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

Laboratory Exercise #10

A Simple Digital Combination Lock

ECEN-248-509

Joaquin Salas

731000141

11-18-2022

TA: Sri Hari Pada Chandanam Kodi

Objectives:

A classic approach to access control is a password. A password can be found on safes, doorways, and all our devices and websites for subscriptions. Here we are dealing with physical security, not cybersecurity. This approach requires that a person wishing to gain access to a particular entryway enter the password. If the password is correct, the entryway unlocks, otherwise, it stays locked while conveying no information about the required sequence. This lab's purpose is to recreate this mechanism in digital form on the ZYBO Z7-10. To do this we will be looking into the Moore machine, a Finite State Machine (FSM). To prototype this combination lock, we will make use of the push buttons and LEDs on the ZYBO- Z7-10 board.

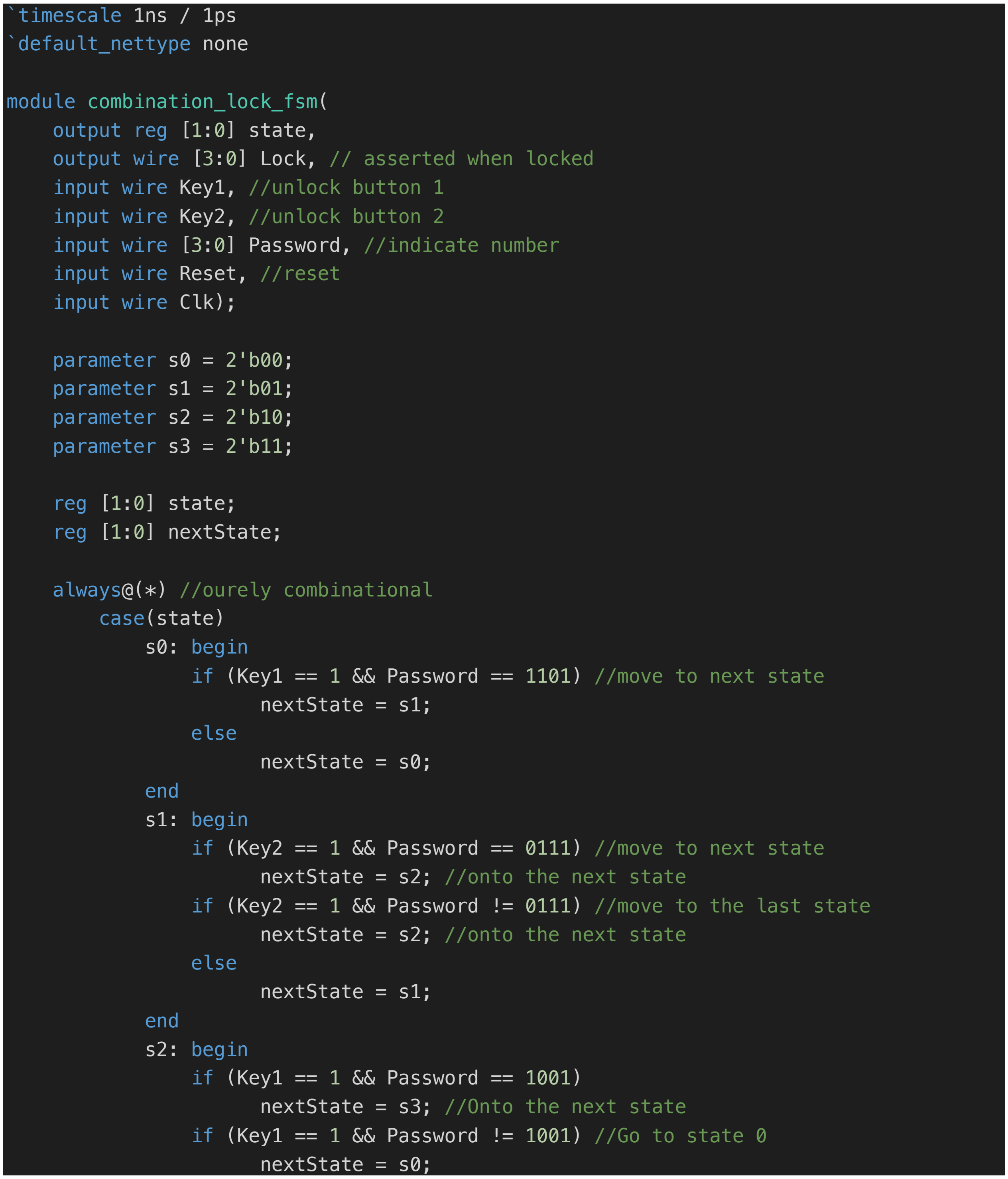
Lab Procedures:

The very first part of this experiment had us create a simulating combination-lock FSM module. This was completed in the prelab for this lab, to see if the combination lock finite state machine is correct, we were instructed to run the testbench through the Verilog code to make sure all tests have been passed. For the second part of this experiment, we were instructed to integrate the code from the previous experiment with the combinational lock FSM to upload the code to FPGA. To accomplish this, we began by running the synthesis on the combinational lock module with the .xdc file. After this is completed, we uploaded the program onto the FPGA. The FPGA is then used to see if the user's inputs would act as a secure combinational lock. After this, we will modify the program to take a 3-number password instead of a 4-number password.

Experiment Part 1:

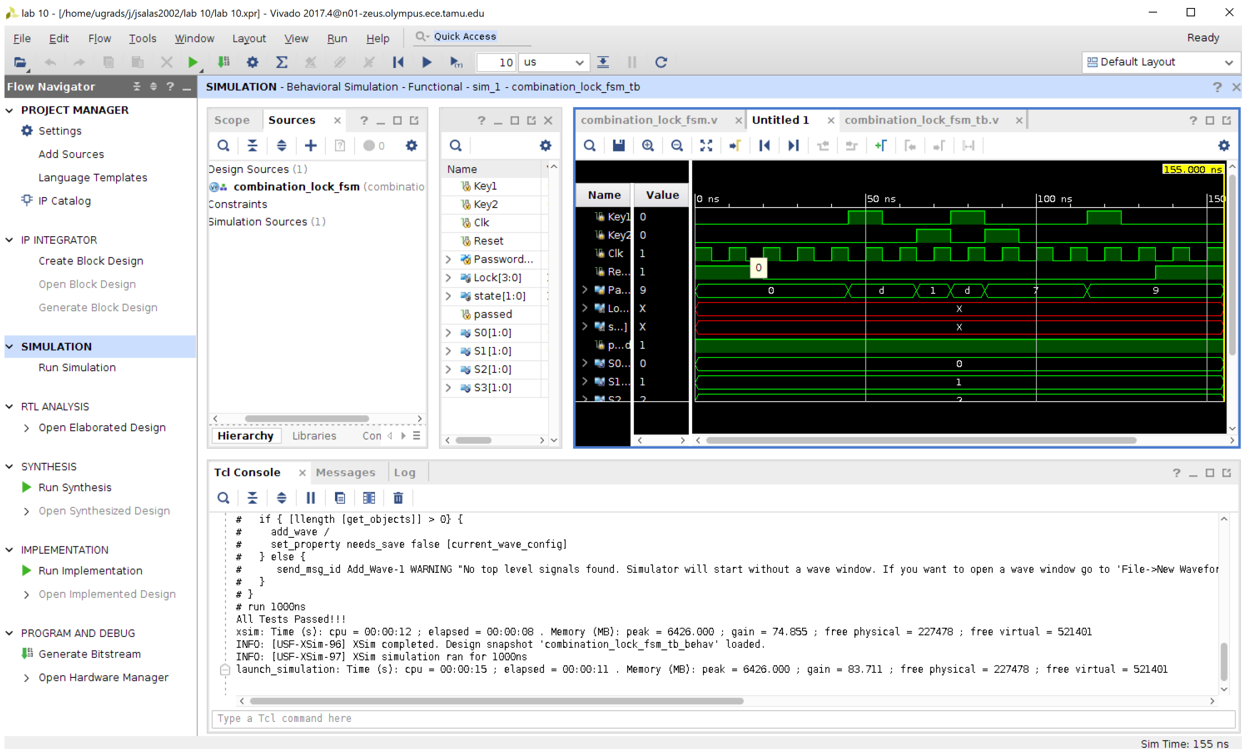
The first experiment in the lab will involve simulating the combination-lock FSM we created in the pre-lab. The results are shown down below and properly labeled.

Code: *combinational lock FSM.*



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

module combination_lock_fsm(
    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);

    parameter s0 = 2'b00;
    parameter s1 = 2'b01;
    parameter s2 = 2'b10;
    parameter s3 = 2'b11;

    reg [1:0] state;
    reg [1:0] nextState;

    always@(*) //ourely combinational
        case(state)
            s0: begin
                if (Key1 == 1 && Password == 1101) //move to next state
                      nextState = s1;
                else
                      nextState = s0;
            end
            s1: begin
                if (Key2 == 1 && Password == 0111) //move to next state
                      nextState = s2; //onto the next state
                if (Key2 == 1 && Password != 0111) //move to the last state
                      nextState = s2; //onto the next state
                else
                      nextState = s1;
            end
            s2: begin
                if (Key1 == 1 && Password == 1001)
                      nextState = s3; //Onto the next state
                if (Key1 == 1 && Password != 1001) //Go to state 0
                      nextState = s0;
                else
                        nextState = s2; //stay in 2nd state
            end
            s3: begin
                if (Reset == 1)
                        nextState = s0;
                else
                        nextState = s3;
            end
        endcase
    assign Lock = (state == s3)? 4'b1111: 4'b0000;
endmodule
```

Waveform: *4-number password waveform.*



```
#   if { [llength [get_objects]] > 0} {
#     add_wave /
#     set_property needs_save false [current_wave_config]
#   } else {
#      send_msg_id Add_Wave-1 WARNING "No top level signals found. Simulator will start without a wave window. If you want to open a wave window go to 'File->New Wavefor
#   }
# }
# run 1000ns
All Tests Passed!!!
xsim: Time (s): cpu = 00:00:12 ; elapsed = 00:00:08 . Memory (MB): peak = 6426.000 ; gain = 74.855 ; free physical = 227478 ; free virtual = 521401
INFO: [USF-XSim-96] XSim completed. Design snapshot 'combination_lock_fsm_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:15 ; elapsed = 00:00:11 . Memory (MB): peak = 6426.000 ; gain = 83.711 ; free physical = 227478 ; free virtual = 521401
```

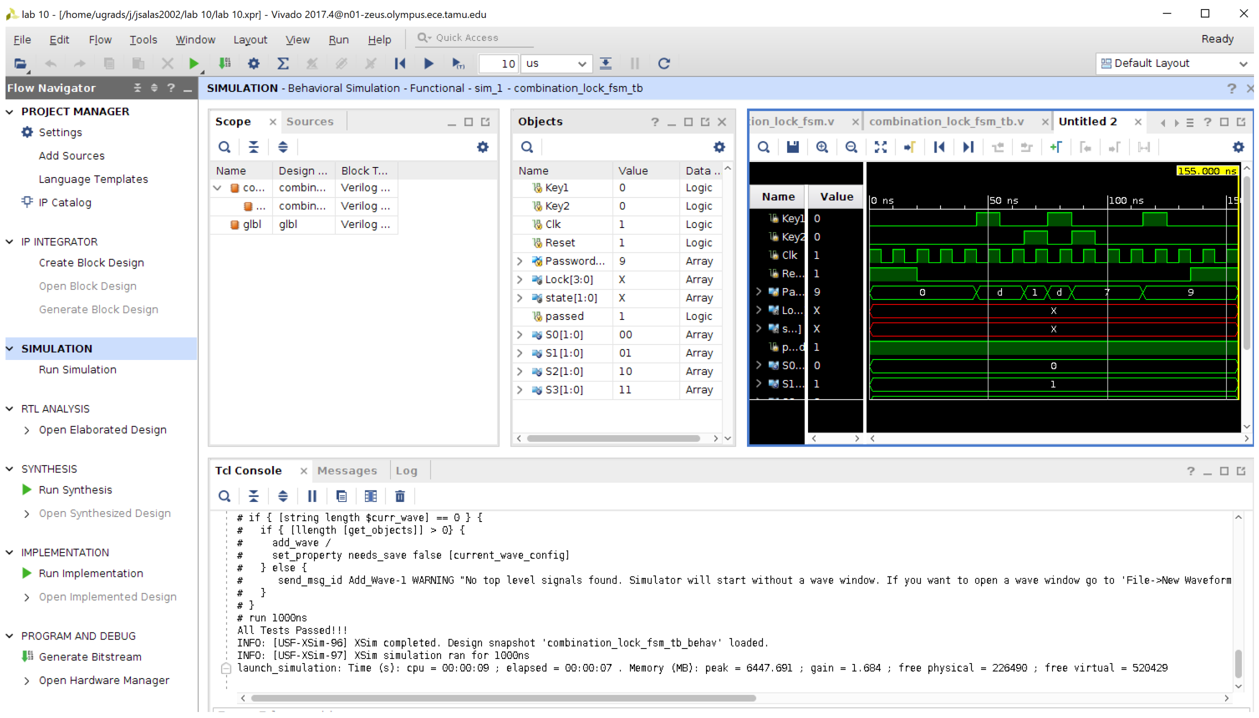
Experiment Part 2:

For the final experiment, you will integrate the modules we simulated in the previous experiment, with the synchronizer into a top-level module.

Code: *combinational lock FSM.*

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

module combination_lock_fsm(
    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
    );
    parameter s0 = 3'b000;
    parameter s1 = 3'b001;
    parameter s2 = 3'b010;
    parameter s3 = 3'b011;
    parameter s4 = 3'b111;

    reg [1:0] state;
    reg [1:0] nextState;

    always@(*) //ourely combinational
        case(state)
            s0: begin
                if (Key1 == 1 && Password == 110) //move to next state
                      nextState = s1;
                else
                      nextState = s0;
            end
            s1: begin
                if (Key2 == 1 && Password == 011) //move to next state
                      nextState = s2; //onto the next state
                if (Key2 == 1 && Password != 011) //move to the last state
                      nextState = s2; //onto the next state
                else
                      nextState = s1;
            end
            s2: begin
                if (Key1 == 1 && Password == 101)
                      nextState = s3; //Onto the next state
                if (Key1 == 1 && Password != 101) //Go to state 0
                nextState = s0;
            else
                nextState = s2; //stay in 2nd state
        end
        s3: begin
            if (Key2 == 1 && Password == 1111)
                nextState = s4;
            if (Key2 == 1 && Password != 1111)
                nextState = s0;
            else
                nextState = s3;
        s4: begin
            if (Reset == 1)
                nextState = s0;
            else
                nextState = s4;
        end
      endcase
    assign Lock = (state == s3)? 3'b111: 3'b000;
endmodule
```

Waveform: *3-number password waveform.*


lab 10 - [/home/ugrads/j/jsalas2002/lab 10/lab 10.xpr] - Vivado 2017.4@n01-zeus.olympus.ece.tamu.edu

SIMULATION - Behavioral Simulation - Functional - sim_1 - combination_lock_fsm_tb

Name	Design ...	Block T...
co...	combin...	Verilog ...
...	combin...	Verilog ...
glbl	glbl	Verilog ...

Name	Value	Data ...
Key1	0	Logic
Key2	0	Logic
Clk	1	Logic
Reset	1	Logic
Password...	9	Array
Lock[3:0]	X	Array
state[1:0]	X	Array
passed	1	Logic
S0[1:0]	00	Array
S1[1:0]	01	Array
S2[1:0]	10	Array
S3[1:0]	11	Array

```
# if { [string length $curr_wave] == 0 } {
#   if { [llength [get_objects]] > 0} {
#     add_wave /
#     set_property needs_save false [current_wave_config]
#   } else {
#      send_msg_id Add_Wave-1 WARNING "No top level signals found. Simulator will start without a wave window. If you want to open a wave window go to 'File->New Waveform
#   }
# }
# 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
launch_simulation: Time (s): cpu = 00:00:09 ; elapsed = 00:00:07 . Memory (MB): peak = 6447.691 ; gain = 1.684 ; free physical = 226490 ; free virtual = 520429
```

Results:

This experiment resulted in a series of waveforms depending on the Verilog code being tested.

Conclusion:

The lab allowed us the students to gain a better understanding of the simple combinational lock FSM. By using the Verilog code, we made a combination lock FSM. We used the FPGA board to determine if the combinational lock was true and worked properly, we then implemented the combinational lock to the ZYBO Z7-10 board and changed the code to accept 3-bit numbers instead of 4-bit numbers.

Post-lab Deliverables:

1. **Include the source code with comments for all modules you simulated and/or implemented in lab.**

The code and comments for all modules are shown above and properly labeled.

2. **Include screenshots of all waveforms captured during simulation in addition to the test bench console output for each test bench simulation.**

Provided above

3. **Answer all questions throughout the lab manual.**

• Likewise, take a look at the simulation waveform and note the tests the test bench performs. Is this an exhaustive test (ie. are all possible cases covered by our test)? Why or why not?

This isn't an exhaustive test since the brute force attack appears only when S3. All other states do not have brute force attacks.

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?**

The brute force attacks will take a significant time for both combinational lock FSM. There's 4096 possible combinations for a combination for three numbers, and 65536 possible combinations for a combination for four numbers.

Important Student Feedback

1. What did you like most about the lab assignment and why? What did you like least about it and why?

I enjoyed learning the difference between the three types of ways that code is written in Verilog, structural, dataflow, and behavioral, the pre-lab did take me a fair amount of time but I believe it was much worth the work. I did not like how the Zybo-10 board did not work with our computer at the end of the lab.

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?

All parts of the lab manual were very clear, except for the last part on experiment 3 where there was a crucial step that was left out to get the Zybo board to work correctly.

3. What suggestions do you have to improve the overall lab assignment?

I have no suggestions for improving the overall lab assignment.