Université d'Ottawa
Faculté de génie

École de science informatique
et de génie électrique

uOttawa

L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical Engineering
and Computer Science

CEG4166/CSI4141/SEG4155 Real-Time System Design Winter 2024

Lab 1: Interactive Traffic Lights

Submission Date: February 2, 2024

Group Number: 5

Team Member List:

- Name 1: Moïse BALEKE, 300207962
- Name 2: Zinah Al-Saadi, 8867078
- Name 3: Decaho Gbegbe, 300094197

Table of Contents

Table of Figures

Table of Tables

Introduction

In the Interactive Traffic Lights Lab for CEG4166/CSI4141/SEG4155 Real-Time System Design, students use the freeRTOS kernel and STM32CubeIDE to create a simulated traffic light system on a nucleo-F446RE board. This lab emphasizes real-time operating system (RTOS) functionalities by requiring students to manage multiple tasks for LEDs representing traffic signals and a pedestrian request button. Through six distinct tasks, learners explore task scheduling, synchronization, and real-time responses, showcasing the crucial role of RTOS in controlling embedded systems. This practical exercise not only enhances their understanding of embedded programming but also highlights the application of RTOS in everyday technology.

Objectives

The main objectives of this lab are to:

1. Implement an interactive traffic light system using the nucleo-F446RE development board.
2. Become familiar with task creation in a real-time operating system (RTOS) environment.

Problem Analysis

The Interactive Traffic Lights Lab tasks students with simulating a traffic light system on a nucleo-F446RE board using FreeRTOS, requiring synchronization of LEDs to mimic vehicle and pedestrian signals, and a push-button to alter signal phases. This simulation emphasizes real-time operating system (RTOS) concepts such as task scheduling, inter-task communication, and synchronization. Utilizing STM32CubeIDE, students must code tasks for each LED and a button-triggered interrupt service routine (ISR) to manage light transitions. The project hinges on FreeRTOS APIs for task creation (xTaskCreate), system start (vTaskStartScheduler), and ISR-safe functions, ensuring accurate real-time behavior and system responsiveness to pedestrian requests.

Component Usage

Identify and explain the role of each component used in the lab, including the nucleo-F446RE development board, LEDs, resistors, and the push-button. Highlight the importance of each in the context of the traffic light simulation.

Algorithmic Approach

Our solution to the Interactive Traffic Lights Lab involves a structured approach to simulate traffic lights using the nucleo-F446RE development board and the FreeRTOS kernel. We aim to control each traffic light through dedicated tasks that manage the timing and state (on/off) of the LEDs representing the traffic signals. Here is an outline of our proposed plan, including the implementation strategy:

Proposed Approach:

1. **Task Creation:** We implement separate tasks for each traffic light. Each task controls the timing and state of one LED.
2. **Timing Control:** The tasks utilize delay functions (osDelay) to manage the duration each LED stays on or off, corresponding to the traffic light phases.
3. **State Management:** We use the HAL_GPIO_TogglePin function to change the state of the LEDs. This function toggles the current state of a pin, effectively turning an LED on if it was off, and vice versa.
4. **Synchronization:** To ensure the traffic lights operate in a realistic manner, the tasks are synchronized using delays that represent the traffic light cycle's timing.

Visual Aids:

To illustrate our design, consider a flowchart depicting the logic of one of the tasks:

- **Start Task → Transmit Data → Toggle LED → Delay (Phase 1) → Toggle LED → Delay (Phase 2) → Repeat**

Methods, Libraries, and Techniques:

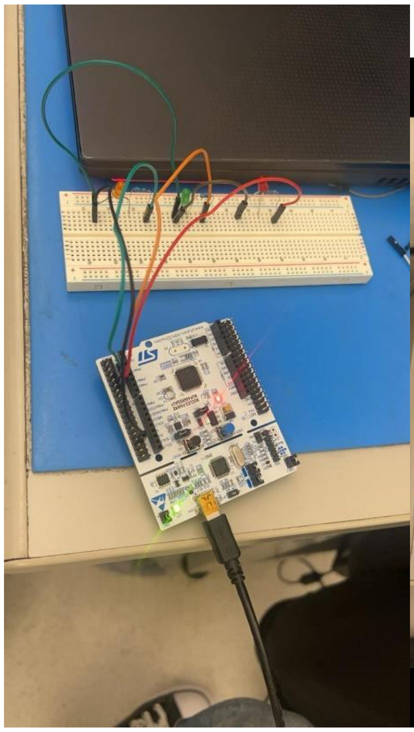
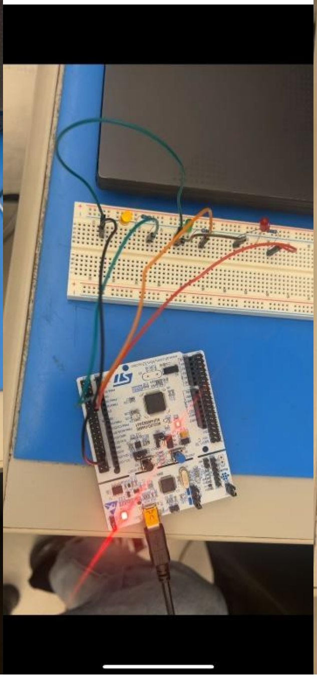
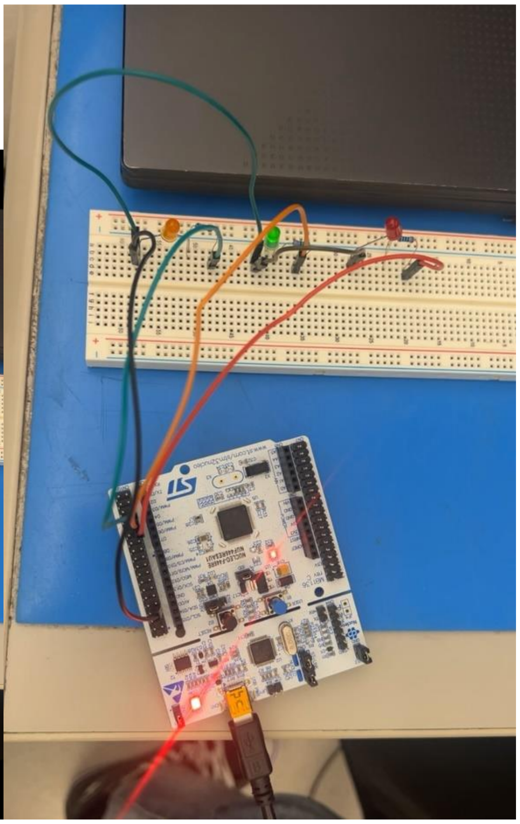
- **HAL Library:** We leverage the STM32 HAL (Hardware Abstraction Layer) library, specifically HAL_GPIO_TogglePin for LED control and HAL_UART_Transmit for debugging messages.
- **FreeRTOS APIs:** The osDelay function from FreeRTOS is used to manage task timing.
- **STM32CubeIDE:** All development is done within this integrated development environment, which provides tools for coding, debugging, and programming the nucleo-F446RE board.

By creating separate tasks for each LED and employing precise timing control, we simulate an interactive traffic light system. This approach not only demonstrates the use of real-time operating system concepts but also provides a hands-on experience with embedded system programming. Our focus on using specific methods and libraries ensures a structured and efficient implementation strategy, leveraging the capabilities of FreeRTOS and the STM32 HAL library to achieve our project objectives.

Implementation Evaluation

Assess the performance of the implemented traffic light system, focusing on the response of the LEDs to the code. Evaluate the system's efficiency, reliability, and user experience based on the code execution and behavior of the hardware components.

Results and Validation

After the launch of the code, we were able to see the changes in the state of the traffic light that we programmed. In terms of functionality, it respects the guidelines imposed by the project. In terms of reliability, the execution shows us a handmade mount that perfectly represents the model that traffic lights follow.

Addressing Challenges

During the implementation of our Interactive Traffic Lights Lab, we faced several challenges that tested our problem-solving skills and understanding of both software and hardware aspects. Two primary issues stood out: inconsistencies in code execution and difficulties with hardware setup, particularly in correctly orienting diodes on the breadboard.

Inconsistent Code Execution

The most perplexing problem was the apparent change in behavior of our code upon subsequent executions without any modifications. This inconsistency raised questions about task synchronization within the FreeRTOS environment and potential issues with the microcontroller's state.

Resolution: To address this, we adopted a systematic debugging approach. We first ensured that all global variables were correctly initialized before being used by tasks. We suspected that residual states from previous executions might affect the current run, so we incorporated explicit initialization routines to reset the system's state at the start of each task. Additionally, we reviewed and adjusted task priorities and delays to ensure a coherent and predictable execution order, mitigating the risk of tasks pre-empting each other in an unexpected manner.

Diode Orientation on the Breadboard

Another challenge was the physical setup of the LEDs on the breadboard. The correct orientation of diodes is crucial for them to function, given their unidirectional conductivity. Initially, identifying the anode and cathode leads for correct placement was a trial-and-error process, which was time-consuming and could potentially damage the components.

Resolution: To overcome this, we referred to the datasheets of the LEDs for clear identification marks of the anode and cathode. We also utilized a multimeter to test the continuity and ensure the correct orientation before powering the circuit. Educating the team on these identification techniques and the importance of careful component handling minimized the errors in setup and streamlined the development process.

These challenges underscored the importance of a thorough understanding of both the software framework and hardware components involved in embedded system projects. By addressing these issues, we not only enhanced our technical skills but also improved our problem-solving strategies, contributing to a successful project outcome.

Conclusion

In conclusion, the interactive traffic light lab, built with the FreeRTOS kernel and the STM32Cube IDE, allowed us to explore in depth the concepts of real-time operating systems (RTOS) and embedded programming. By focusing on task creation, synchronization and interrupt handling, it was possible to develop a practical understanding of the essential functionalities of RTOS in controlling embedded systems. The use of components such as the nucleo-F446RE development board, LEDs and push button allowed a realistic simulation of a traffic light system, highlighting the importance of each element in the success of the simulation.

The structured algorithmic approach, combined with the judicious use of libraries such as the HAL library and FreeRTOS APIs, facilitated the efficient implementation of the traffic light system. The results obtained demonstrated the reliability and effectiveness of the system, while providing us with an enriching learning experience.

The challenges encountered during implementation highlighted the importance of a methodical approach to problem solving, as well as the need for a thorough understanding of both the software and hardware aspects of embedded projects. By overcoming these challenges, students learned technical skills and problem-solving strategies that will prepare them to successfully meet similar challenges in their future career paths. In sum, the Interactive Traffic Light Lab was an enriching educational experience, combining theory and practice to train the engineers of the future in the exciting challenges of embedded engineering. Which will be greatly useful to us in the realization of our future laboratories and our professional career surely.

Workload Distribution

This table show the work undertaken by each team member.

Student name	Student number	Working percent
Moïse BALEKE	300207962	100%
Zinah Al-Saadi	8867078	100%
Decaho Gbegbe	300094197	100%

17. APPENDIX

- Appendix 1 Source Code
  - Code A.1.1: Task 1 Code

```c
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2024 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "cmsis_os.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
UART_HandleTypeDef huart2;

/* Definitions for Task1 */
osThreadId_t Task1Handle;
const osThreadAttr_t Task1_attributes = {
  .name = "Task1",
  .stack_size = 128 * 4,
  .priority = (osPriority_t) osPriorityNormal,
};
/* Definitions for Task2 */
osThreadId_t Task2Handle;
const osThreadAttr_t Task2_attributes = {
  .name = "Task2",
  .stack_size = 128 * 4,
  .priority = (osPriority_t) osPriorityNormal,
};
/* Definitions for Task3 */
osThreadId_t Task3Handle;
const osThreadAttr_t Task3_attributes = {
  .name = "Task3",
  .stack_size = 128 * 4,
  .priority = (osPriority_t) osPriorityNormal,
};
/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
void StartTask1(void *argument);
void StartTask2(void *argument);
void StartTask3(void *argument);

/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
uint8_t dataTask1[] = "Hi, Task1\r\n";
uint8_t dataTask2[] = "Hello, Task2\r\n";
uint8_t dataTask3[] = "Great, Task3\r\n";

/* USER CODE END 0 */
```

```c
/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Init scheduler */
  osKernelInitialize();

  /* USER CODE BEGIN RTOS_MUTEX */
  /* add mutexes, ... */
  /* USER CODE END RTOS_MUTEX */

  /* USER CODE BEGIN RTOS_SEMAPHORES */
  /* add semaphores, ... */
  /* USER CODE END RTOS_SEMAPHORES */

  /* USER CODE BEGIN RTOS_TIMERS */
  /* start timers, add new ones, ... */
  /* USER CODE END RTOS_TIMERS */

  /* USER CODE BEGIN RTOS_QUEUES */
  /* add queues, ... */
  /* USER CODE END RTOS_QUEUES */

  /* Create the thread(s) */
  /* creation of Task1 */
  Task1Handle = osThreadNew(StartTask1, NULL, &Task1_attributes);

  /* creation of Task2 */
  Task2Handle = osThreadNew(StartTask2, NULL, &Task2_attributes);

  /* creation of Task3 */
  Task3Handle = osThreadNew(StartTask3, NULL, &Task3_attributes);

  /* USER CODE BEGIN RTOS_THREADS */
  /* add threads, ... */
  /* USER CODE END RTOS_THREADS */

  /* USER CODE BEGIN RTOS_EVENTS */
  /* add events, ... */
  /* USER CODE END RTOS_EVENTS */

  /* Start scheduler */
  osKernelStart();

  /* We should never get here as control is now taken by the scheduler */
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
            //HAL_UART_Transmit(&huart2, dataTask1, 7, 1000);
            //HAL_Delay(1000);


  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /** Initializes the RCC Oscillators according to the specified parameters
```

```c
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 180;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 2;
  RCC_OscInitStruct.PLL.PLLR = 2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Activate the Over-Drive mode
  */
  if (HAL_PWREx_EnableOverDrive() != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5, GPIO_PIN_RESET);

  /*Configure GPIO pins : PB3 PB4 PB5 */
  GPIO_InitStruct.Pin = GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */
int phase1 = 10000;
```

```c
int phase2 = 2000;
int phase3 = 10000;
int phase4 = 3000;
/* USER CODE END 4 */

/* USER CODE BEGIN Header_StartTask1 */
/**
  * @brief  Function implementing the Task1 thread.
  * @param  argument: Not used
  * @retval None
  */
/* USER CODE END Header_StartTask1 */
void StartTask1(void *argument)
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {
            HAL_UART_Transmit(&huart2, dataTask1, sizeof(dataTask1), 1000);
           // HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5, GPIO_PIN_RESET);
            HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_5);
    osDelay(phase1);
    osDelay(phase2);
            HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_5);
            osDelay(phase3);
            osDelay(phase4);
  }
  /* USER CODE END 5 */
}

/* USER CODE BEGIN Header_StartTask2 */
/**
* @brief Function implementing the Task2 thread.
* @param argument: Not used
* @retval None
*/
/* USER CODE END Header_StartTask2 */
void StartTask2(void *argument)
{
  /* USER CODE BEGIN StartTask2 */
  /* Infinite loop */
  for(;;)
  {
            HAL_UART_Transmit(&huart2, dataTask2, sizeof(dataTask2), 1000);
            osDelay(phase1);
            HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_4);
            osDelay(phase2);
            HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_4);
            osDelay(phase3);
            HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_4);
            osDelay(phase4);
            HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_4);
  }
  /* USER CODE END StartTask2 */
}

/* USER CODE BEGIN Header_StartTask3 */
/**
* @brief Function implementing the Task3 thread.
* @param argument: Not used
* @retval None
*/
/* USER CODE END Header_StartTask3 */
void StartTask3(void *argument)
{
  /* USER CODE BEGIN StartTask3 */
  /* Infinite loop */
  for(;;)
  {
            HAL_UART_Transmit(&huart2, dataTask3, sizeof(dataTask3), 1000);
            osDelay(phase1);
            osDelay(phase2);
            HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_3);
            osDelay(phase3);
            HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_3);
            osDelay(phase2);
  }
  /* USER CODE END StartTask3 */
}

/**
  * @brief  Period elapsed callback in non blocking mode
  * @note   This function is called  when TIM6 interrupt took place, inside
  * HAL_TIM_IRQHandler(). It makes a direct call to HAL_IncTick() to increment
  * a global variable "uwTick" used as application time base.
  * @param  htim : TIM handle
  * @retval None
  */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  /* USER CODE BEGIN Callback 0 */

  /* USER CODE END Callback 0 */
  if (htim->Instance == TIM6) {
    HAL_IncTick();
  }
  /* USER CODE BEGIN Callback 1 */

  /* USER CODE END Callback 1 */
}

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

- Code A.1.2: Task 2 Code

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2024 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "cmsis_os.h"

```

```c
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
UART_HandleTypeDef huart2;

/* Definitions for Task1 */
osThreadId_t Task1Handle;
const osThreadAttr_t Task1_attributes = {
  .name = "Task1",
  .stack_size = 128 * 4,
  .priority = (osPriority_t) osPriorityNormal,
};
/* Definitions for Task2 */
osThreadId_t Task2Handle;
const osThreadAttr_t Task2_attributes = {
  .name = "Task2",
  .stack_size = 128 * 4,
  .priority = (osPriority_t) osPriorityNormal,
};
/* Definitions for Task3 */
osThreadId_t Task3Handle;
const osThreadAttr_t Task3_attributes = {
  .name = "Task3",
  .stack_size = 128 * 4,
  .priority = (osPriority_t) osPriorityNormal,
};
/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
```

```c
void StartTask1(void *argument);
void StartTask2(void *argument);
void StartTask3(void *argument);

/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
uint8_t dataTask1[] = "Hi, Task1\r\n";
uint8_t dataTask2[] = "Hello, Task2\r\n";
uint8_t dataTask3[] = "Great, Task3\r\n";

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Init scheduler */
  osKernelInitialize();

  /* USER CODE BEGIN RTOS_MUTEX */
  /* add mutexes, ... */
  /* USER CODE END RTOS_MUTEX */

  /* USER CODE BEGIN RTOS_SEMAPHORES */
  /* add semaphores, ... */
  /* USER CODE END RTOS_SEMAPHORES */

  /* USER CODE BEGIN RTOS_TIMERS */
  /* start timers, add new ones, ... */
  /* USER CODE END RTOS_TIMERS */

  /* USER CODE BEGIN RTOS_QUEUES */
  /* add queues, ... */
  /* USER CODE END RTOS_QUEUES */


  //this is my shit
  //make sure to turn off all diodes before lunching the code
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_RESET); // Turn off Red LED
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_RESET); // Turn off Yellow/Orange LED
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET); // Turn off Green LED


  /* Create the thread(s) */
  /* creation of Task1 */
  Task1Handle = osThreadNew(StartTask1, NULL, &Task1_attributes);

  /* creation of Task2 */
  Task2Handle = osThreadNew(StartTask2, NULL, &Task2_attributes);

  /* creation of Task3 */
  Task3Handle = osThreadNew(StartTask3, NULL, &Task3_attributes);

  /* USER CODE BEGIN RTOS_THREADS */
  /* add threads, ... */
  /* USER CODE END RTOS_THREADS */

  /* USER CODE BEGIN RTOS_EVENTS */
  /* add events, ... */
  /* USER CODE END RTOS_EVENTS */

  /* Start scheduler */
  osKernelStart();

  /* We should never get here as control is now taken by the scheduler */
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
            //HAL_UART_Transmit(&huart2, dataTask1, 7, 1000);
            //HAL_Delay(1000);


  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 180;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 2;
  RCC_OscInitStruct.PLL.PLLR = 2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Activate the Over-Drive mode
  */
  if (HAL_PWREx_EnableOverDrive() != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
```

```
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5, GPIO_PIN_RESET);

  /*Configure GPIO pins : PB3 PB4 PB5 */
  GPIO_InitStruct.Pin = GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/* USER CODE BEGIN Header_StartTask1 */
/**
  * @brief  Function implementing the Task1 thread.
  * @param  argument: Not used
  * @retval None
  */
/* USER CODE END Header_StartTask1 */
//control cars green light
void StartTask1(void *argument)
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {
    HAL_UART_Transmit(&huart2, dataTask1, sizeof(dataTask1), 1000);
    // HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5, GPIO_PIN_RESET);
    /*HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_5);
    osDelay(1000);*/

    HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_3);

    if(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13) == GPIO_PIN_RESET) {
      // Button is pressed
      HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_3);
      osDelay(15000);
      HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_3);
```

```
          }


    }
    /* USER CODE END 5 */
  }

  /* USER CODE BEGIN Header_StartTask2 */
  /**
  * @brief Function implementing the Task2 thread.
  * @param argument: Not used
  * @retval None
  */
  /* USER CODE END Header_StartTask2 */
  //control pedestrian red light
  void StartTask2(void *argument)
  {
    /* USER CODE BEGIN StartTask2 */
    /* Infinite loop */
    for(;;)
    {

            HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_4);

            if(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13) == GPIO_PIN_RESET) {
      // Button is pressed
      HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_15);
      osDelay(13000);
      HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_15);
    }
    }
    /* USER CODE END StartTask2 */
  }

  /* USER CODE BEGIN Header_StartTask3 */
  /**
  * @brief Function implementing the Task3 thread.
  * @param argument: Not used
  * @retval None
  */
  /* USER CODE END Header_StartTask3 */
  //control cars orange light
  void StartTask3(void *argument)
  {
    /* USER CODE BEGIN StartTask3 */
    /* Infinite loop */
    for(;;)
    {
            if(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13) == GPIO_PIN_RESET) { // Assuming active low
      // Button is pressed
      HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_4); //on
    osDelay(3000);
    HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_4); //off
    osDelay(10000);
    HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_4); //on
    osDelay(2000);
    HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_4); //off

    }
  }
  /* USER CODE END StartTask3 */
}

//control cars red light
void StartTask4(void *argument)
{
  /* USER CODE BEGIN StartTask3 */
  /* Infinite loop */
  for(;;)
  {
        HAL_UART_Transmit(&huart2, dataTask3, sizeof(dataTask3), 1000);
        /*HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_3);
        osDelay(1000);*/
        if(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13) == GPIO_PIN_RESET) { // Assuming active low
            // Button is pressed
            osDelay(3000);
            HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_5); //on
            osDelay(12000);
            HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_5); //off
        }
  }
  /* USER CODE END StartTask3 */
}


//control pedestrians green light
void StartTask5(void *argument)
{
  /* USER CODE BEGIN StartTask3 */
  /* Infinite loop */
  for(;;)
  {
            HAL_UART_Transmit(&huart2, dataTask3, sizeof(dataTask3), 1000);
            /*HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_3);
            osDelay(1000);*/
      if(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13) == GPIO_PIN_RESET) { // Assuming active low
            // Button is pressed
            osDelay(3000);
            HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_14); //on
            osDelay(10000);
            HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_14); //off
        }
  }
```

```
    /* USER CODE END StartTask3 */
  }

  /**
    * @brief  Period elapsed callback in non blocking mode
    * @note   This function is called  when TIM6 interrupt took place, inside
    * HAL_TIM_IRQHandler(). It makes a direct call to HAL_IncTick() to increment
    * a global variable "uwTick" used as application time base.
    * @param  htim : TIM handle
    * @retval None
    */
  void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
  {
    /* USER CODE BEGIN Callback 0 */

    /* USER CODE END Callback 0 */
    if (htim->Instance == TIM6) {
      HAL_IncTick();
    }
    /* USER CODE BEGIN Callback 1 */

    /* USER CODE END Callback 1 */
  }

  /**
    * @brief  This function is executed in case of error occurrence.
    * @retval None
    */
  void Error_Handler(void)
  {
    /* USER CODE BEGIN Error_Handler_Debug */
    /* User can add his own implementation to report the HAL error return state */
    __disable_irq();
    while (1)
    {
    }
    /* USER CODE END Error_Handler_Debug */
  }

  #ifdef  USE_FULL_ASSERT
  /**
    * @brief  Reports the name of the source file and the source line number
    *         where the assert_param error has occurred.
    * @param  file: pointer to the source file name
    * @param  line: assert_param error line source number
    * @retval None
    */
  void assert_failed(uint8_t *file, uint32_t line)
  {
    /* USER CODE BEGIN 6 */
    /* User can add his own implementation to report the file name and line number,
       ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
```

```
-       /* USER CODE END 6 */
-     }
-     #endif /* USE_FULL_ASSERT */
```

- Here is the OneDrive link to the actual code folder: CEG4166Lab1InteractiveTrafficLightsGroupNumber5.docx