# Lab 1 Report

Kerollos Lowandy, Pablo Ruiz, Mohammed Khassim
Completion Date: 02/22/2021
Microprocessor II and Embedded System Design
Lab 1: Traffic Light Controller

**1.INTRODUCTION:**

In this lab, we were tasked with building a traffic light controller using time interrupts with an arduino board. Our design will begin with the red light blinking and will be triggered to start the main traffic light display if a button is pressed. The circuit then cycles through each state, being timed based on a 7 segment display, showing the time until a switch in hexadecimal values. The buzzer is also triggered whenever it is 3 seconds before a color switch. The circuit keeps looping through the red, yellow and green standard states until the button is pressed again. Once it is pressed again at any point, it goes back to only blinking red.

## 2. DESIGN:

### 2.1. HARDWARE:

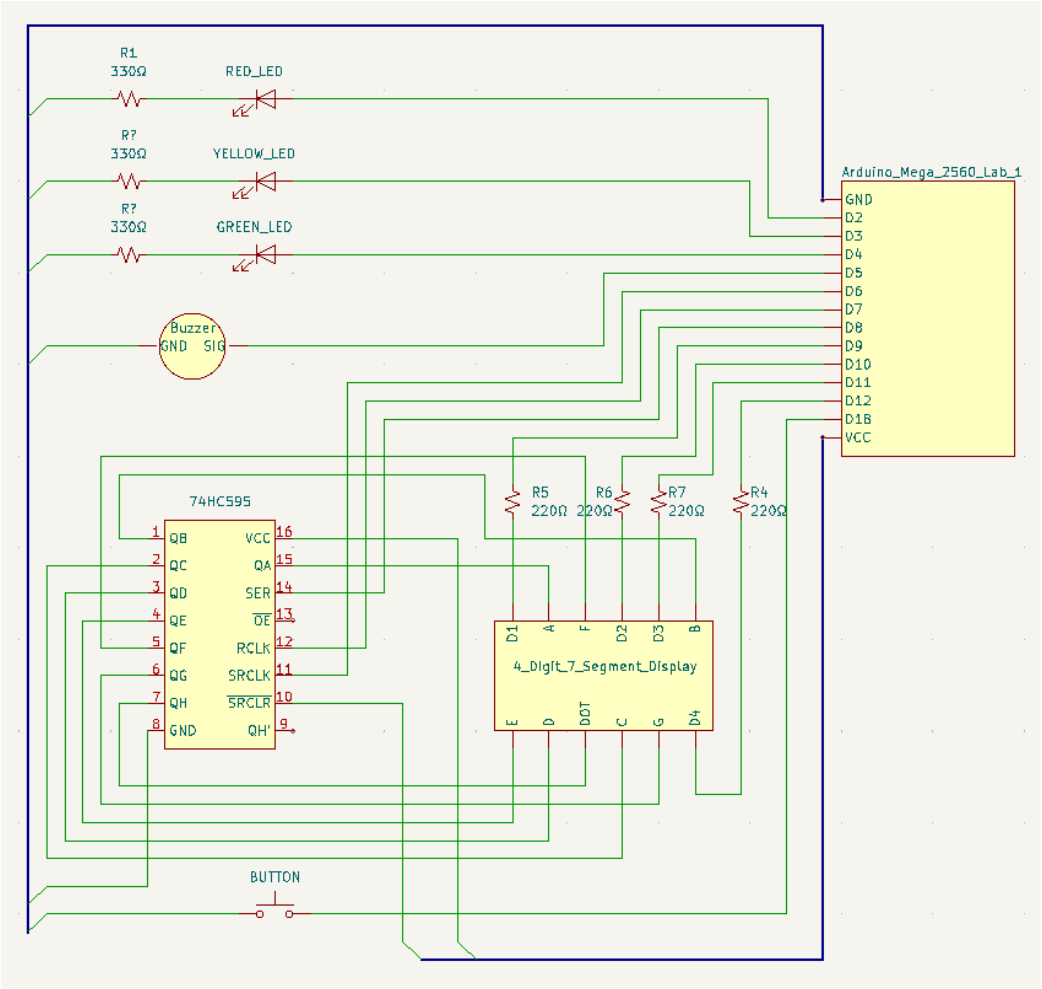


*Figure 1. Lab 1 electrical schematic.*

The main hardware decision in this lab was whether to connect the pins from the 4 digit 7 segment display directly to the arduino which would use 12 digital pins or to use a shift register such as the 74HC595 which adds complexity to the design but allows the arduino to use only 3 pins to control the display. Ultimately, the group decided to use the shift register in order to minimize pins used and to take advantage of the built in shiftOut function included in the Arduino programming language which allows the use of a bit mask instead of having to set all the digital pins, corresponding to the segments, directly every time a digit has to be displayed. Additionally, 220 ohm resistors were used to limit the current to the four pins that enable the LEDs of each of the four digits in the display, and 330 ohm resistors were used to limit the current through the red, yellow and green LEDs. Usually the design would also contain an additional relatively high resistance resistor on the microcontroller side of the button as a pull up resistor but this feature was enabled internally within the Arduino in order to reduce the part count. In total, 10 digital pins and one power and ground pins were used for this design.

**2.2. SOFTWARE (Pablo Ruiz):**

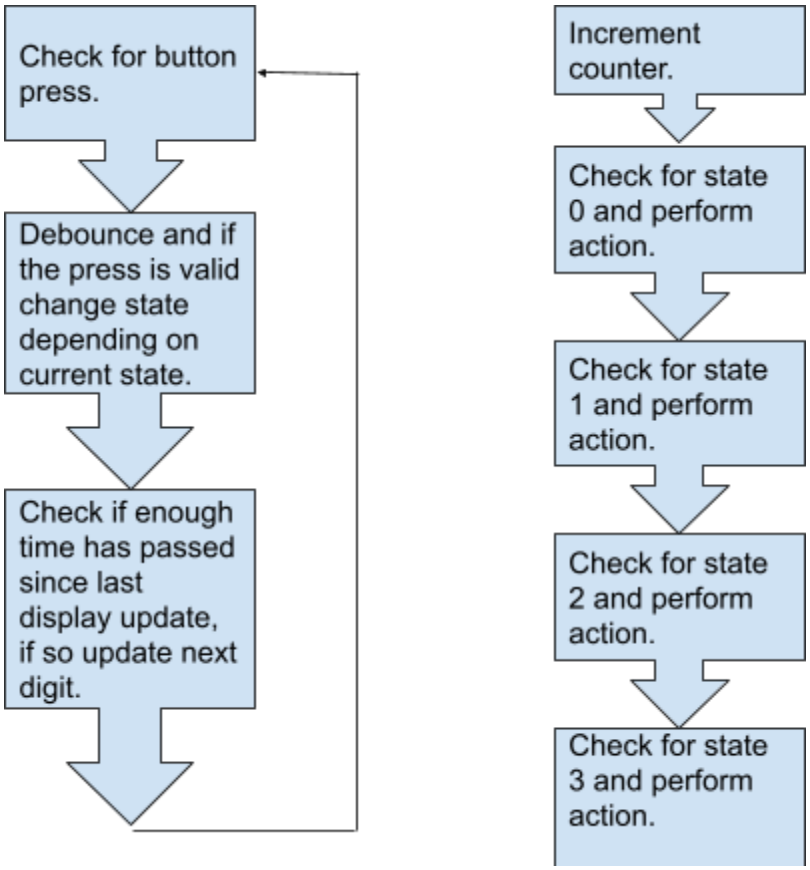


*Figure 2. Lab 1 Program flow chart. Main loop (left) and interrupt (right) logic.*

The software portion of this lab was completed using the Arduino IDE v1.8.15 due to its ease of use and compatibility with the Arduino MEGA2560 microcontroller. The key point that was considered during the design of this program was the necessity to maintain strict timing for the changes of state in the FSM due to it controlling a critical piece of safety equipment such as a stop light. This was done by implementing a timer interrupt for the FSM while the polling for the tactile button and the update routine for the screen remained in the main program loop due to them not being critical and the simplicity of the program ensuring that the loop frequency was already much faster than necessary to provide a seamless user experience.

The timer interrupt for the FSM had to be enabled by manipulating the ATMEGA2560 internal registers directly since the Arduino subset of the C++ language does not contain any functions which abstract this task. Due to the high frequency of the microcontroller's internal oscillator (16MHz) and the relatively high wait time required for the timer defined FSM employed (200ms), even after utilizing the highest clock prescaler available (1024) the value of the compare register (OCR#A) would have to be 3124 which requires the use of a 16 bit timer

register such as timer 1 (TCNT1). The timer was configured in clear timer on compare mode (CTC) by setting the WGM12 bit in register TCCR1B as indicated by the ATMEGA2560 documentation. Additionally as mentioned previously a 1024 prescaler was enabled by setting the CS10 and CS12 bits in register TCCR1B. Within the interrupt service routine (ISR) for the timer a series of if else statements were used in order to determine which action to perform depending on the current state and time elapsed since the last state change. To store the state of the FSM between interrupts an unsigned integer variable stored in RAM (volatile prefix), named state, was used. This system proved reliable and accurate.

Pin 18 had to be used for the button due its ability to enable a pullup resistor which removes the necessity from including one in the hardware design. The polling for the button was performed within the main loop in part due to it being more conducive to effective debouncing techniques than an interrupt. The main loop allows us to still run code while the button is not pressed thus allowing not only for a debounce delay but also a debounce duration threshold (15ms) for the presses. This approach was found to be very effective against long duration presses where the interrupt approach failed intermittently. When the button is pressed the state can change from state 0, where the red light is blinking, to state 1, where the stop light routine begins, or from any other state back to state 0.

The update function for the 4 digit 7 segment display was also included in the main loop since it is not a critical function of the system and in any case in this application the main loop is more than fast enough for human reaction times. An array containing the equivalent bit masks for the 7 segment digits matched to their index was generated in order to facilitate the conversion from decimal to their hex representation. The function waits until a time threshold (5ms) has been exceeded to display the next digit in the display. Therefore each digit is updated every 20ms or at 50Hz. For each state a corresponding timer limit is defined (6 or 20 seconds) and the current value of the FSM counter is subtracted from this limit to determine the value to be displayed. Subsequently the unsigned integer is right shifted the index of the digit (0 indexed) times 4 in order to place the relevant bits as the rightmost 4 bits in the bit array. Then a mask equivalent to the integer representation of 15 is applied to zero all the bits to the left of the rightmost 4. Finally, the value obtained is used as the index to select the bit mask of the equivalent hexadecimal digit from the array mentioned previously, which is then shifted out to the shift register and thus sent to the display. This process happens so fast that to the human eye it looks as if all the digits in the display are updated continuously at the same time without flickering.

Github Link:

https://github.com/tame-potato/Micro-2-Labs/tree/main/Micro_2_Lab_1_Pablo_Ruiz_V2

### 2.3. FINITE STATE MACHINE DESIGN (Kerollos Lowandy):

- States = {Red Blinking, Red, Green, Yellow}
- Inputs = ({Button} -> {0,1})
- Outputs = ({Red, Yellow, Green, Buzzer, } -> {0, 1})
- initialState = {Red Blinking}

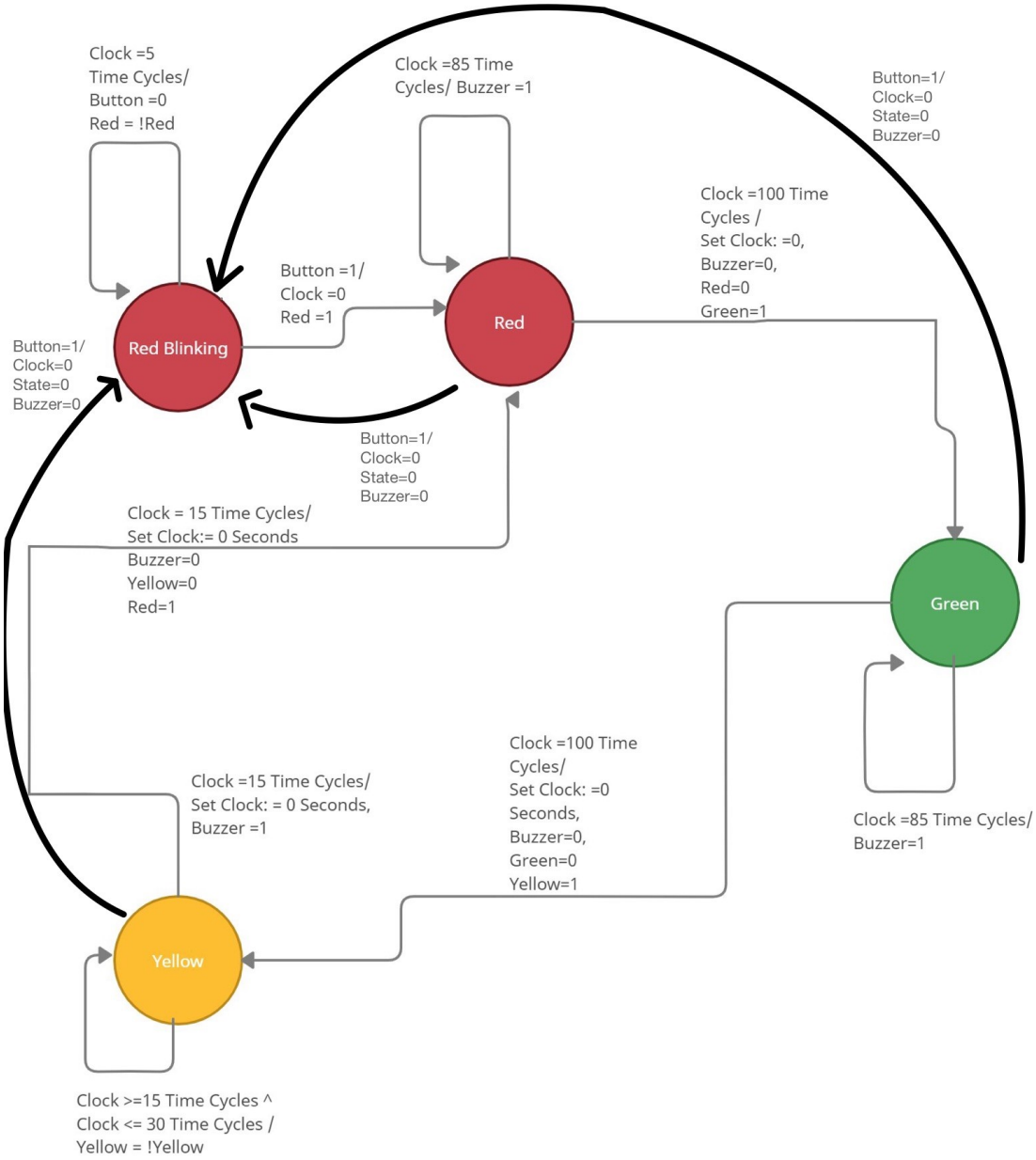

*Figure 3. Lab 1 finite state machine (FSM).*

It should be noted that this is a time triggered model with 200 ms increments. This means that a transition/tick occurs every 200 ms or at 5Hz. This was so that each yellow blink was able to be a single time interrupt. The Clock variable measures ticks and thus its value divided by the frequency, 5Hz, equals the time passed in seconds. The input and output variables such as Red, Yellow, Green, Buzzer, etc. are digital and thus are binary.

**3. RESULTS:**

Demo Video Link:
https://studentuml-my.sharepoint.com/:v:/g/personal/pablo_ruiz_student_uml_edu/EeK3A7M-VTRClyz5j_QVitsBpcWyDLBlY9wld_7XJHIQig?e=0jRjX8

**4. DISCUSSION:**

The main problem that arose in our design process came in incorporating the yellow light switching from a solid light to blinking every 200 ms for 3 seconds. We originally went through and designed the system without this function assuming we would be able to very simply add it in later. However, since our timer up to that point was operating with interrupts of 1 second, (made the most sense for the rest of the design since any action to be executed happened at a multiple of 1 second) the design had to be changed since we could not use 1 second interrupts to perform the yellow blink that would happen every 200 ms. Instead, the route we chose to take is having our time interrupts be every 200 ms. This made each yellow blink represent a single time cycle and everything that was represented with seconds would be represented with 5x as many time cycles. Ex: The buzzer would turn on after the green light had been on for 17 seconds, or 85 time cycles (17 seconds* 5 time cycles/second). Whilst this did not have much of an effect on our hardware design, as expected, our FSM design and the logic within the interrupt had to be adjusted accordingly. Our new FSM triggered the entering into a new state whenever a certain number of 200 ms clock cycles had been achieved as opposed to how many seconds had passed.