California State University Long Beach
Computer Engineering and Computer Science Department
CECS 201 – Computer Logic Design I
Lab 8 – Finite State Machine – FSM

Objectives:

- To implement and test a sequential design using Vivado SW and FPGA board
- To learn how to model **FSM** in Verilog
- To support learning about design, implementation and functioning of FSMs

Theory:

This lab assignment will help you put in practice concepts covered in chapter "Finite State Machines" particularly sub-sections "FSMs", "Capturing behavior", "FSM to circuits" and "State encoding": please make sure to study the corresponding chapter and do corresponding homework assignment in preparation for this lab assignment.

Setup:

Please follow the setup instruction provided in the Lab 1 assignment.

Assignment:

This lab assignment supports learning about design of Finite State Machine (FSM), while focusing on state encoding and 3-segment style of coding. You will need to create a module and a corresponding testbench for an FSM described by state transition diagram using the 3-segment coding style. The states will be encoded using Gray code. Once you confirm that the FSM is functionally correct, you will need to incorporate it in the top module. The clock of the FPGA board is way too fast, thus a human eye cannot difference the states. You will connect a pulse generator to the FSM that will enable/disable state transition. This pulse generator is connected to a control input, **not** a clock signal. In order to preserve clock distribution tree (and facilitate timing analysis, which is out of scope of this course) and to create a synchronous sequential circuit, connect only clock to your clk input. You are provided with the code for pulse generator and fsm_top. Finally, you will **implement** a fsm_top on the board. Follow the instructions in the Steps section and the video tutorials.

In the conclusion portion of the report, write the expected results and reflect if the simulation shows the expected results. Also add if the FPGA board shows expected output value for each test case. If not, try to figure out why and write this in the report as well.

Steps:

Both Student Virtual Lab (SVL) and Vivado are installed on the computers in the lab. Feel free to install them on your laptop or desktop.

1. If you need to use SVL, connect to SVL, as per instructions on BB

2. Create a folder **lab8** in the **labs** folder for lab 8 assignment
   OK,
3. Start Vivado 2016.x and create a project, named lab8-v0

4. For each module in this lab populate the top portion with
   // Company: CECS 201 – Fall 2019
   // Engineer: your name

5. Create a new module with a module name, inputs and outputs as shown in the Figure 2. Implement an fsm_w_pulse using Figure 1. That shows only state transitions, Table 1. That shows values for the output "out" for each state, and pulse input that will, for now, be provided from a testbench and always be 1. Keep in mind, that FSM is a combination of combination and sequential circuits, and therefore it will need a "reset" input to reset a state register, and a "clk" signal to facilitate synchronous sequential operation of the state register. Signal "cnt_up" sill determine the direction: if "1" the FSM will count up S0, S1, ...S8, S0 – as per the Figure 1. If "cnt_up" equals 0, the sequence will be reversed: S0, S8, S7, … S0, S8,… Please refer to accompanying video tutorials on BB for more details.

   Table 1. Output values needed to be produced in each state

state	Out (binary)
S0	0000_0000_0000_0000
S1	0000_0000_0000_0011
S2	0000_0000_0000_1111
S3	0000_0000_0011_1111
S4	0000_0000_1111_1111
S5	0000_0011_1111_1111
S6	0000_1111_1111_1111
S7	0011_1111_1111_1111
S8	1111_1111_1111_1111
default	1000_0000_0000_0000

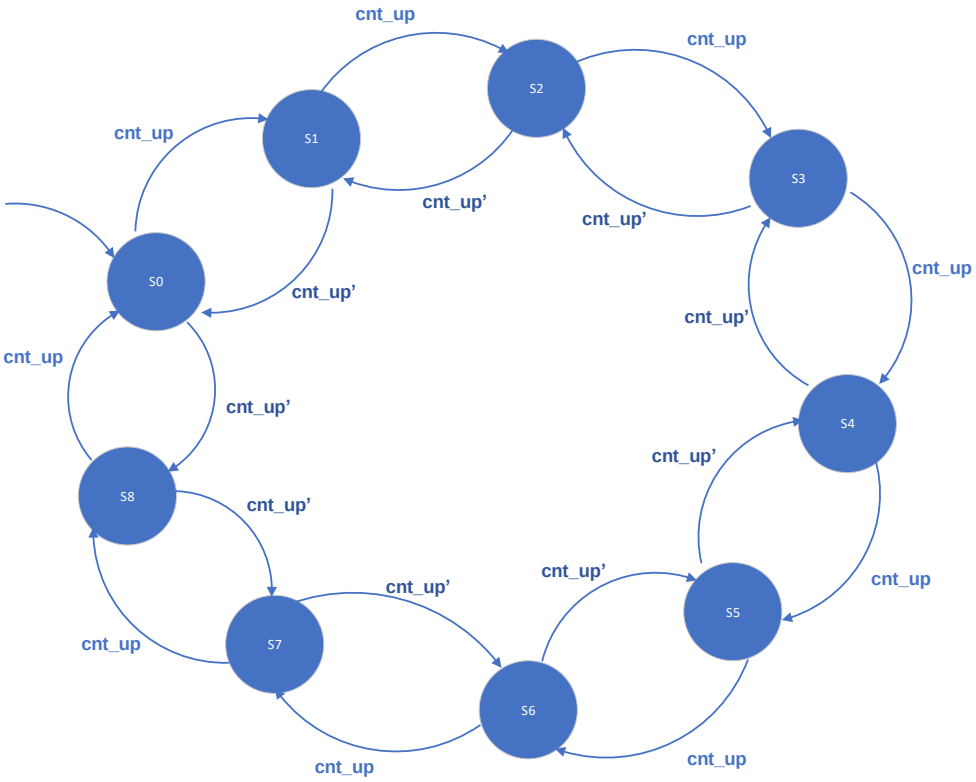

Figure 1. State transition diagram for Moore FSM; the outputs are defined by Table 1.

6. Inside always block for state transition, for each case, you have two possible next states, depending on the values of "cnt_up" signal. For example: S0 transitions to S1 on "cnt_up", but it transitions to S8 on "cnt_up'". To make your code more compact and readable instead of usinf an "if" statement, please use an assignment statement with conditional operator (http://verilog.renerta.com/mobile/source/vrg00010.htm). The expression in the parenthesis (Boolean_var) is evaluated: if it is true, Res gets value of T_expr; and if it is false, Res gets the value of F_expr, as per example below. This operator can be used for scalar and vector variables.

   assign Res = (Boolean_var)? T_expr : F_expr;

7. Note that non-blocking assignment "<=" should be used inside the always block for synchronous sequential logic, i.e. state register, where blocking assignment "=" should be used inside the always block for combinational logic, i.e. next state and output logic.

8. Create a testbench for fsm_w_pulse, as per Figure 3. Instantiate the uut using the variables defined at the beginning of the testbench. Add more tests inside the "initial" statement to change values of "up" and "reset".

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //////////////////////////////////////////////////////////////////////////////////
6
7  module fsm_w_pulse (
8          input clk,
9          input reset,
10         input cnt_up,
11         input pulse,
12         output reg [15:0] out);
13
14     reg [3:0] state;
15     reg [3:0] nstate;
16
17     // explicit state assignment
18     //Gray encoding
19     localparam S0 = 4'b0000,
20                S1 = 4'b0001,
21                S2 = 4'b0011,
22                S3 = 4'b0010,
23                S4 = 4'b0110,
24                S5 = 4'b0111,
25                S6 = 4'b0101,
26                S7 = 4'b0100,
27                S8 = 4'b1100;
28
29      // basic register
30     always @(posedge clk, posedge reset)
31         if(reset)
32             state <= 4'b0;
33         else
34             state <= nstate;
35
36      // state transitions
37     always @(*)//if explicit, you need to lisr all input: state, in
38      begin
39          nstate = state;
40          if(pulse)
41           case(state)
42             S0: nstate = (cnt_up) ? S1:S8;
43             //enter code fot the all remaining states here
44             default: nstate = 4'b1000;
45           endcase
46     end
47
47
48      // output generation
49     always @(*)//if explicit, you need to lisr all input: state, in
50      begin
51        out = 16'b0101_0101_0101_0101;
52        case(state)
53           S0: out = 16'b0000_0000_0000_0000;
54           //enter code fot the all remaining states here
55           default: out = 16'b1000_0000_0000_0000;
56        endcase
57     end
58
59 endmodule
60
```

Figure 2. Module fsm_w_pulse, its inputs and outputs, and the module body.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//////////////////////////////////////////////////////////////////////////////////

module fsm_w_pulse_tb();`timescale 1ns / 1ps

    // Inputs
    reg clk;
    reg reset;
    reg up;
    reg pulse;

    // Outputs
    wire [15:0] out;

    // Instantiate the Unit Under Test (UUT)
    fsm_w_pulse uut (   );

always
 #10 clk = ~clk;

    initial begin
        // Initialize Inputs
        clk = 0;
        pulse = 1;
        reset = 1;
        up = 0;

        #100;
        reset = 0;
        #380;
        up = 1;
        // Add stimulus here
    end

endmodule
```

Figure 3. Testbench fsm_w_pulse_tb

9. Simulate the testbench for at least 1,000 ns, observe the outputs in the simulation window, and include in the report a screenshot of the waveform.

10. Open Schematic (under “RTL Analysis”->” Elaborated Design”) and analyze the output. Include the schematic in the report.

11. The next step is to connect the fsm to the FPGA board. The FSM creates a 16-bit output, but the frequency of FPGA would not allow human eye to notice changes of the output that correspond to the state change. Therefore, you will create a pulse generator and a top module in which you will connect the FSM to the pulse generator. The pule will act as

an “enable” signal, allowing slower state transitions, thus noticeable to human eye. The code for the pulse generator and top module are shown in the Figure 4. and 5. respectively

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //////////////////////////////////////////////////////////////////////////////////
6
7  module pulse_generator(clk, reset, pulse);
8
9      input clk, reset;
10     output pulse;
11     reg [25:0] count;
12
13     assign pulse = (count == 56_250_000);
14
15     always @(posedge clk, posedge reset)
16         if(reset)
17             count  <= 26'b0;
18         else if(pulse)
19             count <= 26'b0;
20         else
21             count <= count + 26'b1;
22
23 endmodule
```

Figure 4. Module pulse_generator

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //////////////////////////////////////////////////////////////////////////////////
6
7  module fsm_top(clk, reset, out, in);
8
9      output [15:0] out;
10     input in;
11     input clk, reset;
12
13     wire pulseConnect;
14
15     pulse_generator p0(.clk(clk), .reset(reset), .pulse(pulseConnect));
16     fsm_w_pulse fsm0(.clk(clk), .reset(reset), .cnt_up(in), .out(out), .pulse(pulseConnect));
17
18 endmodule
```

Figure 5. Top level design fsm_top

12. Draw the block diagram of the inside of the pulse generator and fsm_top and include them into your report. Both modules have been previously tested and they are provided.

13. Open the schematic and include the screenshot of the schematic "as-is" in the report. Expand both module instances by clicking the + sign and provide include the screenshot of the new schematic in the report.

Synthesis:

14. Make sure fsm_w_pulse is functionally correct. "Set as top" fsm_top module.

15. Add a constraint file using "Add source" option "Add or create constraints", press "Next". Name the file "fsm_top _32bit_NexysA7-100T", press "OK" and "Finish". Download file "MasterConstraint_NexysA7-100T.txt" from Lab 4 folder on BB and save it locally. Open the file with the text editor (Notepad or WordPad), copy its content to clipboard, and paste the content inside the "fsm_top_NexysA7-100T.xdc" file in the editor.

16. Modify the file to map the inputs of the fsm_top to the switches, and the outputs to the LEDs, by removing # at the beginning of an appropriate line, and changing parameters for get_ports. Make sure to **map the clock** as well, as shown in the Figure 16. at lines 9 and 10. Save the file.

17. Follow the procedure outlined in the previous labs: "Run Synthesis", "Run Implementation", "Generate Bitstream", connect and power on the Nexys A7-100T board, and finally "Open Hardware Manager". "Open Target" and chose "Auto connect" and "Program device".

18. The "light show" on the board will stop. Move the two rightmost switches (see the .xdc file describing the connections) to test the design: reset the fsm, change the direction of state transition. From time to time reset the fsm. What do you see when "in" is 0, and what when "in" is 1? Write your observation in the report

19. Close "Hardware Manager" by pressing X on the blue nav bar in top right (do not need to close the project or Vivado). Move "POWER" switch on the board to "OFF" position and unplug the board.

Extra Credit (20% more for this lab):

20. Record a short video with music in the background, and use "reset" and "in" to make the LEDs "dance" to any tune/song you choose. The length of the vide should be between 30-90 seconds. Should you need to have switches transition faster or slower, change the

value of the constant in the line 13 of the Figure 4. (do not forget to change the bit-width of the reg and constants accordingly). You are also welcome to change the output pattern to match the tune.

```
## This file is a general .xdc for the Nexys A7-100T
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project

## Clock signal
#set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { CLK100MHZ }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
#create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {CLK100MHZ}];
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
create_clock -add -name clk -period 10.00 -waveform {0 5} [get_ports {clk}];

##Switches
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { reset}]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { in }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
#set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { reset_top }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
#set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { SW[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
#set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { SW[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
#set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { SW[5] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
#set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { SW[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
#set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports { SW[7] }]; #IO_L5N_T0_D07_14 Sch=sw[7]
#set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 } [get_ports { SW[8] }]; #IO_L24N_T3_34 Sch=sw[8]
#set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 } [get_ports { SW[9] }]; #IO_25_34 Sch=sw[9]
#set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports { SW[10] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
#set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 } [get_ports { SW[11] }]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]
#set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33 } [get_ports { SW[12] }]; #IO_L24P_T3_35 Sch=sw[12]
#set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports { SW[13] }]; #IO_L20P_T3_A08_D24_14 Sch=sw[13]
#set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 } [get_ports { SW[14] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
#set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports { SW[15] }]; #IO_L21P_T3_DQS_14 Sch=sw[15]

## LEDs
set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { out[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { out[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { out[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports { out[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { out[4] }]; #IO_L7P_T1_D09_14 Sch=led[4]
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { out[5] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports { out[6] }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { out[7] }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { out[8] }]; #IO_L16N_T2_A15_D31_14 Sch=led[8]
set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33 } [get_ports { out[9] }]; #IO_L14N_T2_SRCC_14 Sch=led[9]
set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports { out[10] }]; #IO_L22P_T3_A05_D21_14 Sch=led[10]
set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports { out[11] }]; #IO_L15N_T2_DQS_DOUT_CSO_B_14 Sch=led[11]
set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports { out[12] }]; #IO_L16P_T2_CSI_B_14 Sch=led[12]
set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33 } [get_ports { out[13] }]; #IO_L22N_T3_A04_D20_14 Sch=led[13]
set_property -dict { PACKAGE_PIN V12   IOSTANDARD LVCMOS33 } [get_ports { out[14] }]; #IO_L20N_T3_A07_D23_14 Sch=led[14]
set_property -dict { PACKAGE_PIN V11   IOSTANDARD LVCMOS33 } [get_ports { out[15] }]; #IO_L21N_T3_DQS_A06_D22_14 Sch=led[15]

```

Figure 6. XDC file

What to submit:

Upload to Lab_8 Dropdox the following files:

a. fsm_w_pulse.v , fsm_w_pulse _tb.v,
b. pulse_generator.v , top_fsm.v,
c. fsm_top_NexysA7-100T.xdc,
d. your video for extra credit, and
e. your lab report.

Refer to the previous labs to locate the sources within the project.

Questions:

The questions to be asked during the demo will be similar, but not limited to, the questions listed below:

1. Show the design being simulated: explain the waveform seen during the simulation.

2. Observe pulse_generator and explain its function?

3. Explain how the implemented designs work on FPGA board. Is this something you expected? Why or why not?

4. Explain sensitivity list for **always** statement for each portion of the 3-segment code.

5. What happens when you change the value of the in switch?

6. If you did an extra credit: What is the song/tune you chose? Did you need states to transition faster or slower?