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

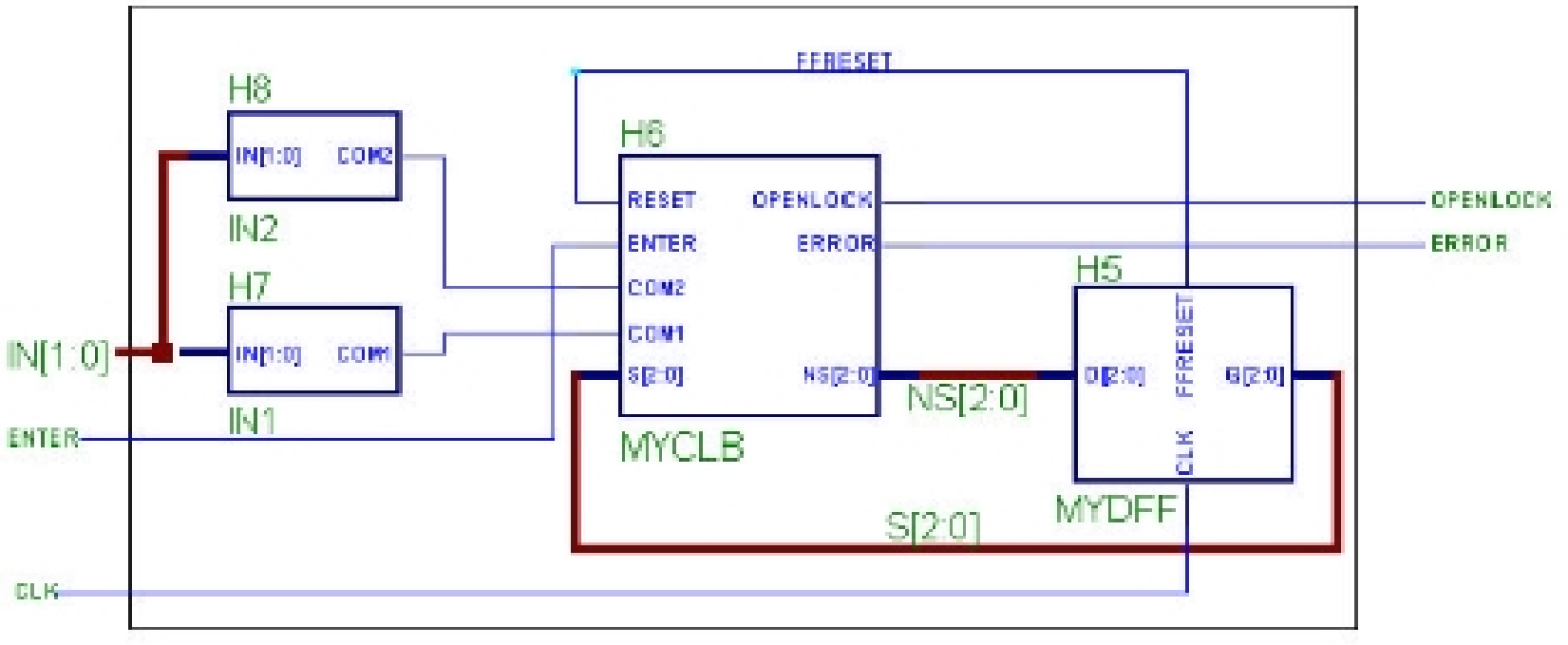

Figure 1: Controller for the combination lock

Figure 1 helps you to visualize your design.

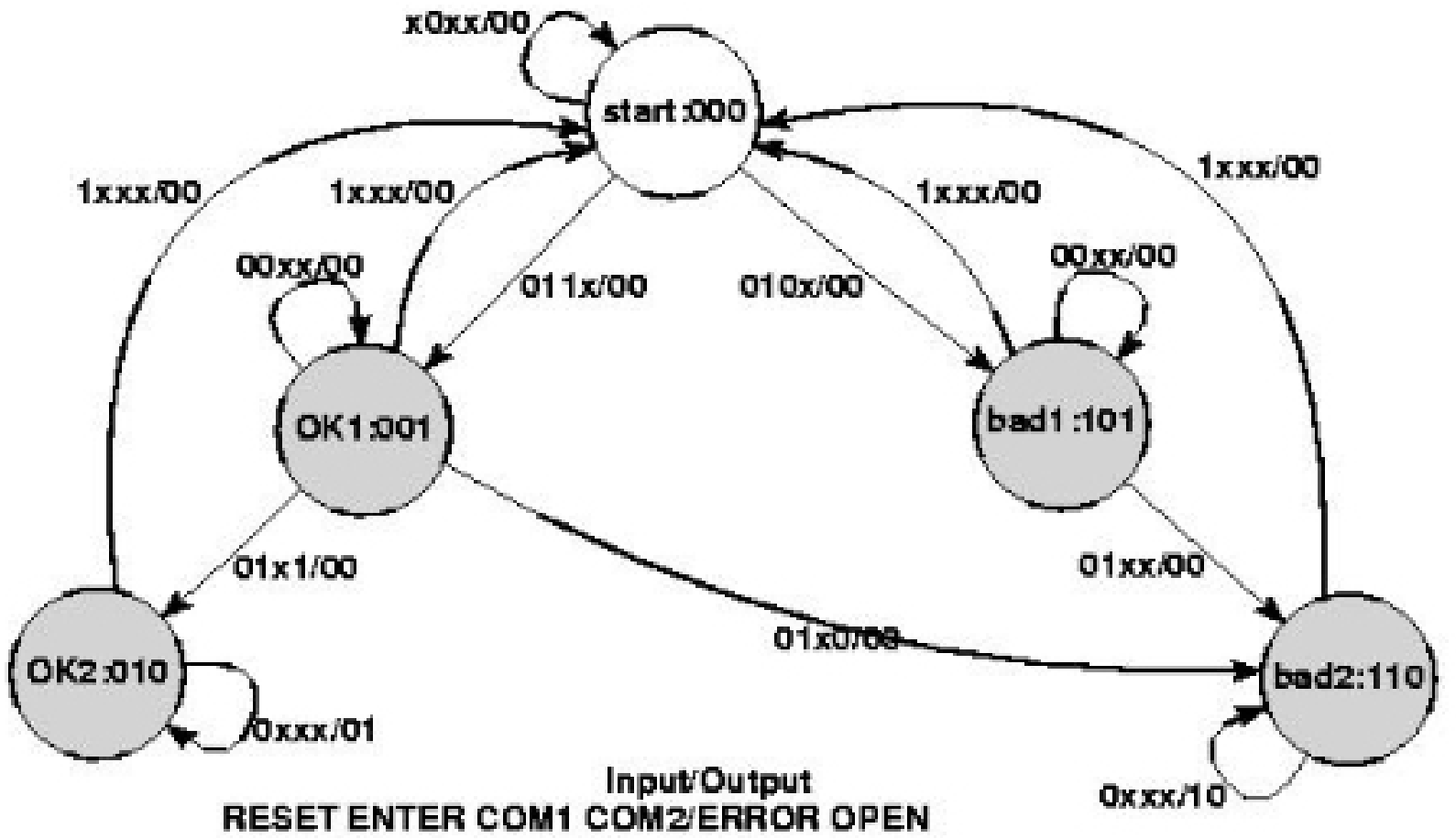

Figure 2: State Transition Diagram

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
```