Laboratory Exercise #11
A Simple Digital Combination Lock

Dilanka Weerasinghe

ECEN 248 – 502

TA: Pankaj Goel

Date: November 25, 2018

Objectives:

The purpose of this lab is to access and control a password. This requires several states that that we have to make for the password for the safe, or doorway to completed. We are creating a mechanism in the digital form on the ZYBO board. In this we are making a Moore machine in an FSM. This will hep us better understand how FSM work in Verilog and teach us how to code for a Moore machine in Verilog.

Design:

In this lab the very first thing we did was understand how the carry lookahead adder is designed before designing it in Verilog. For the new adder it has three main components. These are the generate propagate unit, the carry

In this lab the very first thing we did was understand the flow of this finite state machine. In this specific state machine, we know that there will be 3 passwords to input and when these passwords are input and a key is pressed it will transition to the next state. If the password or key is incorrect the state will return to the first state when nothing is correct.

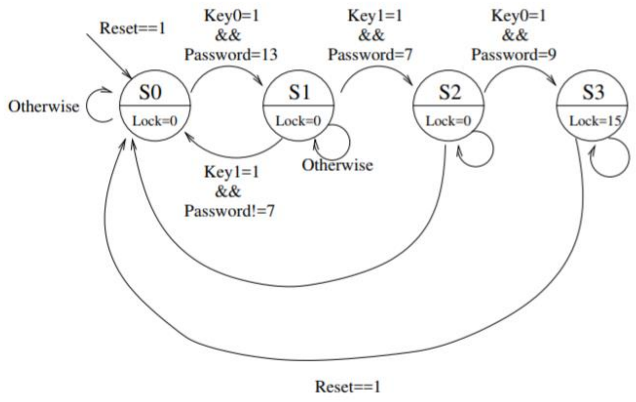


Figure 3: Rotary Combination-Lock State Diagram

This image to the left shows what will happen for each state. In each of these states there is a return and there is a next. In order to replicate these states, we must have several variables to create our digital lock. Given to us were the passwords that had to be used to complete the combination. These three passwords had to be changed into binary so that they could be input: 13, 7, 15. Before we did any of the results in this lab, we first created a Verilog module to begin.

Results:

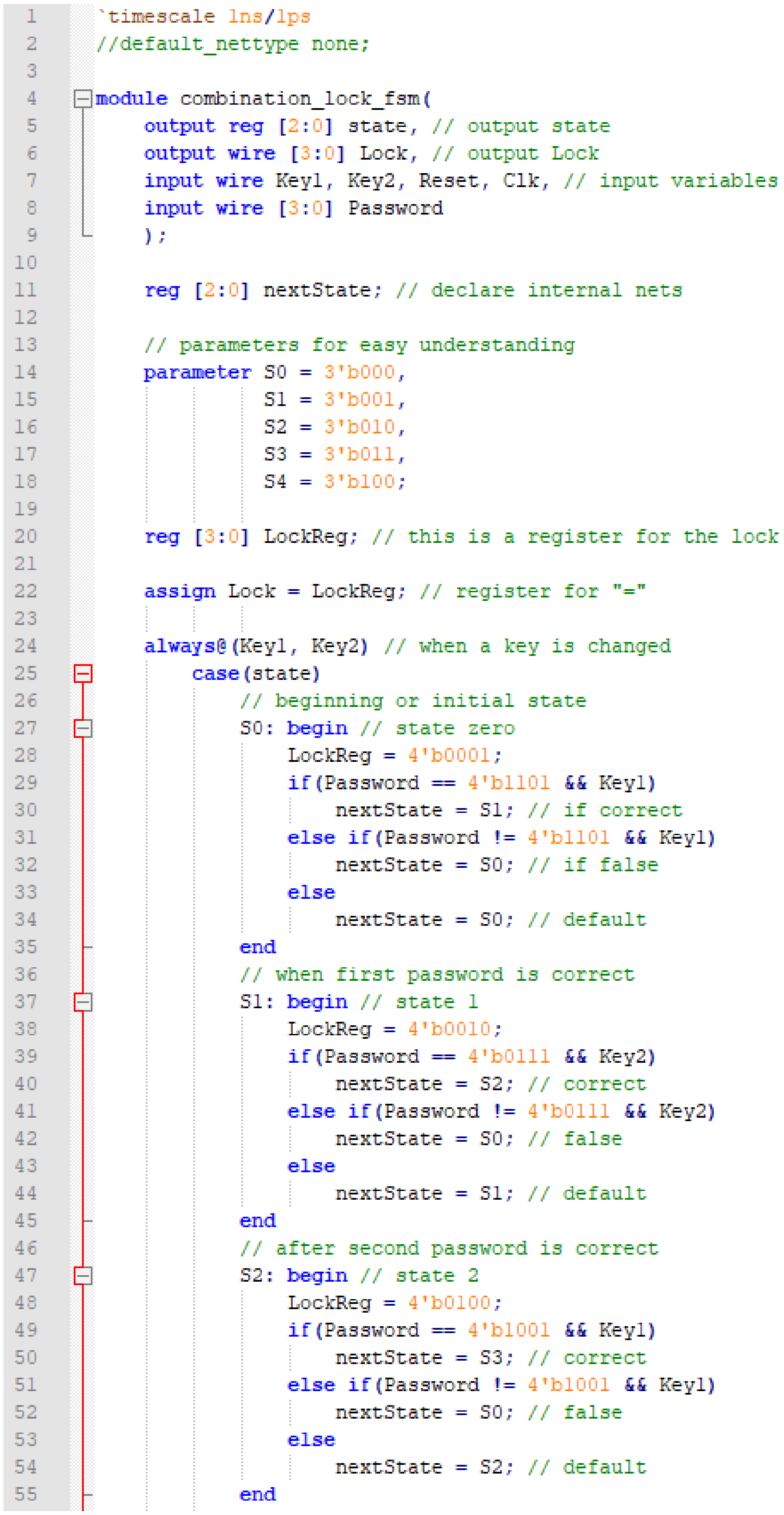
In the creation of this system there was little told on how to create it and a lot of the design and coding was done without instruction. Using what we had learned in class we constructed a Moore machine in Verilog that had an input for the password and for keys. As stated, before the password is input and a key is pressed to continue to the next lock. The first thing we did was test our code created previously and see if it would pass all of the tests. The testbench was not an exhaustive test as it did not test all the outputs and displayed true even when the code would only work in some cases. Through the testbench file given to us we completed the code in working order and continued to creating the bit generation. There was one major issue along the was in that one of the files given to us had an error that was difficult to find. In the top-level file there were no wires and the input for the key2 was set to key1. This spelled disaster for the demo and led to an incomplete product. As soon as this issue was fixed the code worked perfectly and fit the design to the letter. The next portion of the test was to include one more password. This required some significant changes because not the code required 5 states. This is more than our 2-bit state register could handle. Therefore, we changed the register to 3 bit and added the new state completing the design and finishing the lab.

Conclusion:

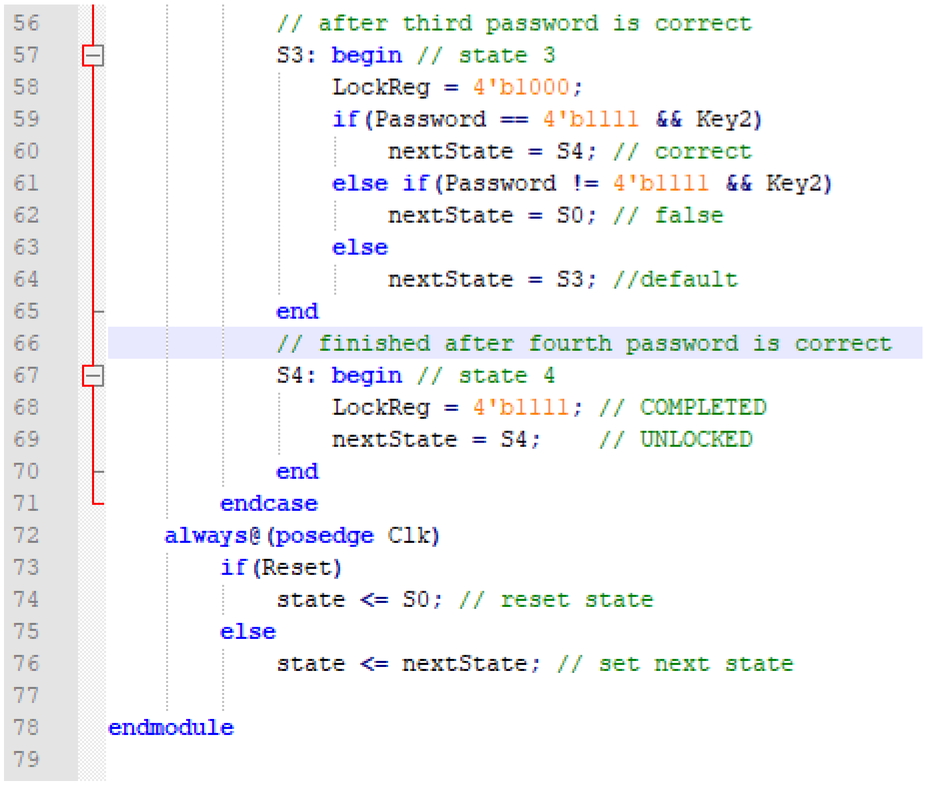
In the lab I implemented a 4-bit carry lookahead adder and a 2 level carry lookahead adder for 16 bits. Throughout the lab I tested using the test bench files and analyzed the logic given to us in the lab manual to understand the concepts. I added comments and subtle pseudocode after to make it easier to understand. This lab let me better understand to the code behaviorally in Verilog and more about how to create a significantly faster adder.

Questions: ** Question 3 is answered and highlighted in the results section.

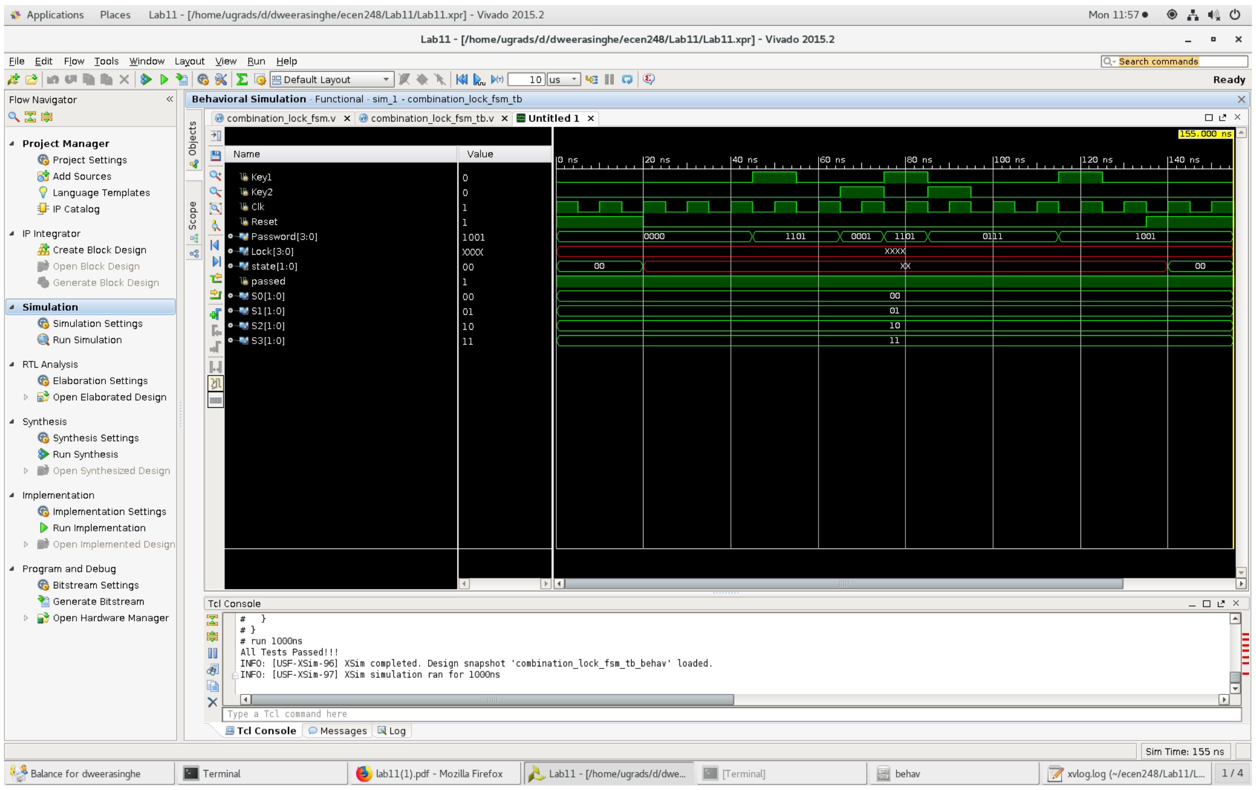
4) For a brute force attack on the 3-state system there are a total of 6750 combinations for a password. If the password has 4 states then there are 101250 combinations. Remember the keys are also included in the calculation for combos. (15*2)3^ = 27000; (15*2)^4 = 81000



```verilog
`timescale 1ns/1ps
//default_nettype none;

module combination_lock_fsm(
    output reg [2:0] state, // output state
    output wire [3:0] Lock, // output Lock
    input wire Key1, Key2, Reset, Clk, // input variables
    input wire [3:0] Password
    );

    reg [2:0] nextState; // declare internal nets

    // parameters for easy understanding
    parameter S0 = 3'b000,
              S1 = 3'b001,
              S2 = 3'b010,
              S3 = 3'b011,
              S4 = 3'b100;

    reg [3:0] LockReg; // this is a register for the lock

    assign Lock = LockReg; // register for "="

    always@(Key1, Key2) // when a key is changed
        case(state)
            // beginning or initial state
            S0: begin // state zero
                LockReg = 4'b0001;
                if(Password == 4'b1101 && Key1)
                    nextState = S1; // if correct
                else if(Password != 4'b1101 && Key1)
                    nextState = S0; // if false
                else
                    nextState = S0; // default
            end
            // when first password is correct
            S1: begin // state 1
                LockReg = 4'b0010;
                if(Password == 4'b0111 && Key2)
                    nextState = S2; // correct
                else if(Password != 4'b0111 && Key2)
                    nextState = S0; // false
                else
                    nextState = S1; // default
            end
            // after second password is correct
            S2: begin // state 2
                LockReg = 4'b0100;
                if(Password == 4'b1001 && Key1)
                    nextState = S3; // correct
                else if(Password != 4'b1001 && Key1)
                    nextState = S0; // false
                else
                    nextState = S2; // default
            end
```



```verilog
                // after third password is correct
                S3: begin // state 3
                    LockReg = 4'b1000;
                    if(Password == 4'b1111 && Key2)
                        nextState = S4; // correct
                    else if(Password != 4'b1111 && Key2)
                        nextState = S0; // false
                    else
                        nextState = S3; //default
                end
                // finished after fourth password is correct
                S4: begin // state 4
                    LockReg = 4'b1111; // COMPLETED
                    nextState = S4;    // UNLOCKED
                end
            endcase
        always@(posedge Clk)
            if(Reset)
                state <= S0; // reset state
            else
                state <= nextState; // set next state

endmodule
```



Lab11 - [/home/ugrads/d/dweerasinghe/ecen248/Lab11/Lab11.xpr] - Vivado 2015.2

Behavioral Simulation - Functional - sim_1 - combination_lock_fsm_tb

```
Name            Value
Key1            0
Key2            0
Clk             1
Reset           1
Password[3:0]   1001
Lock[3:0]       XXXX
state[1:0]      00
passed          1
S0[1:0]         00
S1[1:0]         01
S2[1:0]         10
S3[1:0]         11
```

Tcl Console

```
#   }
# }
# run 1000ns
All Tests Passed!!!
INFO: [USF-XSim-96] XSim completed. Design snapshot 'combination_lock_fsm_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
```

Sim Time: 155 ns