1. **General Information**

**Student Name: Chris Worthley**

**Student ID number: 02024207**

**Team Name/Number: Unknown**

**Team member names: Daniel Burns, Himadri Saha**

**Date of completion: 10/12/25**

**Demonstration method: zoom**

1. **Design:**
   1. **Hardware Design**

Hardware schematics listed below:

A diagram of a circuit

AI-generated content may be incorrect.

**Figure 1. Hardware Schematic (Except 7-Segment Display)**

A circuit board with wires and numbers

AI-generated content may be incorrect.

**Figure 2. 7-Segment Display Wiring Using 74HC595 IC**

The hardware setup uses digital pins 48 through 53 to operate the LEDs for both traffic signals. The main traffic signal is connected to pins 52, 50, and 48 for the green, yellow, and red lights respectively, while the cross-traffic signal uses pins 53, 51, and 49 for its corresponding lights. Each LED is connected to its output pin through a current-limiting resistor, which is 220 ohms, to prevent excessive current draw. The Arduino outputs logic-high signals to illuminate specific LEDs according to the programmed timing sequence.

A 16-button keypad is connected to pins 12 through 5 and serves as a user input device for controlling or modifying light operation modes. The buzzer is attached to pin 13, which is a PWM capable output. The seven-segment display is interfaced through three digital pins using a shift-register configuration to reduce the number of required I/O lines. The clock, latch, and data connections for the display are assigned to pins 4, 3, and 2 respectively.

All components share a common 5V supply and ground, provided directly by the Arduino board. During operation, the Arduino cycles through the programmed states, turning on the main road green light while keeping the cross-traffic red, then proceeding through yellow and red phases before switching priority to the cross-traffic lights. The seven-segment display provides a visible countdown for each phase, while the buzzer sounds briefly to indicate transitions.

* 1. **Software Design**

**Source Code:** <https://github.com/himadri553/Microprocessors-2/tree/main/Lab1/Main_Code_Lab1>

The software design started out as a basic idea of different important functions that need to be addressed. The brainstormed functions we talked about was, the initial state where everything is waiting for an input value and the lights are all red. The next step being the polling state where there are inputs from the keypad are used to set the LEDs for a select amount of time and updates the digital display number. The final step will be outputting the full sequence of the LEDs after the initial data is taken in. The next step in our design was to make a state machine for this lab. This is shown in figure 1 where the first state is the power up state, which is responsible for polling for the LED durations from the keypad. The second state is called the Running state, which is responsible for the whole sequence. Specifically checking when the red light is on the opposite green light should be on, and when the time is up, the lights start blinking. This also includes how the yellow light is supposed to turn on after the green light is done, and keeps the red light on, until the yellow light is done. Once we actually coded the program it got a little more complicated. To take in the keypad inputs, we had to set up a 2D array styled like the keypad itself. Then using the I/O pins on the Arduino we took in the data form the key presses and saved them into a string to be read as a command to control the LEDs and digital display. The digital display is set up with a clock, hex map using hexadecimal values. The display is updated using a digital write and a shiftout function from the Arduino library. The next step of the code is the main loop. The main function is split into two sections. Phase 1 where the code takes in and reads the data and creates the command string from the keypad inputs to control the LEDs. The second phase is the cycle phase. This is where the code loops through the sequence based on the inputs gathered in phase 1. In this phase the timer is set and displaying on the digital display, along with the LEDs lighting up according to the command. As the code goes through the loop, the count decreases on the digital display, and LEDs act accordingly, switching from green to yellow to red, and red to green. The code uses basic case statements in order to get this part done, assigning the LEDs a certain order based on the time, and states of the LEDs prior.

A diagram of a diagram

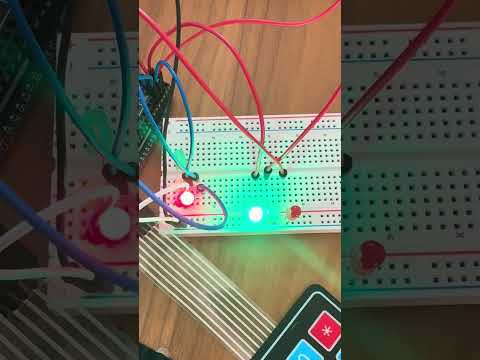
AI-generated content may be incorrect.

**Figure 3. State Machine for the Traffic Lights**

This image is a basic state machine for the entire lab, it goes through all the big processes in the lab. This state machine is used as a skeleton for our code, each state is essentially a step in the program that needed to be addressed.

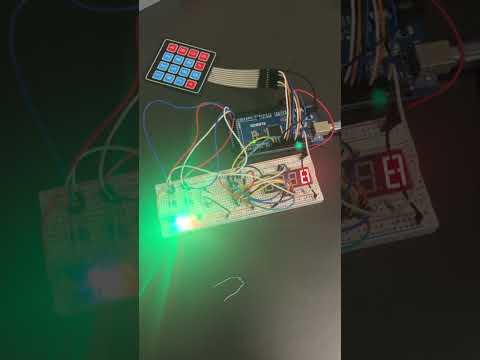
* 1. **Results**

In this lab we successfully created the traffic lights functioning according to the assignment’s requirements. We successfully made the LEDs react in cahoots with keypad and the 7-segment digital display. The only issue we didn’t figure out was how to add a second digit to the digital display. It still returns the correct time and keeps it up to date, but it only displays the last 8 seconds of the timer, since there is only one shift register. Other than that, the traffic lights work properly. The LEDs are connected to each other, changing from red to green, green to yellow, then back to red using either their respective times determined by the keypad, or the three seconds for the yellow lights. The process has the proper interrupts to make the keypad, LEDs, and digital display work properly. Overall the lab was a success and the traffic lights work as intended with only one small issue.

[](https://www.youtube.com/embed/KhIwH64PW10?feature=oembed)

*Video 1: First working prototype of the Traffic Light System*

This video is a demonstration of one our first tests of the basic traffic light system. This model has no interrupts or digital displays. The code used in this video is purely time based with the keypad inputs.

[](https://www.youtube.com/embed/GyIDsukMH8w?feature=oembed)

*Video 2: Final working Traffic Light Circuit*

This video is the final showing of our working circuit with interrupts and the digital display included. The code also includes the working assembly code.

1. **Problems Encountered and Solved:**

In this lab we started by separating our work into different sections of the lab. Our initial idea was to have each group member do a section of the code and someone work on the assembly. This division of labor works well but had a few issues. The first issue came when some parts of the code were made before the others, which led to temporary pseudo code and mistakes. This had to be almost completely rewritten to fit the assembly and digital display portions of the lab. This issue was easily fixed since the framework was made prior, but the actual code needed improvements, so our group put the code together and fixed all the issues with the code including how to call the assembly that turns the LEDs on and off. The next challenge we had during this lab was with the Digital display. Initially we were confused that the 7-segment display had only one shift register. We began troubleshooting with the driver, but our main problem persisted. We got one digit to display properly with interrupts but needed double digits for the assignment. In the end our solution stemmed from the elegoo instructions using all the same pins. The code part was solved by using interrupts and the built in timer. The timer was used to trigger an interrupt every second and decrement another variable used to count the seconds down to then display on the 7 segment digital display. The other major issue we had was during our brainstorming process earlier into the project, we initially made the entire project minus the digital display without interruptions integrated into the code. The LEDs all worked properly with the correct timing, however there was no code for interrupts between the traffic lights for when they switch. Instead, it was made purely from timing commands. This definitely helped showcase that the general system was working properly, however the interrupts are a bit part of the lab and led to a lot recoding and changing the general set up of the whole system. We fixed the timing issue by adding some initial interrupts using the “\*” key to poll for the keypad inputs. The final issue we had to overcome was our keypad wasn’t reading in the data properly. We created the keypad using a 2D array set up like the keypad in real life. The original design had an function for reading in the data, this function as the crux of the data reading issue. The original function called a character variable to get the input from the keypad, then checks an if statement to see if the variable is assigned a value. If that’s true, the function checked if the first character was “#” indicating the start of the keypad sequence. The flaw in this code was that the whole function had to loop to get the full command, rather than have a loop within the function. Without the loop it only found the first character and checked if it was the start, this would be checked each time the function was called, which made it so the only time the code would go through was for the “#” symbol rather than the whole sequence. However, in the new code we initialize the character variable, and set up a infinite while loop. In the loop it sets the variable to the read value from the keypad, then returns the character as a string. This works now because the code isn’t looking for the first character in the read function, instead it looks for it later in the code when it needs to take that data and convert it to different states of the lights.

1. **Personal Contribution to the Lab (Technical Details):**

Chris

For this lab, I did a couple of things. Firstly, I designed the state machine used for the design of the stop light algorithm, which was the basis of the software and of the states that the software would transition through. I set the pin configuration for the software and sourced the drivers for the keypad and wrote code to be able to receive messages from it, which is in the main code in lines 1-65. I created the hardware configuration and hardware schematic (Figures 1 and 2) for our setup, as well as authoring the hardware design section of this lab report. I also created a test function using just the state transitions and timers to test the hardware setup without using interrupts to confirm our hardware design before developing the final software script, which was the skeleton for our final software script. I also put the final lab report together for our group.

Dan

I originally was given the task of making the keypad, however our jobs got mixed up so someone else took that job. After that confusion I was tasked with making the main function. Since this was early in the project, I had no idea how to implement some of the code, and some of the other parts weren’t finished yet. So instead, I made my best attempt at the code. It didn’t include interrupts but rather based off time and input data. When I didn’t know how to call things properly, I added pseudo code so it could be used as a template for how the process would go. The code I made essentially took the data from the keypad in pseudo code, the puts it into a while loop to take all the key presses and put it into one big command sting. Then I took that string and made a case statement for each color in the command based on the letter in the command. Then taking the rest of the command I converted the numbers from the command string to be integers to make the time frame for the lights. Within each of the “if” statements I had the set time frame for each of the LEDs to light up. My code didn’t call the assembly or the 7-segment digital display since those weren’t coded yet. Overall, my code was very flawed, but I provided a skeleton for the future code renditions. I was also in charge of writing the problems and solutions section of the report, the lessons learnt section, and the Software design section.

Himadri

Himadri created the bulk of the final software script and sourced the drivers for the 7-segment display; he developed lines 78-288 of the main software. He implemented the state machine and the buzzer into the final software script with working interrupts to change the times for the LEDs. He did a massive amount of troubleshooting and debugging to get the interrupts to work with the state machine transitions, which was likely the most time-consuming part of developing the software. He also set up the working 7-segment display, which also was a rather difficult part when it came to programming this lab, and got our final software script to its final working state by implementing assembly into the program and getting it to work with the rest of the c script.

1. **Lessons Learnt:**

In this lab we learned a multitude of new software and hardware applications. For the hardware applications, we learned how to use the 7-segment display, keypad, and how to use the Arduino microcontroller. Specifically, we learned how to properly wire up the 7-segment display to the Arduino and how to properly hook up a driver to make the display work. The keypad was pretty straight forward, but we learned how to wire up the keypad to the Arduino’s I/O pins. We also learned the locations and applications of the Arduino I/O and pwm pins of the microcontroller. The software applications we learned mainly had to do three main sections. The first being the keypad. We learned how to set up the keypad arrays to take input characters and how to use them in our code to act as commands. The second software we learned how to use was how to set up the 7-segment digital display. We were able to set up hexadecimal values to set up a hex map for the display, along with setting up a time, prescaler, and interrupts all combined in order to make the display work. The next software problem we overcame and learned from was implementing interrupts with the other newly learned skills in this lab. The interrupts were very similar to other times we have used them, but we still learned how to implement them into our keypad, and digital display. The final software application we learned was how to combine assembly and Arduino IDE code. It was relatively simple the way we implemented it. We made one assembly file in charge of turning the red LED on and then called the function to use it for the initial situation where no button is pressed and both red lights have to be on.