ECE 3270

LAB 2 REPORT

June 13, 2022

Tim Driscoll
Clemson University
Department of Electrical and Computer Engineering
tdrisco@clemson.edu

Abstract

The main goal of lab two is centered around creating a Moore implementation of a finite state machine.The finite state machine was designed to detect a sequence of inputs. Ultimately the FSM was used to represent a simple digital lock. Upon receiving the desired sequence the output was to become a one which represented an unlocked state. The lab was designed in VHDL and mapped to both the FPGA and OpenCL.

1 Introduction

Lab two consisted of two main parts which both had different representations of the digital finite state machine. Part one combined to make up them majority of the lab and involved writing the VHDL code to implement lock. The lock was then mapped to the board. The overall design of the lock used a series off case statements combined with a series of if statements. The case statement worked as a representation to determine which one of the nine states was the current state and the if statement determined which was the next state based on the inputs. After the design was mapped to the board it was uploaded to one of the lab machines and tested. After the initial VHDL design was created an thoroughly tested using both test benches and the actual board implementation part two could be completed. Part two involved mapping the original design of the lock to a given OpenCL project. After mapping the design to OpenCL the executable created through this process was saved to a micro-SD card and used with the FPGA to test the design.

2 Design

2.1 Lock State Diagram

The first step in completing this lab was to create a state diagram that represented the digital lock. The state diagram could be drawn after understating all the major design constraints. The lock was to be designed to detect the sequence depicted in Table 1. Along with detecting this sequence there was a few other design specifications that needed

Table 1: Sequence to Unlock

Input #	Input	Binary Input Representation
1	a	10
2	0	00
3	b	01
4	0	00
5	a	10
6	0	00
7	a	10

to be accounted for when making the state diagram. The first major design choice made was to consider the unrepresented input (11) as an invalid key on the lock and have it be ignored in the state diagram. This means that at any given state besides the last two the input (11) would have no effect. The reason that the input (11) effects the last two states is because of another design specification. The Last two states were the only two states with an output of one which means they represented the unlocked state. Once the lock was in an unlocked state it would re-lock (go back to initial state) after either of the inputs went high. Thus a (11) would represent one of the inputs going high and have an effect on the state. The other major design point in the state diagram is that when an out of sequence input was received the current state wouldn't necessarily reset to the initial state. The current state would get set to the last occurrence of the input. For example if a b was detected as input # seven then the current state would get set to the state that represents receiving a single a and b. The last design point that can be noted in Table 1 is that in between each correct input is a cycle where both the inputs are required to be low (00). When accounting these design constraints the state diagram represented by Figure 1 can be created.

2.2 Lock VHDL Implementation

After creating the state diagram the generic VHDL implementation came relatively easy because it consisted of simply describing the state diagram in VHDL code. The overall design and flow of the code consisted of defining all nine states as a new TYPE call fsm_state, then creating single signal to represent the current state. Then a new process was created to check for changes in both the clk and rst. The clk and rst are the only two values that should trigger the state machine to update. Within this process there was overarching if statement to determine if there was a rst (active low), the else if was there to then determine if ivalid was high and if there was a rising edge on the clock. If both of the last two conditions were met the state would be updated by the current state as specified by a Moore FSM design. A case statement was used to determine what the value of the current state was and with in each case was a series of if statements to determine the next state based on the inputs (a/b). After this process was ended there was simple when-else statement to determine if the output should be brought high or remain low.

2.2.1 Lock VHDL Testing

In order to initially test the design of the lock a test bench was created to test for a series of different conditions. First by looking at Figure 2 a test bench was created to simply show the results of the lock if the correct sequence on the first try. It exemplifies the initial use of reset to make sure the current state is is set to the first state upon starting. It then tests ivalid to make sure no data is saved when ivalid is low. Lastly the test bench tests the final condition for the lock to re-lock (output low). As shown in the test bench the output is held high for a couple clock pulses even after an input transition. The output shows holding high until one of the inputs goes high. Then next test bench, which is represented by Figure 3 shows how the lock handles an incorrect sequence. The sequence that is tested is a(00)b(00)a(00)b where the last b should have been an a. Based on the design of the state machine this will result in the current state being set to the state after the sequence of a(00)b. This means for any given invalid sequence the state machine will reset back to the last valid occurrence of that input. The test bench produces an output of one after the rest of the sequence is correctly inputted ((00)a(00)a).

2.3 Lock FPGA Implementation

The next step in the lab was to map the generic lock implementation to the FPGA. This was done by creating a new VHD file where a lock_board entity was created and in the architecture a lock component was defined. The inputs and output from the lock component were then mapped to the I/O on the FPGA. In the design the two data inputs a and b were mapped to switches 1 and 0 respectively. The rst and ivalid inputs were then mapped to switches 3 and 2 respectively. The last input which represented the clock was mapped to key 0. Lastly the output was mapped to single LED where off represented locked and on represented unlocked.

2.3.1 Lock FPGA Testing

The FPGA implementation of the lock was tested through the process of compiling the VHDL code and uploading it to the FPGA in lab. Once loaded on the board exhaustive testing similar to that described with the test benches in the above section 2.2.1.

2.4 Lock OpenCL Implementation

The final step in the lab which represented part two was to map the lock implementation to OpenCL. This was done by downloading the given OpenCL project folder and navigating through the file structure to find the lock_cl.vhd file where the lock_cl entity was already defined. In this file it was only necessary define a lock component within the architecture and port map the lock to OpenCL. In the design inputs a and b were mapped to datain(1 downto 0) respectively. Then ivalid, clk and rst were mapped to ivalid, clock and resetn respectively. Lastly the output was mapped to dataout(0). Once this file was completed the original lock.vhd file was copied into this folder and the lock_cl.vhd file was added to the lab 2 VHDL project. The next step involved editing the xml file in the OpenCL project folder. This was a simple step that involved adding the lock.vhd file to requirements section of the file. After these edits were made the make script could be run to compile the OpenCL code. After a successful compilation the Direction executable and lock_cl.aocx file could be copied onto a micro-SD card. The SD card could then be ejected and inserted into the FPGA so the OpenCL implementation could be tested.

2.4.1 Lock OpenCL Testing

The OpenCL implementation could be tested through a terminal on one of the lab machines. After completing the necessary software setup the Direction executable test file could be run in order to test the OpenCL design. When the executable file was ran the user was given the options to input an a, b as data inputs or q to quit. It was assumed by the OpenCL design that there was a (00) input between each a or b inputted. When the code receives the sequence abaa it would produce the output "Unlocked" then the next input would reset the sequence detection. Similar tests were run as those implemented by the second test bench 3 in order to verify functionality.

3 Conclusion

The main goal of this lab was to design a Moore finite state machine that would represent a sequence detecting digital lock. This main goal was then applied to two different implementations, both the FPGA directly and OpenCL. Through the basic steps of carefully designing and drawing a state diagram the goal was achieved. After the state diagram to detect the sequence represented in Table 1 there was a seamless transition into the VHDL implementation. Then through testing a two more abstractions of the original lock implementation the final goal was achieved. The final goal represented implementing the lock in OpenCL and testing it's functionality on the FPGA.

The lab provided a great learning experience and allowed for an application of many of the topics that were discussed during lecture. Some of the main lessons that were learned throughout this lab included how to implement a finite state machine in VHDL, use some of the other built in tools included in VHDL and lastly interact and work with the OpenCL implementation. This lab showed the effectiveness of taking time to plan out and design prior to completing any coding in order to minimize errors and provide a clear direction to completeness. Throughout this lab the only errors that occurred were minimal syntax bugs that were easily fixed, and the occasional adjustment to the flow of states. Overall the lab provided a great learning experience where. The final goal was obtained and the lock correctly unlocked upon receiving the correct sequence.

4 References

M.Smith ”Digital Computer Design” Clemson University Holcombe Department of Electrical and Computer Engineering, January 2020

M.Smith ”ECE3270 Digital System Design Lab 2: State Machines” Clemson University Holcombe Department of Electrical and Computer Engineering, January 2020

W. Daily, R. Curtis Harting and T. Aamodt, Digital Design Using VHDL a systems approach. Cambridge University Press

5 Appendix

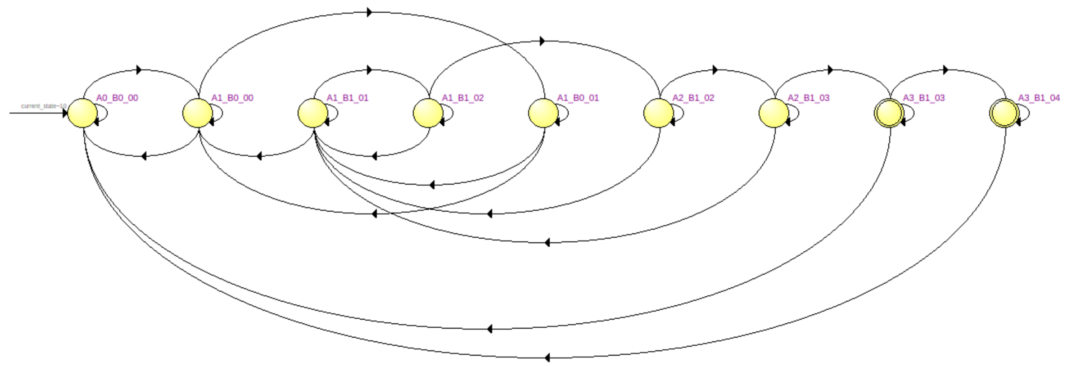
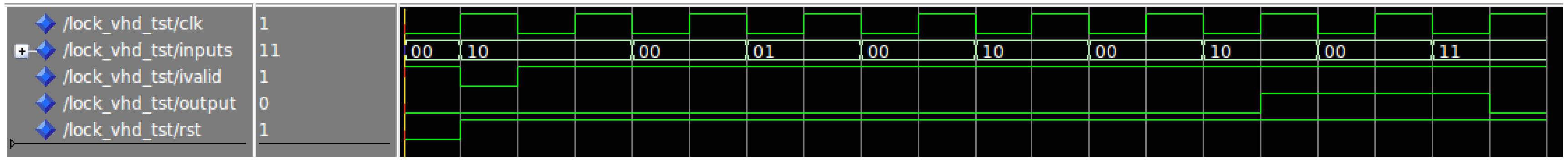


Figure 1: Lock State Diagram



/lock_vhd_tst/clk	1
/lock_vhd_tst/inputs	11
/lock_vhd_tst/ivalid	1
/lock_vhd_tst/output	0
/lock_vhd_tst/rst	1

Figure 2: Lock Test Bench (Correct Sequence)

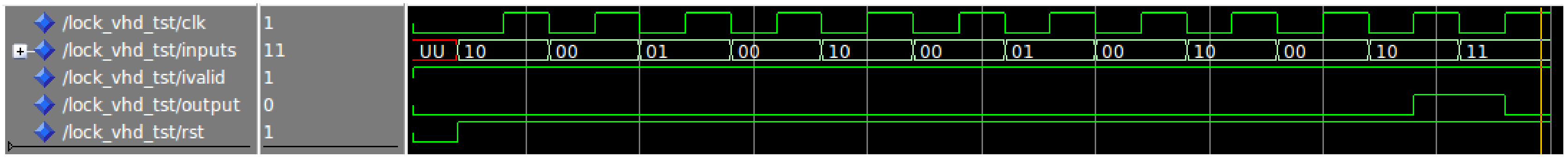


Figure 3: Lock Test Bench (Correct Sequence)