ENGR 478 Final Project: Traffic Light

C-Note

Bryan Thorne

Steven Ta

Kevin Thai

Fall 2025

Xiaorong Zhang

George Anwar

Table of Contents

1 Introduction

The goal of our project was to create a traffic light using the STM32L476RG board. We made two different traffic lights that were used to simulate an intersection of two one-way roads. We also wanted to make it more realistic by adding in a pedestrian walk light. We used an IR break beam sensor to detect the presence of the car, and we used a push button for the pedestrians. For the code, we integrated the use of interrupts for the sensors and buttons, then we used the systick timer for any delay that we needed. Our motivation came from seeing these types of intersections in SF and wondering how we could use the STM32 board to make it.

2 Design and Implementation

This section will explain in detail the hardware used such as the sensors and buttons and their implementation. This section will also dive deeper into how the software was set up such as header files and the main code.

2.1. Hardware

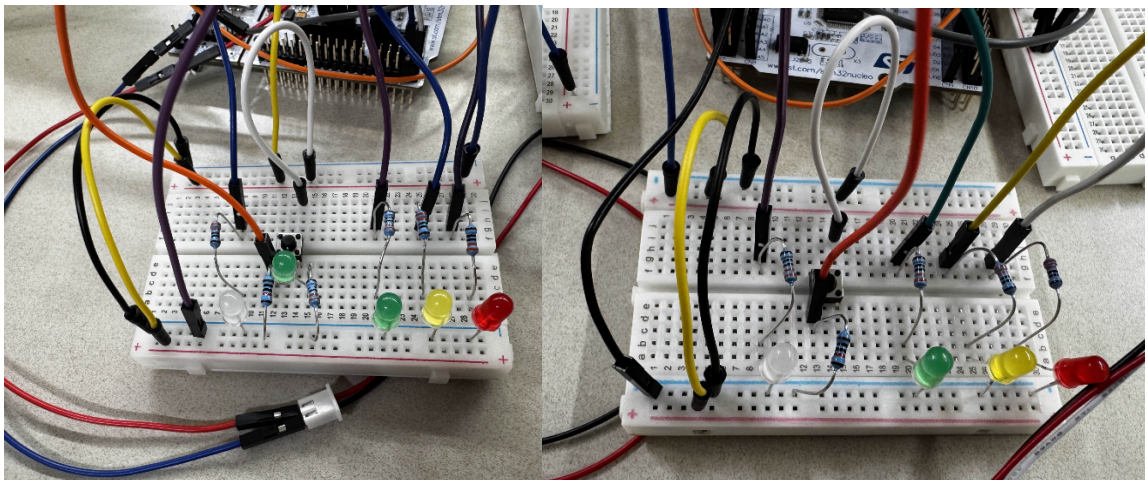

Figure 1: Traffic light connection to breadboard

Components:

- 2 IR break beam sensors
- 2 Push buttons
- Wiring
- 8 LEDs
    - 2 Red
    - 2 Green
    - 2 Yellow
    - 2 Clear
- 2 10kΩ resistors (for the button)
- 8 330Ω resistors (for the LEDs)

We set each traffic light up on a different breadboard for organization, so we could line it up with the correct sensors. We use break beam sensors because it was a great option for the simulation test, a hot wheel car could easily cause an interruption by stopping between the emitter and receiver. We chose push buttons to replicate the use of walk buttons in real life. We connected all the LEDs and buttons in positive logic to make the process easy and less confusing. The sensors are active high, so those are negative logic.

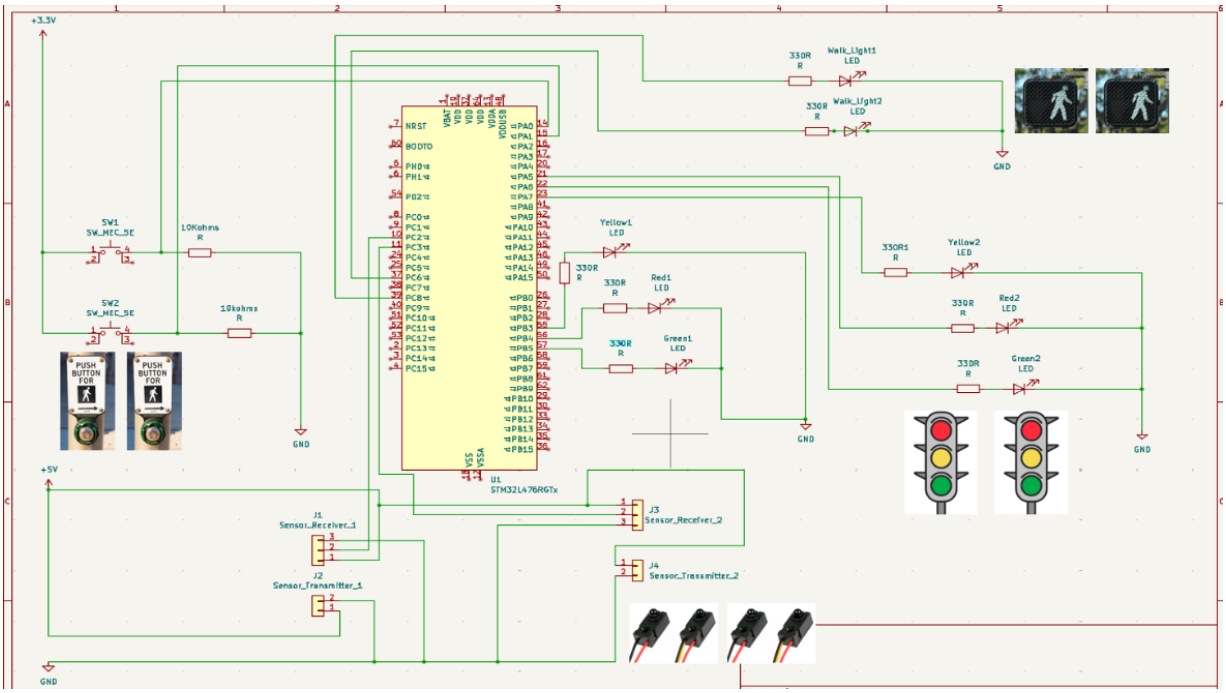

Figure 2: Connection of components and GPIO ports

We set up the traffic lights, buttons, walk lights, and sensors into traffic lights 1 and 2, walk lights 1 and 2, buttons 1 and 2, and then sensors 1 and 2. Sensor 1 was in line with traffic light 1 which turned traffic light 2 red and traffic light 1 green. Sensor 2 did the opposite, turning traffic light 1 red and traffic light 2 green. The buttons controlled both traffic lights and walk lights. Button 1 turned traffic light 2 red, traffic light 1 green, and lastly walk light on and off. Button 2 did the opposite, it's important to remember the input numbers correlate to the output numbers which will be turned green or on. This will be shown better in the flowchart, given in the software section

2.2. Software

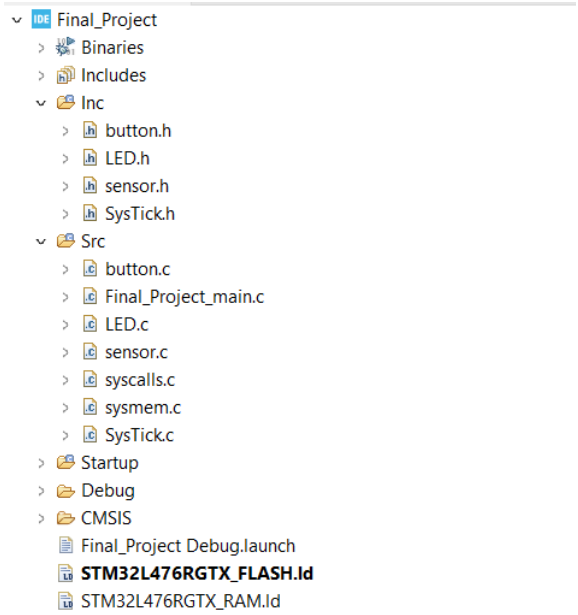

Figure 3: Module hierarchy

For organization purposes we separated the sensor, LED, button, and systick timer configurations into different header and code files. The LED code file is a bit long but clearly notes which lights are being configured. The LED file also includes the functions that turn on and off each light. The EXTI configuration was also done separately in the sensor and button code files. The sensors are active high, so an interruption is caused by a falling edge. The buttons on the other hand are positive logic meaning the interruption is caused by a rising edge. The sensors also require a pull-down resistor, so we implemented that in the configuration. The buttons were set to pull down so that when it's not pressed the value is steadily zero without any errors.

In the main code file called Final_Project_main.c it implements the interruption code going through each option for the sensors and the buttons, shown more clearly in figure 3. Then the main code configures all the inputs and outputs using the functions form the header files. Then lastly the code runs a infinite while loop to wait for an interruption.

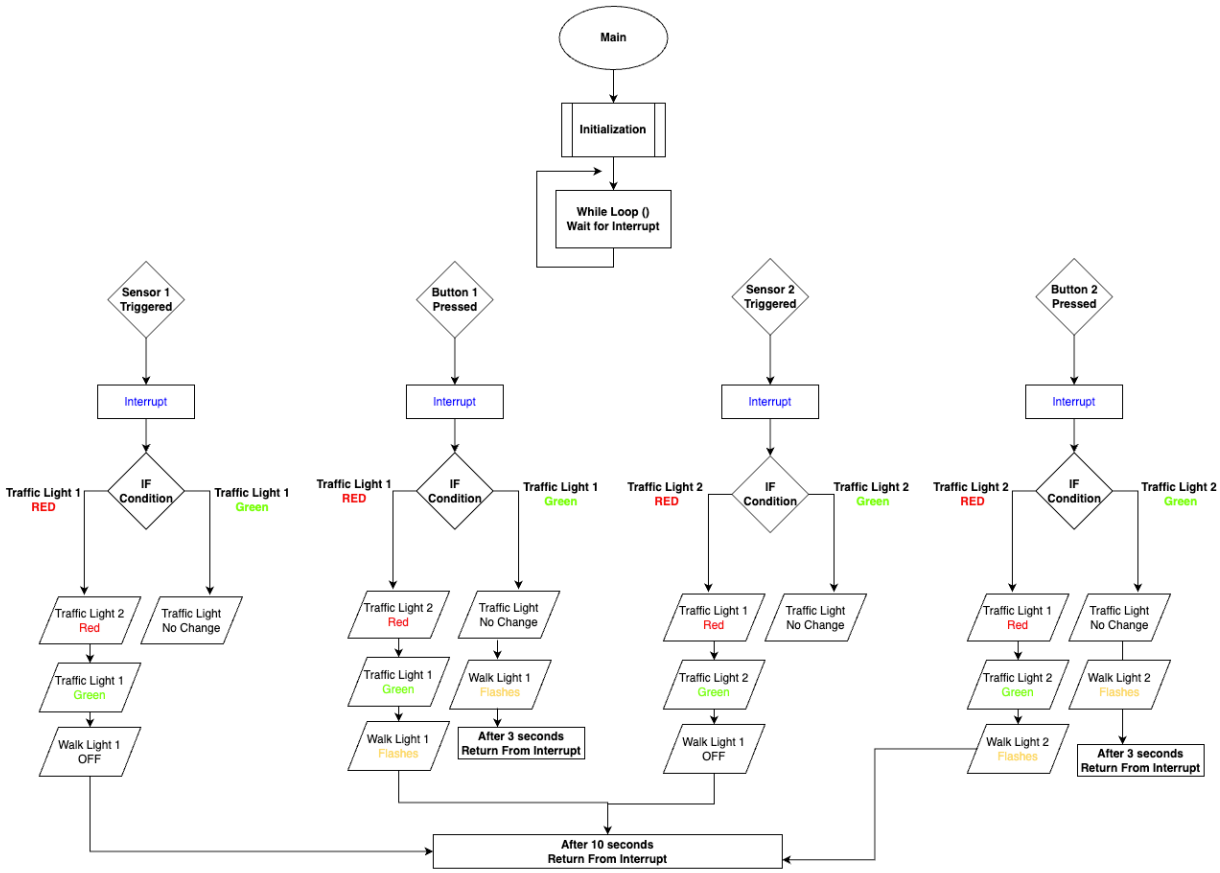

Figure 4: Simplified Flowchart

Figure 3 shows the simplified version of the flowchart; it doesn't include the delays and the turning on and off of the yellow lights. There is an interrupt for each button and sensor, after the interruption the code runs an if-statement. The if-statement checks if the corresponding traffic light is already green. For example, sensor 1 turns traffic light 1 green, but if it's already green nothing has to happen. Then for the buttons, let's say button 1 is pressed it checks if the traffic light 1 green before proceeding. The yellow lights are left out of the flowchart so that it's not to long. The yellow light works like a normal traffic light, the green light turns off, turning the yellow light on, then a delay is used to keep it on for 3 seconds. After that it runs off so the red light can be turned on.

The delays use the systick timer, this is done by changing the configuration of the systick timer. Each interruption uses a 3 second delay before changing any lights, this is so that every light doesn't change right away causing a mess. There is also 200ms delay between turning one light off and another one on so that it isn't instant, and you can actually see the lights change in order. The yellow light also uses a 3 second delay, which is a common amount of time for the yellow light to stay on.

3 Experiments

For the experiments we used the AD2 to make sure each delay is the correct amount of time. We checked the delay for the yellow light, then the delay between the sensors and the yellow light turning on, then the delay for the walk light.

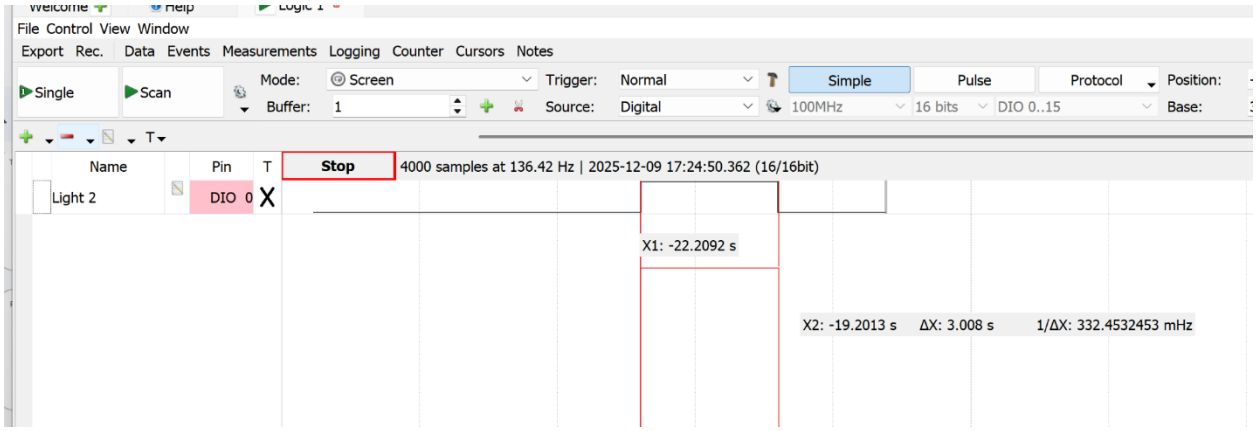

Figure 5: Delay for the yellow light

This delay is 3 seconds because that's the standard time for a yellow light to be on.

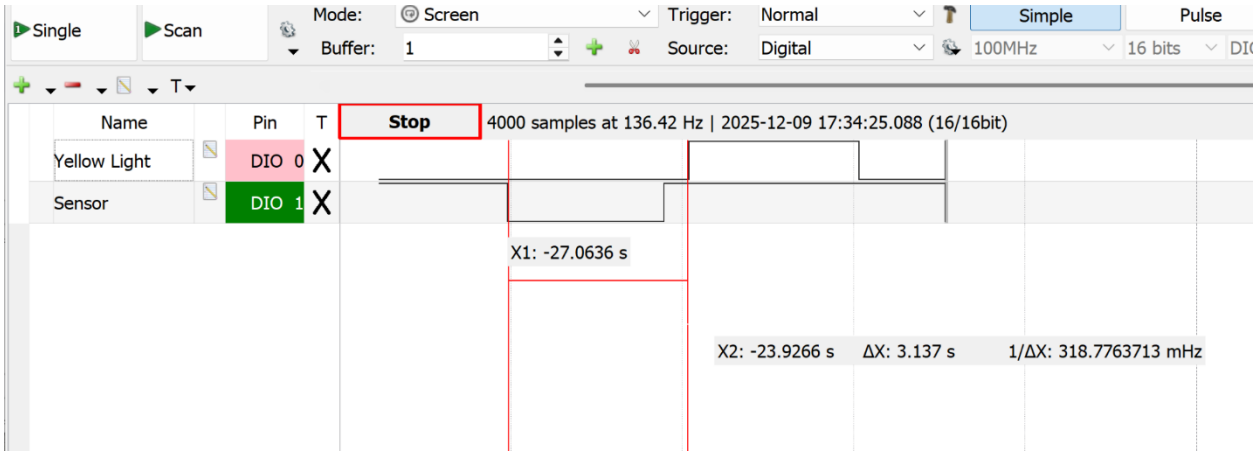

Figure 6: Delay between the sensor and the yellow light being turned on.

This delay should be 3.2 seconds, since the code delays for 3 seconds before turning the green light off, and lastly it delays 200ms before turning the yellow light on.

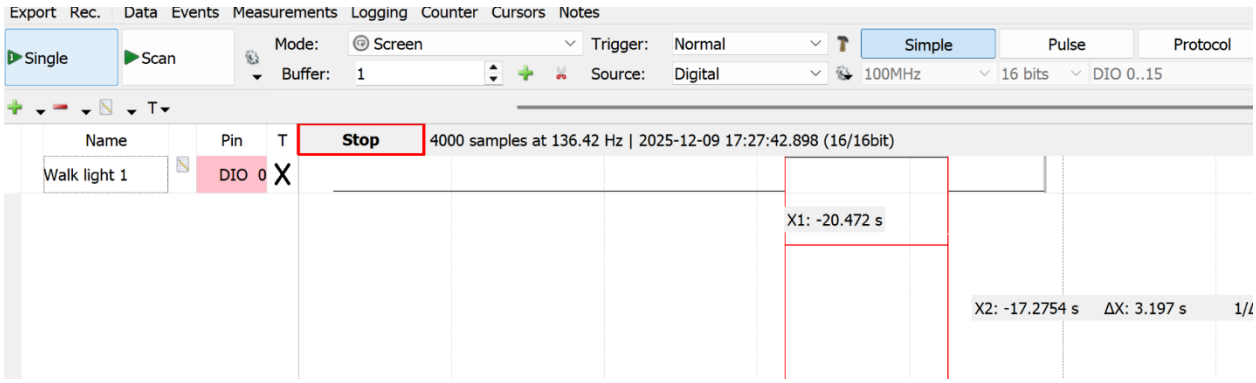

Figure 7: Walk light delay

This delay should be 3.2 seconds since the walk light turns on, delays 200ms to turn on the green light then delays 3 seconds before turning the walk light off.

4 Results and Discussion

The results were successful; we ran a few different tests to show that the sensors and buttons work as intended. We also did a test to imitate two cars approaching at the same time. The sensors, buttons, and interruptions all ran as expected. This all can be shown in the video: https://drive.google.com/file/d/1eMi-ND-s2TatSRykQxVPznd1qfMjvNGL/view?usp=drive_link

For the project, we decided to build around three main constraints: economic, expandability, and safety. This project was a low-cost design of traffic lights, making it fit into the economic constraints. The project is also expandable because it can be produced easily, and it's easy to add on to, such as more traffic lights and buttons. This project also takes safety into account because we're dealing with low voltages and nothing harmful. All these constraints are important in real engineering design, because projects must be affordable, safe to use, and easily produced and expanded.

5 Team Roles

Bryan Thorne: (Lead Engineer) Provided and constructed the coding part of the project and reviewed and approved engineering designs.

Kevin Thai: (Project Manager) Maintained the overall project plan/schedule while communicating between team members and ensuring quality throughout the project

Steven Ta: (Designer Engineer) Contributed to project goals and provided hands-on, detailed support towards the design work, helping to create the prototype for the final product.

6 Conclusion and Future Ideas

In this project we successfully used interrupts and systick timer to simulate an intersection with two traffic lights and two walk lights. The traffic lights were controlled by IR break beam sensors that were interrupted by a car. A button controlled the walk light and was able to change the traffic lights. In the future we could expand the project more by making the intersection two-way roads instead of one-way roads. We could also add in turn lanes. We could improve on the delays that weren't as precise as we wanted, this could also be caused by the experimental process.

7 References

Adafruit Industries. (n.d.). *IR break beam sensor with premium wire header ends – 5 mm LEDs (Product #2168)*. Adafruit. https://www.adafruit.com/product/2168

Magdy, K. (2024, *approx.*). *STM32 SysTick Timer delay_us (Delay Microseconds)*. DeepBluEmbedded. https://deepbluembedded.com/stm32-systick-timer-microseconds-delay-us-delay-function/

We mainly used lecture slides to complete the project.

8 AI Policy

We tried our best to not use any AI because most of the help we needed was in the lecture slides. We did have to use AI to help with using the systick timer as a delay, we tried many options which lead us to AI.