Final Project: Traffic Control System

Carson Jones

December 1st, 2025

Embedded Operating Systems

Introduction

The main goal for this project is to simulate a traffic intersection with four directions that will include traffic lights, pedestrian crossing buttons, and adaptive timing based on sensors. The four lights will include 3 LEDs (red, yellow, and green) while the pedestrian crossing buttons will include 1 white LED to signal that it is okay to cross. The sensors will indicate if there are cars under the lights and reduce the normal time the intersection will switch between a “GO” state. All the processes will be done using FreeRTOS while using concepts like semaphores, mutexes, and queues.

Materials

- 1 Nucleo-F446RE Board
- 16 LEDs (4 Red, 4 Green, 4 Yellow, 4 White)
- 4 Breadboard buttons
- 8 mini bread boards (traffic lights and pedestrian crossing) and 1 large bread board
- 2 large flat cardboard pieces (one for cutting light posts and one for placing the posts on)
- Hot glue gun
- Assortment of wires (there are many wires so plan accordingly)
- 4 HC-SR04 ultrasonic sensors
- 16 220-ohm resistors (for LEDs)
- 4 10k-ohm resistors (for buttons)
- **(Optional)** 4 Toy Cars (for testing)

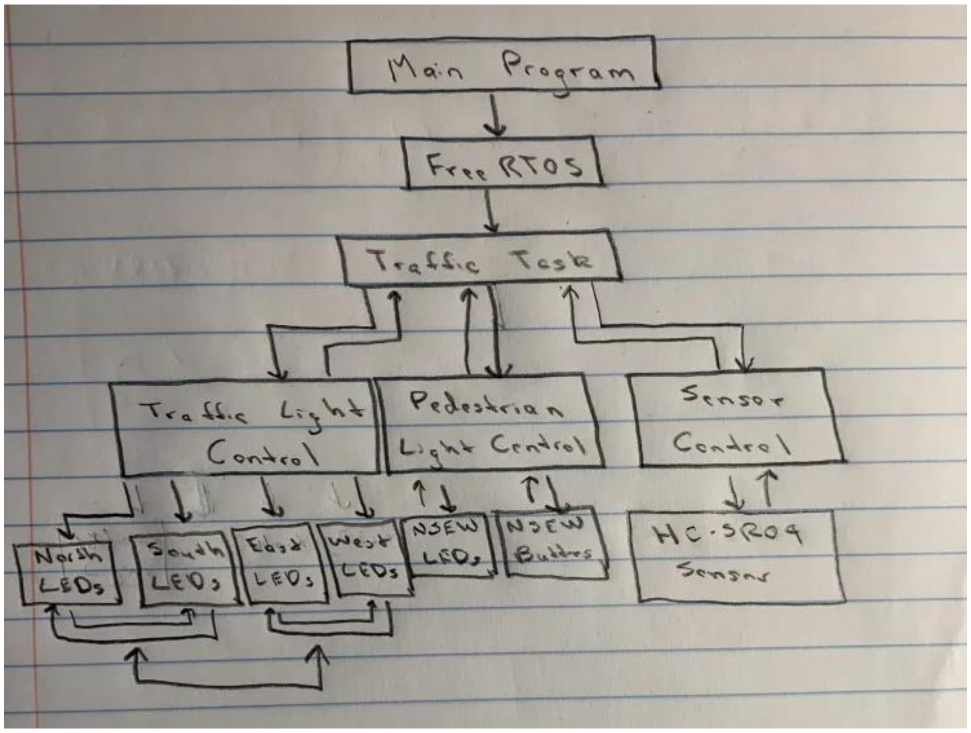
Pricing

- Nucleo-F446RE Board – Roughly \$15 - \$20
- REXQualis Electronics Component Fun Kit (Amazon) - \$13
- 12 Pcs Small Solderable Breadboards (Amazon) - \$12
- Smraza 5pcs Ultrasonic Module HC-SR04 Distance Sensor (Amazon) - \$10
- ELEGOO 17 Values Resistor Kit (Amazon) - \$10
- DiCUNO 450 pcs 5mm LED Diodes (Amazon) - \$12
- **(Optional)** iFunLong 30 Pack Pull Back Cars (Amazon) - \$9

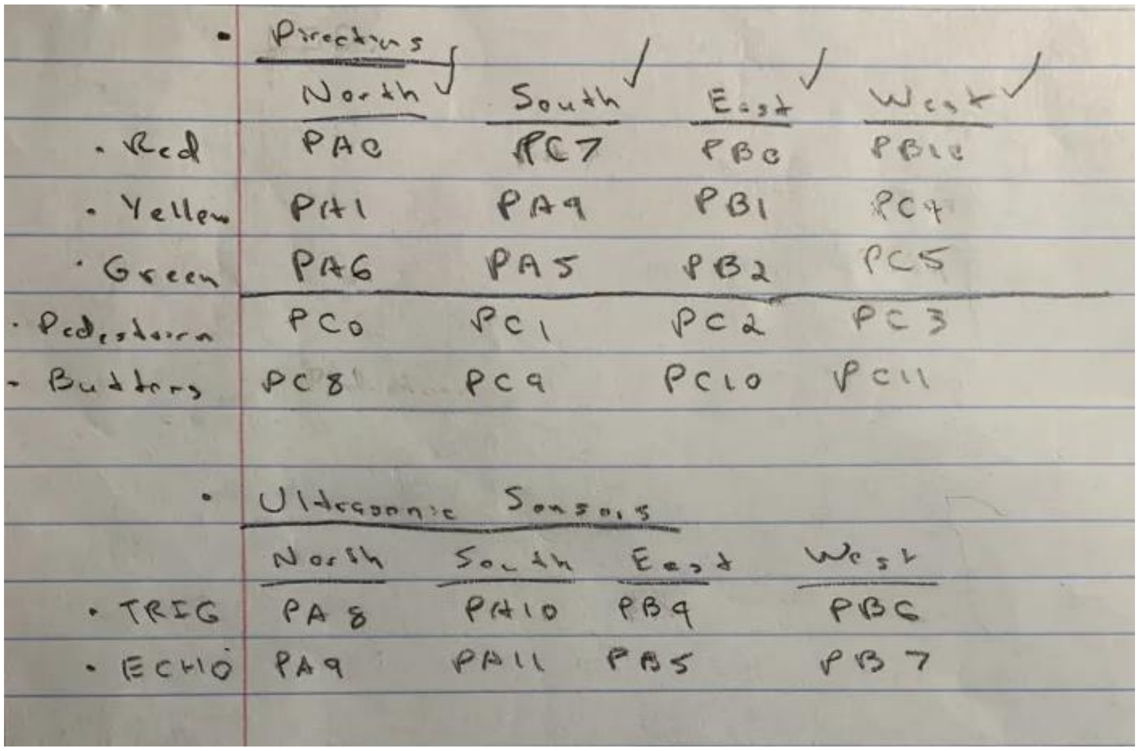
Total Cost: *~\$ 81*

Methods / Results

Block Diagram:



Pinouts for each LED:



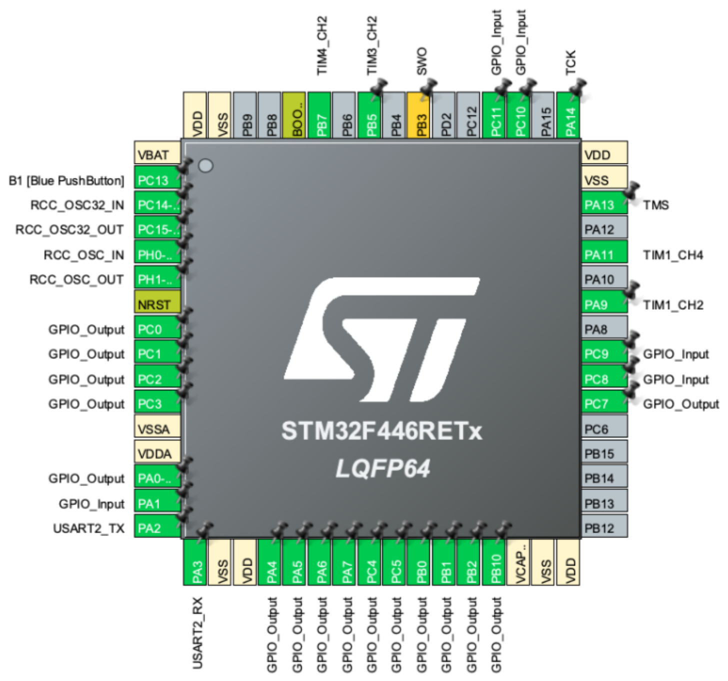
• Directions

	North	South	East	West
• Red	PA0	PC7	PB0	PB10
• Yellow	PA1	PA9	PB1	PC4
• Green	PA6	PA5	PB2	PC5
• Pedestrian	PC0	PC1	PC2	PC3
• Buttons	PC8	PC9	PC10	PC11

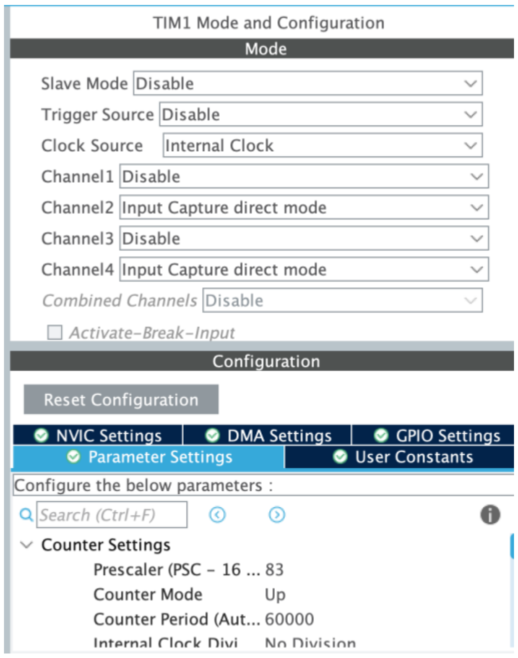
• Ultrasonic Sensors

	North	South	East	West
• TRIG	PA8	PA10	PB4	PB6
• ECHO	PA9	PA11	PB5	PB7

GPIO Pinout diagram on CubeIDE Software:



Timer Initialization and Other Settings:



TIM1 Mode and Configuration

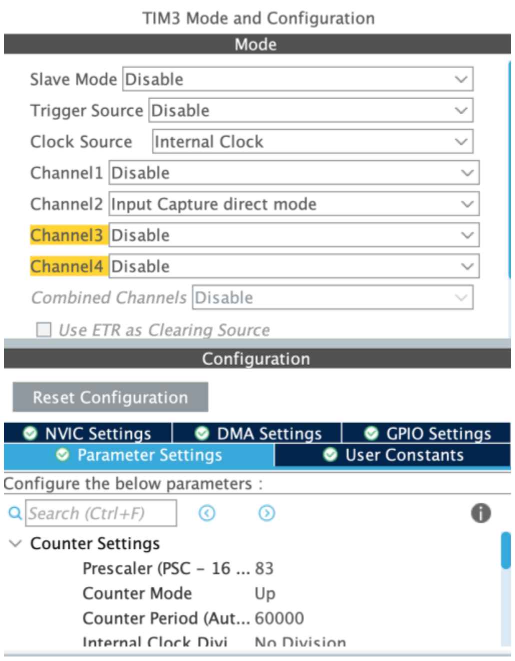
Mode

Slave Mode Disable
Trigger Source Disable
Clock Source Internal Clock
Channel1 Disable
Channel2 Input Capture direct mode
Channel3 Disable
Channel4 Input Capture direct mode
Combined Channels Disable
☐ Activate-Break-Input

Configuration

Configure the below parameters :

Counter Settings
Prescaler (PSC - 16 ... 83
Counter Mode Up
Counter Period (Aut... 60000
Internal Clock Divi No Division



TIM3 Mode and Configuration

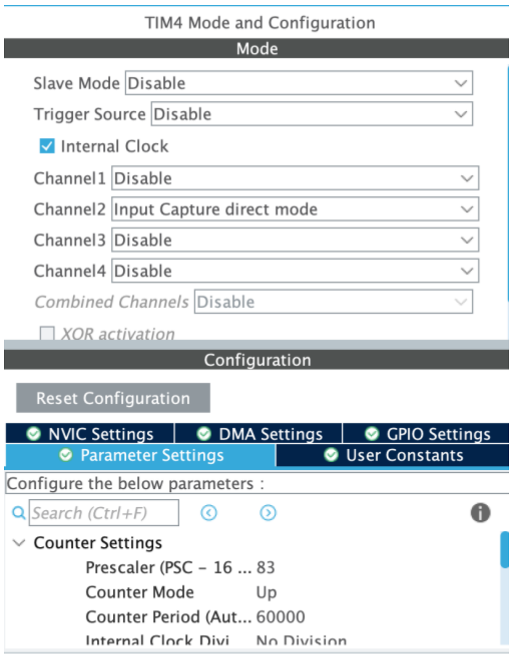
Mode

Slave Mode Disable
Trigger Source Disable
Clock Source Internal Clock
Channel1 Disable
Channel2 Input Capture direct mode
Channel3 Disable
Channel4 Disable
Combined Channels Disable
☐ Use ETR as Clearing Source

Configuration

Configure the below parameters :

Counter Settings
Prescaler (PSC - 16 ... 83
Counter Mode Up
Counter Period (Aut... 60000
Internal Clock Divi No Division



TIM4 Mode and Configuration

Mode

Slave Mode Disable
Trigger Source Disable
☑ Internal Clock
Channel1 Disable
Channel2 Input Capture direct mode
Channel3 Disable
Channel4 Disable
Combined Channels Disable
☐ XOR activation

Configuration

Configure the below parameters :

Counter Settings
Prescaler (PSC - 16 ... 83
Counter Mode Up
Counter Period (Aut... 60000
Internal Clock Divi No Division

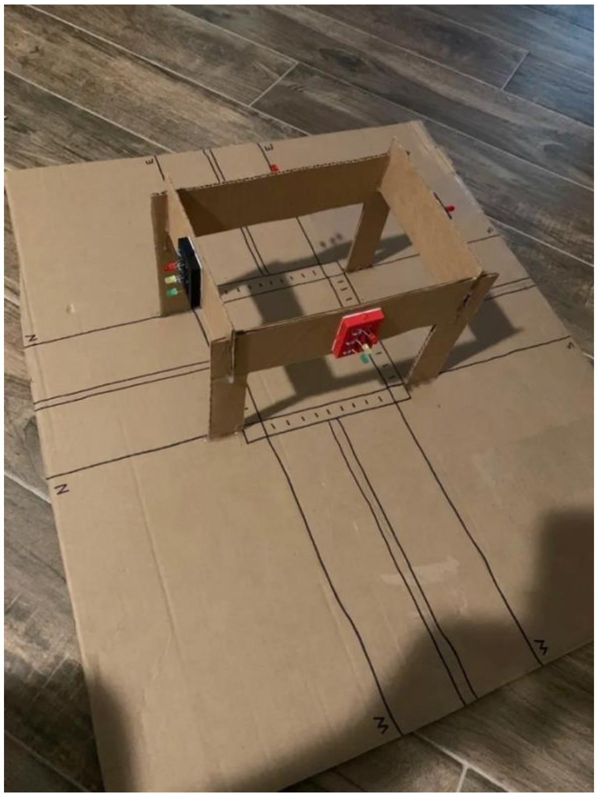
Note: What is shown is what is changed, every other component is the software default. For these timers you will need to activate their global interrupts, you can do this by going to Pinout & Configuration -> System Core -> NVIC then scrolling down to TIM1 capture compare interrupt, TIM3 global interrupt, and TIM4 global interrupt to enable them.

For the block diagram it describes what each task is supposed to do, the main function is an accumulation of the project in which it puts all the pieces together. The FreeRTOS tasks are all centered around the Traffic Task; this task controls the delay response depending on the three other tasks which are the Sensor Task, the Traffic Light Task, and the Pedestrian Task.

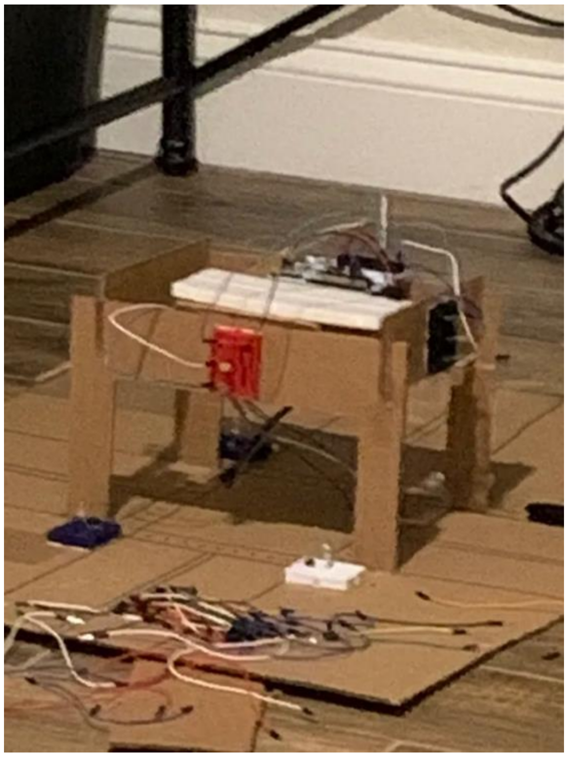
The pinouts listed are what I recommend but can be changed according to your liking, my only recommendation is to not use the pinout PA2 or PA3 since they used TX/RX USART2 respectfully. Doing so will cause problems down the road since they specialize in a specific task, I attempted to make the GPIO_outputs during the initialization process and had nothing but issues trying to get them to function properly.

Photos (Building Process)

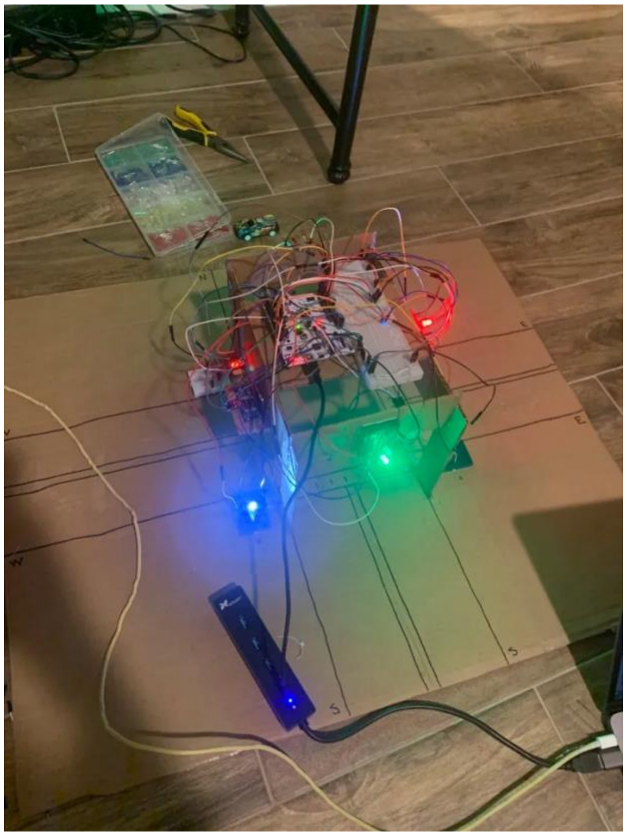
The original concept for intersection (rough sketches).



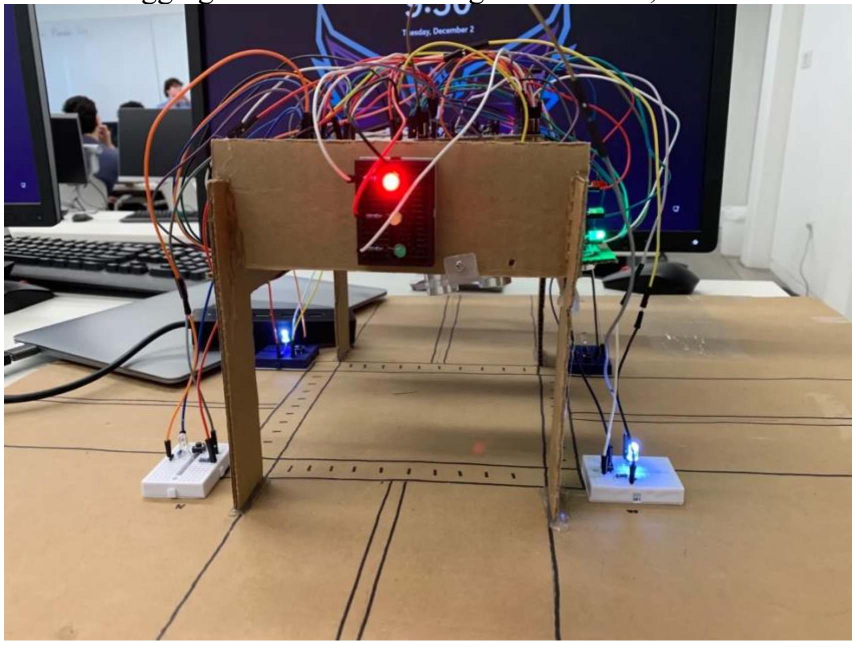
Light posts built with mini breadboards attached to posts (adhesive strip on back).

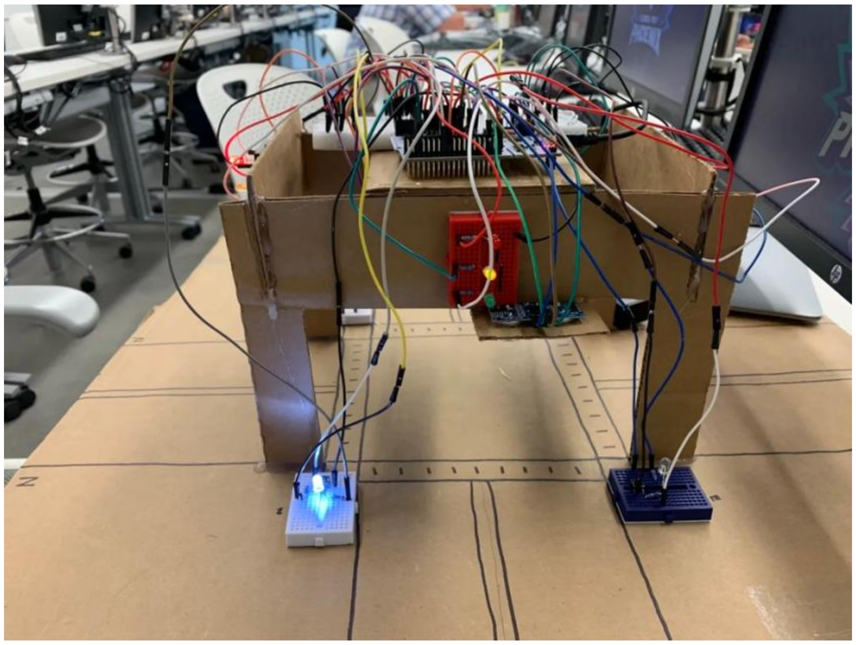


Added cardboard base on top of light posts so Nucleo-F446RE board and Breadboard can stay.



After debugging code and connecting all the wires, the intersection works as intended.





Examples of the process working, light turns yellow right before turning red while the directions change based on what both lights are doing. For example, when the south and north lights are green the east and west lights will turn red (just like a normal four-way intersection). The sensors are attached above each light to determine traffic density, when the lights above the respected sensor are red, the sensors will start detecting cars but will turn off if the lights above them are green. This default process (the lights alternating) happens every 15 seconds, but when the sensor is triggered, it lessens the delay to 7.5 seconds (which is about half the amount of time). When the pedestrian buttons are pressed, the delay is reduced to 12.5 seconds. These pedestrian lights are in sync with the traffic lights and will turn on when the light in the certain direction is red (so they can walk forward).

GitHub Link:

https://github.com/carsonwjo/TrafficLightControl

Conclusion

This project was challenging since I did the entire project solo, it took many hours of debugging for this entire process to work. If I had to change anything in the actual design it would be the sensors since they are finicky, I did not get the sensors working the way I wanted them to. It would inaccurately detect a car when there was nothing there, I tried debugging this process the best I could, but I did not have enough time to continue testing the sensors. Other than that, the entire process was interesting since I learned a lot regarding FreeRTOS and the CubeIDE software (with many mistakes along the way).

References

Arif, Arslan. “HC-SR04 Ultrasonic Sensor with STM32 Nucleo Stm32cubeide.” *Microcontrollers Lab*, 2 Dec. 2025, microcontrollerslab.com/hc-sr04-ultrasonic-sensor-stm32-nucleo-stm32cubeide/.

UM1724 User Manual - STM32 Nucleo-64 Boards (MB1136), www.st.com/resource/en/user_manual/um1724-stm32-nucleo64-boards-mb1136-stmicroelectronics.pdf. Accessed 12 Nov. 2025.