University of California at Berkeley
College of Engineering
Department of Electrical Engineering and Computer Science

EECS 150
Spring 2002

Original Lab By: J.Wawrzynek and N. Weaver
Later revisions by R. Fearing, and J. Shih
Xilinx Foundation 3.0 version: Laura Todd
Xilinx Foundation 3.1 version: Mark Feng

Lab 5
Finite State Machine in Verilog

1 Objectives

You will enter and debug a Finite State Machine (FSM). Using our definition of the problem and logic equations specifying the FSM's operation, you will enter your in the HDL editor and simulate it with the logic simulator. **You are asked to successfully download the design onto the Xilinx board to get checked off**.

2 Prelab

- Complete your **IN1** (INput 1) and **IN2** (INput 2) blocks
- Write a **.cmd** (command) file to test your CLB (Combinational Logic Block).
- Write one single **.cmd** file with all the FSM test scenarios specified in the check-off sheet.
- Do as much as possible before your scheduled lab time.

You are building the controller for a 2-bit serial lock used to control entry to a locked room. The lock has a **RESET** button, an **ENTER** button, and two two-position switches, **CODE1** and **CODE0**, for entering the combination. For example, if the combination is 01-11, someone opening the lock would first set the two switches to 01 (**CODE1** = low, **CODE0** = high) and press **ENTER**. Then s/he would set the two switches to 11 (**CODE1** = high, **CODE0** = high) and press **ENTER**. This would cause the circuit to assert the **OPEN** signal, causing an electromechanical relay to be released and allowing the door to open. Our lock is insecure with only sixteen different combinations; think about how it might be extended.

If the person trying to open the lock makes a mistake entering the switch combination, s/he can restart the process by pressing **RESET**. If s/he enters a wrong sequence, the circuitry would assert the **ERROR** signal, illuminating an error light. S/he must press **RESET** to start the process over.

In this lab, you will enter a design for the lock's controller in a new Xilinx project. Name this lab "lab5". Make **RESET** and **ENTER** inputs. Simulate by pressing the **ENTER** button by forcing it high for a clock cycle. Use a two-bit wide input bus called **CODE[1:0]** for the two switches. (Information on how to use buses will be given later in this handout). The outputs are an **OPEN** signal and an **ERROR** signal.

Figure 1 shows a decomposition of the combination lock controller, whose inputs and outputs are:

Input Signal	Description
RESET	Clear any entered numbers
ENTER	Read the switches (enter a number in the combination)
CODE[1:0]	Two binary switches
Output signal	**Description**
OPEN	Lock opens
ERROR	Incorrect combination

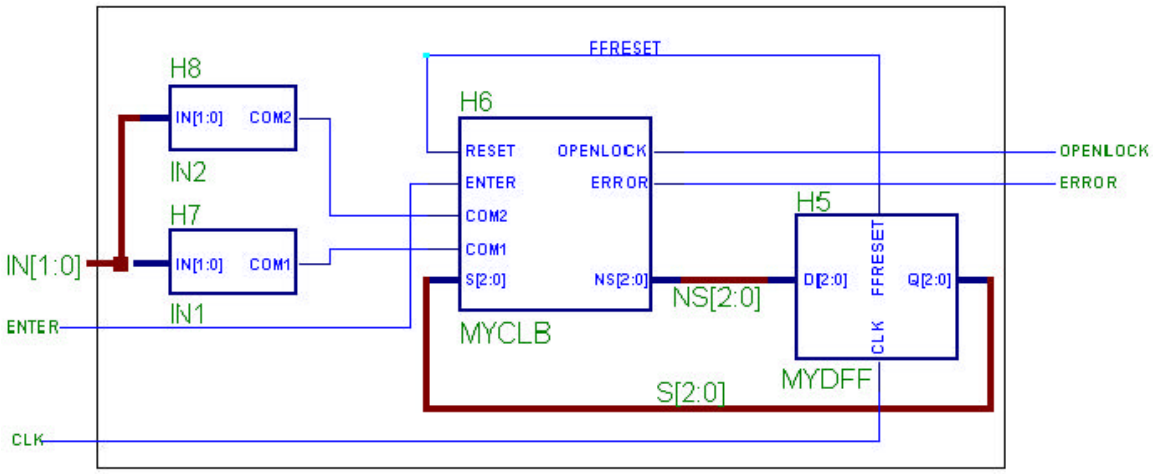

Figure 1: Controller for the combination lock

Figure 1 helps you to visualize your design.

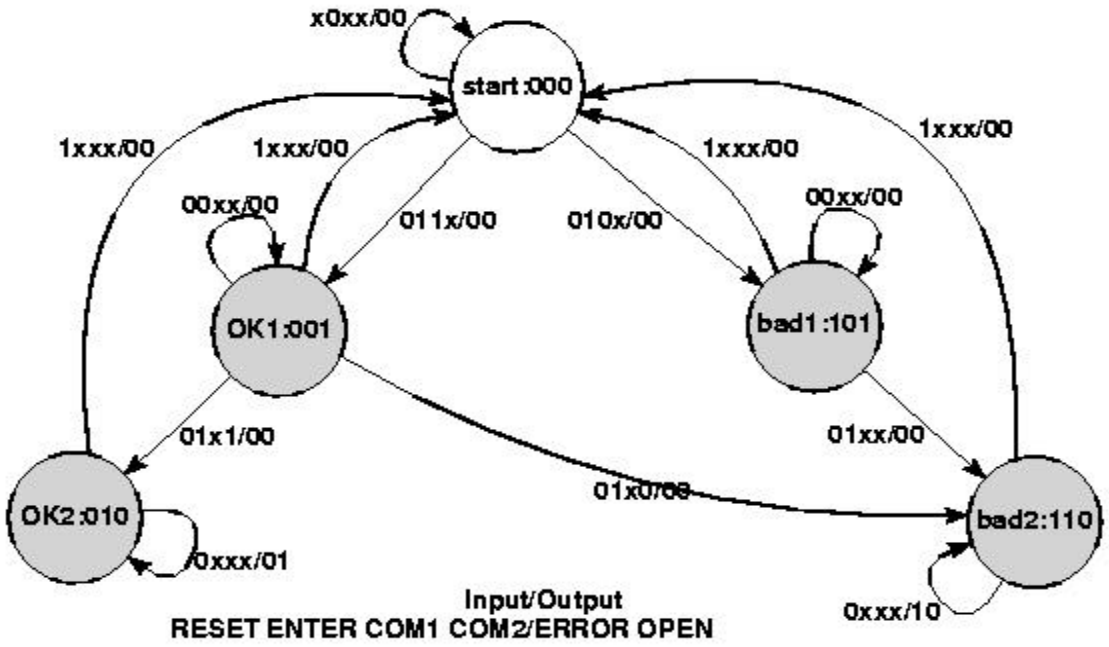

Figure 2: State Transition Diagram

3. Low-level specification

3.1 IN1 (INput 1) and IN2 (INput 2)

Blocks **IN1** and **IN2** process the input signals **COM1** (COMpare 1) and **COM2** (COMpare 2) into a simpler form for the FSM. Specifically, **COM1** is asserted when **CODE[1:0]** is the combination's first

number. Similarly, **COM2** is asserted for the second number. Partitioning the circuit in this way makes the combination easy to change. **Dipswitch[1] and Dipswitch [2] correspond to CODE[1:0].**

Choose your own combination; the two numbers must be different.

This should be a simple block. Use a few AND gates and inverters, but write them in Verilog.

3.2 MYCLB

The MYCLB (MY Combinational Logic Block) block takes **RESET**, **ENTER**, **COM1**, **COM2**, and present state and generates **OPENLOCK** and **ERROR**, as well as the next state. Figure 2 shows the state transition diagram, a Mealy machine since the transitions are labeled with both inputs and outputs. The white circle denotes the rest state (i.e., the state the machine starts in).

Here is a truth table for your FSM, although you may not need to use it for your lab.

RESET	ENTER	COM1	COM2	S[2:0]	NS[2:0]	ERROR	OPEN
1	X	X	X	XXX	000	0	0
0	0	X	X	000	000	0	0
0	1	0	X	000	101	0	0
0	1	1	X	000	001	0	0
0	0	X	X	001	001	0	0
0	1	X	0	001	110	0	0
0	1	X	1	001	010	0	0
0	X	X	X	010	010	0	1
0	0	X	X	101	101	0	0
0	1	X	X	101	110	0	0
0	X	X	X	110	110	1	0

Figure 3: Truth Table for the FSM

3.3 MYDFF

(MY D Flip-Flops)

Use MYDFF to store the states of your design. Think about how many FFs you will need for this project.

4. Tasks

Implement your design for MYCLB in Verilog behavioral model. This means you may not use gates for your design. Here are some important hints. Your MYCLB module is designed entirely in combinational logic, so you may not use clocks in your MYCLB module. Remember that in combinational logic, you can use the **always** statement followed by input signals on to model block behavior.

For example: for a 1 bit adder, you can say

```
always @ (a, b, c) begin
   a ^ b ^ c;
end
```

Since your MYCLB module is a combinational logic module with inputs: enter, com1, com2, s[2:0] you can say

```
always @ (enter or com1 or com2 or s)  begin
   Your code….
end
```

This means whenever one of the above signals change, the **always** block will be executed, and an output can be calculated. In this way, the **always** block works exactly the same way as the logic gates you have laid out in part 4.4. If you want, you can actually specify delays to simulate actual gate delays.

Now think about the purpose of your MYCLB block—it is used to calculate the next transition in the FSM, and return the correct output. The next state value is found through the current state value and the current input value; you can specify this in your MYCLB block.

For example

```
always @ (enter or com1 or com2 or s) begin
        case (s):
                3'b000:  begin          // state 000
```

```
            if (input = A or B)  begin
                    output = C;
                    nextstate = 3'b001;
              end
            end

            3'b101: begin          // state 111
              if (input = X or Y) begin
                    output = Z;
                    nextstate = 3'b111;
             end
            end
        ……
        default: nextstate = 3'b000;
        endcase
end
```

This example illustrates the power of behavioral HDL modeling. By using simple programming techniques you are able to describe the entire behavior of your FSM and later synthesize the design into gate level form.

Built 2 state machines using one-hot encoding and encoded states. Download your design and get checked off by your TA. You can reuse your design for the Com blocks and the MYDFF blocks. Make sure you create all 4 modules in different .v files and instantiate the modules in a final top.v file. Use lab5constraint.exc file as your constraint file. (import it during synthesis)

Finally, you will interface your design to the Xilinx board via the locktop.v file given to you. Once you download your design to the Xilinx board, the number LEDs will display which state you are in to help you to debug. The light LEDs will display **ERROR** or **OPENLOCK**. The right most LED correspond to **OPENLOCK** and the 2nd most right LED correspond to **ERROR**. The other light LEDs is always on.

4.6 Debounce your ENTER signal

In your future projects you will need to debounce certain **external** input signals. In this lab, you are asked to debounce the **ENTER** signal. Debouncing a signal means to assert a signal for only 1 clock cycle. If your clock frequency is 16MHZ and you press the **ENTER** button for 1ms, then your system might interpret the **ENTER** signal has been asserted for thousands of cycles. In our case, since you are not changing the 2-bit input, your design might think you have entered the same combination thousands of times. For your debouncer, if its input has been asserted for 2 clock cycles (this makes sure the input signal is not some random glitch), then it will debounce the signal in the next clock cycle. This means the debounced signal will assert on the positive edge of the 3rd cycle and set low on the positive edge of the 4th cycle. The output of your debouncer will be used to drive the rest of your lock as the **ENTER** signal. You can build this in via FFCEs and logic gates, but **make sure to ground your reset signal for the FFCE blocks**. Finally, add your code for the debouncer in locktop.v.

5. BUS

5.1 Definition and Usefulness

Buses are collections of ordered wires that (for one reason or another) were collected in a group for easy reference. Examples of busses include the two input bits of our combination lock (aka. IN[1:0]), the state and next state of our combination lock (S[2:0], NS[2:0]),or the memory busses for address and data in your personal computer.

Quite often, a bus's wires have similar purposes; the memory address bus in your PC is used to dictate which address in the memory the CPU would like to access. To do so, it needs to send a 32-bit integer to the memory. The easiest way to do so is to connect 32 wires from the CPU to the memory. Each wire corresponds to one bit of data.

Xilinx uses a thicker wire to denote a bus, and it uses a standard naming convention. The convention is:

NAME_OF_BUS **[** *number1* **:** *number2* **]**

The number of bits in the bus is determined by the numbering. A bus called S[2:0] will have 3 wires; a bus called DATA[31:16] will have 16 wires. From our memory address bus example above, if the data bus were called ADDR[31:0], the wires are numbered from 31 to 0, with the 31st wire being the highest-order bit, and the 0th wire being the lowest-order bit. Order of the number matters: if the data bus were called ADDR[0:31], the 31st wire being the lowest-order bit, and the 0th wire being the highest-order bit.

5.2 Using Buses in the Simulator

Bussing related signals makes the circuit easier to read and simulate. When using the command window or writing a command file using the script editor, as you did in lab1, writing:

```
vector data DATA[7:0]
```

(or: `v data DATA[7:0]`) makes the signals **DATA7**, **DATA6**, … **DATA0** into a vector called **data**, which can be treated like any other signal: you can watch vectors and set their values. Use the **`assign`** command to set a vector's value. E.g.: **`a data 3e\h`** (hexadecimal) or **`a data 001111110\b`** (binary).

6. Forcing Internal Signals

In addition to inputs, the logic simulator allows you to force internal signals, those normally driven by components, to particular values. This trick lets you set the state to anything you want. Simply set **NEXTSTATE[2:0]** to the state you want, clock the FSM, and then release **NEXTSTATE[2:0]**.

7. Clks

You can define a clock (an input signal that changes periodically) in your script file or in the command window. For example,

```
clock clk 0 1
```

makes the clock signal **clk** oscillate as the circuit is simulated. To simulate for a single clock period, use **cycle** instead of **sim.** Do not mix **cycle** and **sim**, as it can lead to some very interesting software behavior.

8. Command and Log Files

File → Run Script File… loads a script file and runs each line of it as if you were typing each of those lines in the command window. The most convenient way to create a script file is to use the **Tools → Script Editor**. If you want to use another editor you can, but if you do, make sure to save the file as plain text and that the file extension is **.cmd**.

To save a log of the commands you use in your session, you can create a transcript of your work using the command **log**. Start a log with log **filename.log**, and end a log by typing **log** alone.

9. Naming

- The Xilinx software, DOS, and Windows is case-*in*sensitive, somewhat, although we've used all caps for signals throughout this handout.
- You may use letters, numbers, and underscores (_) in filenames. The period (.) may only appear in certain places (e.g., **yourfile.cmd**, etc). Avoid other punctuation.

Name: _____ Name: _____

Lab Section (Check one)

M: ☐ AM ☐ PM T: ☐ AM ☐ PM W: ☐ AM ☐ PM Th: ☐ PM

Checkoffs: Lab 5

10.1 Design the **IN1**, **IN2**, and **MYCLB** blocks in Verilog (encoded states).

TA: _____(50%)

10.2 Design the **IN1**, **IN2**, and **MYCLB** blocks in Verilog (one-hot states).

TA: _____(50%)

Download the design onto the board and test the following scenarios for **10.1** and **10.2**: You should test your project in simulation first. Each of these scenarios should work in simulation.

(a) A sequence with the first combination number entered wrong.
(b) A sequence with the second combination number entered wrong.
(c) A sequence with the both combination numbers entered wrong.
(d) RESET is asserted after entering just the first number correctly.
(e) RESET is asserted after entering just the first number incorrectly.
(f) ENTER is asserted at state2 and state6, but state does not change.
(g) A successful entry of the combination.

10.3 Compare the number of CLB used by the encoded state and the one hot state machine.

TA: _____(1%)

10.4 Turned in on time

TA: _____
(full credit (100%))

10.5 Turned in one week late

TA: _____
(half credit (50% x points))