# Traffic Light System

Authored By: Muhammad Ali

Date: March 16, 2021

# Introduction

The Traffic Light System project simulates traffic on a one-way, one-lane road with an intersection that has a single traffic light. In this project LED's are used to represent lights and vehicles and a potentiometer is used as an input device to determine the rate at which traffic is generated in the simulation. Figure 1 below taken from the ECE 455 Lab Manual shows the user interface for the simulation.

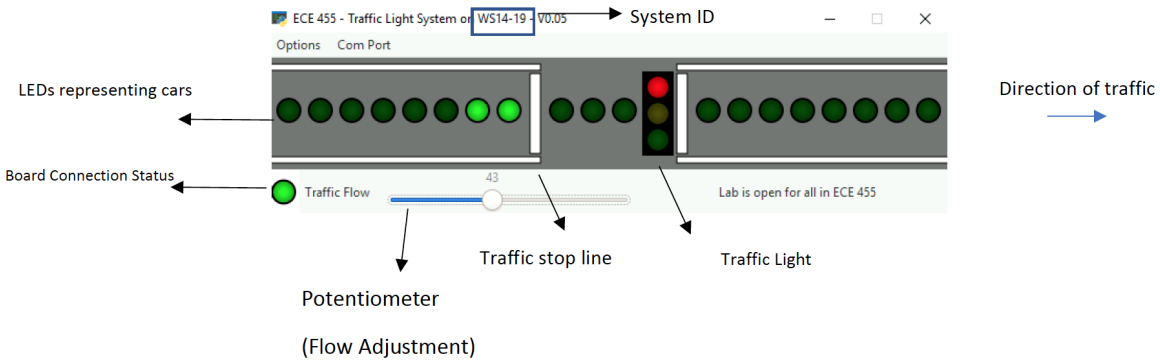


Figure 1 - The user interface of the Traffic Light System simulation

# Initialization of the middleware

## Overview

Pins PC0-PC6 were used for this project. PC0-PC2 were configured as output pins, these pins were used by the application code to display the traffic light state. For example if the current traffic light state was red then PC0 would have a high output signal indicating that the red light was turned on. Pin PC3 was configured as an analog input pin to receive the potentiometer input. PC6-PC8 were the pins associated with the shift register. The shift register (serial in parallel out) was used to provide output to 19 leds which when active each represent the location of a vehicle in the simulation.  The figure below taken from the ECE 455 Lab Manual [1] shows a list of the port pins used and their associated functionality.

ECE 455 Traffic Light Signals

| STM32F0 Port Pin | Signal | Notes |
|---|---|---|
| PC0 | Red Light | Traffic Light |
| PC1 | Amber Light | Traffic Light |
| PC2 | Green Light | Traffic Light |
| | | |
| | | Traffic flow shift register |
| PC8 | Shift Register Reset | Active Low, minimum 1us period |
| PC7 | Shift Register Clock | Falling edge trigger |
| PC6 | Shift Register Data | After clock falling edge data hold time of a minimum of 1 us |
| | | |
| PC3 | Potentiometer input | 0 – 3 Volt Input |

Figure 2 - Pins used for this project with their associated functionality

## Software Setup : GPIO Pins

The STM standard peripheral drivers were used to initialize the GPIO and ADC for this project. The peripheral drivers provide the convenience of not directly having to write to registers. Instead to initialize a GPIO pin an struct is provided by by the driver to configure the GPIO pin settings.

Some of the settings are detailed below:

**GPIO_MODE** - This sets the mode of the pin, it can be set to input, output, analog and other modes.
In this project PC0-PC2 and PC6-PC8 corresponding to the traffic light LED's and shift register were set to output mode. PC3, corresponding to the potentiometer input was set to Analog (ADC).

**OTYPE** - This specified the output type with choices from open drain and push pull.
PushPull was selected for all in the GPIO pins as it appears to be the standard where a low corresponds to the signal being pulled to ground and a high corresponds to the signal being pushed to high. [2]

**PuPd** - The pul down/ pull up settings prevent floating values from occouring by adding a resistor either to VCC or GND.
Pull Down was used for PC0-PC2 and PC6-PC8 corresponding to the traffic light LED's and the shift register. Pull down means that when there is no high signal the state of the signal will be guranteed to be low. For the ADC pin no pull was selected.

**SPEED** - This affects the rise time and fall time of the output.
Options from 2MHz to 100 MHz are available. 2MHz proved to be sufficient for the purpose of this project for all of the GPIO pins.

### Software Setup: ADC initialization

To initialize the ADC the GPIO pin PC3 was initialized as an Analog pin with no Pull followed by the ADC specific settings. One of the settings initialized for the ADC was ADC_Resolution. This setting determines the precision of the input. 12 Bit precision was select which implies that a maximum of 4096 values can be used to represent the voltage input by the potentiometer.

PC3 can be connected to ADC1, ADC2 and ADC 3. ADC  1 was selected for this project and it was connected with channel 13 with a sample time of 56 Cycles.

# Software Design

This project used:

- 3 queues
  - XQueue_State
  - xQueue_light_state
  - xQueue_generate

- 4 tasks
  - Traffic_state
  - Display
  - Traffic_generate
  - Traffic_light_state

## Design Document

The development of this project was a very iterative process.There were many “rough” design documents used. The design document below captures the gist of the design. One of the biggest challenges early in the development of this project was how the traffic state can be represented. After a few iterations an array was selected to represent the state of the traffic. A 1 for an array index indicates that the LED at that index should be on and a 0 indicates it should be off. It was quite clear in the beginning that a queue would be required to send and receive the state of the traffic lights, the amount of new vehicles being generated and to the current state of the vehicles to the display task. A design difference from the beginning and the end was that the display middleware for the traffic lights was called directly from the traffic light task rather than the display task, this was done to avoid using yet another queue.

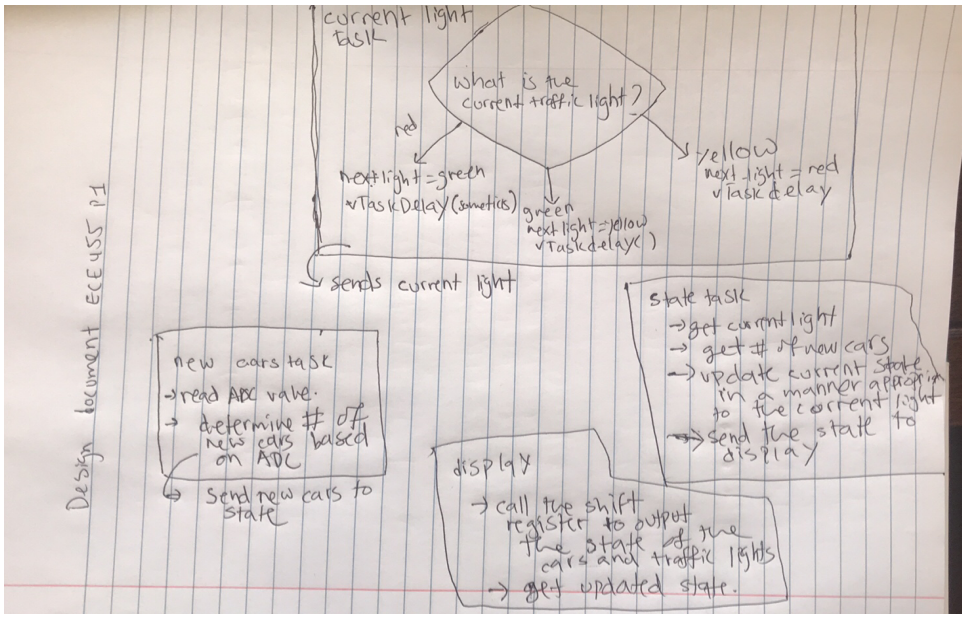


Figure 3 - Design document

## System overview diagram

The following diagram provides an overview of the relationships between the tasks and queues used in the final version of this project.

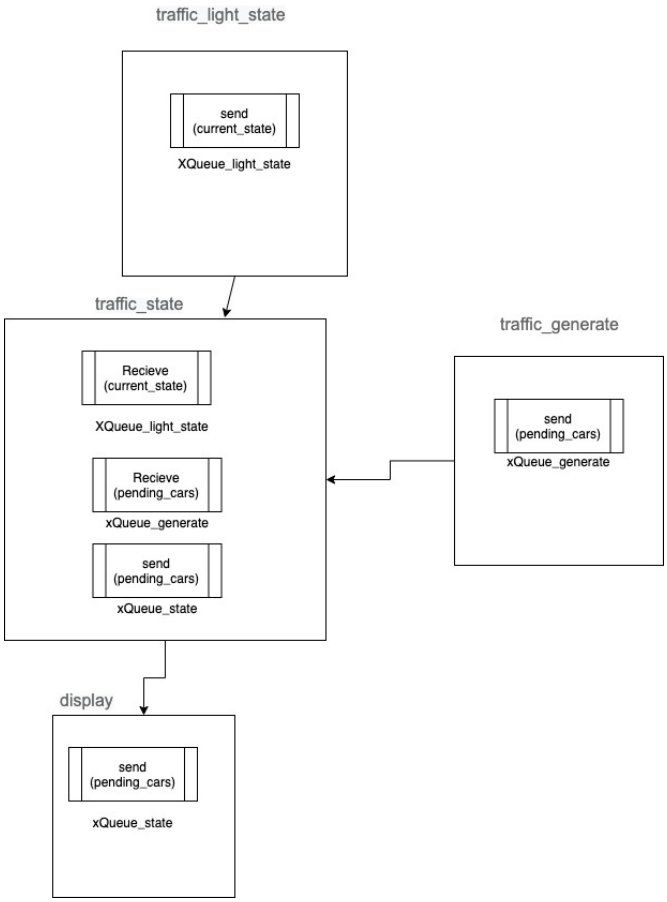


Figure 4 - System overview diagram showing the relationship between the tasks and queues.

## Implementation details

The discussion below summarizes the functionality of each of the tasks and queues.

### Task: traffic_light_state

The traffic_light_state task is the highest priority task (priority =4) of the system. Based on the traffic flow (based on normalized ADC input) the task determines the amount of time to delay the current light. A variable titled “current_state” determines the light that is currently on (a value of 2 corresponds to green, 1 corresponds to yellow and 0 corresponds to red). The task sends the “current_state” value over the “XQueue_light_state” queue and then suspends itself for the amount of time determined by the traffic flow rate and the “current_state”. The “amber light” (current_state=1) has a fixed delay time of 2000 ticks and the “green” and “red” lights (current_state =2 and 0 respectively) have their delay tick time determined by the following formulas:

```
float greenTime = 10 + normalized_adc * 10;

float redTime = 20 - normalized_adc * 10;
```

The formulas dictate that at lowest traffic flow the red light time will be 20(double) and green light time will be 10 and vice versa at the highest traffic flow rate.

### Task: traffic_generate

The “traffic_generate” task has the second highest priority (priority =3). This means that everytime the “traffic_light_state” task is suspended the “traffic_generate” task runs until it suspends its execution or it is pre-empted by the “traffic_light_state” task. This task reads the value from ADC and determines the number of new cars to be created using the formula below:

```
pending_cars = 1 + result * (float)5;
```

Where result is the normalized ADC value and “pending_cars” is the number of vehicles to be generated.

The value of “pending_cars” is sent to the “XQueue_generate” queue and then this task suspends itself for 5000 ticks.

### Task: traffic_state

The “traffic_state task” receives the “pending_cars” value from the “xQueue_generate” queue. It also receives the “current_light”(indicating red,green or amber light currently on) value from the “XQueue_light_state queue”. Thereafter, it executes one of two branches of code depending on the current state ( a separate branch for green light and a branch for yellow and red lights).

The positions of the vehicles are saved in an array of 19 integers named “state”. At the very beginning of the simulation this array is initialized to 0. A supplemental array named “temp” is used to hold the current state and is used to update “state” array for the next 1000 ticks. After the “state” array has been updated its value  is sent over the “xQueue_state” queue and then the task suspends its execution for a time of 1000 ticks.

### Task: display

The display task has the lowest priority. This task receives the current state of the vehicles from the “xQueue_state” queue. It then calls the “display_led” function which is a middleware interacting directly with the shift register to display the current state of the simulation by turning on and off the appropriate LED’s simulating vehicle moment every 1000 ticks.

# Algorithms

## Algorithm: Traffic generating algorithm

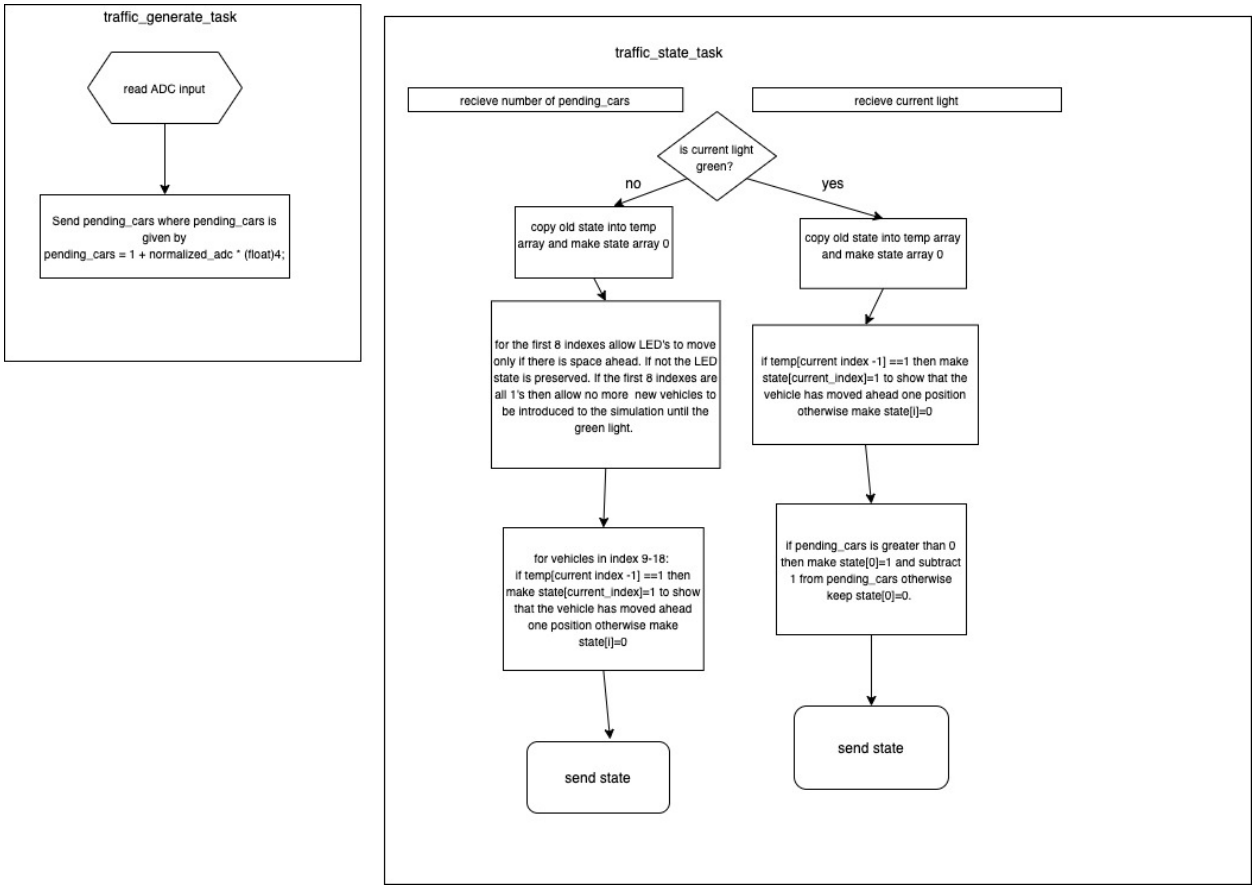


Figure 5 - A flow chart showing the algorithm for traffic generation

## Algorithm: System display algorithm

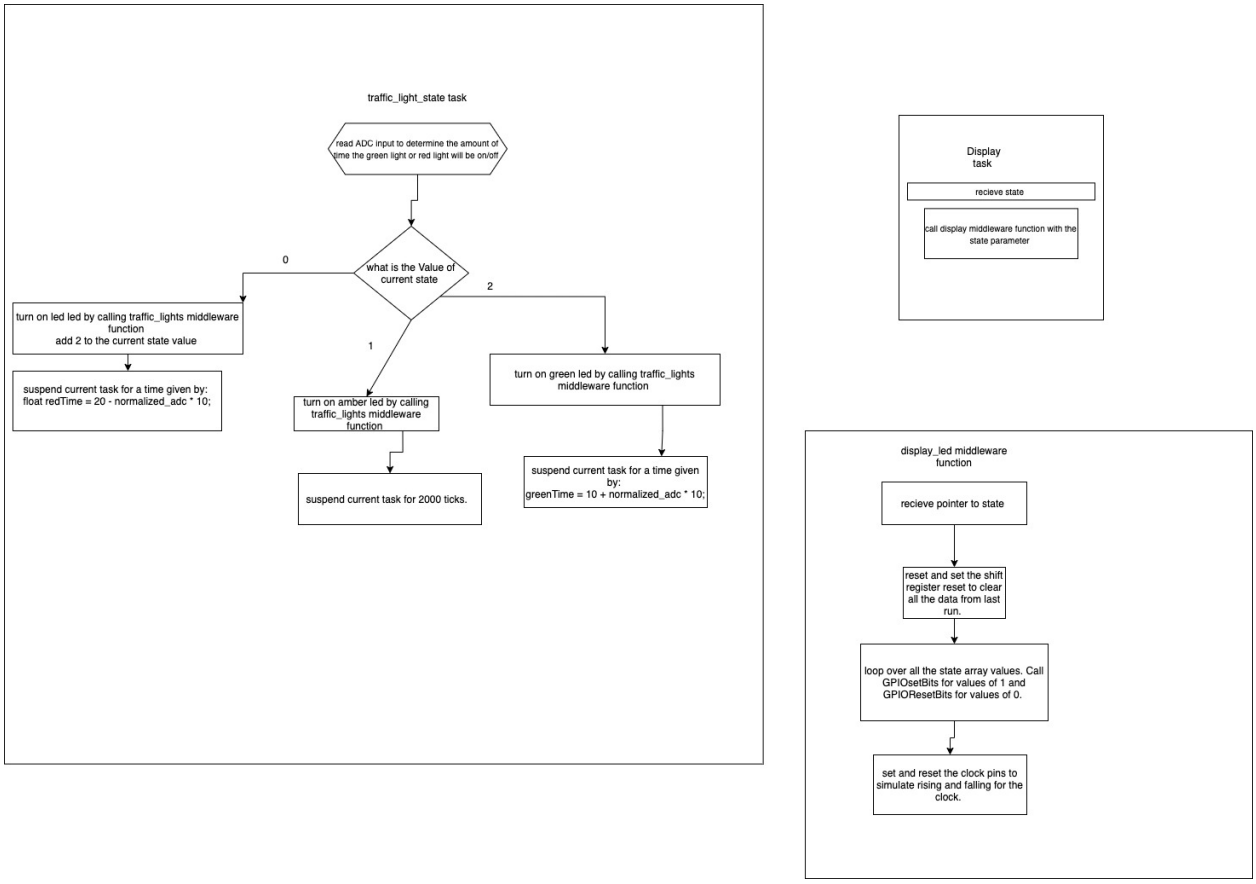


Figure 6 - System display algorithm flow chart

## Limitations and Possible Improvements

The solution could be improved in many ways. Firstly a separate branch can be created for the yellow light state in the “traffic_state” task to improve its readability. It is also possible to implement the “display” task inside the “traffic_state” task by calling the “display_led” middleware function directly from the “traffic_state” task. This would have the improvement of requiring one less queue and one less task thus improving the overall efficiency and responsiveness of the system.

## Summary

The requirements for the Traffic Light System simulation were implemented using 3 queues and 4 tasks. The queues were used for inter-task communication to maintain which traffic light is on, the number of new vehicles and the state of the vehicles currently in the simulation. The 4 tasks were used to control which light would be on, generation of vehicles, maintaining and updating the state of the vehicles on the board and displaying all of the simulation information. System efficiency could be improved by combining some tasks and removing some queues as discussed in the previous section.

## Sources

[1] ECE 455 Lab Manual
[2]https://electronics.stackexchange.com/questions/28091/push-pull-open-drain-pull-up-pull-down