UNIVERSITY OF VICTORIA

Department of Electrical and Computer Engineering

ECE 455 – Real Time Computer Systems

Project 1: Traffic Light System

Report Submitted on: March 20, 2020

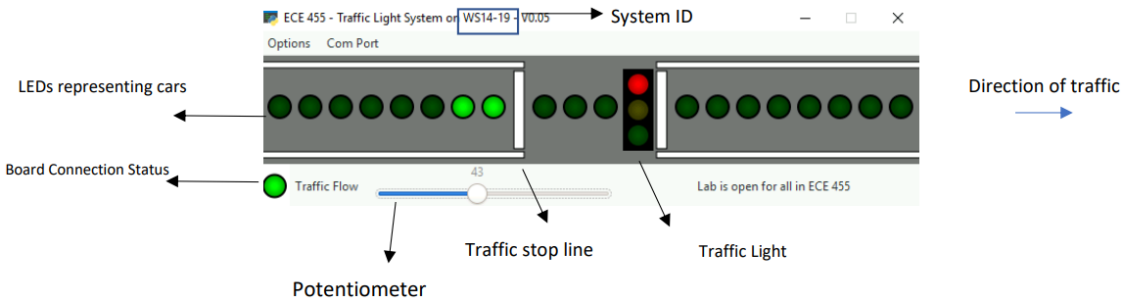
Name: Kutay Cinar - V00******

1.0 Introduction

The Traffic Light System simulates vehicle traffic on a one-way, one-lane road with a simplified intersection that has a single traffic light. The system has three main components:

Potentiometer (Traffic Flow Adjustment):

The potentiometer is used to dynamically adjust the traffic flow rate (i.e. the number of cars created per second) at run-time.



Traffic Light

A traffic light with three LEDs (one green, one yellow, one red) is used to control the traffic at the intersection.

LEDs (Representing Cars)

The LEDs are used to represent the location of cars at various times. An LED being on indicates that there is a car on the road at that LED's location. An LED being off indicates that there is no car at that location. When a vehicle is on the lane it moves by toggling the proper LEDs on or off which allows it to change one spot to the right.

2.0 Design Solution

This project involved 3 major parts. These were the setup of the Hardware (GPIO and ADC), writing of the Middleware and the development of the Application Software parts.

First thing to complete has been to set up the GPIO registers for ports C. As without configuring the registers, there would be no values to play and test with. After all the ports were configured, the three major parts of the project were completed in order in their own functions and tasks.

2.1 Hardware

2.1.1 GPIOC

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. Available pins on the server

Every computer's hardware was pre-configured and linked on the server side to be used by us on labs.engr.uvic.ca . There are a number of pins that can be used with the online board for this project. A list of these pins can be found in Figure 2 above. The traffic lights have three pins, the potentiometer input has one pin, and the shift register has three pins (to load the traffic state). To operate as a single serial unit, three daisy-chained shift registers are used.

Three shift registers are daisy-chained together to form a single serial-to-parallel converter (SPC). Since the discovery board has too few general-purpose input/output (GPIO) pins to map every LED directly to it, SPCs are needed to minimise the number of electrical connections. To maximise the number of outputs from a single input, three shift registers are stacked together.

2.1.2 ADC

The analog voltage signal coming from the potentiometer on the ECE 455 Emulation board was measured by the ADC using GPIOC by using a polling approach. Using the potentiometer's measurements, a resistance value was calculated in the **Traffic Flow Adjustment Task** for loop with the lower and upper limits of the measurable voltage.

```
// Obtain potentiometer value and scale it to a 0-100 range.
pot_value = ( 100 * ( (int) getADCValue() - FLOW_MIN)  / (FLOW_MAX - FLOW_MIN) );
```

This potentiometer value was converted to a 0-100 range by subtracting the lower limit, then diving by the upper limit minus the lower limit, and finally scaling by 100 to obtain this range

2.2 Middleware

A middleware was written that configures the STM32's GPIO pins and the ADC to allow the tasks in the application code to interact with hardware components.

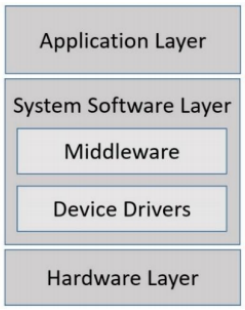


Figure 3: Embedded system model

This required enabling peripheral clocks, defining InitTypeDef structs for initializing both GPIO pins and the ADC in two separate initialization functions (called myGPIOInit and myADCInit, respectively) and writing a function to read the current value of the ADC (called getADCvalue).

2.3 Application Software

Application code is strictly written using FreeRTOS tasks, queues, and timers to manage system resources and simplify hardware and software interactions. The following tasks are created for the Traffic Light System application software:

2.3.1 Traffic Flow Adjustment Task

A potentiometer controls how much traffic flows into the intersection. This task periodically reads the potentiometer's value and passes it on to other tasks. Light traffic is indicated by a low potentiometer resistance, whereas heavy traffic is indicated by a high resistance. This value is then put on a queue to be communicated to other tasks.

2.3.2 Traffic Generator Task

This task produces new traffic at a rate equal to the value of the potentiometer, which is derived from the *Traffic Flow Adjustment Task*. The produced traffic is then routed to a different task to be shown on the lane. An array is used for representing the traffic and cars in the *System Display Task*, however, a queue is used for generating the new cars in this implementation to communicate the new generated cars to arrive at the *System Display Task*.

2.3.3 Traffic Light State Task

This task manages the traffic light control and outputs the current state (green, yellow, and red light). The load of traffic obtained from the *Traffic Flow Adjustment Task* affects the timing of the lights. For more traffic, the length of the green light will be longer proportionally and the duration of the red light will be shorter in an inversely proportional way.

```
xTimer_GREEN     =  xTimerCreate("TMR1", pdMS_TO_TICKS(5000), pdFALSE, 0, vCallbackGreen);
xTimer_YELLOW    =  xTimerCreate("TMR2", pdMS_TO_TICKS(3000), pdFALSE, 0, vCallbackYellow);
xTimer_RED       =  xTimerCreate("TMR3", pdMS_TO_TICKS(5000), pdFALSE, 0, vCallbackRed);
```

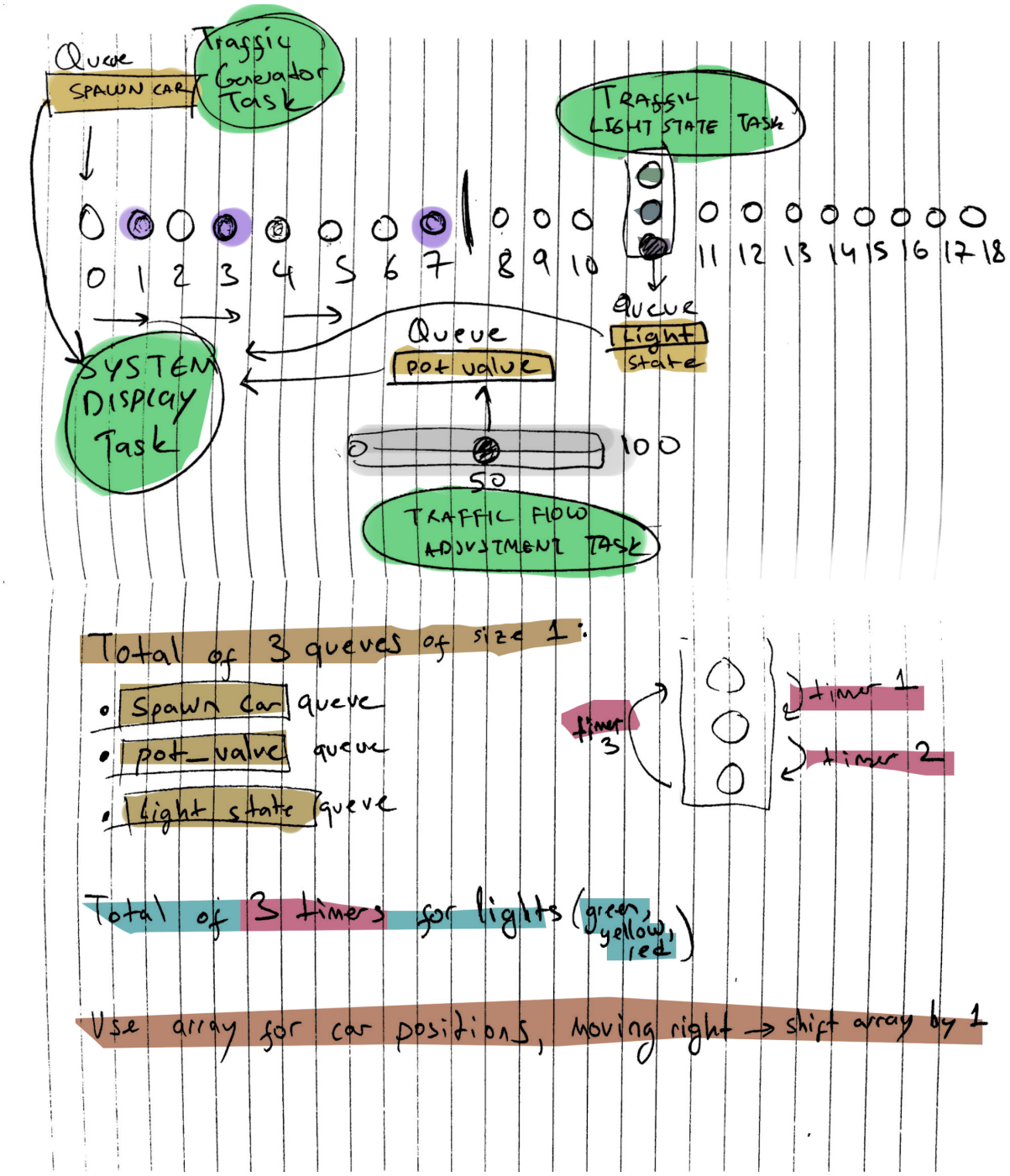
Timers are used for the length of the duration for each of the lights. The green and red light durations are recalculated based on the flow value received from the queue populated by the *Traffic Flow Adjustment Task.*

The callback function of each light's timer starts the next timer in the following cycle where green starts yellow when it finishes, yellow starts red, and red starts green. Additionally, at the end of the yellow light timer, the potentiometer value is read from the queue adjusting the red light timer's duration and at the end of the red timer, the potentiometer value is read from the queue adjusting the green light timer's duration using xTimerChangePeriod.

2.3.4 System Display Task

This task is in charge of visualising both car traffic and traffic lights by controlling all LEDs in the system. It collects data from both the Traffic Generator and the Traffic Light State Tasks with the use of queues and changes the system's LEDs accordingly. This task simulates traffic flow by refreshing the car LEDs at a fixed time.

2.4 Design Document

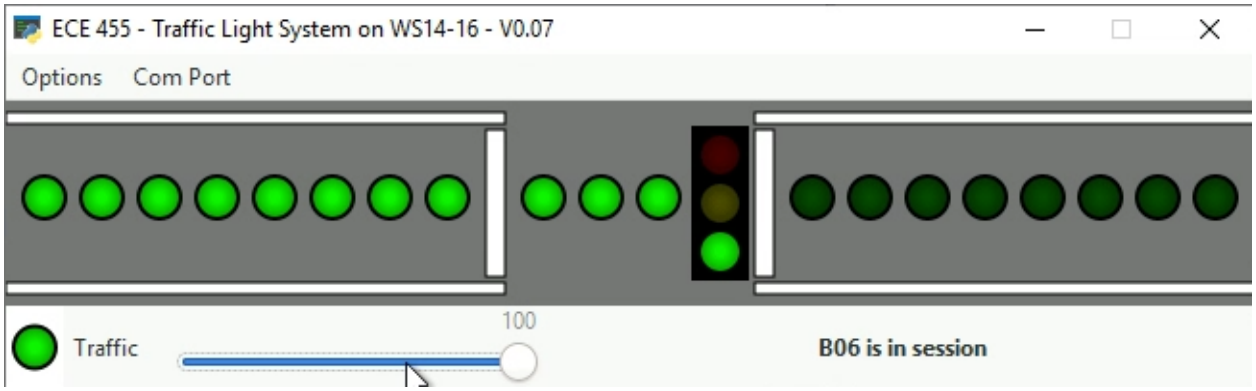


My solution at the end matched my initial design document with the usage of 3 queues (FLOW, LIGHT, GENERATOR), 3 timers (GREEN, YELLOW, RED), and an (int) array to represent traffic and are implemented in my Application Software and were demonstrated in my project demo.

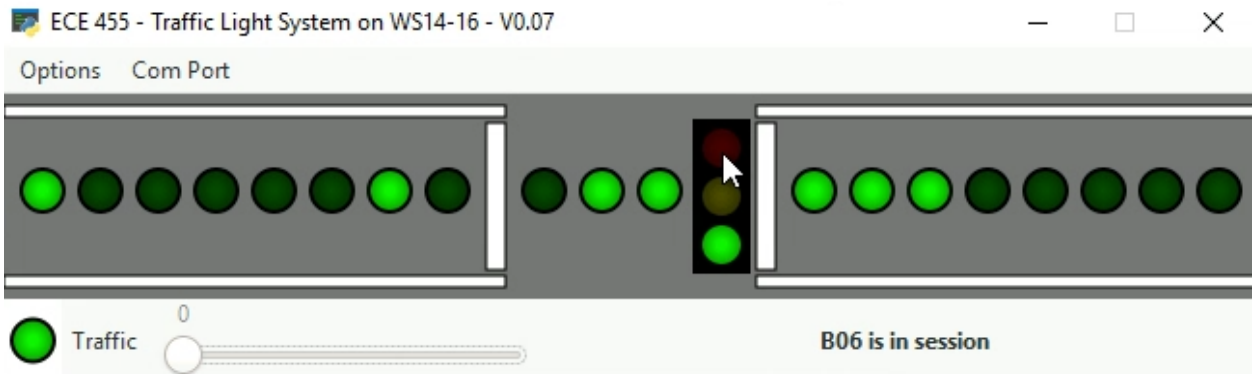
3.0 Testing/Results

The program was tested on the online ECE 455 lab computers using TrueSTUDIO, mainly using ws14-16. Periodically other machines were used when developing the code while ws14-16 was unavailable. Results from ws14-16 machine during the lab demo are as follows:

The corresponding is the result obtained in this project with Traffic Flow at 100.

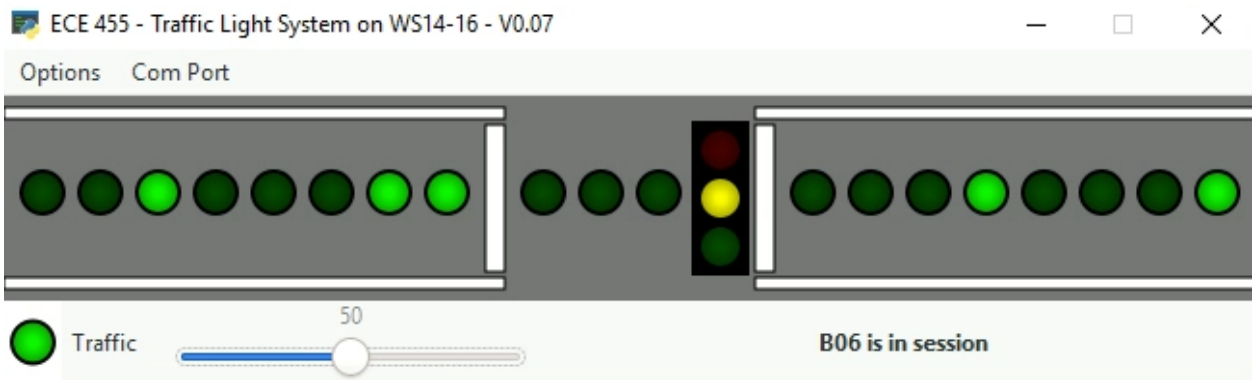


The corresponding is the result obtained in this project with Traffic Flow at 0.

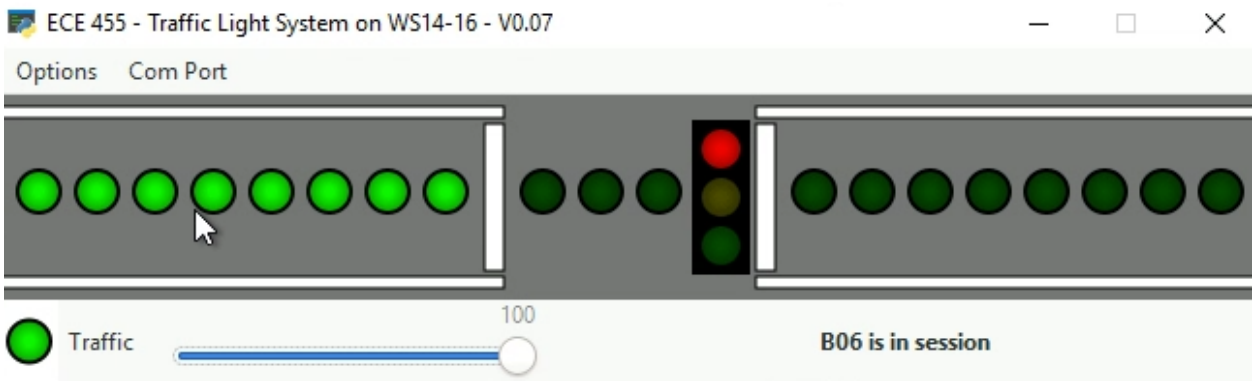


Note the cars will have 5 or 6 spaces between them when coming in from the left-hand side as per requirements.

The corresponding is the result obtained in this project with Traffic Flow at 50.



Note that the cars will start stopping on the left hand side when the light is yellow as per the requirements of the lab manual.



Additionally, as seen above, the cars will stop at red lights and queue up behind the stop sign.

4.0 Discussion

The results obtained in this lab for simulating a one way / one lane traffic and traffic lights are consistent and within the expected limits of lab requirements. Due to the labs being completely online, there were no expectations of creating circuit diagrams and working on breadboards. The virtual nature of the lab also meant that when they were issues, external help had to be relied on by the lab TA or the lab technician.

5.0 Limitations and Possible Improvements

The LCD system and the Signals window on the ECE 455 Emulation board sometimes did not register write operations even though the GPIO ports were set up correctly. This hindered development at times when no other machines were available on labs.engr.uvic.ca. It previously was pointed out in ECE 355 by the lab technician that this issue could be fixed by resetting the board as found out when demonstrating my final project demo.

Potential improvements to this project would be adding new features to reflect a more comprehensive real life traffic system such as varying speeds to cars so that not all cars move at a constant speeds, and additional lanes (LEDS) in the same or opposite directions to expand the project to work on a larger scale and replicate a more real life like environment.

6.0 Summary

The summary of this project is to design and implement a Traffic Light System using Hardware, Middleware, Application Software using FreeRTOS features such as tasks, queues, and software timers. Four main tasks were created in the Application Software, being the *Traffic Flow Adjustment Task*, the *Traffic Generator Task*, the *Traffic Light State Task*, and the *System Display Task*. Three timers were used for the duration of each of the traffic lights and adjusted according to the value read from the potentiometer.

My design in the end did not majorly differ from my initial design document. Overall the lab project has been a fun and enjoyable experience that supported the class material and assignments, and contributed to my learning and understanding of the course.

Appendix

Source code

```
// Kutay Cinar
// V00******

/* Standard includes. */
#include <stdint.h>
#include <stdio.h>
#include "stm32f4_discovery.h"
#include "stm32f4xx_adc.h"
#include "stm32f4xx_gpio.h"

/* Kernel includes. */
#include "stm32f4xx.h"
#include "../FreeRTOS_Source/include/FreeRTOS.h"
#include "../FreeRTOS_Source/include/queue.h"
#include "../FreeRTOS_Source/include/semphr.h"
#include "../FreeRTOS_Source/include/task.h"
#include "../FreeRTOS_Source/include/timers.h"

/*-----------------------------------------------------------*/

#define FLOW_MIN    40
#define FLOW_MAX    4000

#define TICK_RATE   pdMS_TO_TICKS(100)

#define RED                 GPIO_Pin_0
#define YELLOW              GPIO_Pin_1
#define GREEN               GPIO_Pin_2

#define POT                 GPIO_Pin_3

#define DATA                GPIO_Pin_6
#define CLOCK               GPIO_Pin_7
#define RESET               GPIO_Pin_8

#define YES                 1
#define NO                  0

/*-----------------------------------------------------------*/
```

```c
void Traffic_Flow_Adjustment_Task      ( void *pvParameters );
void Traffic_Light_State_Task          ( void *pvParameters );
void Traffic_Generator_Task            ( void *pvParameters );
void System_Display_Task               ( void *pvParameters );

void vCallbackGreen       ( TimerHandle_t xTimer );
void vCallbackYellow      ( TimerHandle_t xTimer );
void vCallbackRed         ( TimerHandle_t xTimer );

void myGPIOC_Init   (void);
void myADC_Init     (void);

uint16_t getADCValue(void);

xQueueHandle xQueue_FLOW = 0;
xQueueHandle xQueue_LIGHT = 0;
xQueueHandle xQueue_GENERATOR = 0;

TimerHandle_t xTimer_GREEN  = 0;
TimerHandle_t xTimer_YELLOW = 0;
TimerHandle_t xTimer_RED    = 0;

/*-----------------------------------------------------------*/

int main(void)
{
	// Initialization functions
	myGPIOC_Init();
	myADC_Init();

	// Create the queue
	xQueue_FLOW  = xQueueCreate( 1, sizeof(int) );
	xQueue_LIGHT = xQueueCreate( 1, sizeof(uint16_t) );
	xQueue_GENERATOR = xQueueCreate( 1, sizeof(int) );

	if( xQueue_FLOW != NULL && xQueue_LIGHT != NULL && xQueue_GENERATOR != NULL)
	{
		xTaskCreate( Traffic_Flow_Adjustment_Task, "Traffic_Flow_Adjustment",
			configMINIMAL_STACK_SIZE, NULL, 1, NULL);
		xTaskCreate( Traffic_Light_State_Task,        "Traffic_Light_State",
			configMINIMAL_STACK_SIZE, NULL, 1, NULL);
		xTaskCreate( Traffic_Generator_Task,          "Traffic_Generator",
			configMINIMAL_STACK_SIZE, NULL, 1, NULL);
		xTaskCreate( System_Display_Task,                    "System_Display",
			configMINIMAL_STACK_SIZE, NULL, 1, NULL);
```

```
            xTimer_GREEN  =     xTimerCreate("TMR1", pdMS_TO_TICKS(5000), pdFALSE,
                                      0, vCallbackGreen);
            xTimer_YELLOW =     xTimerCreate("TMR2", pdMS_TO_TICKS(3000), pdFALSE,
                                      0, vCallbackYellow);
            xTimer_RED    =     xTimerCreate("TMR3", pdMS_TO_TICKS(5000), pdFALSE,
                                      0, vCallbackRed);

            /* Start the tasks and timer running. */
            vTaskStartScheduler();

      }

      for( ;; );

}

/*-----------------------------------------------------------*/

void myGPIOC_Init()
{
      /* Enable clock for GPIO C */
      RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);

      /* Initialize GPIO Structure */
      GPIO_InitTypeDef GPIO_InitStruct;

      /* Enable PC0-PC2 Output */
      GPIO_InitStruct.GPIO_Pin =                        RED | YELLOW | GREEN;
      GPIO_InitStruct.GPIO_Mode =                       GPIO_Mode_OUT;
      GPIO_InitStruct.GPIO_Speed =                      GPIO_Speed_50MHz;
      GPIO_Init(GPIOC, &GPIO_InitStruct);

      /* Enable PC3 Analog */
      GPIO_InitStruct.GPIO_Pin =                        POT;
      GPIO_InitStruct.GPIO_Mode =                       GPIO_Mode_AN;
      GPIO_InitStruct.GPIO_PuPd =                       GPIO_PuPd_NOPULL;
      GPIO_Init(GPIOC, &GPIO_InitStruct);

      /* Enable PC6-PC8 Output */
      GPIO_InitStruct.GPIO_Pin =                        DATA | CLOCK | RESET;
      GPIO_InitStruct.GPIO_Mode =                       GPIO_Mode_OUT;
      GPIO_InitStruct.GPIO_Speed =                      GPIO_Speed_50MHz;
      GPIO_Init(GPIOC, &GPIO_InitStruct);
}
```

```
/*-----------------------------------------------------------*/

void myADC_Init()
{
	/* Enable clock for ADC */
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

	/* Initialize ADC Structure */
	ADC_InitTypeDef ADC_InitStruct;

	/* ADC1 Structure */
	ADC_InitStruct.ADC_ContinuousConvMode =       DISABLE;
	ADC_InitStruct.ADC_DataAlign =                ADC_DataAlign_Right;
	ADC_InitStruct.ADC_Resolution =               ADC_Resolution_12b;
	ADC_InitStruct.ADC_ScanConvMode =             DISABLE;
	ADC_InitStruct.ADC_ExternalTrigConv =         DISABLE;
	ADC_InitStruct.ADC_ExternalTrigConvEdge =     DISABLE;

	/* Apply initialization */
	ADC_Init(ADC1, &ADC_InitStruct);

	/* Enable ADC */
	ADC_Cmd(ADC1, ENABLE);

	/* ADC Channel Config */
	ADC_RegularChannelConfig(ADC1, ADC_Channel_13, 1, ADC_SampleTime_144Cycles);
}

/*-----------------------------------------------------------*/

uint16_t getADCValue(void)
{
	ADC_SoftwareStartConv(ADC1);

	/* Checking for EoC (End of Conversion) flag */
	while (!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC));

	return ADC_GetConversionValue(ADC1);
}

/*-----------------------------------------------------------*/

void Traffic_Flow_Adjustment_Task( void *pvParameters )
{
	int pot_value;
```

```
	for( ;; )
	{
		// Obtain potentiometer value and scale it to a 0-100 range.
		pot_value = ( 100 * ( (int) getADCValue() - FLOW_MIN)  / (FLOW_MAX -
				FLOW_MIN) );

		// Overwrite existing flow in queue with new potentiometer value
		if( xQueueOverwrite(xQueue_FLOW, &pot_value) )
		{
			vTaskDelay( TICK_RATE );
		}
	}
}

/*-----------------------------------------------------------*/

void Traffic_Light_State_Task( void *pvParameters )
{
	xTimerStart( xTimer_GREEN, 0);

	uint16_t LIGHT = GREEN;
	xQueueOverwrite(xQueue_LIGHT, &LIGHT);

	for( ;; )
	{
		vTaskDelay(100 * TICK_RATE );
	}

}

/*-----------------------------------------------------------*/

void vCallbackGreen ( TimerHandle_t xTimer)
{
	xTimerStart( xTimer_YELLOW, 0 );

	uint16_t LIGHT = YELLOW;
	xQueueOverwrite(xQueue_LIGHT, &LIGHT);
}

/*-----------------------------------------------------------*/

void vCallbackYellow ( TimerHandle_t xTimer)
{
	int flow;
	BaseType_t xStatus = xQueuePeek( xQueue_FLOW, &flow, TICK_RATE );
```

```
	if ( xStatus == pdPASS )
	{
		xTimerChangePeriod( xTimer_RED, pdMS_TO_TICKS(10000 - 50 * flow), 0 );

		uint16_t LIGHT = RED;
		xQueueOverwrite(xQueue_LIGHT, &LIGHT);
	}
}

/*-----------------------------------------------------------*/

void vCallbackRed ( TimerHandle_t xTimer)
{
	int flow;
	BaseType_t xStatus = xQueuePeek( xQueue_FLOW, &flow, TICK_RATE );

	if ( xStatus == pdPASS )
	{
		xTimerChangePeriod( xTimer_GREEN, pdMS_TO_TICKS(5000 + 50*flow), 0 );

		uint16_t LIGHT = GREEN;
		xQueueOverwrite(xQueue_LIGHT, &LIGHT);
	}
}

/*-----------------------------------------------------------*/

void Traffic_Generator_Task( void *pvParameters )
{
	BaseType_t xStatus;

	int flow, car = YES;

	for ( ;; )
	{
		xStatus = xQueuePeek( xQueue_FLOW, &flow, TICK_RATE );

		if ( xStatus == pdPASS )
		{
			xQueueOverwrite(xQueue_GENERATOR, &car);

			vTaskDelay( pdMS_TO_TICKS(4000 - 35 * flow) );
		}
	}
```

```
}

/*-----------------------------------------------------------*/

void System_Display_Task( void *pvParameters )
{
	BaseType_t xStatus_LIGHT, xStatus_GENERATOR;

	uint16_t LIGHT;

	int car;
	int cars[20] = {NO};

	GPIO_SetBits(GPIOC, RESET);

	for( ;; )
	{

		// Traffic Lights LED pins
		xStatus_LIGHT = xQueuePeek( xQueue_LIGHT, &LIGHT, TICK_RATE );

		if ( xStatus_LIGHT == pdTRUE )
		{
			GPIO_ResetBits(GPIOC, RED);
			GPIO_ResetBits(GPIOC, YELLOW);
			GPIO_ResetBits(GPIOC, GREEN);

			GPIO_SetBits(GPIOC, LIGHT);
		}

		// Traffic Car LEDS shift register
		for (int i = 20; i>0; i--)
		{
			if ( cars[i] == YES )
				GPIO_SetBits(GPIOC, DATA);
			else
				GPIO_ResetBits(GPIOC, DATA);

			// Cycle
			GPIO_SetBits(GPIOC, CLOCK);
			GPIO_ResetBits(GPIOC, CLOCK);
		}

		// Moving ALL cars on GREEN Lights
		if ( LIGHT == GREEN )
		{
```

```
                for (int i = 19; i > 0; i--)
                        cars[i] = cars[i-1];
        }

        // Conditions for stopping on RED and YELLOW lights
        else
        {
                // Shifting cars before stop sign
                for (int i = 8; i > 0; i--)
                {
                        if (cars[i] == NO)
                        {
                                cars[i] = cars[i-1];
                                cars[i-1] = NO;
                        }
                }

                // Shifting cars after stop sign
                for (int i = 19; i > 9; i--)
                {
                        cars[i] = cars[i-1];
                        cars[i-1] = NO;
                }
        }

        // Check the Traffic Generator Queue to see if a new car has arrived
        xStatus_GENERATOR = xQueueReceive( xQueue_GENERATOR, &car,
                                           pdMS_TO_TICKS(100) );

        cars[0] = NO; // no car by default

        if ( xStatus_GENERATOR == pdTRUE )
        {
                if ( car == YES)
                        cars[0] = YES; // add car if arrived
        }

        vTaskDelay( 5 * TICK_RATE );

    }

}

/*-----------------------------------------------------------*/

void vApplicationMallocFailedHook( void )
```

```
{
	/* The malloc failed hook is enabled by setting
	configUSE_MALLOC_FAILED_HOOK to 1 in FreeRTOSConfig.h.

	Called if a call to pvPortMalloc() fails because there is insufficient
	free memory available in the FreeRTOS heap.  pvPortMalloc() is called
	internally by FreeRTOS API functions that create tasks, queues, software
	timers, and semaphores.  The size of the FreeRTOS heap is set by the
	configTOTAL_HEAP_SIZE configuration constant in FreeRTOSConfig.h. */
	for( ;; );
}

/*-----------------------------------------------------------*/

void vApplicationStackOverflowHook( xTaskHandle pxTask, signed char *pcTaskName )
{
	( void ) pcTaskName;
	( void ) pxTask;

	/* Run time stack overflow checking is performed if
	configconfigCHECK_FOR_STACK_OVERFLOW is defined to 1 or 2.  This hook
	function is called if a stack overflow is detected.  pxCurrentTCB can be
	inspected in the debugger if the task name passed into this function is
	corrupt. */
	for( ;; );
}

/*-----------------------------------------------------------*/

void vApplicationIdleHook( void )
{
	volatile size_t xFreeStackSpace;

	/* The idle task hook is enabled by setting configUSE_IDLE_HOOK to 1 in
	FreeRTOSConfig.h.

	This function is called on each cycle of the idle task.  In this case it
	does nothing useful, other than report the amount of FreeRTOS heap that
	remains unallocated. */
	xFreeStackSpace = xPortGetFreeHeapSize();

	if( xFreeStackSpace > 100 )
	{
		/* By now, the kernel has allocated everything it is going to, so
		if there is a lot of heap remaining unallocated then
		the value of configTOTAL_HEAP_SIZE in FreeRTOSConfig.h can be
```

```
            reduced accordingly. */
        }
}

/*-------------------------------------------------------------*/
```

References

[1] Department of Electrical and Computer Engineering, "Lab Manual ECE 455: Real Time Computer Systems Design Project", University of Victoria, Victoria, 2019.

[2] "STM32F4 reference manual", STMicroelectronics, 2014.

[3] "STM32F4 data sheet", STMicroelectronics, 2014.