California State University, Northridge
ECE 524L - Advanced FPGA Design
Fall 2022

Lab 8: Digital Lock FSM Design

November 18th, 2022
Instructor: Saba Janamian

Table of Contents

Introduction:

The purpose of the experiment was to improve our knowledge of finite state machines that we had learned in ECE 320 and ECE 420. In the two previous classes we learned the basic concepts of what an FSM is and learned how to design it using VHDL. In this class and with this lab we will refine our understanding of FSMs and learn how to implement the concepts using an FPGA board. Unlike in ECE 420, this lab will have more states to work with as well the use of buttons that when a specific button is pressed a different state will occur.

Procedure:

To implement this lab we will need at minimum two modules. One module will be a top module which will implement the state diagram provided and the second module will implement a debouncer. We need the debouncer because we will be working with buttons and we need to make sure that we have a way of capturing the input correctly. Because the FPGA has a 125MHz clock it can accidentally capture the same button multiple times instead of just once.

The debouncer module will be following a design similar to a digilent or digikey designs that we have used in other labs. We are making some slight modifications to signals that we will create. It will still follow the same basic principles of the debouncers we used previously where we will have two flip flop registers that will be XOR together.

To set up the top module, we need to instantiate the debouncer module four times. We do this because we need to debounce all four of the button inputs. From that we create an instantiation of an edge detector which will detect if a specific button was pressed or not. The edge detector requires a register and pulse signal. A pulse signal is generated when the button is pressed and the register captures that signal and it gets captured at the rising edge of the clock.

Next a FSM process is made to create the state machine and implement the given diagram. This process block will handle which leds need to be illuminated based on the state, which is determined when a button is pressed.

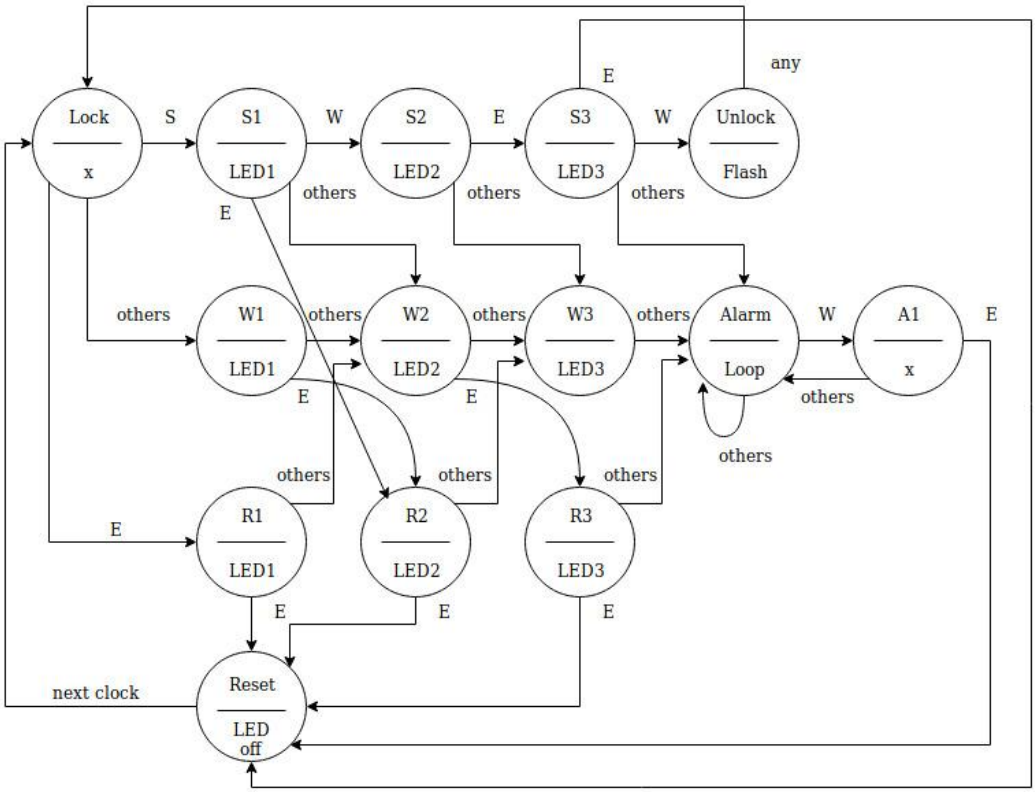

Fig 1. Given FSM Diagram

The FSM diagram was given for this lab. The “Lock” state is the starting point of the state machine diagram. For S1-3, W1-3 and R1-3 we want a specific led to illuminate. For example, when a button is pressed, regardless if it is part of the correct sequence or not, we want the first led to illuminate (Assuming it's the very first button pressed). And when the second button is pressed, the next led 2 is illuminated and so on. When the state machine is in state Alarm or Unlock we want the system to be in a loop. We want it to be in a loop so that an led pattern can be displayed multiple times and keep displaying for a long time. If any button is

pressed in the unlock state, return the state machine back to the locked state. Essentially acting as a reset feature without resetting prematurely.

The states are broken down into categories. Firstly, there is the “S” or Safe or correct category where if the user enters a correct button it moves to the next correct state. But as soon as the user enters a wrong button it moves to the “W” or wrong state. In the wrong state, the user can enter E twice to reset their inputs or will keep entering the wrong button combination and enter the alarm state. The user will not know that they are in the correct state or wrong state until a specific led pattern is displayed. The “R” or reset state is where the user presses E to reset the system or their inputs. The A1 state is an extension of the alarm state where if the user presses W and then later E to get them out of the alarm state entirely and reset the system.

Conclusion:

In this lab we learned further implementations of FSMs and learned how to apply them to an FPGA. I learned how to design if/else blocks where in each section we weren’t just limited to possible two states like in ECE 420. Each of the if/else blocks allowed the machine to choose from three to four possible states. While there were a couple of small bugs like pressing E during the Alarm state reset the whole system making it so that pressing W and E was a waste. Pressing W and E like the originally intended would still reset the Alarm state. The other small bug was when you pressed any two buttons, for example N + S and then did E + E, it would enter an alarm state. Apart from that the system worked as intended.

Appendix:

Github: https://github.com/csun-ece/fa22-e524-lab8-troykerim

Youtube: https://www.youtube.com/watch?v=1_UPpySfc7k