Students will be notified via CCLE if any changes are made.

CS M152A Project 3 **Due Date: 11/29 11:59 PM**

Design a Parking Meter

In this lab, you will design and implement a finite state machine (FSM) for a parking meter.

Introduction

In this lab, you are required to design an FSM to model a parking meter which simulates coins being added and displays the appropriate time remaining.

FSM Design:

Overview and Example

Finite State Machines (FSMs) are a powerful tool that can be used to model many real-world systems and are particularly useful for the behavioral modeling of sequential circuits. An FSM has a finite number of 'states' and can be in any one of these states at a given time. The machine transitions from one state to another based on the inputs it receives and the state that it is currently in. The machine must begin operation in an initial state. Given below is a simple example of an FSM for a turnstile, and the Verilog code to implement it. This FSM is a Moore machine because the output (*is_locked*) depends only on the present state. A Mealy machine is an FSM whose outputs depend on both the present state and input.

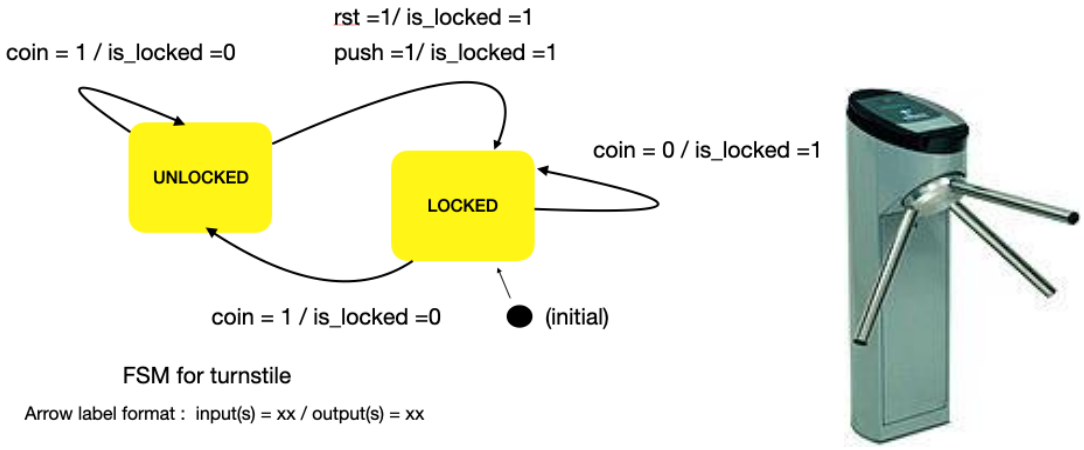

Fig 1 : FSM for a Turnstile (allows a user to push through when a coin is inserted)

Students will be notified via CCLE if any changes are made.

```
module turnstile(rst,clk,coin,push,is_locked);
input rst,clk,coin,push;
output reg is_locked;
parameter LOCKED = 1'b0;
parameter UNLOCKED = 1'b1;

reg current_state;
reg next_state;

// always block to update state : sequential - triggered by clock
always@(posedge clk)
begin
if(rst)
      current_state <= LOCKED;
else
      current_state <= next_state;

end

// always block to decide next_state : combinational- triggered by
state/input

always @(*)
case(current_state)
LOCKED : begin
            if(coin)
                  next_state = UNLOCKED;
            else
                  next_state = LOCKED;
             end
UNLOCKED:
         begin
            if(coin)
                  next_state = UNLOCKED;
            else if(push)
                  next_state = LOCKED;
            else
                  next_state = UNLOCKED;  //stay unlocked until push
             end

endcase

//always block to decide outputs : triggered by state/inputs; can be
comb/seq.
always @(*)
case(current_state)
LOCKED : begin
            is_locked = 1'b1;
             end
UNLOCKED:
         begin
            is_locked = 1'b0;
             end

endcase
endmodule
```

Students will be notified via CCLE if any changes are made.

System Specifications:

The specifications for the parking_meter module have been described below.
The input buttons represent different coin denominations and the seven-segment LED display will display the time remaining before the meter expires in seconds.

The inputs to the system have been listed in the table below

Inputs	Function
add1	add 60 seconds
add2	add 120 seconds
add3	add 180 seconds
add4	add 300 seconds
rst1	reset time to 16 seconds
rst2	reset time to 150 seconds
clk	frequency of 100 Hz
rst	resets to the initial state

As soon as a button is pushed, the time should be added immediately. The output is modeled as 4 seven segment displays which display the time remaining. Each add and reset button is high for at most one clock cycle.

- In the **initial state**, the seven-segment displays should be flashing 0000 with period 1 sec and duty cycle 50% (on for 0.5 sec and off for 0.5 sec).
- When any add button (add0, add1, add2, or add3) is pressed, the display adds to the corresponding time and starts counting down.
- When less than 180 seconds are remaining, the display should flash with a period of 2 seconds and 50% duty cycle. You should have alternate counts on the display

like 180, blank,178, blank,176,…). Make sure you blink such that even values show up and odd values are blanked out.

- When the time has expired, the display should flash 0000 with period 1 sec and duty cycle 50% (on for 0.5 sec and off for 0.5 sec).
- For example, if add4 is then pushed, the display should read 300 seconds and begin counting down (at 1 Hz). When the timer counts down to 180 seconds and add2 is pushed, the display should then read 300 seconds (120 + 180) and continue counting down. If rst1 goes high, then the display gets reset to 16 seconds and starts flashing accordingly while counting down.
- The max value of time will be 9999 and any attempt to increment beyond 9999, should result in the counter latching to 9999 and counting down from there.
- Use input clock(**clk**) frequency as 100 Hz
- Include a global input reset (**rst**) which takes the FSM to the **initial state**.
- Do not account for multiple inputs being pressed at the same time.

Though you do not have access to the FPGA, you will be required to design the output module which displays the time on the 4 seven segment displays for full credit.

Output Module:

The output module consists of a seven-segment vector **led_seg(7-bit vector)** which displays the actual value fed to the 4 segments corresponding to the digits being displayed. The order of the mapping is from CA to CG with CA being the most significant bit. (refer to the link below)

The anodes driving each of these segments are (one bit signals) **a1,a2,a3,a4**. Please refer to the Xilinx Nexys 3 reference manual to design the seven segment display module. You will need to multiplex the seven segment displays with the anodes as would be required in actual hardware.

https://reference.digilentinc.com/reference/programmable-logic/nexys-3/reference-manual

Include 4 other output ports (**val1, val2, val3, val4**) each a 4-bit vector, which display the actual digit in BCD (binary coded decimal) corresponding to each of the segments. You will be given partial credit for these ports even if your seven-segment implementation is not accurate.

Students will be notified via CCLE if any changes are made.

Deliverables

When you finish, the following should be submitted for this lab:

1. **Verilog source code** for the "parking_meter" module. The file should be named exactly as "parking_meter.v". If you are using sub-modules, define them within parking_meter.v. Please DO NOT submit them as separate files.
2. **Verilog testbench** you used to evaluate your design. Note that your testbench is graded based on the correctness of the waveforms generated in your report. Please name the file "testbench_UID.v" where UID is your UCLA ID. Make sure you test **all the** special cases.

3. **Lab Report**: Please refer to the syllabus for the basic components of your lab report.

    1. Include the FSM diagram and explain the states and transitions of your FSM.
    2. Explain how you test your design and include simulation waveforms
    3. Schematics can be generated from ISE but please explain how your Verilog code results in the RTL generated. Include the 'Design Summary' section of the synthesis report and the summary of your implementation (map) report and write 1-2 lines on the conclusions you draw from these reports. requirement
    4. Please name your report "Firstname_Lastname_Project3_UID.pdf" where UID is your UCLA ID.
4. **Video**: Please refer to 'Syllabus' for details.

Submission Checklist :

- Please submit ONLY the files in this table
- There is no late submission for this project.
- It is recommended that you have your submission ready an hour or two before the deadline so that you do not face problems due to long upload times etc.

Type	SUBMIT THESE FILES	Contents
Report	Firstname_Lastname_Project3_UID.pdf	-
Design	Firstname_Lastname_Project3_codes.zip	parking_meter.v
Testbench		testbench_UID.v

Students will be notified via CCLE if any changes are made.

Video	Firstname_Lastname_Project3_video.zip	video_UID.() (extension : some video format - ex. mp4)