Student Name: Keegan Smith, Casey Merola, Jeffrey Nguyen

Student ID#:

Team Name/Number:

Team Members:

Date of Completion:

Demonstration Method: Zoom

1. Design

1.1) Hardware Design

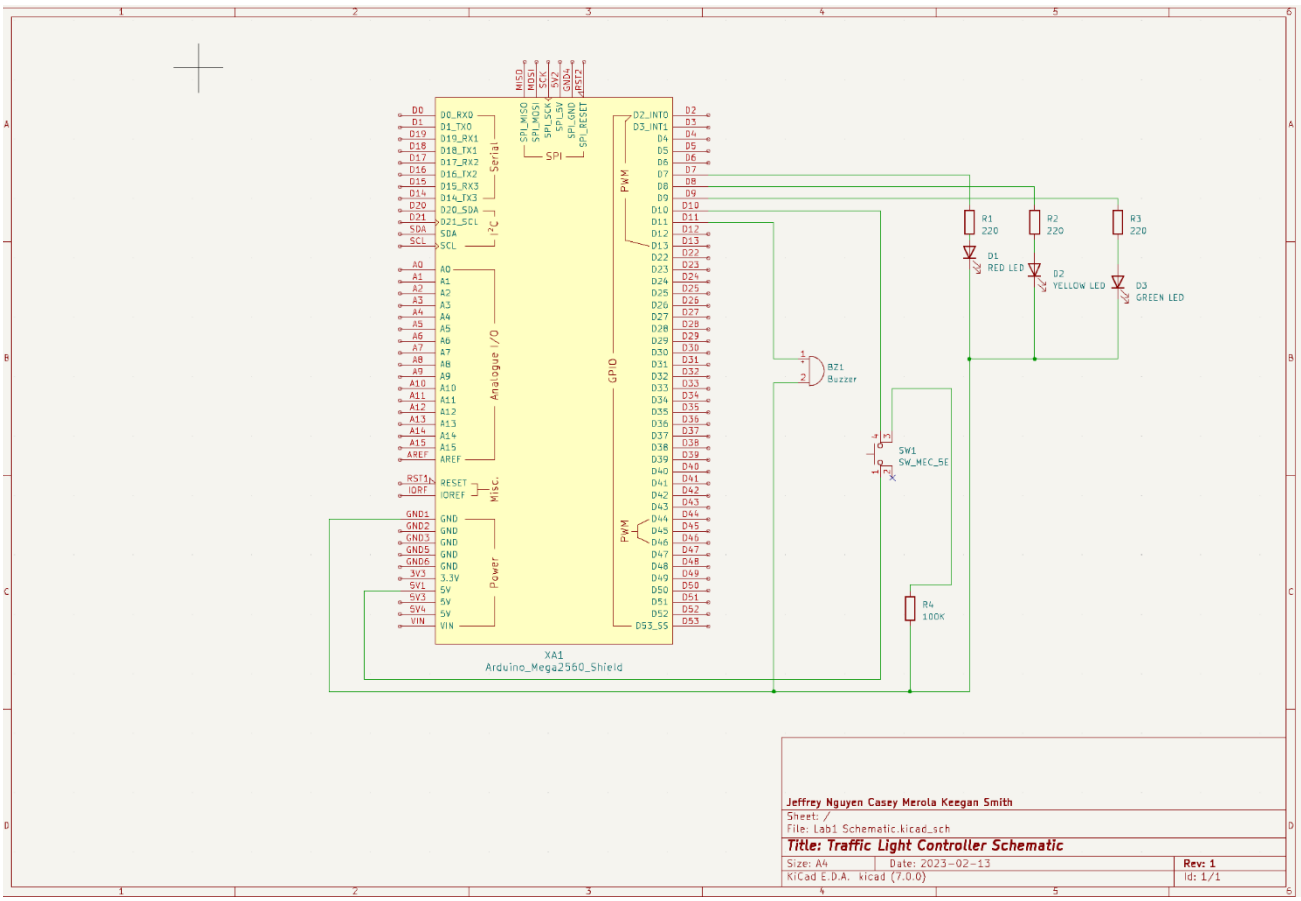

Figure 1. Traffic Light Controller Schematic

For this lab we used the PWM/Digital pins seven through eleven on the Arduino Mega. These pins were chosen at random, and any PWM/Digital I/O pin would work if they were properly assigned in the software. Using equation 1, the resistors calculated for the most current to the LEDs were 220Ω. A higher resistance may be used, but the brightness of the LEDs would decrease.

$$R = \frac{(Vss - Vf)}{I} \qquad 1.$$

V_{ss} = Supply Voltage, V_f = forward voltage on the LED, I = Desired Current

A 100 kΩ resistor is used as a pull-down resistor to GND to keep the push button signal LOW until the button is pushed. The resistance was chosen as to keep the current as low as possible, since resistance is inversely proportional to current.

1.2) Software Design

A switch statement is used for the main code loop, with each state given a number from 0 to 3 for the initial state, red state, green state, and yellow state. This was chosen for the ease of coding the state machine and allows for the "default" case to return the system to the started state if anything goes wrong. The rest of the code, such as the timer and button, are handled using interrupts. This solution makes the most sense since we already have interrupts enabled and assigning interrupts to digital pins is more manageable and elegant than creating a polling function just for the button. The interrupt on the button also allows for a seamless transition into the debounce function, which ensures that the button is read properly.

Each state is programmed as close as possible to the FSM, and the system evaluates the next state every time the loop function repeats, setting the state integer to the proper value and turning off the LEDs and buzzer if necessary.

GitHub Link: https://github.com/CaseyMerola/Lab-1.git

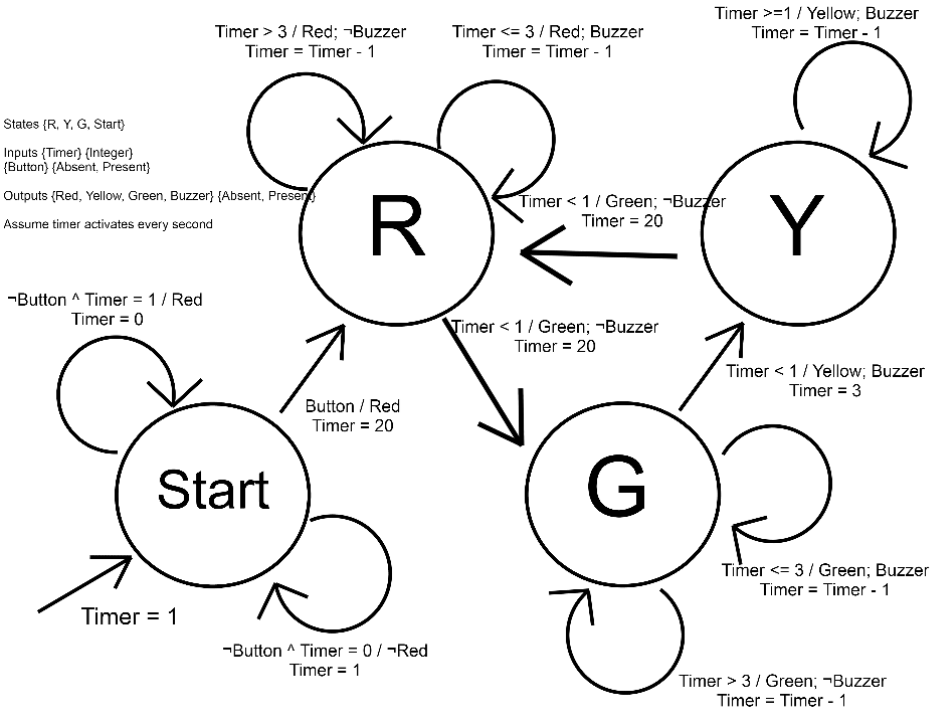

Figure 2. Traffic Light Controller FSM

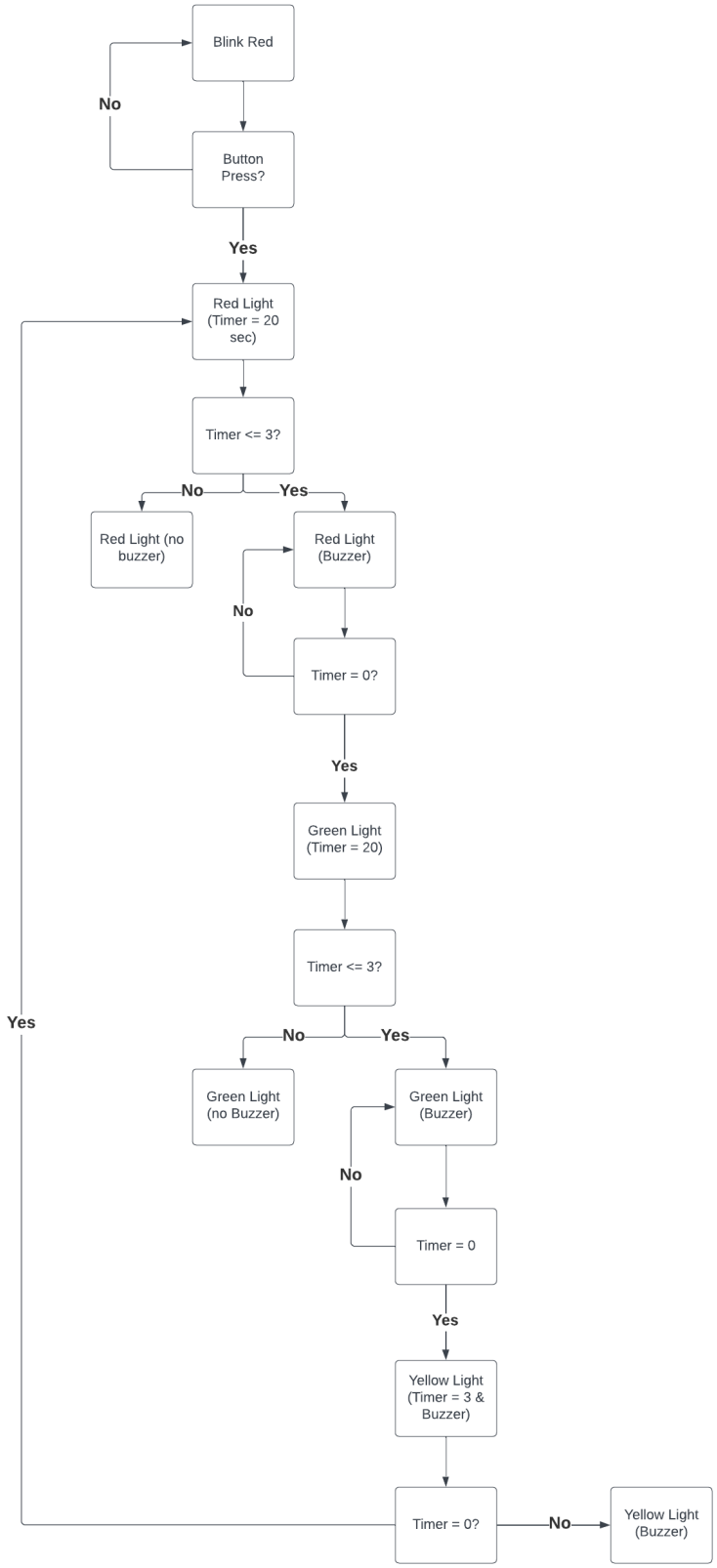
Blink Red
No
Button Press?
Yes
Red Light (Timer = 20 sec)
Timer <= 3?
No
Yes
Red Light (no buzzer)
Red Light (Buzzer)
No
Timer = 0?
Yes
Green Light (Timer = 20)
Timer <= 3?
Yes
No
Yes
Green Light (no Buzzer)
Green Light (Buzzer)
No
Timer = 0
Yes
Yellow Light (Timer = 3 & Buzzer)
Timer = 0?
No
Yellow Light (Buzzer)
Yes

Figure 3. Traffic Light Controller Flow Diagram

1.3) Results

The traffic light controller works according to the required specifications. It blinks red every second waiting for the button to be pressed. Once the button is pressed the red LED stays on for 20 seconds and the buzzer will beep for that last three seconds indicating a change in light. When the timer hits zero, the green light will come on and stay on for 20 seconds and the buzzer will beep for the last three seconds indicating a change in light to yellow. The yellow light will turn on and stay on for three seconds, and the buzzer will beep the whole time until the next light is on, which is red. The buzzer stays on for the duration of the yellow light because the yellow light is only on for three seconds, and the buzzer is triggered when there are three seconds left on the timer.

2. Problems Encountered and Solved

One of the problems that we came across were false triggers on the push button. This was solved by adding a switch debouncing function as well as a pull-down resistor to ensure the input pin is grounded until the button is pressed.

Another problem encountered was properly programming the timer interrupt. The information on programming the interrupt was readily available in the microprocessor's datasheet, but with so many registers to program, it got convoluted at times. Also, the timer count register did not originally reset in the interrupt service routine, so the interrupt only triggered when the register rolled over, and since there were more than one timer register, it was difficult to tell which one indicated that it was programmed wrong. This problem was solved by printing a test message to the console whenever the timer interrupt activated that displayed the timer value, which indicated when it was causing the interrupt to activate, helping the debugging process.

3. Personal Contributions

Casey:

1) Drafted FSM
2) Started code with writing configurations and skeleton
3) Created group GitHub repository
4) Programmed timer interrupt
5) Tested code
6) Contributed to report

Jeff:

1) Made schematic
2) Created Flow Diagram
3) Tested code
4) Contributed to report

Keegan:

1) Finished, committed, pushed, and merged code
2) Programmed button interrupt
3) Started report
4) Contributed to report

4. Lesson Learned

This lab helped us understand how to initialize and use interrupts, with the program utilizing a timer interrupt and a rising edge interrupt. The interrupt process also taught us how to gain the most information from other sources, like the debouncing function that was adapted from another source. Also, we learned how to properly read the datasheet for the microprocessor, which can be difficult for someone who does not have a lot of experience with this type of writing.

The lab also introduced us to using KiCad, a PCB/Schematic software which was used to build/create the schematic, showing how the traffic light controller is wired.

The lab also introduced us to the concept of drafting a program by hand before programming it, which simplified the time it took to program significantly. By having a finite state machine already designed, converting the machine to a program was incredibly easy, and just turned each arrow into an if statement with the variables already written.