Class:	**CPE300L - Digital System Architecture and Design Lab**	Semester:	**Fall 2025**

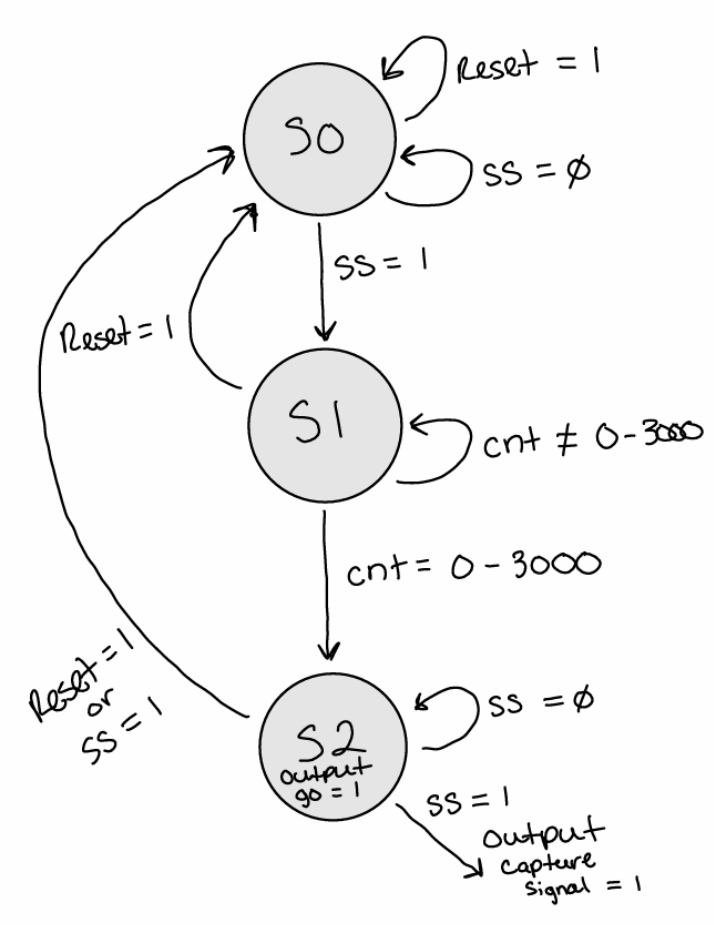
Points		Document author:	**Darryll Mckoy**
		Author's email:	**Mckoyd1@unlv.nevada.edu**
		Document topic:	**Postlab 2**

Instructor's comments:

1. Hours spent on this lab

About 7 hours

2. FSM state transition diagram



3. FSM System Verilog code

C: > altera > 14.1 > FSM.sv

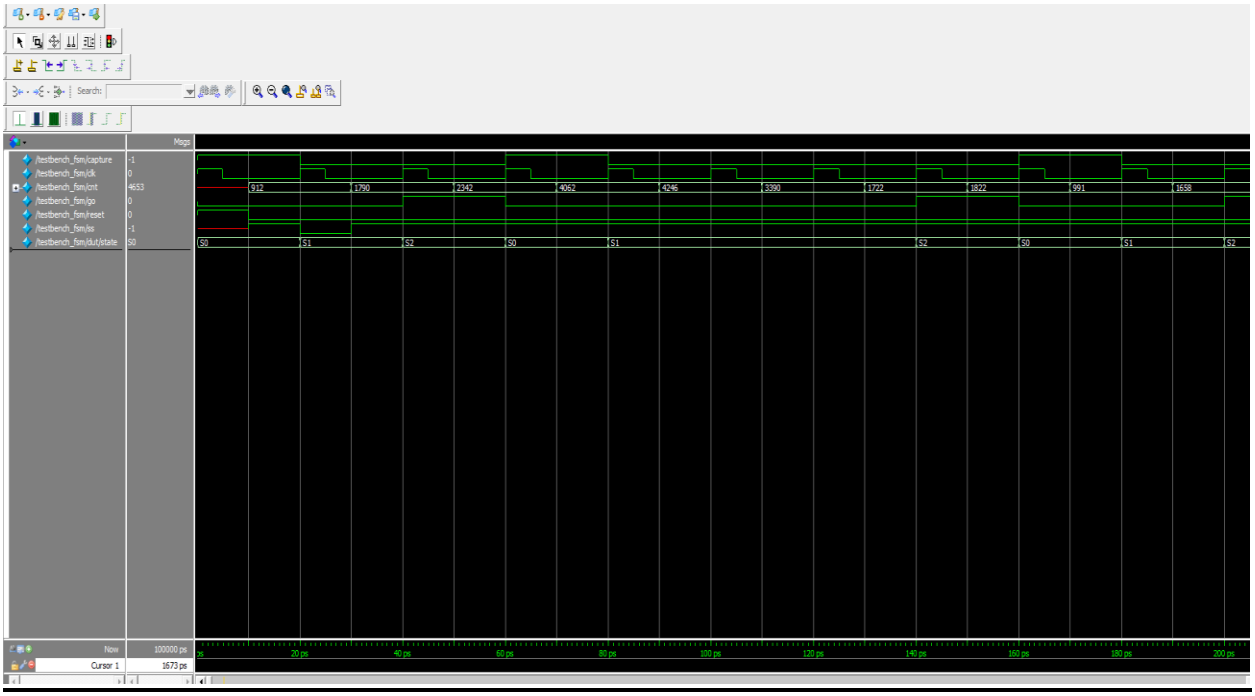
```
1   //CPE 300L, Lab2 FSM
2
3   module FSM(input  logic clk, reset,
4              input  logic ss,
5           input  logic [31:0] cnt,
6              output logic go, capture);
7
8    typedef enum logic [1:0] {S0, S1, S2} statetype;
9    statetype state, nextstate;
10
11   // state register
12   always_ff @(posedge clk, posedge reset)
13     if (reset) begin
14       state <= S0;
15       $display("reset pressed moving to state S0");
16     end
17
18     else       state <= nextstate;
19
20    // next state logic
21    always_comb
22      case (state)
23        S0:      if (ss) begin
24                   nextstate = S1;    //if ss pressed then move on to next state and generate random value 0-3000;
25                   $display("ss pressed moving to state S1");
26        end
27           else begin
28          nextstate = S0;   //if ss not pressed then stay in current state.
29          $display("ss not pressed staying in state S0");
30          end
31
32        S1:      if (cnt > 32'h0 && cnt < 32'hBB8) begin
33                   nextstate = S2;
34                   $display("cnt between 0-3000 asserting go and moving to state S2");
35        end
36           else begin
37             nextstate = S1;
38             $display("Error! cnt not between 0-3000 staying in S1");
39           end
40
41        S2:      if (ss) begin
42           nextstate = S0;
43           $display("ss pressed asserting capture and moving to state S0");
44        end
45
46      else begin
47        nextstate = S2;
48        $display("ss not pressed staying in state S2");
49      end
50
51        default: nextstate = S0;
52      endcase
53
54    // output logic
55    assign capture = (state == S0);
56    assign go = (state == S2);
57  endmodule
```

FSM testbench code

C: > altera > 14.1 > testbench_FSM.sv

```
1   // 4.39: testbench example 3
2
3   module testbench_fsm();
4     logic        clk, reset;
5     logic        ss;
6     logic [31:0] cnt;
7     logic go, capture;
8
9     // instantiate device under test
10    FSM dut(clk, reset, ss, cnt, go, capture);
11
12    // generate clock
13    always
14      begin
15        clk = 1; #5; clk = 0; #5;
16        cnt = $urandom%5000; #10;
17      end
18
19    // at start of test, load vectors
20    // and pulse reset
21    initial
22      begin
23      reset = 1; #10; reset = 0;
24      ss = 1; #10;
25
26      ss = 0; #10;
27      ss = 1; #10;
28      end
29
30  endmodule
```

4. FSM waveform



5. 1kHz clock system Verilog code

C: > altera > 14.1 > clk1kHz.sv

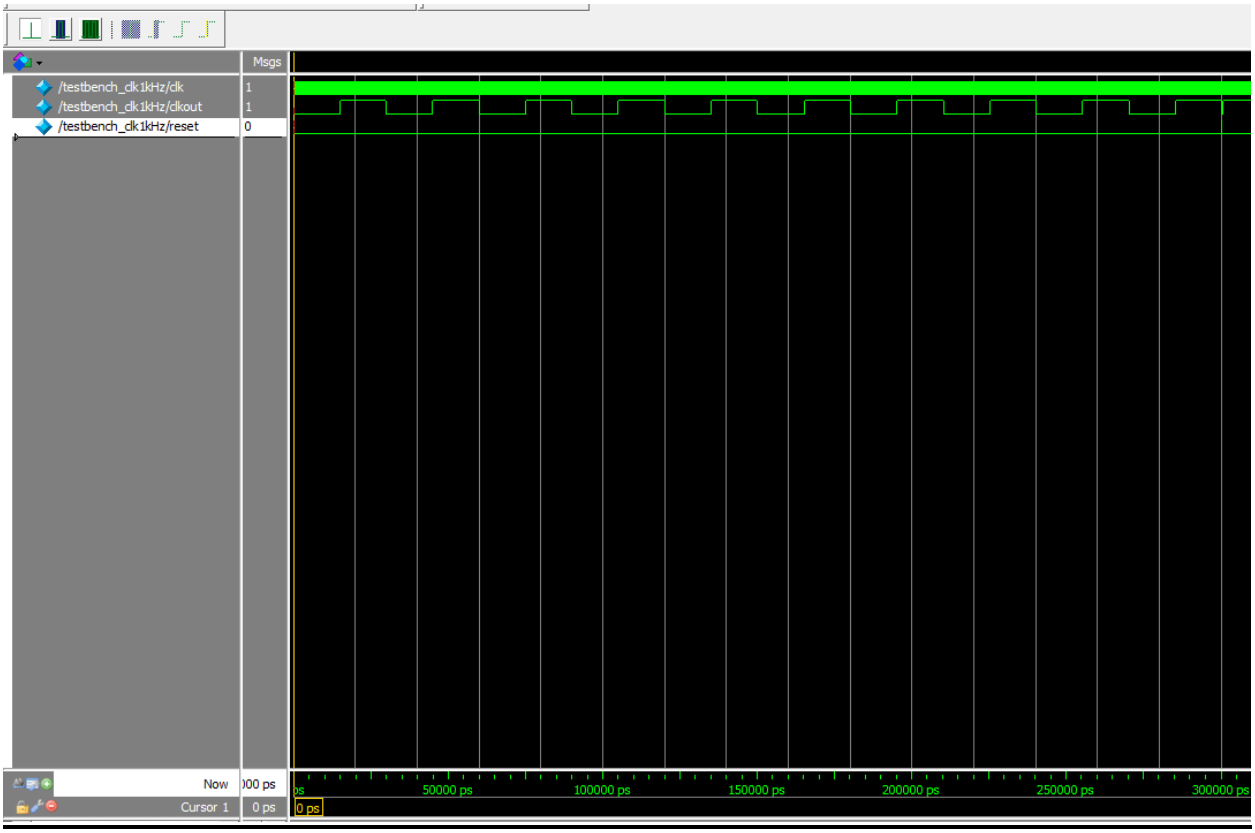
```
1
2   module clk1kHz(input  logic clk, reset, output logic clkout);
3
4     logic [15:0] count = 0;   // Counter to count clock cycles (e.g., 20 bits for 1M counts)
5     int value = 1000; // The number of clock cycles for 1 kHz
6     int clk_20kHz = 20000;
7
8   ////////////////////////////////////////////////////////////
9
10      // Frequency divider logic
11      always @(posedge clk or posedge reset) begin
12
13          if (reset) begin
14              count <= 16'd0;
15              clkout <= 1'b0;
16          end
17
18
19          if (count == value) begin
20                  count <= 16'd0;
21                  clkout <= ~clkout;
22              end
23
24              else begin
25                  count <= count + 1;
26
27              end
28
29      end
30
31
32  endmodule
```

1kHz clock testbench code

C: > altera > 14.1 > clk1kHz_testbench.sv

```
1   //1kHz testbench
2
3   module testbench_clk1kHz();
4     logic        clk, reset;
5     logic        clkout;
6
7
8     // instantiate device under test
9     clk1kHz dut(clk, reset, clkout);
10
11
12    // generate clock
13    always
14      begin
15        clk = 1; #5 clk = 0;
16      end
17
18
19   initial
20      begin
21        reset = 1; #5 reset = 0;
22      end
23
24    //toggle clock at positive clock edge
25    always @(posedge clk)
26      begin
27      //reset = 1; #5 reset = 0;
28      clk = 1; #5; clk = 0; #5;
29
30      end
31
32  endmodule
```

6. 1kHz clock waveform



7. Millisecond counter system Verilog code

C: > altera > 14.1 > mscnt.sv

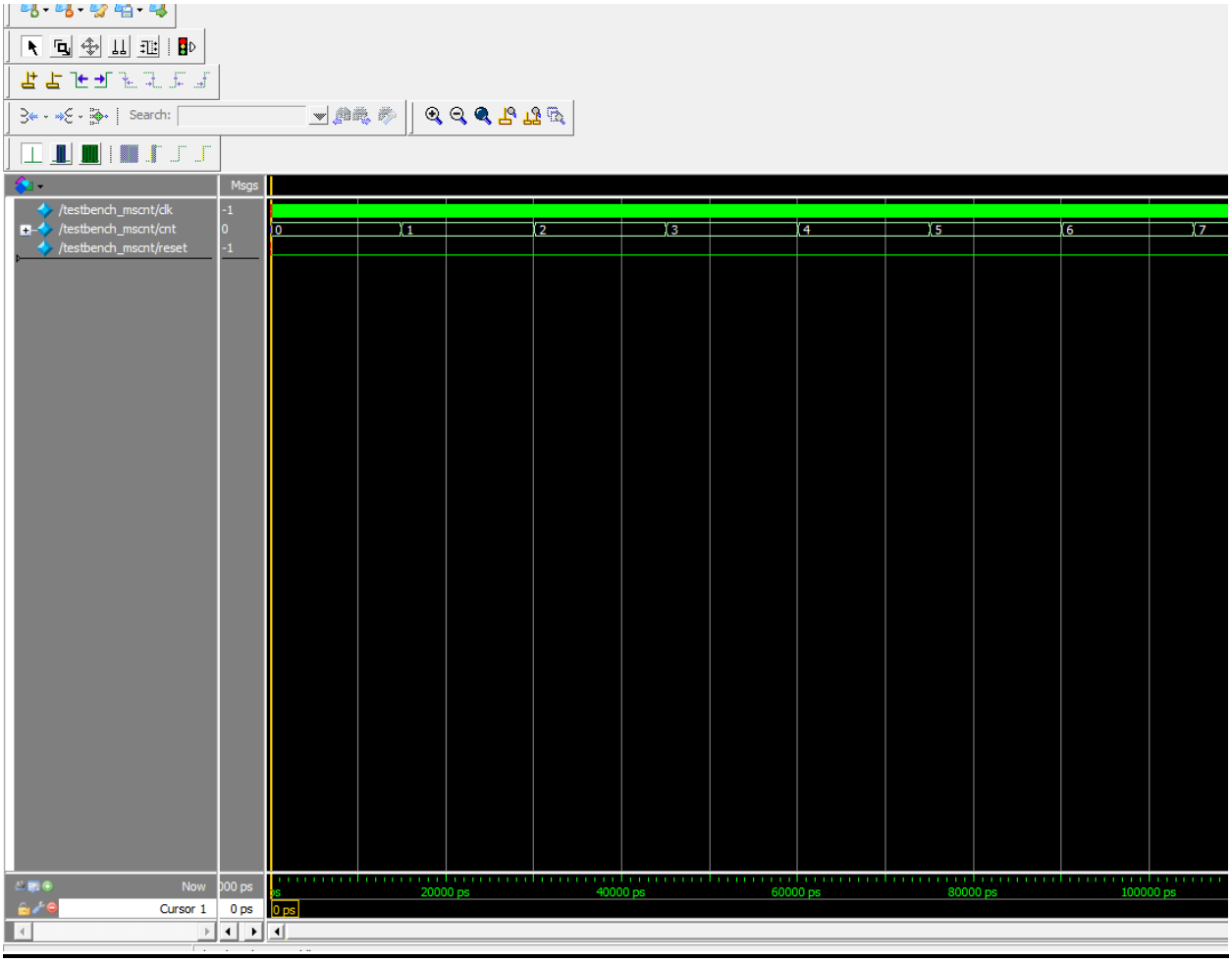
```
1   module mscnt(input  logic clk, reset,
2   output logic [31:0] cnt);
3
4
5   logic [15:0] count = 0;   // Counter to count clock cycles
6     int value = 1000; // The number of clock cycles for 1 kHz
7     int clk_20kHz = 20000;
8
9   ////////////////////////////////////////////////////////////
10
11
12      always @(posedge clk or posedge reset) begin
13
14          if (reset) begin
15              count <= 16'd0;
16              //clkout <= 1'b0;
17              cnt <= 32'b0;
18          end
19
20
21          if (count == value) begin
22                  count <= 16'd0;
23                  //clkout <= ~clkout;
24                  cnt <= cnt + 1;
25              end
26
27              else begin
28                  count <= count + 1;
29
30              end
31
32      end
33
34
35  endmodule
```

Millisecond counter system Verilog testbench code

C: > altera > 14.1 > mscnt_testbench.sv

```
1   //millisecond counter testbench
2
3   module testbench_mscnt();
4     logic        clk, reset;
5     logic        [31:0] cnt;
6
7
8     // instantiate device under test
9     mscnt dut(clk, reset, cnt);
10
11
12    // generate clock
13    always
14      begin
15        clk = 1; #5 clk = 0;
16      end
17
18
19   initial
20      begin
21        reset = 1; #5 reset = 0;
22      end
23
24    //toggle clock at positive clock edge
25    always @(posedge clk)
26      begin
27      //reset = 1; #5 reset = 0;
28      clk = 1; #5; clk = 0; #5;
29
30      end
31
32  endmodule
```

8. Millisecond counter waveform



9. Feedback

No complaints.