Home Security System

Samantha Fakhouri, Thomas Filarski, Nathan Kelley, Mathew Plaza
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
E-mails: sfakhouri@oakland.edu, tfilarski@oakland.edu, nkelley@oakland.edu, mathewplaza@oakland.edu

***Abstract*: The goal of this project was to create a Home Security System for use in home or business. Through the use of a Finite State Machine, along with some independent research on arduino based Sensors a home security system that detects motion, asks for a switch input, and then proceeds to either trigger an alarm or disarm itself until the user rearms it was created successfully. This project is simpler than a full security system, but it is a baseline that could be extended to serve larger purposes with different components and knowledge.**

I. Introduction

The original idea for this home security system came from an EGR 240 test problem the group encountered during their studies. In it, it asked us to use a decoder to map a series of inputs, as well as the output, and then determine how the security system should react. It was a simple system, but it certainly had a few problems. During the test time, when this question was on the exam, many people had various reasons why a certain set of inputs should be reacted to in different ways. For example, if no code is entered in time, but the sensor didn't pick up anything, should that be considered an error? Or just an alarm?

This project hopes to take this problem in a different manner. Instead of using an asynchronous decoder, we used our knowledge that we learned in class about Finite State Machines to completely overhaul the design process of this problem and improve upon it. By using a Finite State Machine, we are able to set triggers to occur for different situations, rather than just have a series of outputs based on a single input, which, is prone to errors, and was ultimately complicated to implement.

What we had to learn on our own was how to implement external inputs and outputs mainly. Our external input, the sensor was the hardest to achieve, and used an arduino both as a power supply as well as some code. The PMOS pins were a bit unintuitive to use at first, but in the end we successfully made a working sensor as an input. External wiring also had a few quirks that we had to learn. For example, because we used both an arduino and an external sensor, the ground for the Nexus 4 board and the arduino had to be connected, else the signal would change due to different voltage reference points.

This project is a Home security system. It is primal, in the sense that normally you would use a smaller board to create something like this, but the biggest advantage is that this system could be easily changed with some knowledge of Computer Hardware Design to implement more sensors such as the accelerometer as well as infrared sensors or sound based sensors. With some increased knowledge, and a bit of knowledge with electrical wiring, this system could easily be expanded to cover multiple rooms of a house. Finally, with a bit of refinement, the switches could be attached to a keypad, to create a passcode lock. This system is just a model for what security systems can be created, but this model is the base for a variety of security systems for various purposes.

II. Methodology

This project consists of a series of switch inputs, along with a sensor. The internal microcontroller contains an FSM, comparator, register for passcode storage, as well as a decoder to control the outputs: an LED and a piezo buzzer.

The basic structure of this system is that the input switches go to a comparator to compare their values to the stored passcode. This outputs a signal that goes to the finite state machine. The finite state machine takes this input, as well as the sensor input and timer input, and based on the logic discussed in its section, sends an encoded 2 bit code to the output LED decoder and alarm to determine which led should turn on, telling the user which state the system is in. If an alarm is determined to be necessary, it also tells the buzzer to turn on. For a complete block diagram see Appendix A.

A. Initial Inputs

The inputs of the alarm system consist of eight switches used to check if the person entering is the owner, a motion sensor that detects entry and starts the countdown for the correct password to be entered, a reset button which resets the entire system. Finally, a softreset button which sets the system back to armed.

The eight switches used for the passcode go to a register and a comparator. When the correct password is entered the comparator will output a '1'. When the softreset button is pressed and the correct passcode has been entered the password will be changed in the registers to be used later. The motion sensor is set up on an Arduino board because the motion sensor uses a 5V input to be powered. Then the sensor detects motion in outputs a high value which is read by the Arduino. The Arduino then outputs a HIGH value which is sent to the FPGA and causes the finite state machine to go from state 1 to state 2. When the reset button is pressed the entire system is reset, this means the password is

changed back to 0000 0000 and the FSM is changed back to state 1.

B. *Finite State Machine*

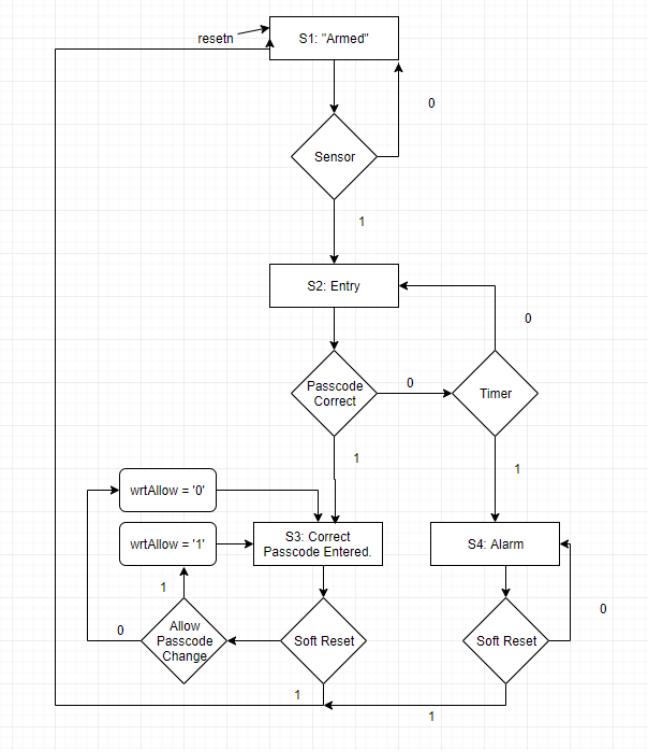
The first thing to create when designing the security system was to define the different states of operation. This naturally would lead to a Finite State Machine. When the security system is armed, but the sensors detect nobody in the house, there should be a state that constantly looks for potential intruders. Thus this state we deemed the "Armed" state or the "Inactive" state, as there was no passcode functionality active. In this state it needs to detect any potential intruder, and thus when the sensor picks up motion, it shifts into the second state "Entry mode".

In the second state, "Entry mode" the system desires the correct passcode. When designing this state it was important to realize in a security system there are parameters in which to trigger an alarm, and parameters in which to accept the passcode. In the security system, if the passcode is entered correctly, it should transfer into the "Correct Password Entered" state. However, in this security system's state, if a correct passcode is not entered in a certain period of time, then it should trigger the "alarm".

In the third state "correct password entered", the system has recognized that the correct passcode has been entered. Thus, there are no timers to trigger any alarms. However, this state also is meant to change the passcode. In a security system. the password should only be able to be changed once the correct passcode has been entered. Thus, this functionality is only allowed within this state. This is a separate trigger, outside of this state, tied to a switch, and thus it makes the resulting FSM a mealy machine. In addition in the case of a user reset, it will move back to the "Inactive" state.

In the fourth state "alarm", the system has timed out before a correct passcode was entered. Thus, the system triggers an alarm, in this case, a piezo buzzer in addition to locking down the system until a reset is given.

These states are encoded into a 2 bit code that is sent to a decoder. Each one of these states are handled differently by the decoder. The figure below shows the FSM created in ASM format.



C. *Password Storage Register*

The password storage register is a synchronous component. Its inputs are the 8 bit input from the entry switches, and the “wrtAllow” input from the FSM, as well as the clock signal and the resetn. Its default state on startup is the passcode 0000 0000. As stated earlier, wrtAllow is only high if the passcode has been entered properly (“correct password entered state”) and the user activates the allow password change switch. If these conditions are met, the current state of the switches is converted to the new password in memory. If it is not high, the current password in memory is maintained. If the CPU reset button is pushed, the passcode is cleared and set to the default 0000 0000 passcode.

```
if resetn = '0' then
    pass <= (others => '0');
    else
   if clk'event and clk = '1' then
      if  wrtAllow = '1' then
    pass <= SW;
    else
    end if;
```

Figure: Basic Logic of password storage register.

D. *Comparator.*

The comparator is a simple asynchronous component. Its inputs are the 8 bit number from the register, and the 8 bit input from the entry switches. It simply takes the passcode from the register, compares it bit by bit to the

current state of the switches (in binary). If they are equal, the comparator outputs a '1'. If they are not equal, it outputs a 0.

```
if (SW = pass) then
cmp <= '1';
else
cmp <= '0';
end if;
```

Figure: simple logic of comparator

E. Timer.

The timer is a series of clock dividers followed by a counter. The clock divider inputs a 10MHz clock, and through frequency division. First it scales the clock frequency down from 10MHz to 400Hz through a 25,000:1 counter frequency divider. It then scales the clock down to a 1Hz clock by using a 400:1 frequency divider. The final block takes this 1Hz input and outputs a 4 bit number (from 10 to 0) that starts at "1010"(10) and decrements one every time it receives a rising edge clock tick. This means that the output of the timer block is a 4 bit number ranging from 10 to 0 that is then sent to the 7segment decoder to be displayed on the arduino. This number continues to decrement, and when it hits 0, the timer signal is turned high, triggering the alarm state.[1]

F. Decoder LED

The LED decoder looks at the outputs of the finite state machine to determine which LED color(s) should be activated. When the FSM is in state 1 the output is A1 = 0 and A2 = 0 which will activate the red and blue LEDs creating a purple color to show that the security system is Armed. When the FSM is in state 2 the output is A1 = 0 and A2 = 1 which will activate the red and green LEDs creating yellow to show that there has been an entry. This will cause the timer to start and the user will have to enter the password. When the FSM goes into state 3, A1=1 and A2 = 0, which means that the correct password was entered within the allotted time causing the green LED to light up to show that the correct password was entered. Then the FSM goes into state 4, A1 = 0 and A2 = 0 then the red LED will light up to show that someone has broken in and will have the FPGA output a high value to activate the piezo buzzer.

III. Experimental Setup

For testing we used the Xilinx Vivado software to create a timing diagram based off a testbench. The testbench starts in the inactive state, triggers a sensor to move it to the entry state. It then inputs the correct passcode 0000 0000 to send it to the correct passcode entered state. It then sets allow password change to true, and rewrites the passcode. It then soft resets and repeats the process, but with the new passcode. Finally, it soft resets a third time, repeating the process, but instead overriding the timer (for testing purposes) to trigger the alarm.

See Appendix B for waveform result.

IV. Results

The project gave the results that were expected except for the buzzer. When the state changed in the FSM, the LED color changed as expected. Also when state 2 was entered the timer would count down from ten to zero. When the alarm state was entered the FPGA output a voltage but it was not powerful enough to make the buzzer as loud as desired.

Figure: Sample LED Showing Alarm State.

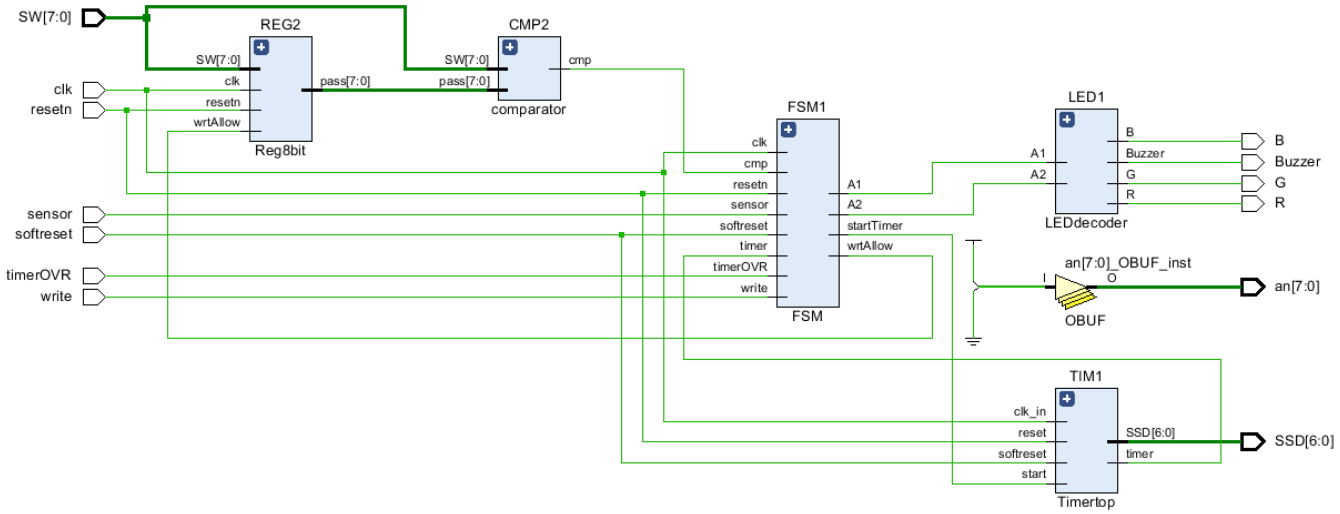
Conclusions

In conclusion, our design of the security system was an overall success. First, the sensor recognizes movement and start a timer. Before the sensor, the RGB LED is yellow, and after it is purple. If the correct 4-digit code was entered by the switches 0 to 7 (simulating a keypad) before the timer reaches 30 seconds, the RGB LED turns green. If the same situation happens but the code is incorrect or if no code is typed in within the 30 second window, the LED turns red (signaling the alarm). As for the topology, it starts with the sensor and keypad. The sensor is connected to a voltage divider and then to the FPGA. The timer is displayed on the 7-segment display. The switches on the FPGA are then switched on then immediately off (similar to that of a keypad). The default code is 0000, but if the signal WriteAllow is activated, a code can be set, which is stored in registers and is compared to future codes. The output of the comparator (either a logic 0 or 1) is sent to a finite state machine, and depending on the inputs, the finite state machine will control the color of the LED. Some limitations include the lack of a keypad and errors with the alarm. Although an actual keypad was not present, it could still be simulated with the switches, so the addition of a keypad would not have affected the performance of the system. As for the alarm, when activated, it would produce a very high frequency pitch in which only a few could hear. A different alarm or different frequency of the clock might have improved the performance of the alarm and therefore the system.

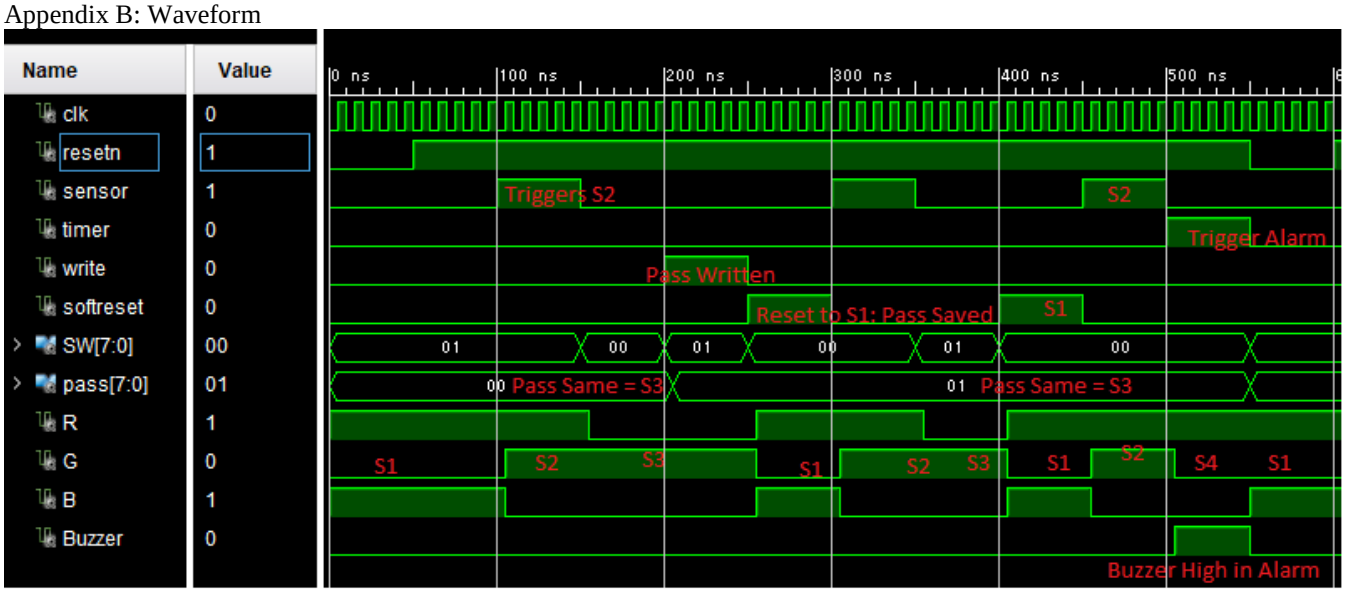
References

[1] C. A. Ramos, "Frequency Divider with VHDL," Frequency Divider with VHDL - CodeProject. [Online]. Available: https://www.codeproject.com/Tips/444385/Frequency-Divider-with-VHDL. [Accessed: 20-Apr-2018

Appendix A: Full Block Diagram



Appendix B: Waveform



Armed: S1: R&B LEDs Entry: S2: R&G LEDs Correct: S3: G LED Alarm: S4: R LED