University of Victoria
Department of Electrical and Computer Engineering
ECE 455 Spring 2023
B01

Lab Project
Project 1: Traffic Light System

Merlin M'Cloud - V0094624
Jordan Atchison - V00915743
March 7, 2023

Introduction

The goal of this project is to implement a traffic simulation using hardware and software. The hardware components consist of LED's, three shift registers, a potentiometer, resistors, and wires. The software is written in C and uses FreeRTOS, a real time operating system for microcontrollers. The features used for the project include tasks, queues, and software timers.

The project simulates a one way vehicle traffic intersection (no cross street). The car positions are represented by a horizontal line of green LED's. When the LED is on, a car is present at that position, and when an LED is off, no car is currently at the position. Splitting the car LED's into two halves is three other LED's (green, yellow, and red) that represent traffic lights. The cars proceed through the intersection when the light is green or yellow, but must stop at the intersection when the light is red. This means that while the traffic light is red, the cars before the intersection bunch up filling any empty positions. The cars after the intersection aren't affected by the traffic light.

The rate at which traffic is generated is controlled by a potentiometer. The potentiometer value is also proportional to the duration of the traffic lights. This project relies heavily on the flow of data between the hardware and software components.

Design Solution

Our design involved breaking the problem down into smaller subtasks. These could be categorized as circuit design, GPIO initialization, ADC initialization, traffic flow adjustment, traffic generation, LED display, traffic light timers.

Design document

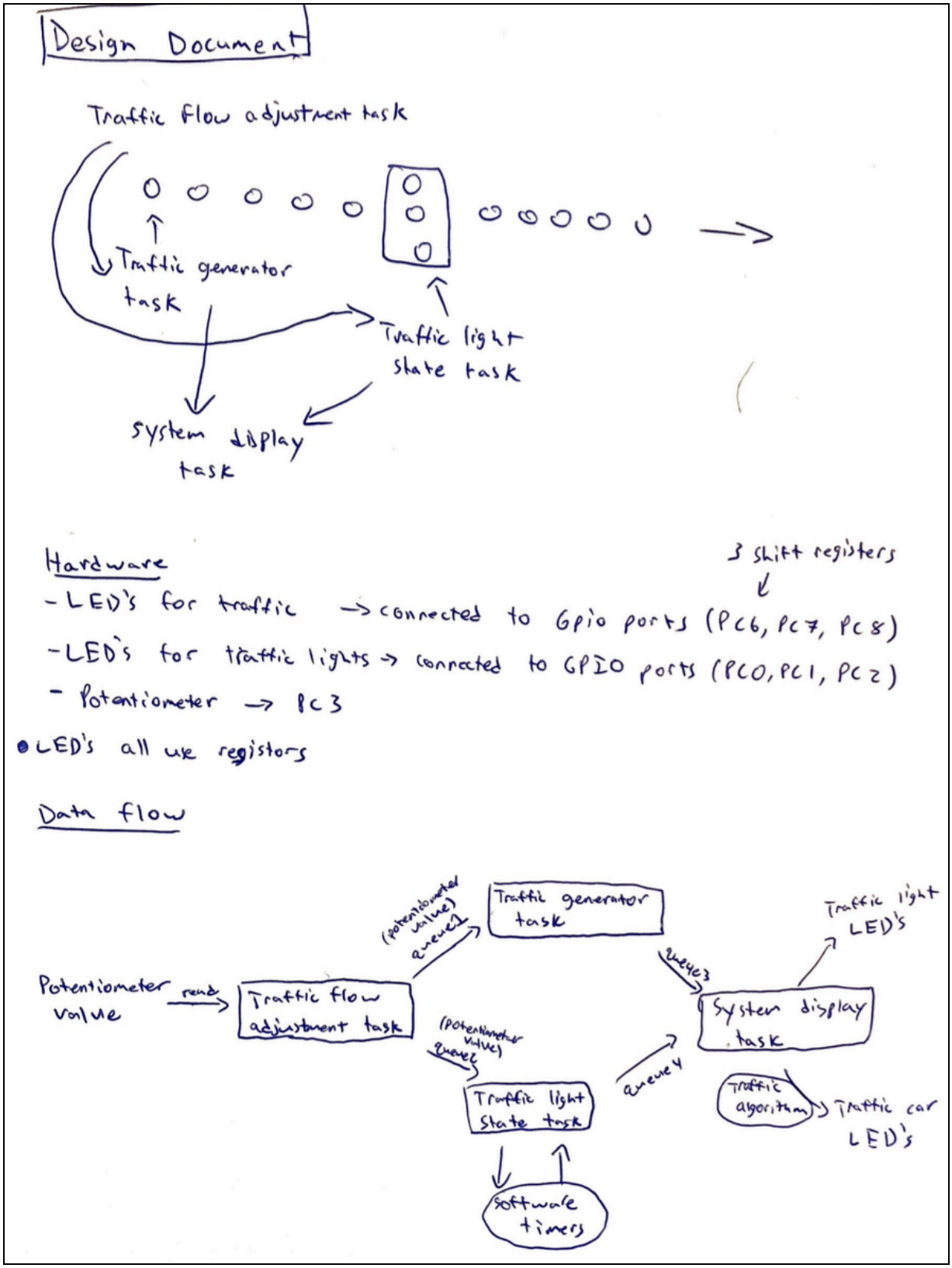

Image 1: Design document

Our design process began with creating an early sketch of the overall data flow and description of how the four tasks and queues worked together. This is shown in image 1 above. Overall, the sketch is a good representation of the final design. The most significant change that we made was the removal of the “traffic light state task”. We instead used software timer callback functions to control the switching and duration of the traffic lights and the task became unnecessary.

Circuit design

Our circuit involved three serialized 74HC164 shift registers. These are serialized by synchronizing the clock and reset pins of all three shift registers, as well as connecting the H pin to the next shift registers data pin. Each output pin from the shift registers was connected to an LED representing a car. Our traffic lights were controlled directly by GPIOB ports. The potentiometer was set up in a standard way, drawing 5 volts and sending its output to GPIOA.

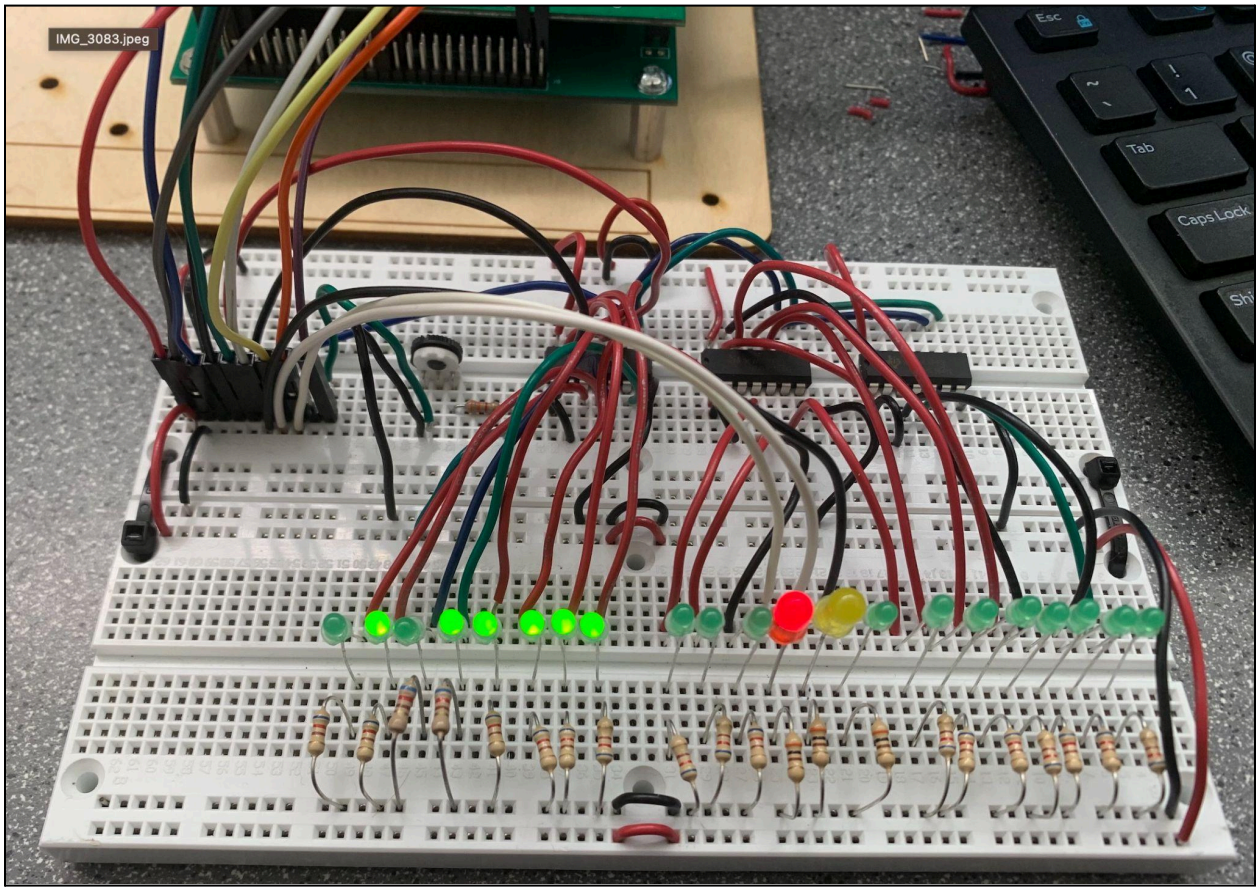

Image 2: circuit design of system.

GPIO initialization

To initialize our GPIO, we first enabled all of the GPIO clocks. We then configured GPIOC for the shift registers, and GPIOB for the traffic light control. These were configured exactly as described in the provided slides. We also made sure to set the shift register reset bit to 0 at this point.

ADC initialization

To initialize our ADC, we first enabled the ADC clock. We then configured the ADC exactly as described in the provided slides. We then enabled the ADC, and set it to channel 1.

Traffic flow adjustment

Our traffic flow adjustment task was responsible for detecting a change in the value coming from the ADC, and if a change was detected, sending this new value to the traffic light timers via a queue.

```
while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC));
        new_adc_val = ADC_GetConversionValue(ADC1);
        if(abs(new_adc_val - adc_val) > 100){
        xQueueReset(adc_light_queue);
            xQueueSend(adc_light_queue,&new_adc_val,1000);
        }
}
```

Code snippet 1: Traffic flow adjustment

The code snippet shows reading the ADC value and updating the value in the queue if the change was significant (bigger than 100 difference).

Traffic generation

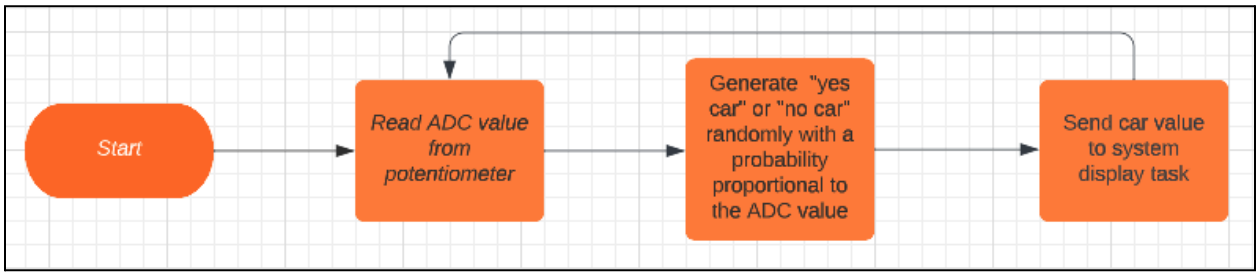

Image 3: Flow chart of traffic generating algorithm

Image 3 depicts a flow chart of our traffic generating algorithm. Each time the task is run it sends a 1 or 0 to the car_position_queue, which is read by the system display task. To generate a random car value, we use the adc value read from the potentiometer to increase or decrease

the probability of a car being sent. Below is a code snippet showing the generation of this value.

```
//generate random between 0 and 5000
uint16_t random = rand() % 4000;
if(random < adc_val + 1000){ //send car
     i = 1;
}
else{ //send no car
     i = 0;
}
```

Code snippet 2: Generating random car value

Our traffic generator task was responsible for generating a random number between 0 and 5000. If the ADC value was less than this random number, it would send a 1 to the LED system display task via a queue. If the ADC value was greater than the random number, it would send a 0 to the LED system display task via queue.

LED system display

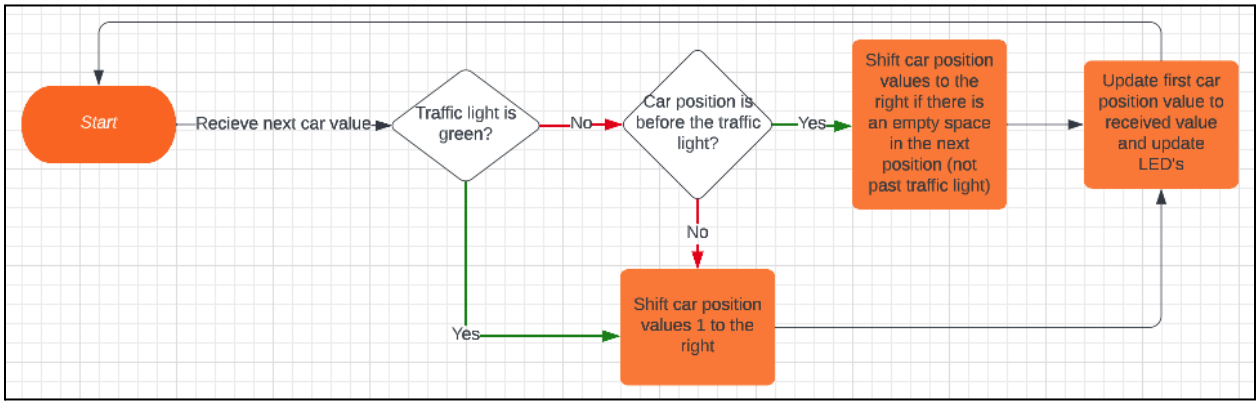

Image 4: Flow chart of system display algorithm

The system display algorithm controls the LED's representing the car position values. A car position LED can be on or off, representing a car at that position or no car at that position. A flow chart describing the algorithm is seen in image 4.

The LED system display task first creates an array of size 19 to represent the cars in traffic and initializes it to be filled with 0s. The first value of the array is set to the value received from the traffic generation task. If the light is green, the array is looped through backwards, setting every value in the array to be equal to the value of the index directly below it. This simulated the movement of traffic. If the light is yellow or red, the array entries after the traffic light are looped through backwards and updated to simulate normal traffic movement, but the entries before the traffic lights are treated differently. They are looped through backwards, but the values are not

updated to simulate traffic movement unless a value of 0 is encountered, at which point a flag is set to 1, and all values encountered after this are updated to simulate traffic movement. This simulates traffic coming to a stop at the red light, and then stacking up close together.

Traffic light timers

We created three timers, one to represent each traffic light color. We then created three callback functions. At the start of our program the light was set to green, and our green light callback function was called.
Once the set time had elapsed, the green light callback function was executed. This function set the light to yellow, sent this value to other functions via queue, and then called the amber light callback function.
Once the set time had elapsed, the yellow light callback function was executed. This function set the light to red, and sent this value to other functions via queue. The period for the red light timer was then calculated based on the ADC value (received from a queue). The period was calculated to be inversely proportional to the ADC value. The red light callback function was then called.
Once the set time had elapsed, the red light callback function was executed. This function set the light to green, and sent this value to other functions via queue. The period for the green light timer was then calculated based on the ADC value (received from a queue). The period was calculated to be directly proportional to the ADC value. The green light callback function was then called.

Discussion

In this project, we attempted to use the specifications given in the lab manual and lab slides as closely as possible. This way, we could focus on ensuring our software was well planned and developed, rather than worrying about if problems were coming from venturing away from the lab manual specifications.

Our implementation was smooth for the most part. Early on, we ran into some trouble getting started with the circuitry and getting LEDs to turn on. This turned out to be mostly due to incorrect initialization functions. Once we got these sorted out, we improved our project steadily throughout the implementation project.

Before writing any software, we planned out how our code would be split into tasks and functions. This made the development process much more efficient, as there were very few instances of creating esoteric and overly complex code. We also made use of the oscilloscope for this project. This was very useful at certain points, as it was at times not clear our software was failing to pick up on an input signal, or if the signal was failing to be sent.

Limitations and Possible Improvements

Looking back on this project, we could have improved our breadboard circuitry techniques. We got caught up in the development cycle and failed to plan well at the start, which created a confusing mess of wires. This was improved later on, but if we had planned out our circuitry before starting the development process it would have saved a lot of headaches and debugging.

Summary

The purpose of this project was to get hands-on experience with real time systems by creating a simulated intersection using LEDs to represent cars and a potentiometer to control the flow of traffic. The project was somewhat difficult to start, as we had limited experience with some of the hardware modules. But due to proper design planning at the beginning, once we got started our project moved along steadily. Most of our bugs originated at the intersection of software and hardware. It was difficult to know if the hardware was failing or if the software was written incorrectly. However, the oscilloscope was very useful when diagnosing these issues.
Despite some setbacks early on, we were able to implement a fully functional project with time to spare before the deadline, which allowed us to spend some time cleaning up our circuitry, as well as our code.

Overall, a very satisfying project once all of the hardware and software components were working well together.

Appendix (with source code)

Source Code

```
/* Standard includes. */
#include <stdint.h>
#include <stdio.h>
#include "stm32f4_discovery.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 mainQUEUE_LENGTH 100
```

```
#define green   0
#define amber   1
#define red     2

#define shift_reg_GPIO_port     GPIOC
#define shift_reg_DATA_pin      GPIO_Pin_6
#define shift_reg_CLOCK_pin     GPIO_Pin_7
#define shift_reg_RESET_pin     GPIO_Pin_8

#define traffic_light_GPIO_port GPIOB
#define red_light_pin           GPIO_Pin_4
#define amber_light_pin         GPIO_Pin_5
#define green_light_pin         GPIO_Pin_6

#define ADC_GPIO_port           GPIOA
#define ADC_pin                 GPIO_Pin_1


static void prvSetupHardware( void );

static void my_ADC_Init();
static void my_GPIO_Init();
static void Traffic_Flow_Adjustment_Task( void *pvParameters );
static void Traffic_Generator_Task( void *pvParameters );
//static void Traffic_Light_State_Task( void *pvParameters );
static void System_Display_Task( void *pvParameters );
static void update_traffic(int n, uint16_t car_positions[]);
static void green_timer_callback();
static void amber_timer_callback();
static void red_timer_callback();

xQueueHandle adc_light_queue = 0;
xQueueHandle adc_traffic_queue = 0;
xQueueHandle traffic_light_queue = 0;
xQueueHandle car_postion_queue = 0;

xTimerHandle green_light_timer;
xTimerHandle amber_light_timer;
xTimerHandle red_light_timer;

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

```c
int main(void)
{

    my_GPIO_Init();
    my_ADC_Init();


    /* Configure the system ready to run the demo.  The clock configuration
    can be done here if it was not done before main() was called. */
    prvSetupHardware();


    /* Create the queue used by the queue send and queue receive tasks.
    http://www.freertos.org/a00116.html */
    adc_light_queue = xQueueCreate(      mainQUEUE_LENGTH,        /* The number of items the queue can hold. */
                                sizeof( uint16_t ) );    /* The size of each item the queue holds. */

    adc_traffic_queue = xQueueCreate(    mainQUEUE_LENGTH,        /* The number of items the queue can hold. */
                                    sizeof( uint16_t ) );    /* The size of each item the queue holds. */

    traffic_light_queue = xQueueCreate(      mainQUEUE_LENGTH,        /* The number of items the queue can hold. */
                                    sizeof( uint16_t ) );    /* The size of each item the queue holds. */

    car_postion_queue = xQueueCreate(    mainQUEUE_LENGTH,        /* The number of items the queue can hold. */
                                        sizeof( uint16_t ) );    /* The size of each item the queue holds. */

    green_light_timer = xTimerCreate("timer_green", pdMS_TO_TICKS(1000), pdFALSE, (void*)0, green_timer_callback );
    amber_light_timer = xTimerCreate("timer_amber", pdMS_TO_TICKS(1500), pdFALSE, (void*)0, amber_timer_callback );
    red_light_timer = xTimerCreate("timer_red", pdMS_TO_TICKS(1000), pdFALSE, (void*)0, red_timer_callback );
```

```
    /* Add to the registry, for the benefit of kernel aware debugging. */
    vQueueAddToRegistry( adc_light_queue, "adc_light_queue" );
    vQueueAddToRegistry( adc_traffic_queue, "adc_traffic_queue" );
    vQueueAddToRegistry( traffic_light_queue, "traffic_light_queue" );
    vQueueAddToRegistry( car_postion_queue, "car_postion_queue" );

    xTaskCreate( Traffic_Flow_Adjustment_Task, "Traffic_Flow",
configMINIMAL_STACK_SIZE, NULL, 3, NULL);
    xTaskCreate( Traffic_Generator_Task, "Traffic_Generator", configMINIMAL_STACK_SIZE,
NULL, 2, NULL);
    //xTaskCreate( Traffic_Light_State_Task, "Traffic_Light_State",
configMINIMAL_STACK_SIZE, NULL, 1, NULL);
    xTaskCreate( System_Display_Task, "System_Display", configMINIMAL_STACK_SIZE, NULL,
1, NULL);

    //turn on green light, start green light timer
    GPIO_SetBits(traffic_light_GPIO_port, green_light_pin);
    xTimerStart(green_light_timer, 0);

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



    return 0;
}


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

static void my_GPIO_Init(){

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOE, ENABLE);

    GPIO_InitTypeDef Shift_Init_Struct;

    Shift_Init_Struct.GPIO_Pin = shift_reg_DATA_pin | shift_reg_CLOCK_pin |
shift_reg_RESET_pin;
```

```
    Shift_Init_Struct.GPIO_Mode = GPIO_Mode_OUT;
    Shift_Init_Struct.GPIO_OType = GPIO_OType_PP;
    Shift_Init_Struct.GPIO_PuPd = GPIO_PuPd_NOPULL;
    Shift_Init_Struct.GPIO_Speed = GPIO_Speed_2MHz;
    GPIO_Init(GPIOC, &Shift_Init_Struct);
    GPIO_SetBits(shift_reg_GPIO_port, shift_reg_RESET_pin);

    GPIO_InitTypeDef Shift_Init_Struct_light;

    Shift_Init_Struct_light.GPIO_Pin = red_light_pin | amber_light_pin |
green_light_pin;
    Shift_Init_Struct_light.GPIO_Mode = GPIO_Mode_OUT;
    Shift_Init_Struct_light.GPIO_OType = GPIO_OType_PP;
    Shift_Init_Struct_light.GPIO_PuPd = GPIO_PuPd_NOPULL;
    GPIO_Init(GPIOB, &Shift_Init_Struct_light);

    GPIO_SetBits(traffic_light_GPIO_port, red_light_pin);
    GPIO_ResetBits(traffic_light_GPIO_port, red_light_pin);

    GPIO_SetBits(traffic_light_GPIO_port, amber_light_pin);
    GPIO_ResetBits(traffic_light_GPIO_port, amber_light_pin);

    GPIO_SetBits(traffic_light_GPIO_port, green_light_pin);
    GPIO_ResetBits(traffic_light_GPIO_port, green_light_pin);

}

static void my_ADC_Init(){
    GPIO_InitTypeDef ADC_GPIO_Init_Struct;
    ADC_InitTypeDef ADC_Init_Struct;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

    ADC_GPIO_Init_Struct.GPIO_Pin = ADC_pin;
    ADC_GPIO_Init_Struct.GPIO_Mode = GPIO_Mode_AN;
    ADC_GPIO_Init_Struct.GPIO_PuPd = GPIO_PuPd_NOPULL;
    GPIO_Init(ADC_GPIO_port, &ADC_GPIO_Init_Struct);

    ADC_Init_Struct.ADC_ScanConvMode = DISABLE;
    ADC_Init_Struct.ADC_Resolution = ADC_Resolution_12b;
    ADC_Init_Struct.ADC_ContinuousConvMode = ENABLE;
    ADC_Init_Struct.ADC_ExternalTrigConv = DISABLE;
```

```
    ADC_Init_Struct.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None;
    ADC_Init_Struct.ADC_DataAlign = ADC_DataAlign_Right;
    ADC_Init_Struct.ADC_NbrOfConversion = 1;
    ADC_Init(ADC1, &ADC_Init_Struct);

    ADC_Cmd(ADC1, ENABLE);
    ADC_RegularChannelConfig(ADC1, ADC_Channel_1, 1, ADC_SampleTime_3Cycles);
}

static void green_timer_callback(){
    uint16_t traffic_light_val = amber;
    if( xQueueSend(traffic_light_queue, &traffic_light_val, 1000)){
        GPIO_ResetBits(traffic_light_GPIO_port, green_light_pin);
        GPIO_SetBits(traffic_light_GPIO_port, amber_light_pin);
    }

    xTimerStart(amber_light_timer, 0);

}

static void amber_timer_callback(){
    uint16_t adc_val;
    uint16_t traffic_light_val = red;
    if( xQueueSend(traffic_light_queue, &traffic_light_val, 1000)){
        GPIO_ResetBits(traffic_light_GPIO_port, amber_light_pin);
        GPIO_SetBits(traffic_light_GPIO_port, red_light_pin);
    }
    if(xQueuePeek(adc_light_queue, &adc_val, 100)){
        int period = 1000*(8000/adc_val);
        if(period > 4000){
            period = 4000;
        }
        printf("adc_val: %d\n", adc_val);
        printf("Red light period: %d\n", period);
        xTimerChangePeriod(red_light_timer, pdMS_TO_TICKS(period), 0);
    }
    xTimerStart(red_light_timer, 0);
}

static void red_timer_callback(){
    uint16_t adc_val;
    uint16_t traffic_light_val = green;
```

```
    if( xQueueSend(traffic_light_queue, &traffic_light_val, 1000)){
        GPIO_ResetBits(traffic_light_GPIO_port, red_light_pin);
        GPIO_SetBits(traffic_light_GPIO_port, green_light_pin);
    }
    if(xQueuePeek(adc_light_queue, &adc_val, 500)){
        int period = adc_val + 1000;
        printf("Green light period: %d\n", period);
        xTimerChangePeriod(green_light_timer, pdMS_TO_TICKS(period), 0);
    }
    xTimerStart(green_light_timer, 0);
}

static void Traffic_Flow_Adjustment_Task( void *pvParameters ){

    uint16_t adc_val = 0;
    uint16_t new_adc_val;

    while(1){

        ADC_SoftwareStartConv(ADC1);
        while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC));

        new_adc_val = ADC_GetConversionValue(ADC1);

        if(abs(new_adc_val - adc_val) > 100){

            xQueueReset(adc_light_queue);
            if( xQueueSend(adc_light_queue,&new_adc_val,1000))
            {
                //printf("ADC Light Queue : %u\n", new_adc_val);
            }
            else
            {
                printf("Traffic_Flow_Adjustment_Failed!\n");
            }
        }

        if( xQueueSend(adc_traffic_queue,&new_adc_val,1000))
        {
            //printf("ADC Traffic Queue : %u\n", new_adc_val);
        }
        else
```

```
        {
            printf("Traffic Flow Adjustment Failed!\n");
        }
        vTaskDelay(1000);
    }
}

static void Traffic_Generator_Task( void *pvParameters )
{
    uint16_t adc_val;
    uint16_t i = 0;


    while(1)
    {
        uint16_t random = rand() % 4000; //generate random between 0 and 5000
        printf("Random: %d\nadc_val + 1000: %d\n", random, adc_val + 1000);
        if(random < adc_val + 1000){
            printf("Sending car.\n");
            i = 1;
        }
        else{
            i = 0;
        }
        if(xQueueReceive(adc_traffic_queue, &adc_val, 500))
        {
            //printf("ADC val received: %u\n", adc_val);
            if( xQueueSend(car_postion_queue, &i, 1000)){
                //printf("car position queue: %u\n", i);
            }
            else{
                printf("car position failed\n");
            }
        }
        vTaskDelay(1000);
    }
}

static void System_Display_Task( void *pvParameters )
{
    uint16_t car_positions[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}; //len = 19
    uint16_t traffic_light_val;
```

```
    uint16_t new_car_position;
    uint16_t before_light = 8;
    uint16_t total_light = 19;
    uint16_t space_in_traffic = 0;
    while(1)
    {
        space_in_traffic = 0;
        if(xQueueReceive(traffic_light_queue, &traffic_light_val, 500))
        {
            //printf("Traffic light received: %u\n", traffic_light_val);
        }

        if(xQueueReceive(car_postion_queue, &new_car_position, 500))
        {
            car_positions[0] = new_car_position;
            for(int i = total_light - 1; i > 0; i--){

                if(traffic_light_val == green){

                    car_positions[i] = car_positions[i - 1];

                }else{
                    if(i > before_light){
                        //past traffic light
                        if(i == 9){
                            car_positions[i] = 0;
                        }else{
                            car_positions[i] = car_positions[i - 1];
                        }

                    }else if(i <= before_light){
                        //before traffic light
                        if(car_positions[i] == 0){
                            space_in_traffic = 1;
                        }
                        if(space_in_traffic){
                            car_positions[i] = car_positions[i - 1];
                        }
                    }
                }
            }
            update_traffic(total_light, car_positions);
```

```
            vTaskDelay(1000);
        }
    }
}

static void update_traffic(int n, uint16_t car_positions[]){
    uint16_t position;

    for (int i = n - 1; i > 0; i--){
        position = car_positions[i];
        if (position == 0){
            GPIO_ResetBits(shift_reg_GPIO_port, shift_reg_DATA_pin);
        }
        else{
            GPIO_SetBits(shift_reg_GPIO_port, shift_reg_DATA_pin);
        }
        GPIO_SetBits(shift_reg_GPIO_port,  shift_reg_CLOCK_pin); //clock
        GPIO_ResetBits(shift_reg_GPIO_port,  shift_reg_CLOCK_pin);
    }
}

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

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. */
    }
}
/*-----------------------------------------------------------*/

static void prvSetupHardware( void )
{
    /* Ensure all priority bits are assigned as preemption priority bits.
    http://www.freertos.org/RTOS-Cortex-M3-M4.html */
    NVIC_SetPriorityGrouping( 0 );

    /* TODO: Setup the clocks, etc. here, if they were not configured before
    main() was called. */
}
```