Don’t Forget Your Bits

Combinational Lock

Authors (Zachary Hill, Zachary Jump, Aidan Gallagher, Stephen Hayes)
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: zhill@oakland.edu, zjump@oakland.edu, aidangallagher@oakland.edu, stephenhayes@oakland.edu

***Abstract*—This combinational lock project will take a combination of three initial four bit inputs (four numbers/letters 0-9 and A-F) from the user and store it as the unlock sequence. The user inputs this combination with the sixteen switches on the Nexys A7. In the “set-up” state of the lock, a button (BtnL) will store the user's combination. During the unlocking state, the user will attempt to unlock the combination using the same switches and the RGB LED will light up red if the safe is still locked and green if the user has inputted the correct combination. A reset button will clear the user’s combination and restart the circuit. In each state of the lock, each set of 4 switches will be displayed as a hexadecimal value on the 7 segments displays. As a general rule of this system, the user will control the states themselves using the far left switch on the FPGA board. Once their combination is made and BtnL is pushed, the user is recommended to set all the switches back to the down position before changing the state switch (SW15) to the guessing stage.**

I. Introduction

The scope of this project is to design and demo a combinational circuit. The project uses synchronous circuits through VHDL implementation. Input feedback and validation outputs are generated through the combinational circuit for user interaction.

Our team wanted to design a fundamental circuit design that covers the most frequently used implementation techniques. Some of the essential components include decoders, a demultiplexer, a multiplexor, a comparator, a counter,a register, and two Finite State Machines.

The application of our circuit is to model a lock for a safe that is accessed with an FPGA. A user may choose to either memorize a sequence of up and down switches (these switches will represent a hexadecimal number) or they may choose to use a combination which reflects a sequence of numbers (0-9) and letters (A-F). We are employing three seven segment displays to show the user what combination of numbers and letters they are inputting. This will allow a user who does not know hexadecimal to be able to keep track of their combination or their guesses at the combination. The color of the RGB LED will dictate if the sequence was inputted correctly or not.

On the application and digital level, our circuit can be split into three distinguishable states. Our circuit is representative of what we learned in our labs and lectures throughout the semester, as well as topics found in Daniel Llamocca’s website[1]. Our design is centered around a 12-bit register, a component that is very familiar to students that have taken ECE 2700, and the 12-bit comparator, which draws the connection between the combination selecting and combination unlocking states of the design. One of the larger challenges of the project was determining how to get all three of the seven segment displays to show different numbers representing the respective four switches below them, all updating in real time. This was done by utilizing a “serializer” design, which consists of a counter, FSM, multiplexor select, and a 2 to 4 decoder[3]. The function of a serializer is to switch the select of a multiplexor every millisecond while also enabling the multiplexor input’s respective seven segment display. Since all of the seven segment displays on the FPGA board are all controlled by the same output, the serializer essentially changes all of the displays on the board every millisecond, but only enables one at a time so that the user has the illusion of a real time display of what the switches on the board represent. The project will utilize several parts of the FPGA board. 3 of the seven segment displays, 13 of the switches, and 2 buttons will be implemented to facilitate the user’s interactions with the combination lock.

II. Methodology

A. *Component Gallery*

As mentioned in the introduction, this project’s design is composed of a register, hexi-decimal to seven segment display decoders, a 2 to 4 decoder, a counter, two FSMs, logic gates, a comparator, a demultiplexer and a multiplexor. The function of the register in the design is to store the input from the user in the combination selecting phase[4]. The 12-bit register is enabled by a switch of input “E” where a value of ‘1’ will let the combination of the user pass through into storage upon the press of BtnL (“E” and “BtnL” are inputs to an “AND” gate and the output of this gate is the input to the register). Once the user has stored their combination and the enable switch is flipped to the down position, the RGB LED will light up red. The 12 bit comparator will be used to compare the 12 bits representing the user-chosen combination and the 12 bits representing the attempt at unlocking the safe. The comparator outputs a ‘1’ when the bits are all equal and a ‘0’ when the bits are not all equal. The comparator is also enabled by a signal that is the

enable switch “E” going through a not gate so that the comparator will operate when the register is no longer enabled and vice versa. When the comparator is off it will output a ‘0’. The demultiplexer is used to direct the top file input “X” to either the register’s input “D” or the comparator’s input “x”. The demultiplexer select is controlled by the enable switch “E” (SW15). The multiplexor (3 to 1) serves alongside the 2 to 4 decoder, Hex to 7 segment decoder, counter, and FSM to act as a serializer so that the user may keep track of the hexadecimal representation of the switches.

B. Component Interactions

The biggest variable in this design is the output of the far left switch, “E”. This output, when on, will allow a combination from the user to pass into the register at the instance of a button push (btnL), controls the select to the demultiplexer, enables the comparator, and serves as an input to the RGB FSM. Top file input “X”, which is equal to the values represented by the first twelve switches on the FPGA (SW0-SW11), is the input to the multiplexer and demultiplexer, which eventually travels to the register and comparator. The output of the comparator is also the input to the RGB FSM, whose output is the top file output “RGB” which dictates which color shines on LED16. To describe the serializer structure in detail, it begins with the counter, which is designed to output a ‘1’ every millisecond. The counter receives top file inputs “clk” and “resetn” just as the register and both Finite State Machines do. The output of the counter also serves as the input enable to the serializer’s FSM. This FSM’s output “sel” serves as the input select to the multiplexer and the input to the 2 to 4 decoder. As the FSM changes the “sel” every millisecond, the multiplexer will cycle through which input (x_0, x_1, x_2) goes into the Hex to 7 segment decoder and then to the top file output “R”, which dictates what value shows up on the seven segment display.

C. Finite State Machines of the Design

Two Finite State Machines (FSMs) are used to control the RGB LED FSM and the Seven Segment FSM(s) [2]. Both of the FSMs consist of three states which direct the components behaviors and actions. The RGB LED FSM consists of a transition influenced by “resetn”, “clock”, “s”, and “SW15”. The output is the RGB value. In state one, the FSM in setup mode which turns the LED off “000”. During state one “SW15” is the enable switch. If “SW15” is equal to one, we return back to the start of state one. However, if “SW15” is zero, we move into state two. In state two, the circuit is in a locked state awaiting for the correct user combination and will display a red LED “100”. This is controlled by “s” which is the output from the comparator. While “s” is valid, we remain in state three, else we return to state two. In the final state three, the circuit is in an unlocked state which indicates the combination was correct, thus displaying a green LED “010”. The influence of staying in state three depends on “SW15” being false and the value of “s” to be true, else we remain in a locked state. If “SW15” is true, we return back to state one and repeat the RGB LED FSM.

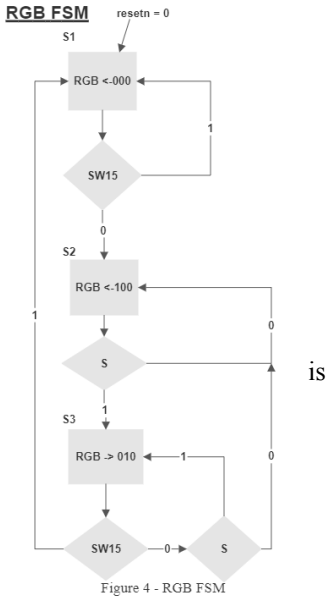
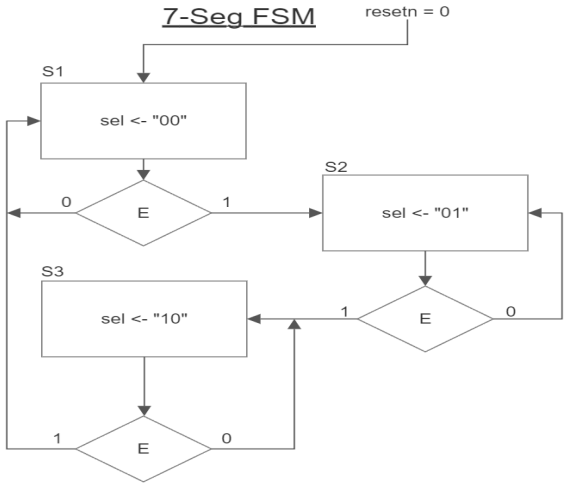


Figure 4 - RGB FSM

Figure 3 - Seven Segment FSM



The other FSM that the circuit uses is the Seven Segment FSM which has transitions that are controlled by “resetn”, “clock”, and “E”. The output is the value of “sel” which is controlled by “E” where “E” is the output of the counter. “sel” itself will influence the MUX and Decoder that allows the seven segment display to show different values on each display. In state one, the value of “sel” is “00” and remains as such if “E” is also ‘0’. If “E” becomes true in state one, we transition into state two. State two gives the value of “01” to “sel”. Again, if “E” is false, the circuit remains in state two, else we transition into state three. In state three, the value of “sel” changes to “10”. In this last state, if “E” is ‘0’ the circuit remains in state three, else the circuit will return to state one and restart the seven segment FSM. In practice, the FSM will advance in state every millisecond as long as the FPGA is powered on and the “resetn” button is not pushed.

III. Experimental Setup

Indicate the setup you used to verify the functioning of your project. What software/hardware tools did you use? What was the specific configuration of those tools? What are the expected results?

Our group used the Nexys A7 100T FPGA as a source for testing the physical functionality of the project. We also used Vivado's Simulation software to implement a behavioral simulation to which we can validate the circuit. Within this testbench, the user input "X" displays 420 onto the three 7-segment displays which is then locked. The test reflects three inputs after the program was locked. Two of these inputs are 666 and 101 which were proven to fail and the third being 420 resulting in a successful test along with an unlocked program. Also included within this testbench was signal R to represent the segments within the display that would turn on, RGB to represent the color of the LED, and "checkcode" to help visualize the correct code a user may input into the program.

IV. Results

As seen in the testbench and as explained in the Experimental Setup of this report, we utilized the behavioral simulation software in Vivado to debug and check the functionality of the circuit. We found after the initial simulation of the test bench we needed to assign a button press duration so the users code would not be continuously entered into the register. This also provided a clear visual representation of the button press in the simulation; mimicking a real world physical input. Our team was initially worried about propagation delays with a 1 ms counter and the interaction this would have with the button press from the user.

Our team also included a physical demonstration of the combinational lock physically working on the FPGA. This demonstration can be found at the end of the appendix section of the report.

Conclusions

The process of creating a combinational lock within an FPGA board sounds simple but is proven to be very tedious and time consuming. This tedious work can be found within every component along with certain routes that components must take for the lock to properly function. Every component involved within this combinational lock plays an important role. One of the more important components include but is not limited to the comparator since it decides whether or not a user input aligns with the one that originally locked the program. One issue that we came across was mapping different components together. Likewise, a simple mistake such as the demux outputs being flip flopped can cause the entire program to fail. Looking back, if we were to expand this project farther, we would implement more components such as a breadboard, speaker, and a locking mechanism to help better display our project and its purpose.

References

[1] D. Llamocca, VHDL Coding for FPGAs. [Online] http://www.secs.oakland.edu/~llamocca/VHDLforFPGAs.html

[2] D. Llamocca, "VHDL Coding for FPGAs Unit 6," Reconfigurable Computing Research Laboratory (RECRLab), Electrical and Computer Engineering Department, Oakland University.[Online]. https://www.secs.oakland.edu/~llamocca/Tutorials/VHDLFPGA/Unit%206.pdf

[3] D. Llamocca, "VHDL Coding for FPGAs Unit 7," Reconfigurable Computing Research Laboratory (RECRLab), Electrical and Computer Engineering Department, Oakland University.[Online] https://www.secs.oakland.edu/~llamocca/Tutorials/VHDLFPGA/Unit%207.pdf

[4] D. Llamocca, "VHDL Coding for FPGAs Unit 5," Reconfigurable Computing Research Laboratory (RECRLab), Electrical and Computer Engineering Department, Oakland University.[Online]. https://www.secs.oakland.edu/~llamocca/Courses/ECE2700/Notes%20-%20Unit%206.pdf

APPENDIX

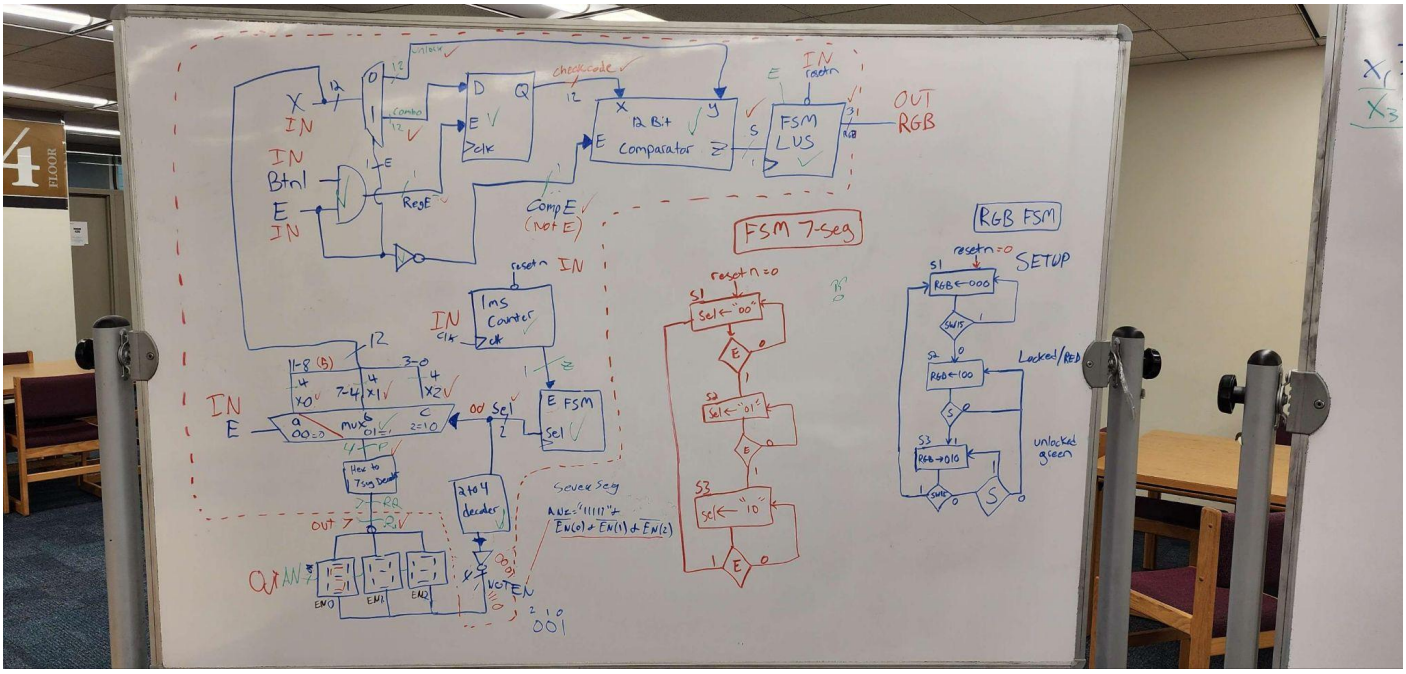


Figure 1 – Top File Block Diagram & FSMs

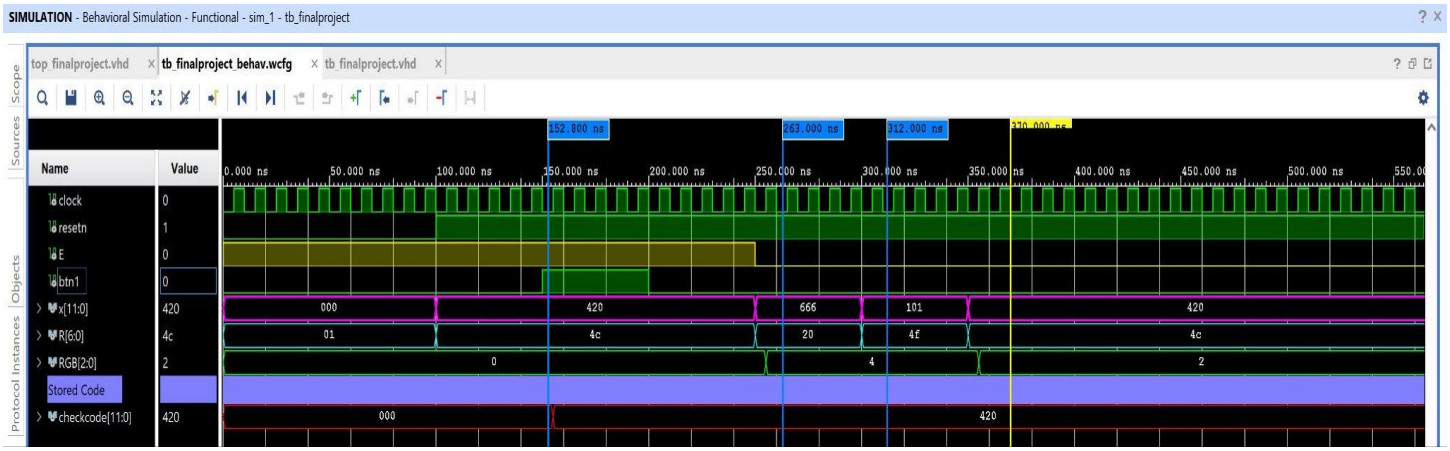


Figure 2 – Behavioral Simulation Waveform

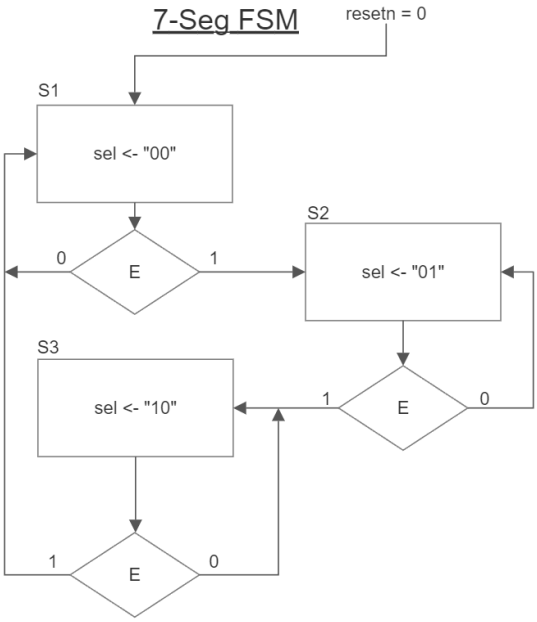


Figure 3 - Seven Segment FSM

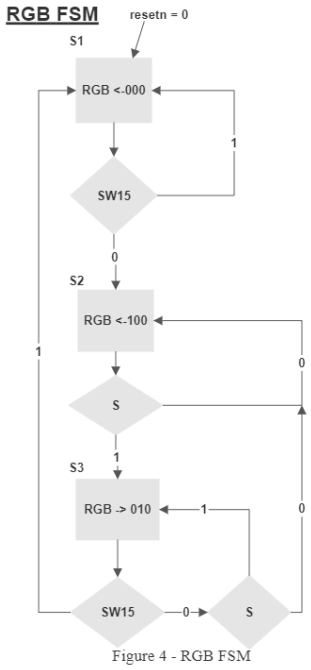


Figure 4 - RGB LED FSM

Demonstration of Combinational Lock URL: https://youtu.be/KmQWPVBU4QQ