
            { rsttb, Xtb } = 2'b0_1; @(posedge clktb) expectedResult ( 1'b1 );
            { rsttb, Xtb } = 2'b0_0; @(posedge clktb) expectedResult ( 1'b1 );
            { rsttb, Xtb } = 2'b0_0; @(posedge clktb) expectedResult ( 1'b1 );
            { rsttb, Xtb } = 2'b0_0; @(posedge clktb) expectedResult ( 1'b0 );
            { rsttb, Xtb } = 2'b0_0; @(posedge clktb) expectedResult ( 1'b0 );
            { rsttb, Xtb } = 2'b1_0; @(posedge clktb) expectedResult ( 1'b1 );
            { rsttb, Xtb } = 2'b1_1; @(posedge clktb) expectedResult ( 1'b0 );
            $display ( "*****Test PASSED*****" );
            $finish;
        end
endmodule
```

This is my testbench Verilog code for the FSM machine table.


DVE - TopLevel.2 - [Wave.1] /home/batuj1/.../lab5/fsmTable/simv@csimcluster.ee.unlv.edu
File Edit View Simulator Signal Scope Trace Window Help
x100ps
Any Edge
C1:1
REF
Sim:1200
(1199)
Name
Value
Group1
clktb St1
rsttb St1
Xtb StX
Ztb StX
New Group
100
200
300
400
500
600
700
800
900
1000
1100

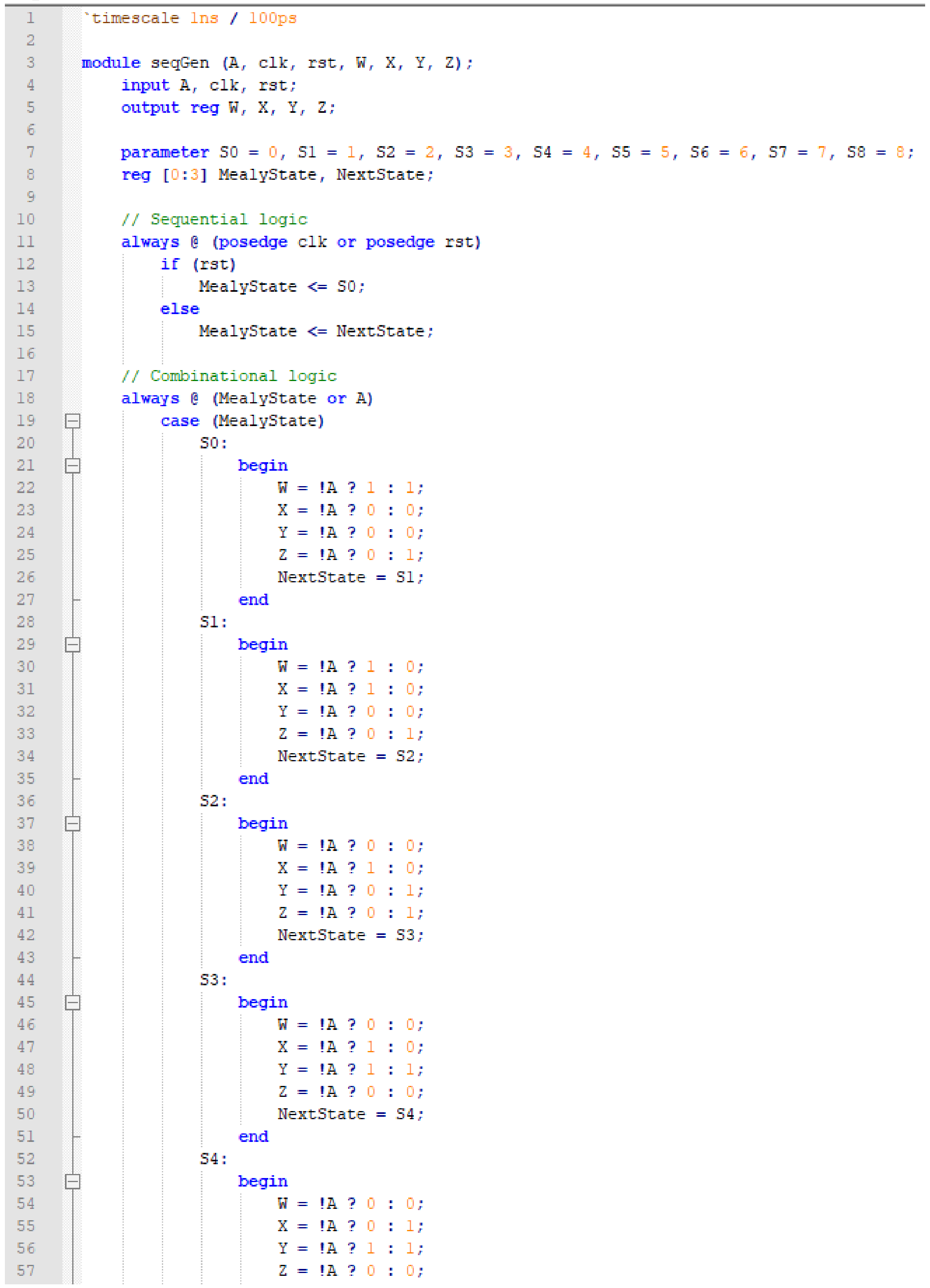
These are my VCS waveforms for the FSM machine table. The red blocks represent delay.



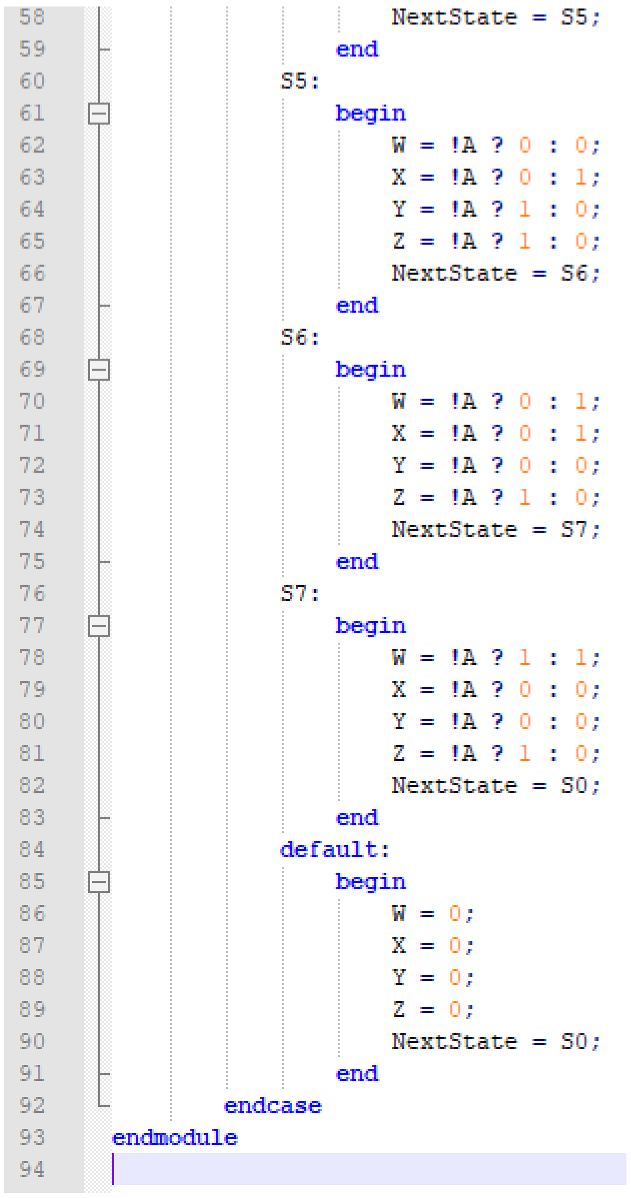
```
Chronologic VCS simulator copyright 1991-2017
Contains Synopsys proprietary information.
Compiler version N-2017.12-SP2-14; Runtime version N-2017.12-SP2-14;  Sep 29 10:33 2021
VCD+ Writer N-2017.12-SP2-14 Copyright (c) 1991-2017 by Synopsys Inc.
The file '/home/batuj1/Fall2021/lab5/fsmTable/inter.vpd' was opened successfully.
time=    0.0ns    Xtb = x     rsttb = 1     Ztb = x
time=   10.0ns    Xtb = 1     rsttb = 0     Ztb = 0
time=   30.0ns    Xtb = 1     rsttb = 0     Ztb = 1
time=   40.0ns    Xtb = 1     rsttb = 0     Ztb = 0
time=   50.0ns    Xtb = 1     rsttb = 0     Ztb = 1
time=   60.0ns    Xtb = 0     rsttb = 0     Ztb = 1
time=   80.0ns    Xtb = 0     rsttb = 0     Ztb = 0
time=  100.0ns    Xtb = 0     rsttb = 1     Ztb = 1
time=  110.0ns    Xtb = 1     rsttb = 1     Ztb = 0
*****Test PASSED*****
$finish called from file "fsmTableTB.v", line 52.
$finish at simulation time   120.0ns
Simulation complete, time is 120000 ps.
```

This is my VCS console for the FSM machine table.

Experiment 3

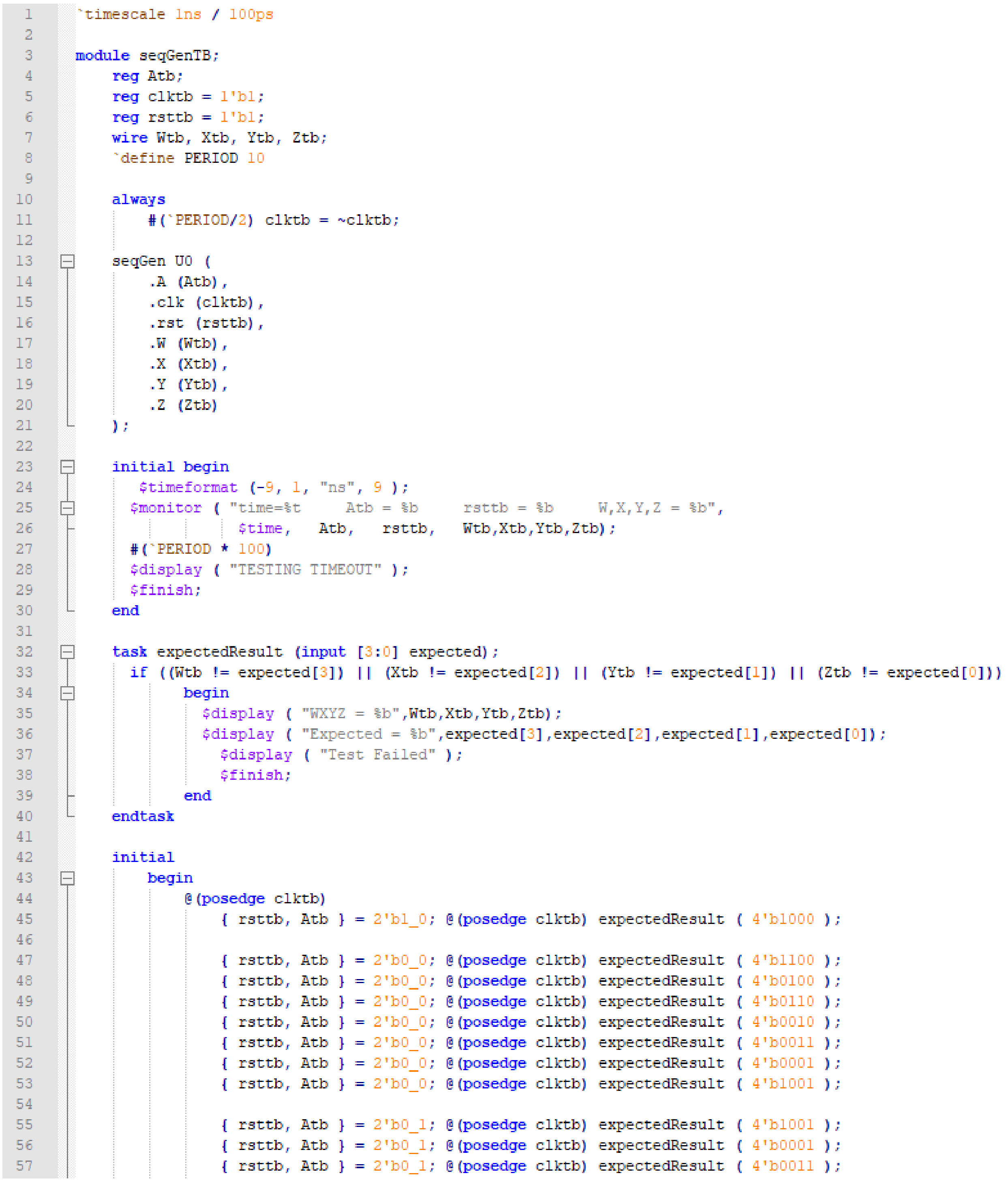


```verilog
`timescale 1ns / 100ps

module seqGen (A, clk, rst, W, X, Y, Z);
    input A, clk, rst;
    output reg W, X, Y, Z;

    parameter S0 = 0, S1 = 1, S2 = 2, S3 = 3, S4 = 4, S5 = 5, S6 = 6, S7 = 7, S8 = 8;
    reg [0:3] MealyState, NextState;

    // Sequential logic
    always @ (posedge clk or posedge rst)
        if (rst)
            MealyState <= S0;
        else
            MealyState <= NextState;

    // Combinational logic
    always @ (MealyState or A)
        case (MealyState)
            S0:
                begin
                    W = !A ? 1 : 1;
                    X = !A ? 0 : 0;
                    Y = !A ? 0 : 0;
                    Z = !A ? 0 : 1;
                    NextState = S1;
                end
            S1:
                begin
                    W = !A ? 1 : 0;
                    X = !A ? 1 : 0;
                    Y = !A ? 0 : 0;
                    Z = !A ? 0 : 1;
                    NextState = S2;
                end
            S2:
                begin
                    W = !A ? 0 : 0;
                    X = !A ? 1 : 0;
                    Y = !A ? 0 : 1;
                    Z = !A ? 0 : 1;
                    NextState = S3;
                end
            S3:
                begin
                    W = !A ? 0 : 0;
                    X = !A ? 1 : 0;
                    Y = !A ? 1 : 1;
                    Z = !A ? 0 : 0;
                    NextState = S4;
                end
            S4:
                begin
                    W = !A ? 0 : 0;
                    X = !A ? 0 : 1;
                    Y = !A ? 1 : 1;
                    Z = !A ? 0 : 0;
```

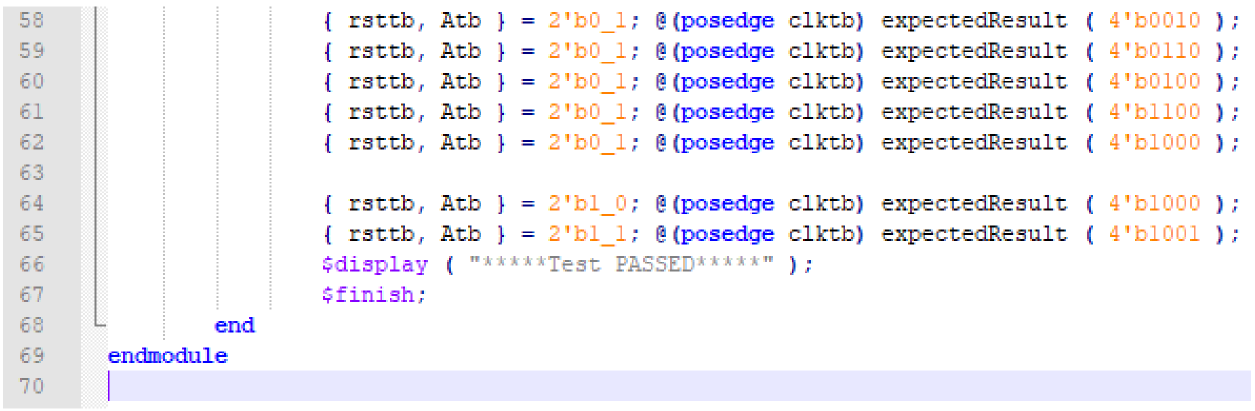


```
                NextState = S5;
            end
        S5:
            begin
                W = !A ? 0 : 0;
                X = !A ? 0 : 1;
                Y = !A ? 1 : 0;
                Z = !A ? 1 : 0;
                NextState = S6;
            end
        S6:
            begin
                W = !A ? 0 : 1;
                X = !A ? 0 : 1;
                Y = !A ? 0 : 0;
                Z = !A ? 1 : 0;
                NextState = S7;
            end
        S7:
            begin
                W = !A ? 1 : 1;
                X = !A ? 0 : 0;
                Y = !A ? 0 : 0;
                Z = !A ? 1 : 0;
                NextState = S0;
            end
        default:
            begin
                W = 0;
                X = 0;
                Y = 0;
                Z = 0;
                NextState = S0;
            end
    endcase
endmodule
```

This is my Verilog code for an FSM that models a sequence generator.

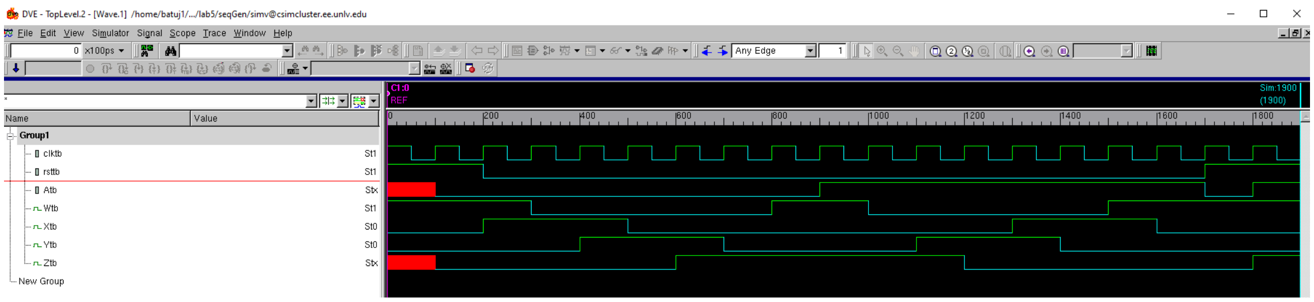


```verilog
`timescale 1ns / 100ps

module seqGenTB;
    reg Atb;
    reg clktb = 1'b1;
    reg rsttb = 1'b1;
    wire Wtb, Xtb, Ytb, Ztb;
    `define PERIOD 10

    always
        #(`PERIOD/2) clktb = ~clktb;

    seqGen U0 (
        .A (Atb),
        .clk (clktb),
        .rst (rsttb),
        .W (Wtb),
        .X (Xtb),
        .Y (Ytb),
        .Z (Ztb)
    );

    initial begin
       $timeformat (-9, 1, "ns", 9 );
      $monitor ( "time=%t     Atb = %b     rsttb = %b     W,X,Y,Z = %b",
                 $time,   Atb,   rsttb,   Wtb,Xtb,Ytb,Ztb);
      #(`PERIOD * 100)
      $display ( "TESTING TIMEOUT" );
      $finish;
    end

    task expectedResult (input [3:0] expected);
      if ((Wtb != expected[3]) || (Xtb != expected[2]) || (Ytb != expected[1]) || (Ztb != expected[0]))
            begin
              $display ( "WXYZ = %b",Wtb,Xtb,Ytb,Ztb);
              $display ( "Expected = %b",expected[3],expected[2],expected[1],expected[0]);
                $display ( "Test Failed" );
                $finish;
            end
    endtask

    initial
        begin
            @(posedge clktb)
                { rsttb, Atb } = 2'b1_0; @(posedge clktb) expectedResult ( 4'b1000 );

                { rsttb, Atb } = 2'b0_0; @(posedge clktb) expectedResult ( 4'b1100 );
                { rsttb, Atb } = 2'b0_0; @(posedge clktb) expectedResult ( 4'b0100 );
                { rsttb, Atb } = 2'b0_0; @(posedge clktb) expectedResult ( 4'b0110 );
                { rsttb, Atb } = 2'b0_0; @(posedge clktb) expectedResult ( 4'b0010 );
                { rsttb, Atb } = 2'b0_0; @(posedge clktb) expectedResult ( 4'b0011 );
                { rsttb, Atb } = 2'b0_0; @(posedge clktb) expectedResult ( 4'b0001 );
                { rsttb, Atb } = 2'b0_0; @(posedge clktb) expectedResult ( 4'b1001 );

                { rsttb, Atb } = 2'b0_1; @(posedge clktb) expectedResult ( 4'b1001 );
                { rsttb, Atb } = 2'b0_1; @(posedge clktb) expectedResult ( 4'b0001 );
                { rsttb, Atb } = 2'b0_1; @(posedge clktb) expectedResult ( 4'b0011 );
```

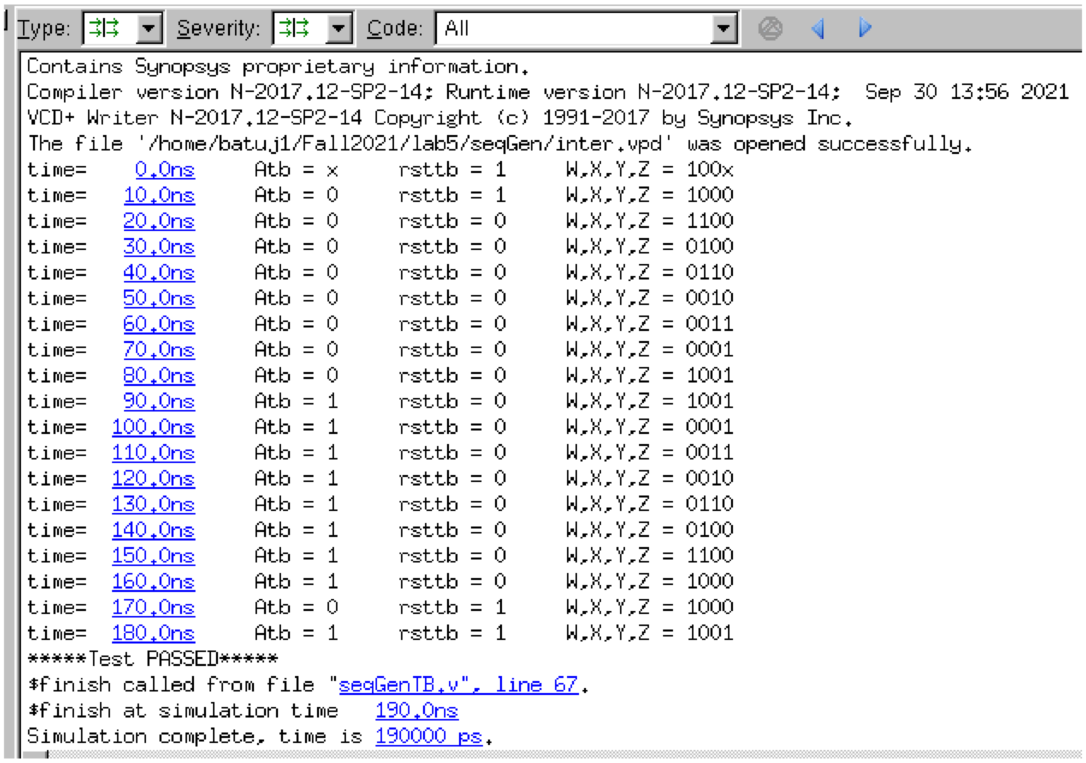


```
				{ rsttb, Atb } = 2'b0_1; @(posedge clktb) expectedResult ( 4'b0010 );
				{ rsttb, Atb } = 2'b0_1; @(posedge clktb) expectedResult ( 4'b0110 );
				{ rsttb, Atb } = 2'b0_1; @(posedge clktb) expectedResult ( 4'b0100 );
				{ rsttb, Atb } = 2'b0_1; @(posedge clktb) expectedResult ( 4'b1100 );
				{ rsttb, Atb } = 2'b0_1; @(posedge clktb) expectedResult ( 4'b1000 );

				{ rsttb, Atb } = 2'b1_0; @(posedge clktb) expectedResult ( 4'b1000 );
				{ rsttb, Atb } = 2'b1_1; @(posedge clktb) expectedResult ( 4'b1001 );
				$display ( "*****Test PASSED*****" );
				$finish;
		end
endmodule
```

This is my testbench Verilog code for an FSM that models a sequence generator.

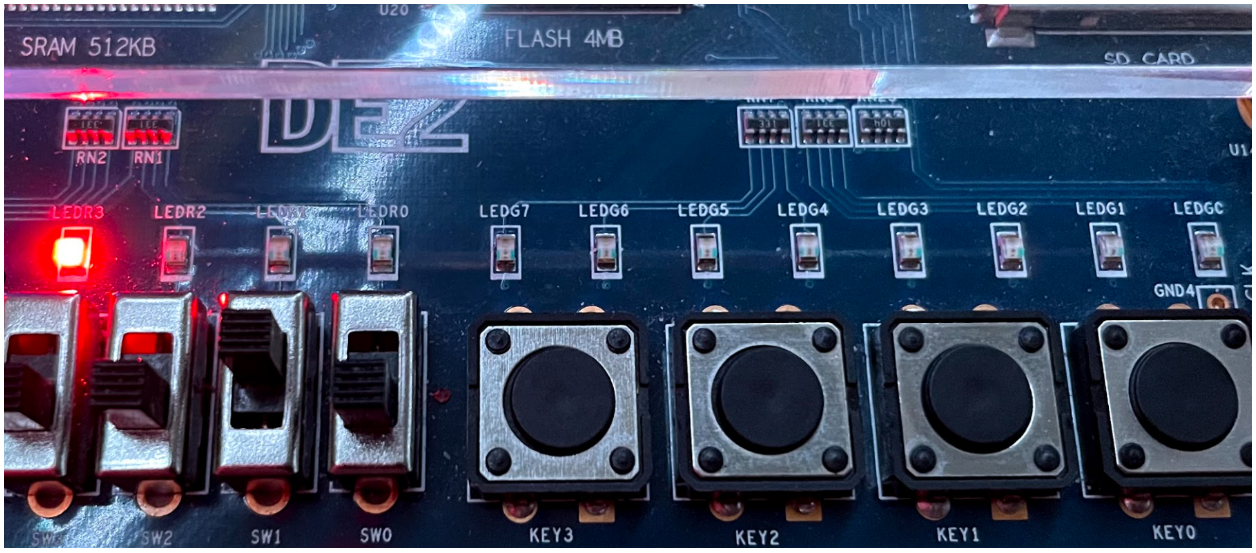

DVE - TopLevel.2 - [Wave.1] /home/batuj1/.../lab5/seqGen/simv@csimcluster.ee.unlv.edu
File Edit View Simulator Signal Scope Trace Window Help
Name
Value
Group1
clktb St1
rsttb St1
Atb StX
Wtb St1
Xtb St0
Ytb St0
Ztb StX
New Group
C1:0
REF
Sim:1900
(1900)
200
400
600
800
1000
1200
1400
1600
1800

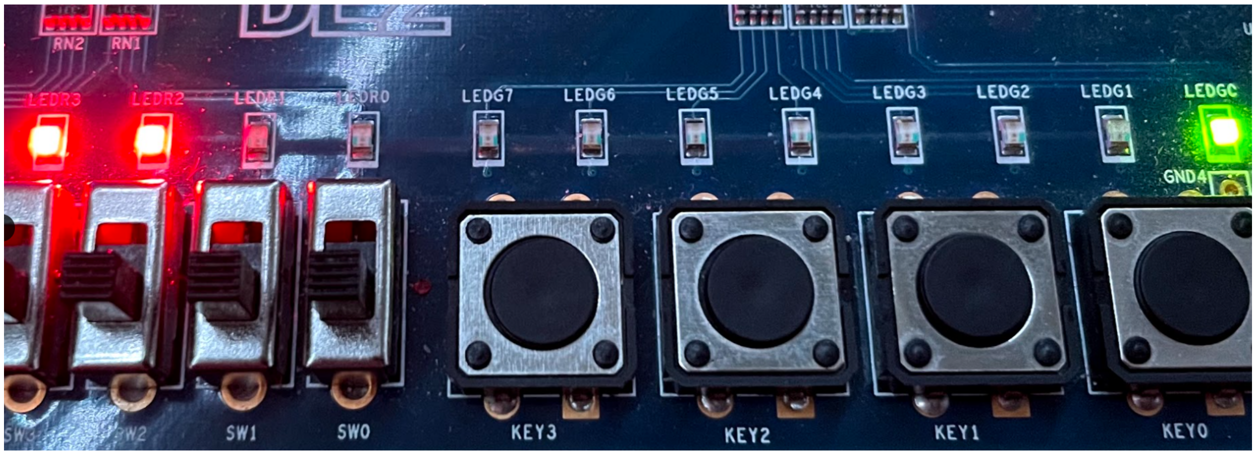
These are my VCS waveforms for an FSM that models a sequence generator. The red blocks indicate delay.

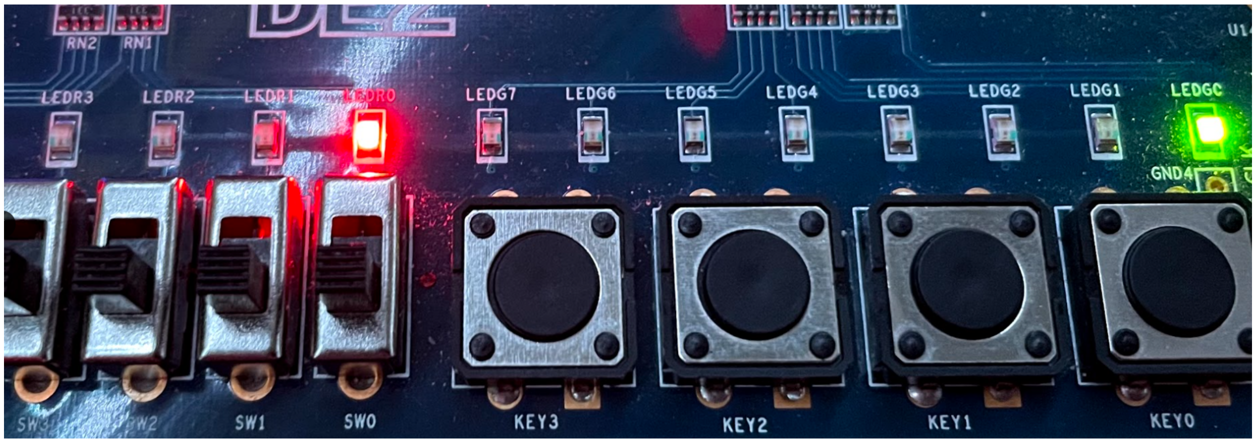


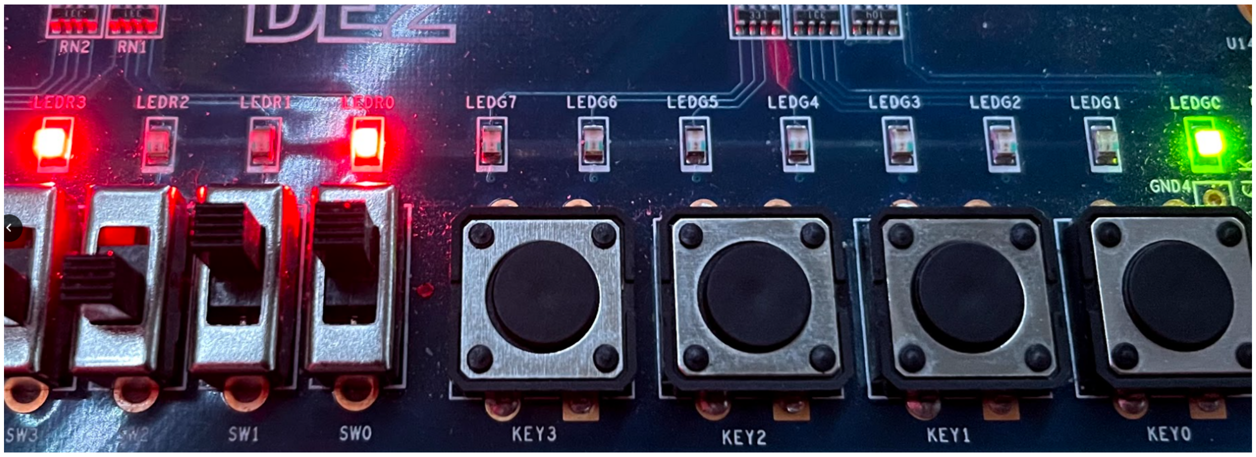
```
Contains Synopsys proprietary information.
Compiler version N-2017.12-SP2-14; Runtime version N-2017.12-SP2-14;  Sep 30 13:56 2021
VCD+ Writer N-2017.12-SP2-14 Copyright (c) 1991-2017 by Synopsys Inc.
The file '/home/batuj1/Fall2021/lab5/seqGen/inter.vpd' was opened successfully.
time=   0.0ns     Atb = x    rsttb = 1     W,X,Y,Z = 100x
time=  10.0ns     Atb = 0    rsttb = 1     W,X,Y,Z = 1000
time=  20.0ns     Atb = 0    rsttb = 0     W,X,Y,Z = 1100
time=  30.0ns     Atb = 0    rsttb = 0     W,X,Y,Z = 0100
time=  40.0ns     Atb = 0    rsttb = 0     W,X,Y,Z = 0110
time=  50.0ns     Atb = 0    rsttb = 0     W,X,Y,Z = 0010
time=  60.0ns     Atb = 0    rsttb = 0     W,X,Y,Z = 0011
time=  70.0ns     Atb = 0    rsttb = 0     W,X,Y,Z = 0001
time=  80.0ns     Atb = 0    rsttb = 0     W,X,Y,Z = 1001
time=  90.0ns     Atb = 1    rsttb = 0     W,X,Y,Z = 1001
time= 100.0ns     Atb = 1    rsttb = 0     W,X,Y,Z = 0001
time= 110.0ns     Atb = 1    rsttb = 0     W,X,Y,Z = 0011
time= 120.0ns     Atb = 1    rsttb = 0     W,X,Y,Z = 0010
time= 130.0ns     Atb = 1    rsttb = 0     W,X,Y,Z = 0110
time= 140.0ns     Atb = 1    rsttb = 0     W,X,Y,Z = 0100
time= 150.0ns     Atb = 1    rsttb = 0     W,X,Y,Z = 1100
time= 160.0ns     Atb = 1    rsttb = 0     W,X,Y,Z = 1000
time= 170.0ns     Atb = 0    rsttb = 1     W,X,Y,Z = 1000
time= 180.0ns     Atb = 1    rsttb = 1     W,X,Y,Z = 1001
*****Test PASSED*****
$finish called from file "seqGenTB.v", line 67.
$finish at simulation time   190.0ns
Simulation complete, time is 190000 ps.
```

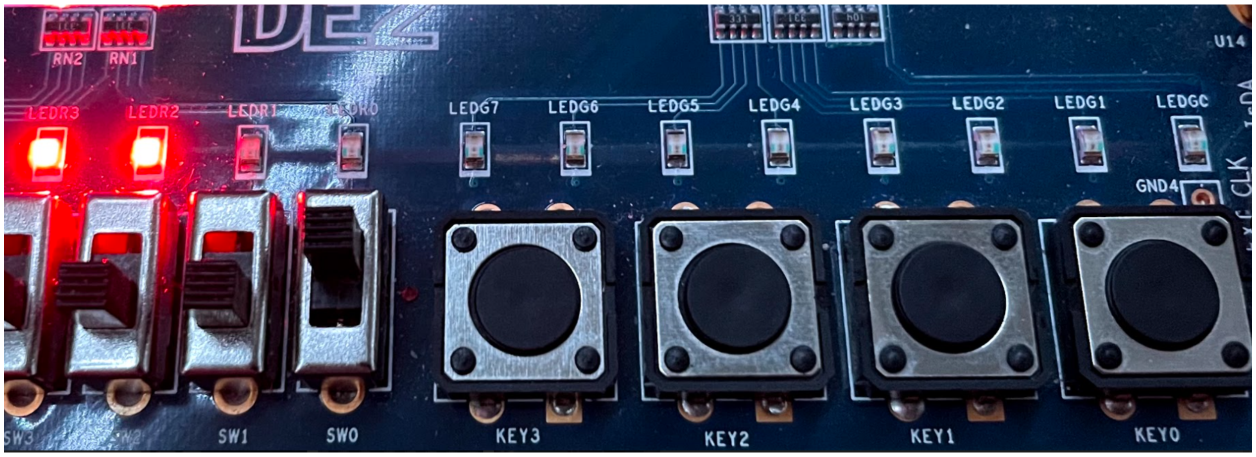
This is my VCS console for an FSM that models a sequence generator.

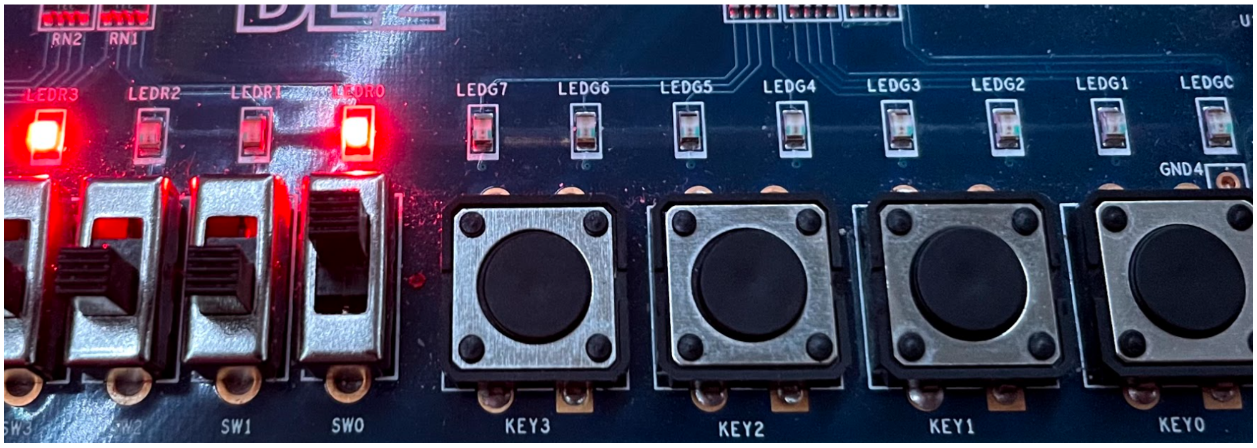

SRAM 512KB
FLASH 4MB
SD CARD
DE2
RN2
RN1
LEDR3
LEDR2
LEDR0
LEDG7
LEDG6
LEDG5
LEDG4
LEDG3
LEDG2
LEDG1
LEDG0
GND4
SW2
SW1
SW0
KEY3
KEY2
KEY1
KEY0


RN2
RN1
LEDR3
LEDR2
LEDR1
LEDR0
LEDG7
LEDG6
LEDG5
LEDG4
LEDG3
LEDG2
LEDG1
LEDG0
GND4
SW1
SW0
KEY3
KEY2
KEY1
KEY0


RN2
RN1
LEDR3
LEDR2
LEDR1
LEDR0
LEDG7
LEDG6
LEDG5
LEDG4
LEDG3
LEDG2
LEDG1
LEDG0
GND4
SW1
SW0
KEY3
KEY2
KEY1
KEY0

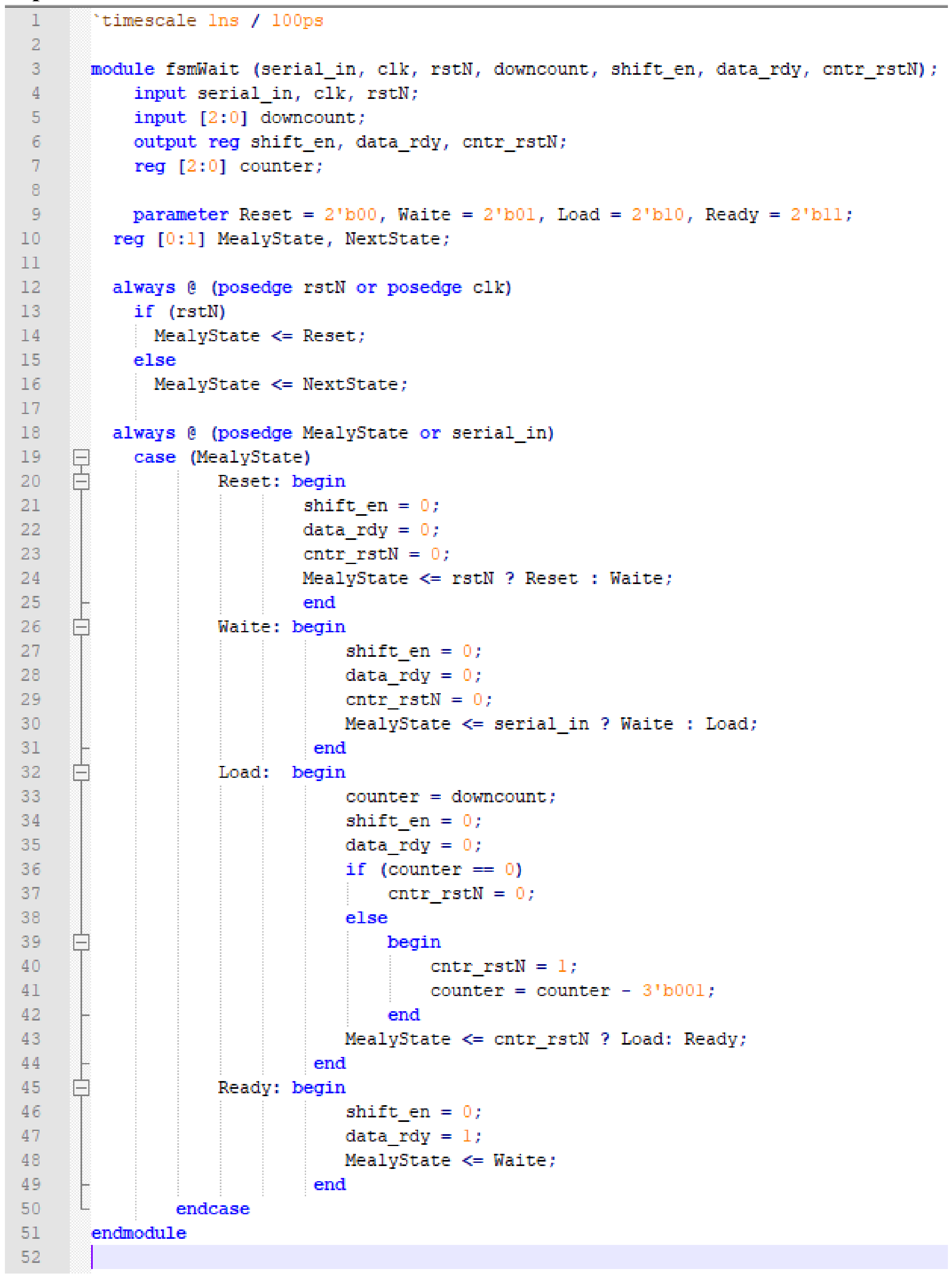






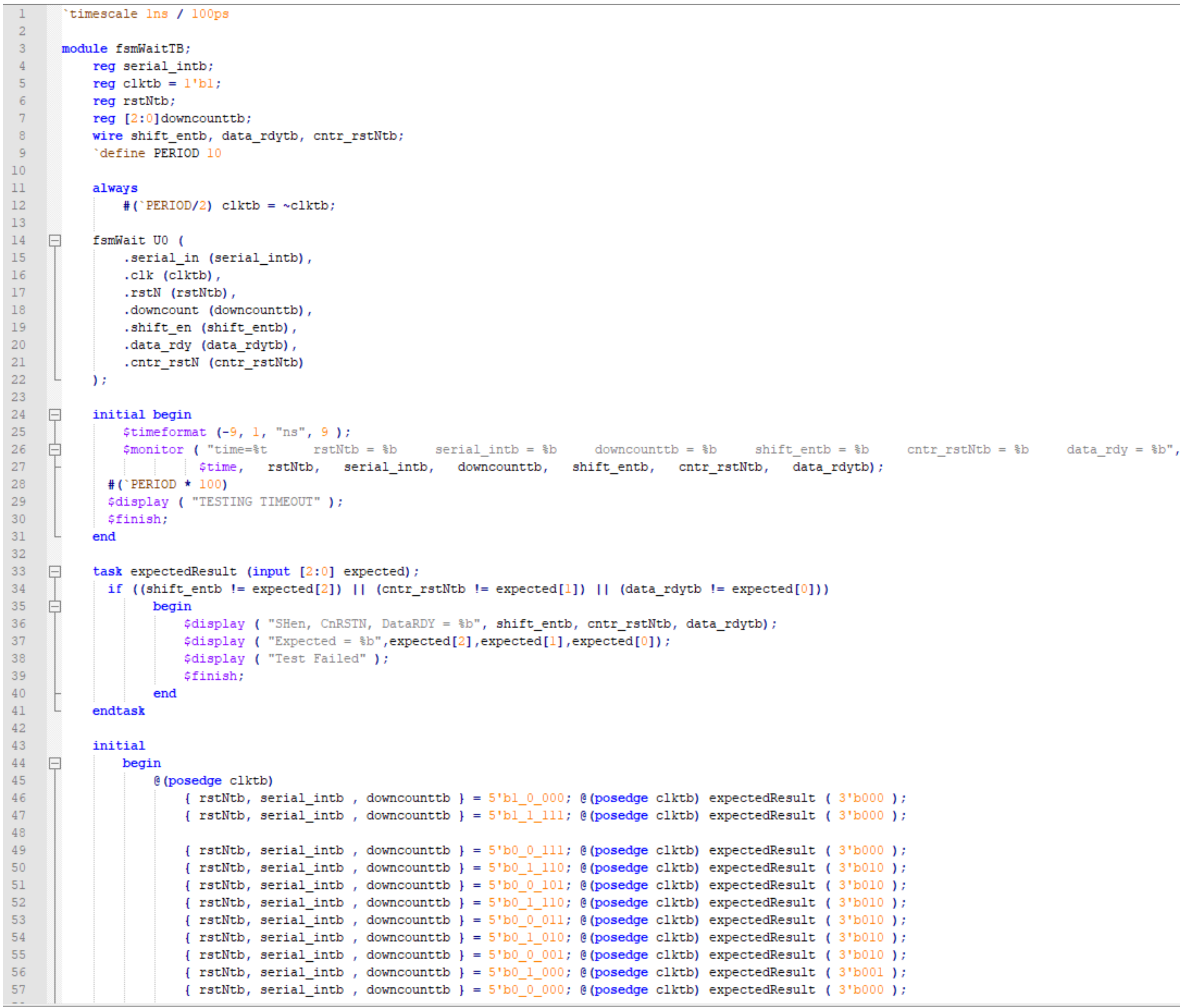
These are my images of the sequence generator. The first and fourth image show where the sequence starts and when reset is 1 for both cases. The 2nd and 3rd image start the sequence of decreasing from 1000 to 0001 then back up to 1000. The 5th and 6th image start the sequence of increasing from 0001 to 1001 then back down to 0001.

Experiment 4

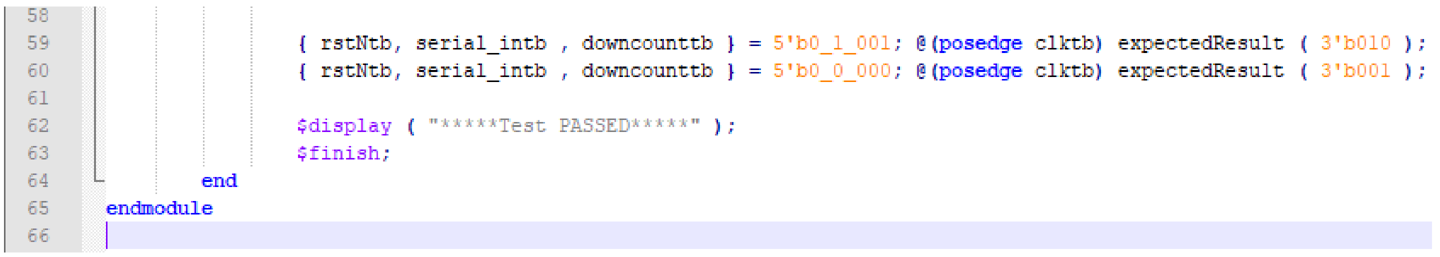


```
`timescale 1ns / 100ps

module fsmWait (serial_in, clk, rstN, downcount, shift_en, data_rdy, cntr_rstN);
    input serial_in, clk, rstN;
    input [2:0] downcount;
    output reg shift_en, data_rdy, cntr_rstN;
    reg [2:0] counter;

    parameter Reset = 2'b00, Waite = 2'b01, Load = 2'b10, Ready = 2'b11;
  reg [0:1] MealyState, NextState;

  always @ (posedge rstN or posedge clk)
    if (rstN)
      MealyState <= Reset;
    else
      MealyState <= NextState;

  always @ (posedge MealyState or serial_in)
    case (MealyState)
            Reset: begin
                    shift_en = 0;
                    data_rdy = 0;
                    cntr_rstN = 0;
                    MealyState <= rstN ? Reset : Waite;
                    end
            Waite: begin
                        shift_en = 0;
                        data_rdy = 0;
                        cntr_rstN = 0;
                        MealyState <= serial_in ? Waite : Load;
                     end
            Load:  begin
                        counter = downcount;
                        shift_en = 0;
                        data_rdy = 0;
                        if (counter == 0)
                            cntr_rstN = 0;
                        else
                            begin
                                cntr_rstN = 1;
                                counter = counter - 3'b001;
                            end
                        MealyState <= cntr_rstN ? Load: Ready;
                     end
            Ready: begin
                        shift_en = 0;
                        data_rdy = 1;
                        MealyState <= Waite;
                     end
        endcase
endmodule
```

This is my Verilog code for the FSM figure implementation.

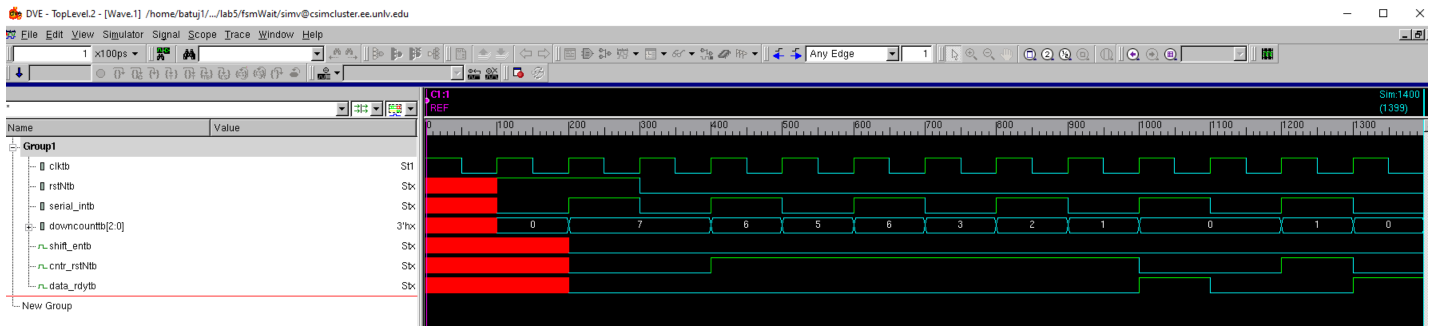


```verilog
`timescale 1ns / 100ps

module fsmWaitTB;
    reg serial_intb;
    reg clktb = 1'b1;
    reg rstNtb;
    reg [2:0]downcounttb;
    wire shift_entb, data_rdytb, cntr_rstNtb;
    `define PERIOD 10

    always
        #(`PERIOD/2) clktb = ~clktb;

    fsmWait U0 (
        .serial_in (serial_intb),
        .clk (clktb),
        .rstN (rstNtb),
        .downcount (downcounttb),
        .shift_en (shift_entb),
        .data_rdy (data_rdytb),
        .cntr_rstN (cntr_rstNtb)
    );

    initial begin
        $timeformat (-9, 1, "ns", 9 );
        $monitor ( "time=%t      rstNtb = %b      serial_intb = %b      downcounttb = %b      shift_entb = %b      cntr_rstNtb = %b      data_rdy = %b",
                  $time,    rstNtb,    serial_intb,    downcounttb,    shift_entb,    cntr_rstNtb,    data_rdytb);
      #(`PERIOD * 100)
      $display ( "TESTING TIMEOUT" );
      $finish;
    end

    task expectedResult (input [2:0] expected);
      if ((shift_entb != expected[2]) || (cntr_rstNtb != expected[1]) || (data_rdytb != expected[0]))
            begin
                $display ( "SHen, CnRSTN, DataRDY = %b", shift_entb, cntr_rstNtb, data_rdytb);
                $display ( "Expected = %b",expected[2],expected[1],expected[0]);
                $display ( "Test Failed" );
                $finish;
            end
    endtask

    initial
        begin
            @(posedge clktb)
                { rstNtb, serial_intb , downcounttb } = 5'b1_0_000; @(posedge clktb) expectedResult ( 3'b000 );
                { rstNtb, serial_intb , downcounttb } = 5'b1_1_111; @(posedge clktb) expectedResult ( 3'b000 );

                { rstNtb, serial_intb , downcounttb } = 5'b0_0_111; @(posedge clktb) expectedResult ( 3'b000 );
                { rstNtb, serial_intb , downcounttb } = 5'b0_1_110; @(posedge clktb) expectedResult ( 3'b010 );
                { rstNtb, serial_intb , downcounttb } = 5'b0_0_101; @(posedge clktb) expectedResult ( 3'b010 );
                { rstNtb, serial_intb , downcounttb } = 5'b0_1_110; @(posedge clktb) expectedResult ( 3'b010 );
                { rstNtb, serial_intb , downcounttb } = 5'b0_0_011; @(posedge clktb) expectedResult ( 3'b010 );
                { rstNtb, serial_intb , downcounttb } = 5'b0_1_010; @(posedge clktb) expectedResult ( 3'b010 );
                { rstNtb, serial_intb , downcounttb } = 5'b0_0_001; @(posedge clktb) expectedResult ( 3'b010 );
                { rstNtb, serial_intb , downcounttb } = 5'b0_1_000; @(posedge clktb) expectedResult ( 3'b001 );
                { rstNtb, serial_intb , downcounttb } = 5'b0_0_000; @(posedge clktb) expectedResult ( 3'b000 );
```

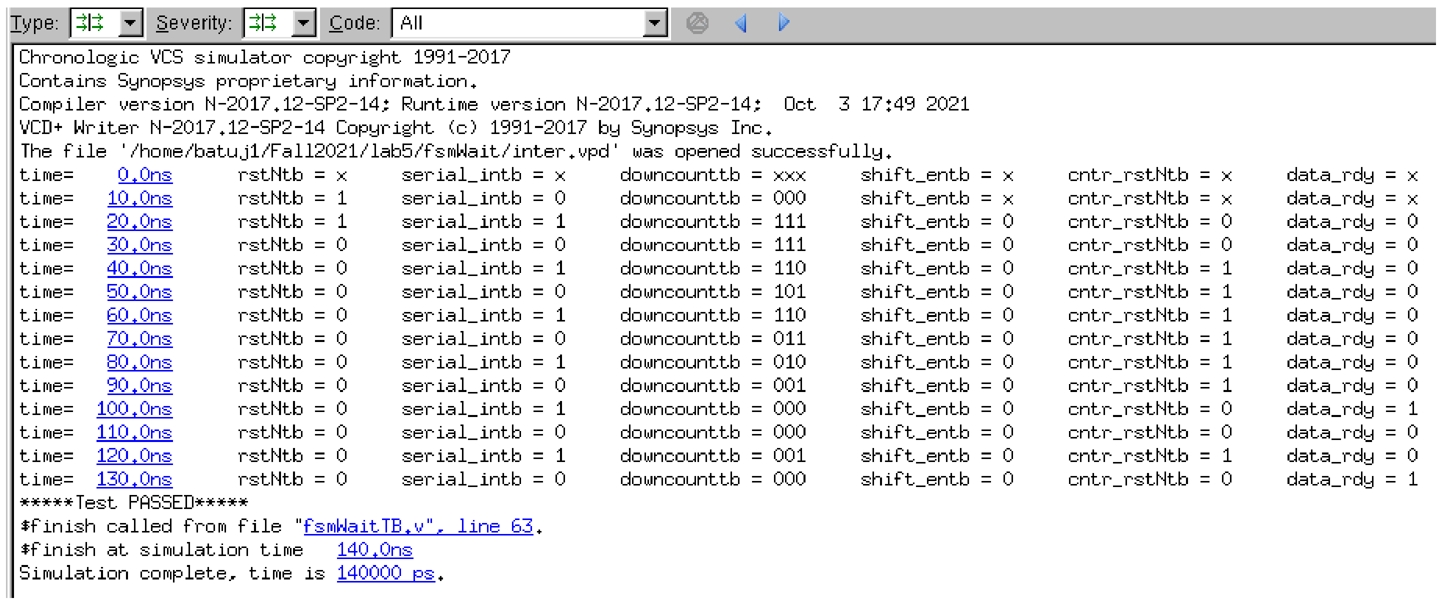


```verilog

                { rstNtb, serial_intb , downcounttb } = 5'b0_1_001; @(posedge clktb) expectedResult ( 3'b010 );
                { rstNtb, serial_intb , downcounttb } = 5'b0_0_000; @(posedge clktb) expectedResult ( 3'b001 );

                $display ( "*****Test PASSED*****" );
                $finish;
        end
endmodule

```

This is my testbench Verilog code for the FSM figure implementation.



These are my VCS waveforms for the FSM figure implementation.



```
Chronologic VCS simulator copyright 1991-2017
Contains Synopsys proprietary information.
Compiler version N-2017.12-SP2-14; Runtime version N-2017.12-SP2-14;  Oct  3 17:49 2021
VCD+ Writer N-2017.12-SP2-14 Copyright (c) 1991-2017 by Synopsys Inc.
The file '/home/batuj1/Fall2021/lab5/fsmWait/inter.vpd' was opened successfully.
time=    0.0ns     rstNtb = x     serial_intb = x     downcounttb = xxx     shift_entb = x     cntr_rstNtb = x     data_rdy = x
time=   10.0ns     rstNtb = 1     serial_intb = 0     downcounttb = 000     shift_entb = x     cntr_rstNtb = x     data_rdy = x
time=   20.0ns     rstNtb = 1     serial_intb = 1     downcounttb = 111     shift_entb = 0     cntr_rstNtb = 0     data_rdy = 0
time=   30.0ns     rstNtb = 0     serial_intb = 0     downcounttb = 111     shift_entb = 0     cntr_rstNtb = 0     data_rdy = 0
time=   40.0ns     rstNtb = 0     serial_intb = 1     downcounttb = 110     shift_entb = 0     cntr_rstNtb = 1     data_rdy = 0
time=   50.0ns     rstNtb = 0     serial_intb = 0     downcounttb = 101     shift_entb = 0     cntr_rstNtb = 1     data_rdy = 0
time=   60.0ns     rstNtb = 0     serial_intb = 1     downcounttb = 110     shift_entb = 0     cntr_rstNtb = 1     data_rdy = 0
time=   70.0ns     rstNtb = 0     serial_intb = 0     downcounttb = 011     shift_entb = 0     cntr_rstNtb = 1     data_rdy = 0
time=   80.0ns     rstNtb = 0     serial_intb = 1     downcounttb = 010     shift_entb = 0     cntr_rstNtb = 1     data_rdy = 0
time=   90.0ns     rstNtb = 0     serial_intb = 0     downcounttb = 001     shift_entb = 0     cntr_rstNtb = 1     data_rdy = 0
time=  100.0ns     rstNtb = 0     serial_intb = 1     downcounttb = 000     shift_entb = 0     cntr_rstNtb = 0     data_rdy = 1
time=  110.0ns     rstNtb = 0     serial_intb = 0     downcounttb = 000     shift_entb = 0     cntr_rstNtb = 0     data_rdy = 0
time=  120.0ns     rstNtb = 0     serial_intb = 1     downcounttb = 001     shift_entb = 0     cntr_rstNtb = 1     data_rdy = 0
time=  130.0ns     rstNtb = 0     serial_intb = 0     downcounttb = 000     shift_entb = 0     cntr_rstNtb = 0     data_rdy = 1
*****Test PASSED*****
$finish called from file "fsmWaitTB.v", line 63.
$finish at simulation time   140.0ns
Simulation complete, time is 140000 ps.
```

This is my VCS output console for the FSM figure implementation.

4. Answers to questions

Question 1:

Mealy FSMs have a benefit over the Moore FSM when initializing the output instantaneously rather than the output depending on the state; however, the main disadvantage of the Mealy FSM is that the outputs are not held after each clock cycle. This means that Mealy FSMs are dependent on both the state and the inputs rather than only the state. Since the Mealy FSM considers the state and the inputs, there are more faults that may happen upon the output of the Mealy FSM.

Question 2:

Encoding styles of an FSM refer to how the states are represented for clarity and ease of maintenance. A few examples of encoding styles include binary encoding, one-hot encoding, and gray coding. These different coding styles depend on how the FSM is designed. Binary encoding has the states enumerated with binary numbers such as 0000, 0100, 0110, 1010, etc. One-hot encoding refers to states as bit patterns with only one '1'. An example of this would be:

00001, 00010, 00100, 01000, or 10000. Gray encoding only differs from one bit such as 00, 01, 10, or 110. Each of these encoding styles have benefits over one another. Binary encoding minimizes the length of state vectors. One-hot encoding is faster, but uses more registers and less logic. Gray encoding reduces faulty errors within an FSM. Encodings basically represent how the states are encoded within an FSM's design.

Question 3:
The combinational section of an FSM is to determine the next state logic. It's to show how one state transitions from another as well as where the output would be determined inside the FSM. Within the combinational section, there are two different logics: next state and output. Sometimes, the combinational section can be split up into one for the next state and one for output. The sequential section of an FSM is to store the current state of the FSM. It is to show when one state goes to the next rather than how one state goes to the next. For example, the sequential section of an FSM may include a reset where the current state will be set to the beginning, then the next state will start when no reset is initiated and at the positive clock edge.

5. Conclusions & Summary

Overall, this was probably one of the most time consuming labs. It wasn't necessarily hard understanding what was being asked for within the lab; however, there was just a lot of work to be done for the lab. The main issues for the first experiment were implementing the clock divider into the code again, but it was a simple fix that took about 10 minutes. In addition, trying to input the correct code within the DE2 was also rough as the next state transitioned at the clock edge, so I had to input the code very quickly on the DE2 board. The second and third experiments were quite easy. The fourth experiment took the most time trying to understand what was being outputted. The video provided did clarify some of my questions on what some of the inputs and outputs meant. My main concern was with the down counter because I had no idea how to implement this within an FSM. My first idea was to create the down counter in a separate module, but that idea did not work; therefore, I included the down counter within the state itself and made the next state condition based on cntr_rstN. Choosing between Mealy and Moore ended up being very confusing, but I remember now which one to use.