ECEN 248: Introduction to Digital Design
Department of Electrical and Computer Engineering
Texas A&M University

Laboratory Exercise #10
A Simple Digital Combination Lock

Lab exercise created and tested by
Abbas Fairouz, He Zhou, Joshua Mashburn, and Sunil P. Khatri

1 Introduction

A classic approach to access control is the password, which can be found on safes, doorways etc. This typical approach requires that a person wishing to gain access to a particular entryway enter the password. If the password is correct, the entryway will unlock; otherwise, the entryway will remain locked, while conveying no information about the required sequence. In lab this week, we will attempt to recreate this conventional mechanism in digital form on the ZYBO Z7-10 board. To accomplish such a feat, we will introduce the *Moore machine*, which is a type of Finite State Machine (FSM). To prototype our combination-lock, we will make use of the push buttons and and LEDs on the ZYBO Z7-10 board. The exercises in this lab serve to reinforce the concepts covered in lecture.

2 Background

Background information necessary for the completion of this lab assignment will be presented in the next few subsection. The pre-lab assignment that follows will test your understanding of the background material.

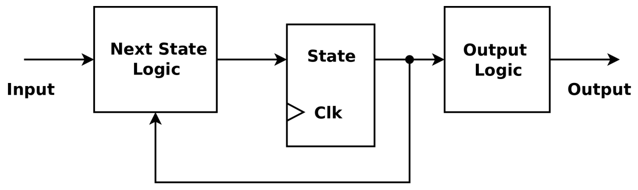
2.1 The Moore Machine

A Finite State Machine (FSM) is an abstract way of representing a sequential circuit. Often times the terms, FSM and sequential circuit, are used interchangeably; however, an FSM more formally refers to the model of a sequential circuit and can model higher-level systems, such as computer programs, as well. An FSM can be broadly classified into one of two types of machines, namely *Moore* or *Mealy*. Simply put, a *Moore* machine is an FSM such that the output depends solely on the state of the machine, whereas a *Mealy* machine is an FSM such that the output depends not only on the state but also the input.

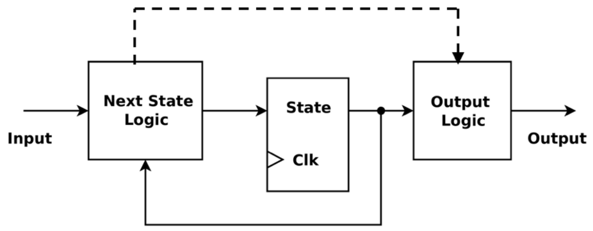
Figure 1 differentiates between the *Moore* and *Mealy* machines with a dotted wire, which on the *Mealy* machine, connects the input to the output logic. Other than the dotted wire, the two machines are identical. As shown, combinational logic generates the next state based on the current state and input to the machine, while the flip-flops store the current state. The output logic is purely combinational as well and depends on the state in both machine. For this week's lab, we will design a *Moore* machine because it fits our application quite well; however, for the sake of comparison, we will design a *Mealy* machine next week.

2.2 State Diagrams and Behavioral Verilog

The operation of an FSM can be described with a directed graph such that the vertices represent the states of the machine and the edges represent the transitions from one state to the next. This sort of graphical representation of an FSM is known as a *state diagram*. Figure 2 depicts the state diagram for a 2-bit



(a) Moore Machine



(b) Mealy Machine

Figure 1: Moore vs. Mealy Machine

saturating counter. If you recall from previous labs, a saturating counter is one that will not roll over but rather, will stop at the minimum or maximum counter value. Let us examine the state diagram and convince ourselves that this is the case.

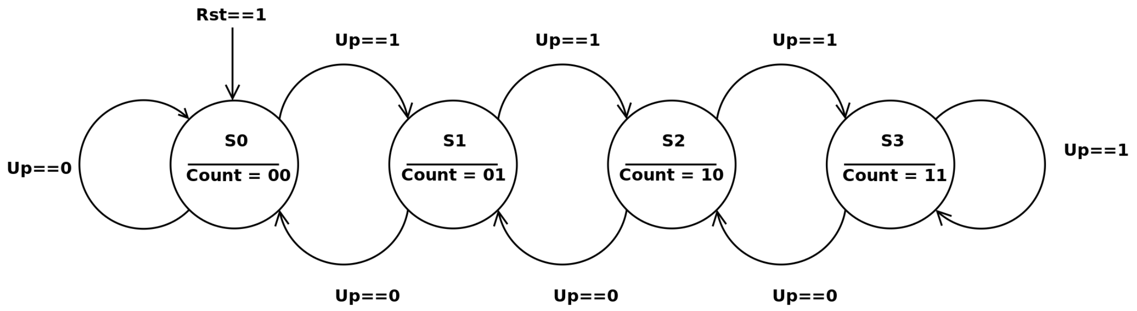


Figure 2: 2-bit Saturating Counter State Diagram

Each transition or edge of the graph in Figure 2 is marked by are particular input value that causes that transition. For this simple machine, there is only one input (with the exception of *Rst*) so we are able to explicitly represent all input combinations in the diagram. As we will see later on, this is not always the case. For each vertex, the state is clearly labeled (i.e. $S0$, $S1$, $S2$, and $S3$); likewise, the output for each state can be found under the state label. Remember that this is a *Moore* machine so the outputs are dependent only on the state of the machine. Given the state diagram of an FSM, the question to be answered now is how to go from a state diagram to a Verilog description? The next few paragraphs will demonstrate that exact process.

In lecture, you have learned how to implement an FSM by mapping the design into state tables. Then by performing logic minimization on the next-state and output expressions, you are able to realize your FSM with gates and flip-flops. In lab, however, we will take a different approach to implementing an FSM. Because we will be describing our final circuit in Verilog and synthesizing it for the ZYBO Z7-10 FPGA, the process would be much more efficient and less error-prone if we simply described the functionality of our FSM directly in Verilog using behavioral constructs. The flip-flops for holding the state of the machine can be described with a positive-edge triggered **always** block as done in previous labs, while the next-state and output logic can be eloquently described with **case** statements contained within **always@(*)** blocks. The code below demonstrates the use of these behavioral constructs to describe the FSM in Figure 2.

```
`timescale 1ns / 2 ns
`default_nettype none

/*This is a  behavioral Verilog description of*
 *a 2-bit saturating counter.                 */
module saturating_counter(
    /*output and input are wires*/
    output wire [1:0] Count; //2-bit output
    input wire Up; //input bit asserted for up
    input wire Clk, Rst; //the usual inputs to a synchronous circuit

    /*we haven't talked about parameters much but as you can see *
     *they make our code much more readable!                     */
    parameter S0 = 2'b00,
              S1 = 2'b01,
              S2 = 2'b10,
              S3 = 2'b11;

    /*intermeidate nets*/
    reg [1:0] state; //4 states requires 2-bits
    reg [1:0] nextState; //will be driven in always block!

    /*describe next state logic*/
    always@(*) //purely combinational!
        case(state)
            S0: begin
                if(Up) //count up
                    nextState = S1;
                else //saturate
                    nextState = S0;
            end
            S1: begin
                if(Up) //count up
                    nextState = S2;
                else //count down
                    nextState = S0;
            end
            S2: begin
                if(Up) //count up
                    nextState = S3;
                else //count down
                    nextState = S1;
            end
            S3: begin
                if(Up) //saturate
                    nextState = S3;
                else //count down
                    nextState = S2;
            end
            //do not need a default because all states
            //have been taken care of!
        endcase

    /*describe the synchronous logic to hold our state!*/
    always@(posedge Clk)
        if(Rst) //reset state
        state <= S0;
        else
        state <= nextState;

    /*describe the output logic, which in this case *
     *happens to just be wires                      */
    assign Count = state;

endmodule // gg ez
```

2.3 Digital Combination Lock on the ZYBO Z7-10

For interacting with the user, we will utilize of the push buttons, switches and LEDs on the ZYBO Z7-10 board.

3 Pre-lab

The objective of the pre-lab assignment is to describe in Verilog the FSM that you will load onto the ZYBO Z7-10 board during the lab. Therefore, the following subsections will describe the design we are attempting to create in detail. The pre-lab deliverables at the end will state what it is you are required to have prior to attending your lab session.

3.1 Digital Combination-Lock

In lab this week, we will design a digital combination-lock that can be used on a safe or access-controlled doorway. You will be responsible for describing the combination-lock FSM in Verilog. The easiest way to describe the combination-lock FSM is via a *state diagram*. Switches on the ZYBO Z7-10 board will be used to enter input to the lock. You will configure switches with a number(password) in binary format. Since we have only 4 switches on the ZYBO Z7-10 board, the password will only be 4 bits wide i.e 0-15. However, before delving into the diagram, let us take a high-level look at how our digital combination-lock is expected to operate.

1. Set up the switches with 13 in binary format.
2. Press the button **BTN0** which is connected to the signal '**Key1**'
3. Set up the switches with 7 in binary format.
4. Press the button **BTN1** which is connected to the signal '**Key2**'
5. Set up the switches with 9 in binary format.
6. Finally press the button **BTN0** which is connected to the signal '**Key1**' to unlock.

Now that we understand how we want our combination-lock to operate, we can represent the FSM for our combination-lock in a succinct manner with a state diagram. Figure 3 depicts the state diagram of the combination-lock with some transition labels missing. Let us take a moment to examine the incomplete diagram so that you can determine how to complete it.

The reset state is S0. This is indicated by the arrow labeled 'Reset == 1' pointing to the S0 node. This arrow differs from the other arrows in the diagram in that it does not represent a transition from one state to the next. Rather, it signifies the fact that no matter what state the machine is in, it will return to S0 when reset is asserted (i.e. 'Reset == 1'). We can see that the FSM will remain in state S0 until the *Password* input is equal to '13' and *Key1* is equal to 1 . When this happens, the FSM will transition to S1, where it will remain until the the *Key2* is pressed. When the *Key2* is pressed, if the Password is equal to 7, the FSM will move to S2, else it will move back to S0 indicating a wrong password. When FSM reaches S3, *Lock* goes high indicating a correct password entry.

Hopefully, it is now clear how the FSM continues until the entire combination has been entered correctly. The final event that moves the FSM into the unlocked state, S3, is when the *Key1* button is pressed, at which point the FSM will remain in that state until the lock is reset by pressing the 'Reset' button. The output logic of this FSM is quite simple. If the current state equals anything other than S3, 'Lock == 0'. This can be described with a simple **assign** statement. Complete the state diagram by adding the missing labels, including the output designations for states S2 and S3.

Now that you have a completed state diagram, use it and the example FSM implementation in the Background section to describe the combination-lock FSM in behavioral Verilog. The module interface

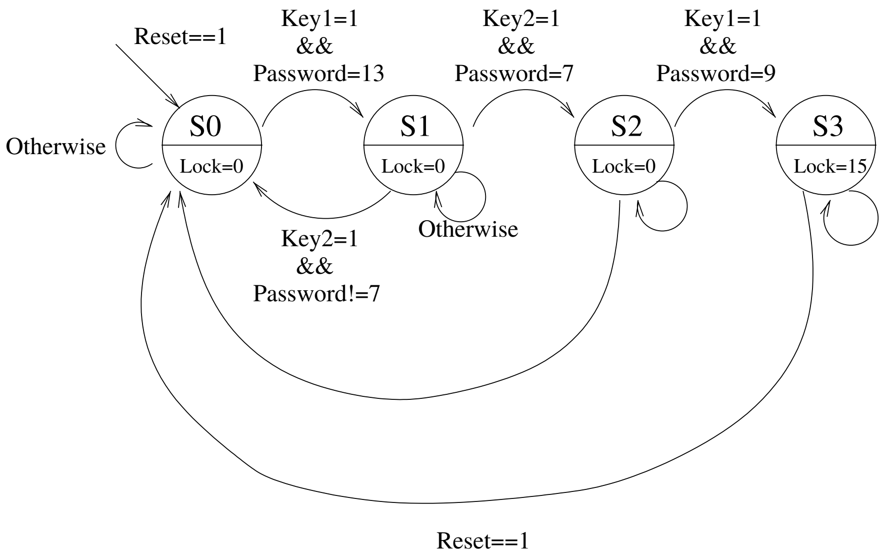


Figure 3: Rotary Combination-Lock State Diagram

below should get you started. Notice that we are exposing the state nets in the module interface. This is to aide in debugging! **Hint**: Next state logic should be triggered when 'Key' is pressed

```
/*This module describes the combination-lock  *
 *FSM discussed in the prelab using behavioral*
 *Verilog                                     */
module combination_lock_fsm(

    /*for ease of debugging, output state     */
    output reg [1:0] state,
    output wire [3:0] Lock, //asserted when locked
    input wire Key1, //unlock button 1
    input wire Key2, //unlock button 2
    input wire [3:0] Password, //indicate number
    input wire Reset, //reset
    input wire Clk, //clock
);
```

Note: You need a way to output to the LEDs to indicate the status of the lock (locked or unlocked). To do so, use the following line of code. This assign statement uses a **ternary operator**, which reads like "Is the current state s3? If so, assign Lock as 4'b1111, else assign Lock as 4'b0000". This also means that Lock should not be written to inside of your state machine logic.

```
assign Lock = (state == s3)? 4'b1111: 4'b0000;
```

3.2 Pre-lab Deliverables

Include the following items in you pre-lab submission in addition to those items mentioned in the *Policies and Procedures* document for this lab course.

1. Binary numbers for 13, 7 and 9.
2. The completed state diagram for the combination-lock FSM.
3. The combination-lock FSM Verilog module. Refer to the above sections to complete this.

4 Lab Procedure

The experiments below will take you through a typical design process in which you describe your design in Verilog, simulate it in Vivado, and synthesize it for the ZYBO Z7-10 board.

4.1 Experiment Part 1

The first experiment in the lab will involve simulating the combination-lock FSM module you described in the pre-lab to ensure it works properly before attempting to load it on the ZYBO Z7-10 board.

1. The following steps will guide you through the process of simulating the FSM that you created in the pre-lab.
   (a) Create a new Vivado project called 'lab10' and add your combination-lock FSM module to that project.
   (b) Copy the test bench file, 'combination_lock_fsm_tb.v', from the course directory into your lab10 directory.
   (c) Add the test bench file you just copied over to your Vivado project and simulate its operation.
   (d) Examine the console output of the test bench and ensure all of the tests passed.
   (e) Likewise, take a look at the simulation waveform and take note of the tests that the test bench performs. Is this an exhaustive test (ie. are all possible cases covered by our test)? Why or why not?

4.2 Experiment Part 2

For the final experiment, you will integrate the modules you simulated in the previous experiment with the synchronizer into a top-level module. You will then synthesize and implement the top-level module and load it onto the FPGA.

1. Synthesize and implement the combination-lock top-level module with the following steps:

(a) Copy over the following files from the course directory into your lab10 directory:
    - 'combination_lock.v', the top-level Verilog module
    - 'combination_lock.xdc', the XDC for the top-level
    - 'synchronizer.v', the synchronizer module for our asynchronous inputs

(b) Add the aforementioned files along with your combination-lock FSM module to your Vivado project.

(c) In "combination_lock.v", fix lines 9-11. The keyword "wire" is missing from these lines.

(d) Set the **combination_lock** module as the top-level module and kick off the synthesis process with "Generate Bitstream". Correct any errors with your design.

(e) Once your design builds without errors and warnings, load it onto the FPGA board.

2. With the combination-lock design loaded onto the FPGA, test its operation.

    (a) Run through the combination sequence shown in the pre-lab with the switches. All the LEDs should glow when the correct combination has been entered. Ensure the design operates as expected. If it does not, you may want to connect the logic analyzer up to JC (as in Lab 8) and walk through the state transitions of your design.
    Note: When your design works properly, demonstrate your progress to the TA.

3. At this point, we have demonstrated our prototype to our customer, and they liked what they saw. However, they mentioned that they would like to use the lock to secure sensitive company information and would require a more secure combination. Thus, we will modify our existing design to accept a four number combination instead of just three.

    (a) Modify the Verilog source to require the input of a fourth number of your choosing in the combination. This will require the addition of one more state in your FSM. Use **BTN1** to enter the new added password.

    (b) Re-synthesize and implement your design in Vivado.

    (c) Load the design on the FPGA and ensure that it works properly. As with the previous test, you may want to use the logic analyzer to aide in debugging. **Note:** When your modifications work properly, demonstrate your progress to the TA.

5 Post-lab Deliverables

Please include the following items in your post-lab write-up in addition to the deliverables mentioned in the *Policies and Procedures* document.

1. Include the source code with comments for **all** modules you simulated and/or implemented in lab. You do **not** have to include test bench code that was provided! Code without comments will not be accepted!

- 3-password combination lock code
- 4-password combination lock code

2. Include screenshots of all waveforms captured during simulation in addition to the test bench console output for each test bench simulation.
   - 3-password combination lock simulation waveform
3. Answer <u>all</u> questions throughout the lab manual.
   - Experiment Part 1, 1e
4. A possible attack on your combination-lock is a brute-force attack in which every possible input combination is tried. Given the original design with a combination of three numbers between 0 and 15, how many possible input combinations exist? How about for the modified design with a combination of four numbers?
5. **(Honors)** Include a state diagram for the modified combination-lock FSM with the four number combination.
6. **(Honors)** Another way to improve the security of our rotary combination-lock is to increase the range of input numbers. If we wanted to allow the user to enter numbers from 0 to 45, what changes would we have to make to our current design? In this case, what would be the total number of possible input combinations assuming a four number combination?

6 Important Student Feedback

The last part of the lab requests your feedback. We are continually trying to improve the laboratory exercises to enhance your learning experience, and we are unable to do so without your feedback. Please include the following post-lab deliverables in your lab write-up.
Note: If you have any other comments regarding the lab that you wish to bring to your instructor's attention, please feel free to include them as well.

1. What did you like most about the lab assignment and why? What did you like least about it and why?
2. Were there any sections of the lab manual that were unclear? If so, what was unclear? Do you have any suggestions for improving the clarity?
3. What suggestions do you have to improve the overall lab assignment?