California State University, Fresno
Lyles College of Engineering
Electrical and Computer Engineering Department

TECHNICAL REPORT

Experiment Title: Multitasking

Instructor: Dr. Nan Wang

Course Title: Ece 144

Date Submitted: 12/11/2020

Prepared By:

Nirmala Sinha

INSTRUCTOR SECTION

Comments: ____

____

____

____

Final Grade: Team Member 1: ____________________

Team Member 2: ____________________

Table of Contents

Section **Page #**

List of Figures

1. Statement Of Objective:

The objective of this assignment is to build a project which uses FreeRTOS of Nucleo-L32KC. In this project students will be learning about FreeRTOS The project will be divided into three segments. Through this project students will learn to create multiple tasks that will be sharing the same resources. To ensure while one task is running and using the resource it will be locked with mutex. In this project students will learn to implement the concepts that were taught in the beginning of class.

2. Background Information:

Nucleo L432K development board is easy to program a low-cost development board. Students can build different projects using different prototypes, commands, and functions. This board has three LED ports where LED1 shows the connection and loading of the program, LED2 is for the power input and LED3 is the user used LED. It also has other ports such as SB for other ports (soldier bridge), processor,UART connections, etc. A Nucleo-L432KC uses a processor ARM Cortex M-4, which is a 32-bit processor.

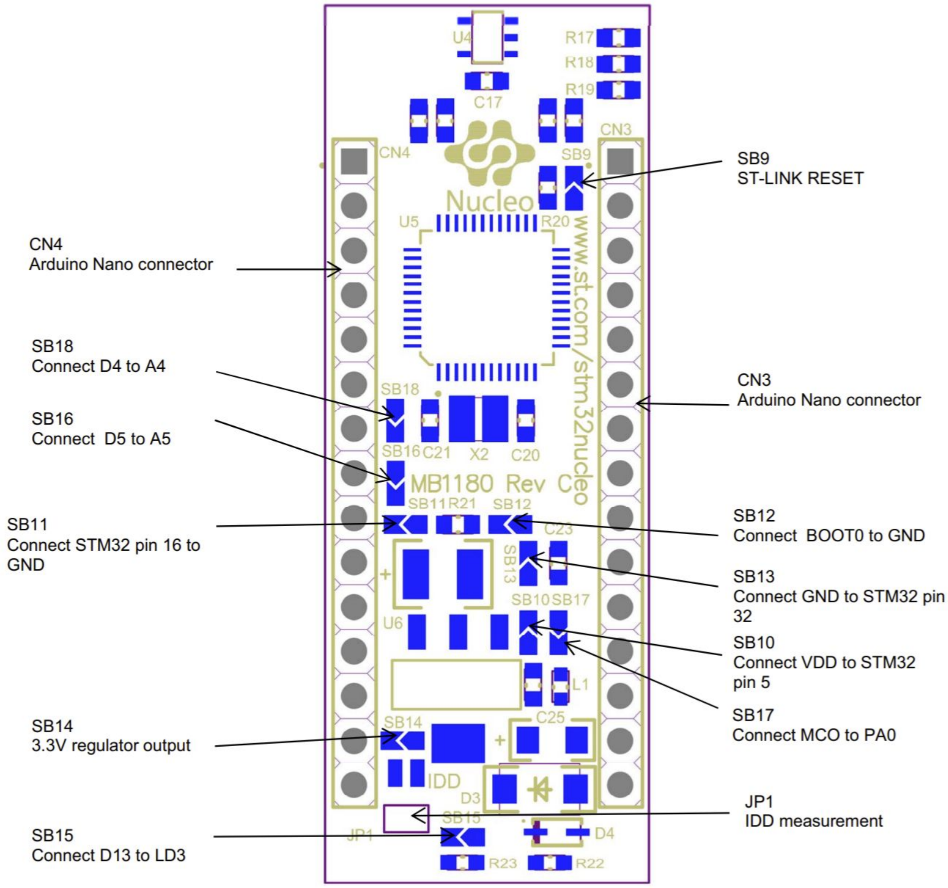

Figure1: STM32 board layout

This board also has many different peripherals. These peripherals can be used for different purposes. This board also includes General purpose Input/Output pins which allows to push or pull the data from the board or allows to input data to the board.

This board also has a feature called FreeRTOS which students can use to implement the project. FreeRTOS belongs to one of the classes of the RTOS. It allows the system to perform different functions, tasks, threads, memory allocations, schedulting, etc, in real time. The FreeRTOS also helps to split the program functions into independent tasks and then are executed when the user asks for it. This board is able to execute many inbuilt functions such as performing multitasks, threads, scheduling, etc. A task in an operating system means a job which is provided to the CPU to execute. This task may include a use of resources such as uart, gpios, LED’s, etc. An operating system also supports multitasking that allows the user to perform tasks of different priorities or same priorities to run and output the result.

In multitasking, main memory is loaded with some number of tasks. Once all the tasks are loaded into the main memory, the CPU executes each and every task with some quantized time as shown in the Figure 2 below. The CPU executes the tasks according to their priority.

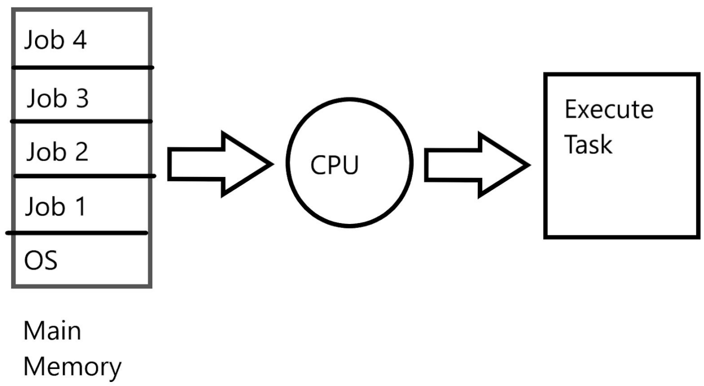

Figure 2: Working of Tasks

If a task has higher priority than the other tasks then that task will be executed first. Sometimes tasks with the same priority and sharing resources creates a deadlock.

3. Experimental Procedure:

To start working on the project, students need to perform the necessary steps. First, the students have to download and install STMcubeMX,Tera term and KEIL. Once that is done, then students can carry out the following process.

3.1 Equipment Used:

- STMcubeMX
- Keil
- Tera Term
- Mini USB
- LED’s
- Wire
- Oscilloscope
- Resistor

- STM32 board
- Word doc/ Google Doc

3.2 Experimental Procedure:

a. STMcubeMX

After downloading and installing the STMcubeMX, students need to click on the Files->New project. This will open a new dialogue box in which students have to click on the board selector, which is present on the top bar, then select the board that students will use, for example, as shown in Figure 3 below.

In the part number section, students need to type and select NUCLEO-L432KC. On the right side, students would be able to see the board appearing.

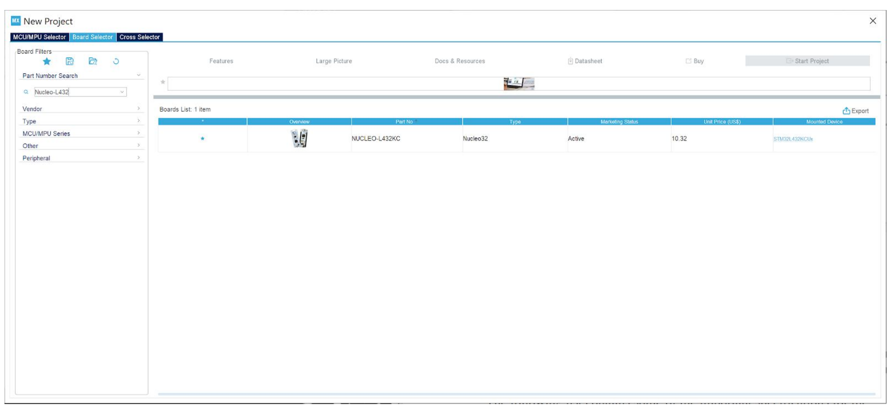

Figure 3: Part number search box

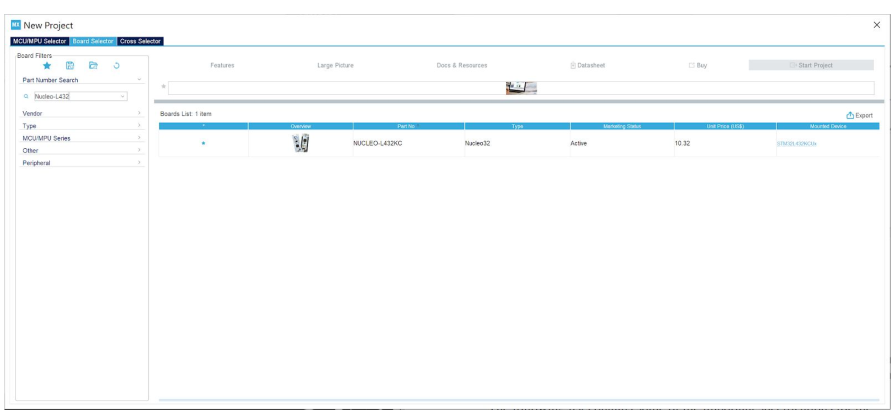

Figure 4: Select the board and click get started with the board

Click on the board and then click on start a new project. It will open a new screen, as shown in the figure 5 below.

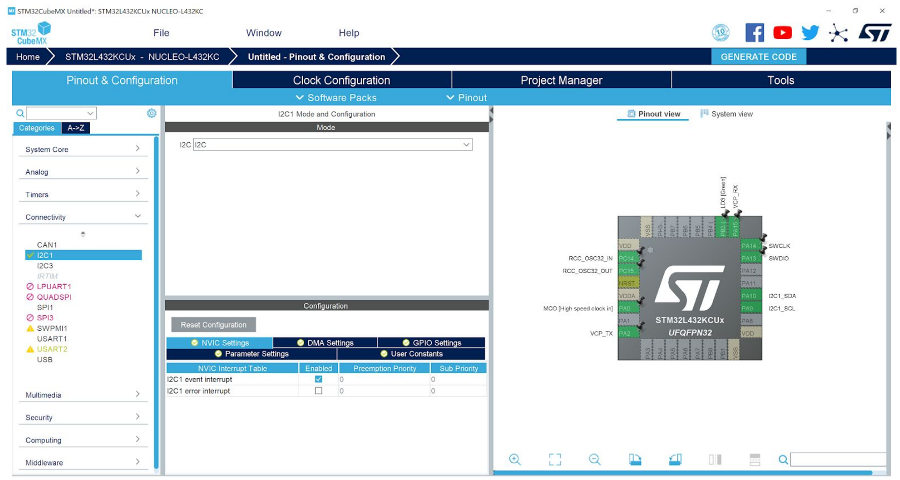

Figure 5: Main screen to set pins.

As shown in Figure 5, students can set any configuration first. For this project, as shown in Figure 5, set the I2C1 interface. The I2C is also known as a synchronous serial data link. I2C is a serial protocol, which contains a master and a slave. In this I2C1 mode and configuration, students have to go into the NVIC settings and enable the I2C event to interrupt.

In the next figure, Figure 6, students have to set SPI1 which stands for Serial Peripheral Interface. Then first, set the mode, Full-Duplex Master. The second step for setting SPI1 is to set up Hardware NSS Signal to disable. Set the parameters for the SPI1.

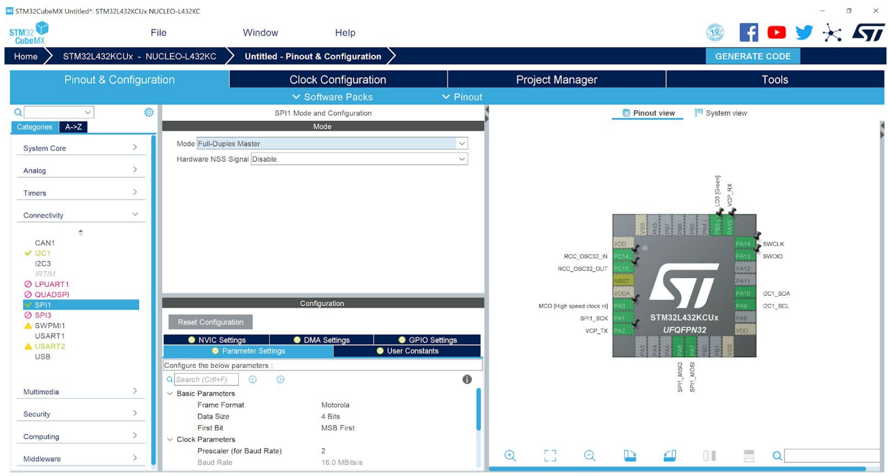

Figure 6: Setting SPI1

After this, students need to set USART. This can be done by setting up the mode to asynchronous. Students need to enable the USART1 global interrupt. This can be seen in Figure 7 shown below.

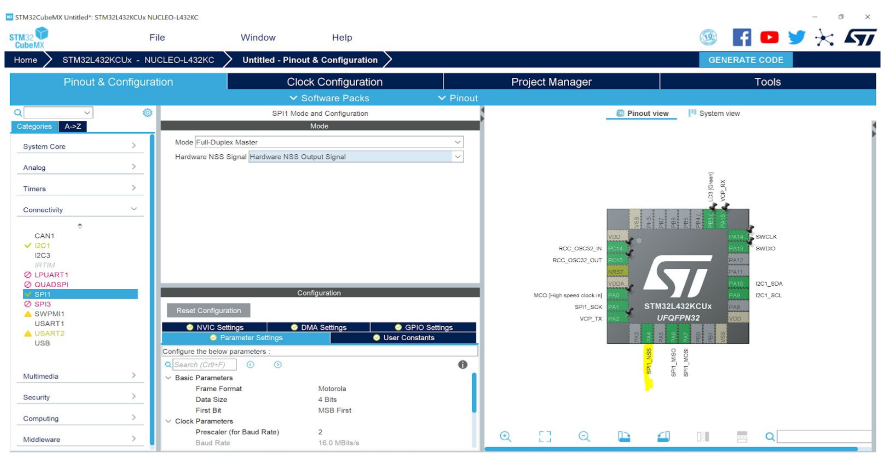

Figure 7: Setting SP2

STMCubeMX also allows the user to select any pin on the board and enable it. After setting up the USART, students need to set up the FreeRTOS, which is under Mindware. Config parameter needs to be set, as shown in Figure 8 below. Make sure to set the interface CMSIS_V1 or the one which is available to the system.

Figure 8: Settings for FreeRTOS

Students also have to add semaphores and mutexes to the system, as shown in Figure 9 below.

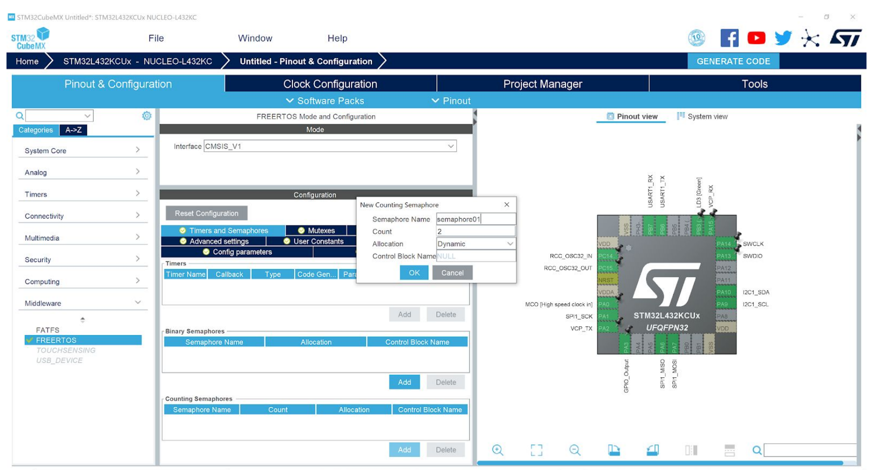

Figure 9: Adding a semaphore

After adding semaphores and mutexes. Students need to add different tasks with the same or different priorities. This can be seen in Figure 10 shown below. It is important to note that students can add the tasks here or they can add while they are coding in Keil. One of the specialities of the system is that if students miss to add any extra task, mutex, thread or semaphore they can always write in Keil and add them there.

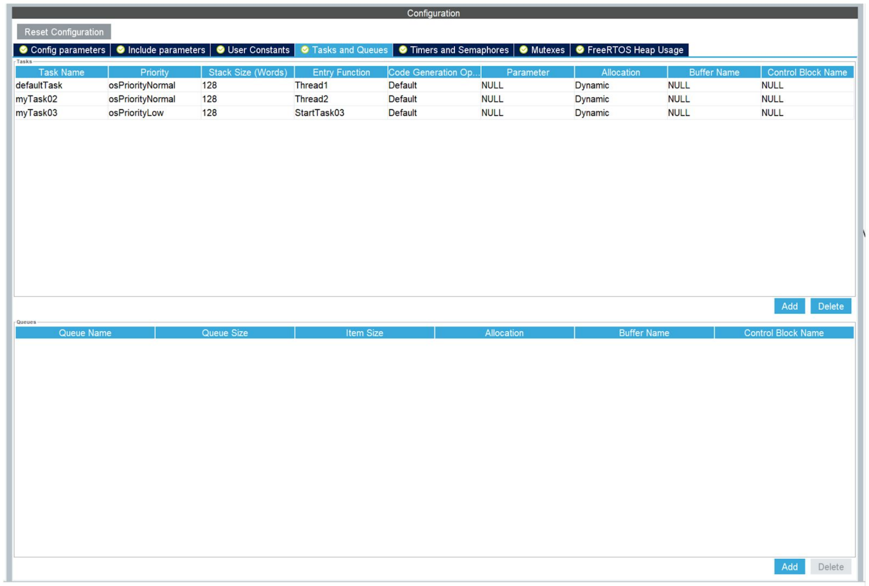

Figure 10: Setting up tasks

Then by clicking on the clock configuration change it from HSI to HSE. set it up to maximum as shown in Figure 11.

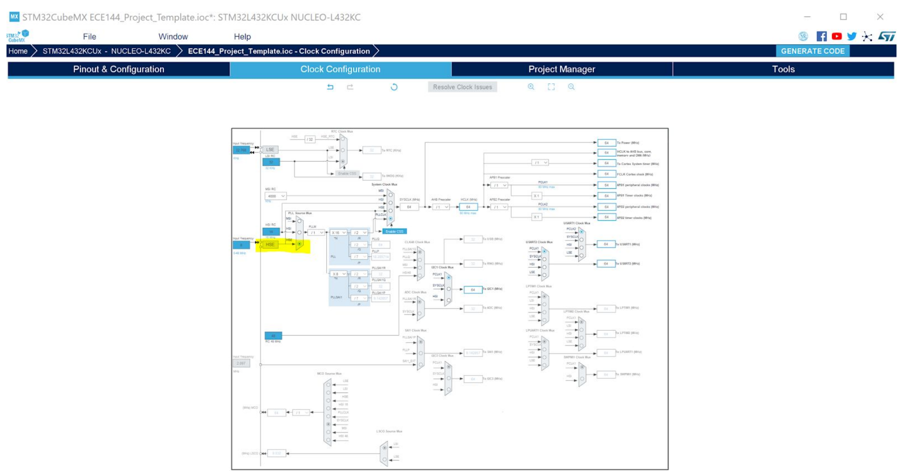

Figure 11: Setting up the clock configuration.

Once all the settings are done, the students can click on the generate code and save them to an accessible folder.

b. Keil

For keil students doesn't have to make much changes. If it is the first time a user is using it, keil might need to download and install some files. Once when the code is generated it will directly open the files in the keil.

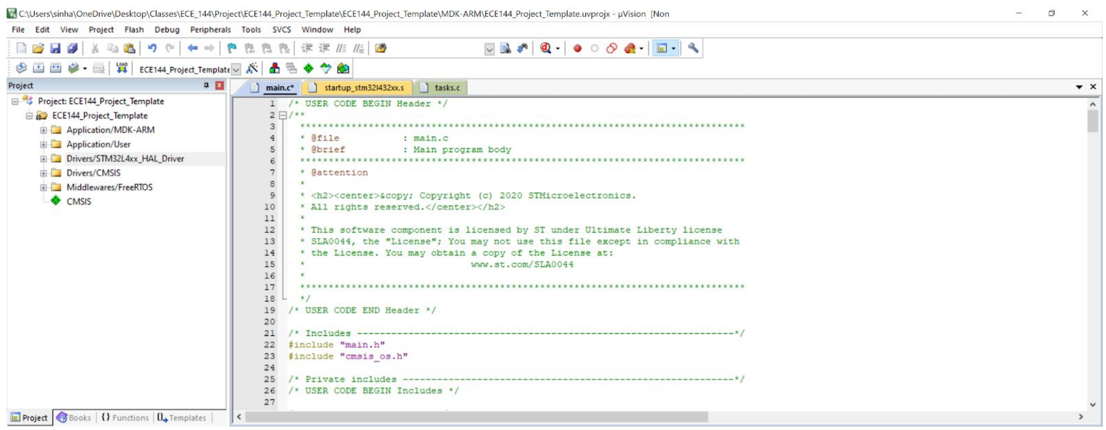

Figure 12: Keil homepage

c. Tera Term

Download and Install the Tera term. Once it is downloaded, and installed properly, students need to select Serial as shown in Figure 10 and select the port which includes ST-Link. Once that is done, using the code which is present in the Keil for UART set the setting for the serial port. So, students have to change the speed of UART to the speed shown in Keil. The steps for setting the system are shown below in Figure 11.

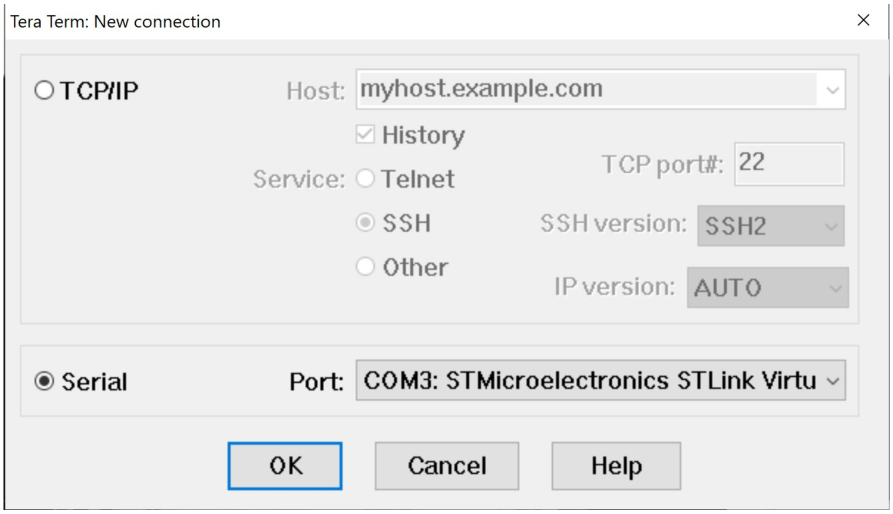

Figure 13: Tera term

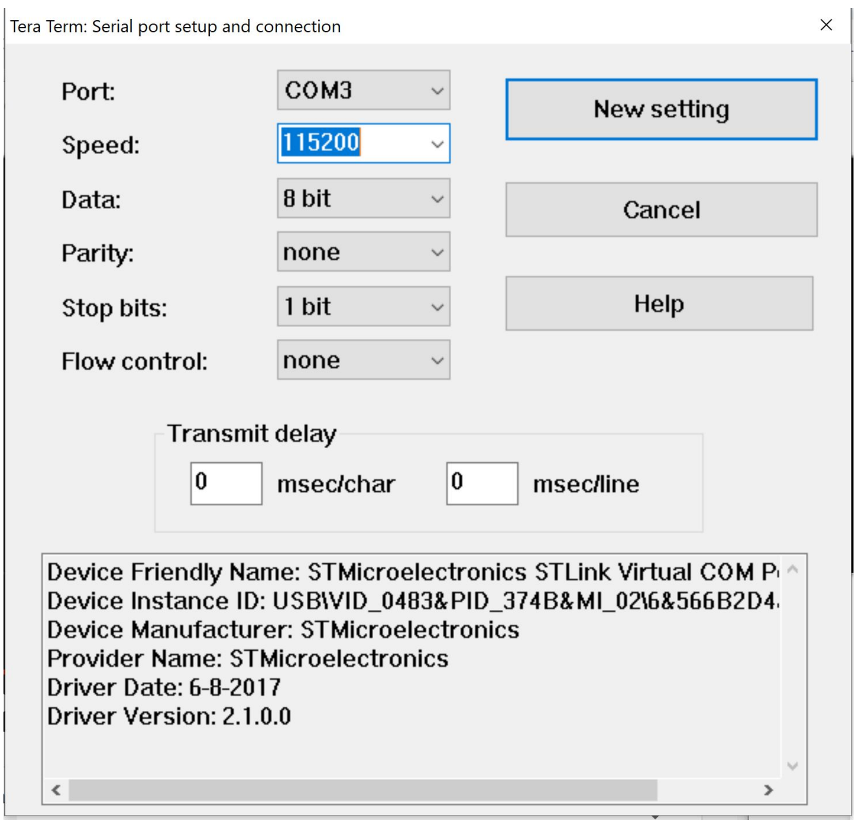

Figure 14: Setting serial port for Tera term

Make sure to save these settings in the same folder and under the files that . This will allow the user to restore the settings and doesn't have to be set up everytime it uses the tera term.

4. Data Analysis

a. Results :

i. Part 1- Creating multiple Tasks & mutex

After setting up the system, in the keil, first include the header files which will be necessary for executing the program. The file that needs to be included is shown in the Figure 15 below. These header files will allow the interface parts to work properly in the operating system

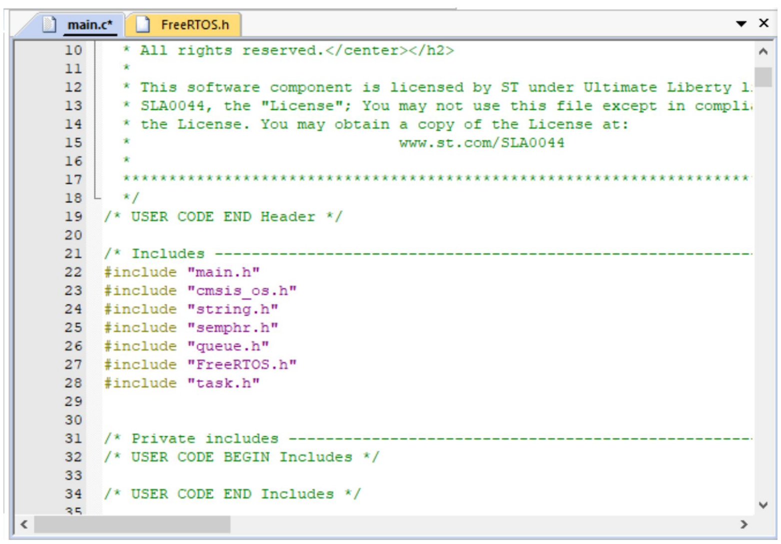

Figure 15: Header files

Once all the header files are included then the students can remove all the default task related functions as the main aim of this project is to build multitask functions. Then students can start off by first making a handler for mutex. For this project the defined mutex handler is called SimpleMutex. Then create task handlers as well. For this project, five tasks will be executed, so five handlers are made as shown in the Figure 16 below.

```
main.c*   FreeRTOS.h


/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------
/* USER CODE BEGIN 0 */
SemaphoreHandle_t SimpleMutex;    // define the semaphore handler

TaskHandle_t First_Task_Handler;           // defined three tasks
TaskHandle_t MiddlePT_Handler;           //  using task Handler
TaskHandle_t LessPT_Handler;           //
TaskHandle_t Fourth_Handler;
TaskHandle_t Fifth_Handler;



void First_PT (void *argument);
void MiddlePT_Task (void *argument);
void LessPT_Task (void *argument);
void Fourth_Task (void *argument);
void Fifth_Task (void *argument);


void Send_Uart (char *str)
```

Figure 16: Creating the handlers

The next step is to build functions for these. So that the user can type code in those functions. Then create the mutex as shown in Figure 17.

```

void Send_Uart (char *str)
{
  xSemaphoreTake(SimpleMutex, portMAX_DELAY);
  HAL_UART_Transmit(&huart2,str, strlen(str), HAL_MAX_DELAY);
  HAL_Delay(2000);
  xSemaphoreGive(SimpleMutex);
}

```

Figure 17: Create mutex function

Using xsemaphore type function create a mutex function. Once that is made students need to check if the mutex is created or not. This can be checked using various methods. For this project, an if loop is created to check if the mutex is created or not. If it is created it should return any value other than Null. In the code the UART will transmit a command to

UART2 when mutex is created. Stating to print a sting. This can be seen in the Figure 18 below.

```
if (SimpleMutex != NULL)
{
  HAL_UART_Transmit(&huart2,myTxData,15 ,1000);

}
```

Figure 18: Check for mutex

Now tasks will be created. For this students need to use the command as shown below in Figure 19. Using xtasksan inbuilt function will be used and the following parameters will be passed to it. The first parameter that is passed is the task code. Task code is basically the function that was created for the task. In the second parameter the name of the task is entered. In the third parameter the stack size is passed. The fourth and fifth parameters deal with the priority of the task. For this project the priority for the first task is higher and the fourth task is lower priority.

Once these are set the scheduler understands which task needs to be run first and which one needs to be after it and so on. In the sixth parameter students need to mention the reference. This will be done for all the tasks. After this write a command for starting the scheduler.

```c
/* We should never get here as control is now taken by the scheduler */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
SimpleMutex = xSemaphoreCreateMutex();

if (SimpleMutex != NULL)
{
  HAL_UART_Transmit(&huart2,myTxData,15 ,1000);

}
// Creating tasks

xTaskCreate(First_PT,"First_Task",128,NULL,3,&First_Task_Handler);
xTaskCreate(MiddlePT_Task,"MiddlePT",128,NULL,2,&MiddlePT_Handler);
xTaskCreate(LessPT_Task,"LessPT",128,NULL,1,&LessPT_Handler);
xTaskCreate (Fourth_Task,"Fourth_task",128,NULL,0,&First_Task_Handler);

vTaskStartScheduler();



while (1)
{
  /* USER CODE END WHILE */
```

Figure 19: Creating tasks

After this step the next step is to create a function which will allow us to use the critical section of our function. So, before any operation is performed, mutex uses the xsemaphore function. In this function we mutex is first taken. This will ensure once the task is in here no other task can take the resource until the task has performed its activity. Later the task releases the mutex; This means now the resource is free and other tasks can use it. In this function we also pass the time delay which is for time when mutex is not available. The following Figure 20 shows this

```c
void Fourth_Task (void *argument);
void Fifth_Task (void *argument);


void Send_Uart (char *str)
{
  xSemaphoreTake(SimpleMutex, portMAX_DELAY);
  HAL_UART_Transmit(&huart2,str, strlen(str), HAL_MAX_DELAY);
  HAL_Delay(2000);
  xSemaphoreGive(SimpleMutex);
}

/* USER CODE END 0 */

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

  /* USER CODE END 1 */


```

Figure 20: Using xsemaphore function

The acquiring of mutex, will transmit the string to UART. Then the mutex will be released. To understand the concepts add a delay of 2 second to observe and gain knowledge how mutex is blocking other tasks to use the UART.

Once all the tasks have been created the next step to write the task related code. In this code, strings will be printed once the control enters the tasks. In this the UART will transmit commands to uart2 by passing the first parameter, in the second parameter the string needs to be passed. In the third parameter the length of string will be passed and in the last parameter the user needs to pass how much delay should be there. After this call the function that was created above in the figure 20 and pass the string to it. Later when the control comes back to this function it will need to print the task is complete and is ended. The last step will be to add a delay to the task. This can be seen in the Figure 21 below. Do the same for multiple tasks. Follow the steps mentioned above and shown in Figure 21.

```
void First_PT (void *argument)
{ char *strtosend = "In First_Task=====================\r\n";
  while (1)
  {
    char *str = "Enter First_Task and about to enter in to mutex \r\n";
    HAL_UART_Transmit(&huart2, str, strlen(str),HAL_MAX_DELAY);

    Send_Uart(strtosend);

    char *str2 = "Leaving First_Task=====================\r\n ";
    HAL_UART_Transmit(&huart2, str2, strlen(str2),HAL_MAX_DELAY);
    vTaskDelay(1500);
  }
}

```

Figure 21: Task function

Click on the build button on the top left of the program. Once it is built click on the debug mode and load the program to the board.

The results for this part will be shared and explained in the results and discussion shown below.

ii. Part 2- Changing the priorities

Once the results are obtained for the part1. To make the project a little interesting, students can change the priority of the tasks. So, as the code shown in figure 19 above. Students need to modify the code. Where the priority of the first task is given the highest priority. It can be changed. Same is for the other task the priority of each task can be changed and controls will be passed according to the tasks. This can be seen in the Figure 22 below.

```
// priority changed

xTaskCreate(First_PT,"First_Task",128,NULL,2,&First_Task_Handler); // priority changed from 3 to 2
xTaskCreate(MiddlePT_Task,"MiddlePT",128,NULL,1,&MiddlePT_Handler); // priority changed from 2 to 1
xTaskCreate(LessPT_Task,"LessPT",128,NULL,0,&LessPT_Handler); // priority changed from 1 to 0
xTaskCreate (Fourth_Task,"Fourth_task",128,NULL,3,&First_Task_Handler); // priority changed from 0 to 3

vTaskStartScheduler();
```

Figure 22: Priority Changed

iii. Results and Discussions

iv. Justification

1. **Results of Part 1- Creating multiple Tasks & mutex**

   To understand the concepts in depth it is important to put breakpoints and run the code in segments. So that one easily understands and sees what is actually happening. This will also allow us to understand how the control is being passed from one task to another. Once the code starts to run. It will first show if the mutex is created or not. Then control will be given to the higher task. It will start to execute its function and print out the string. The results can be seen Figure 23.

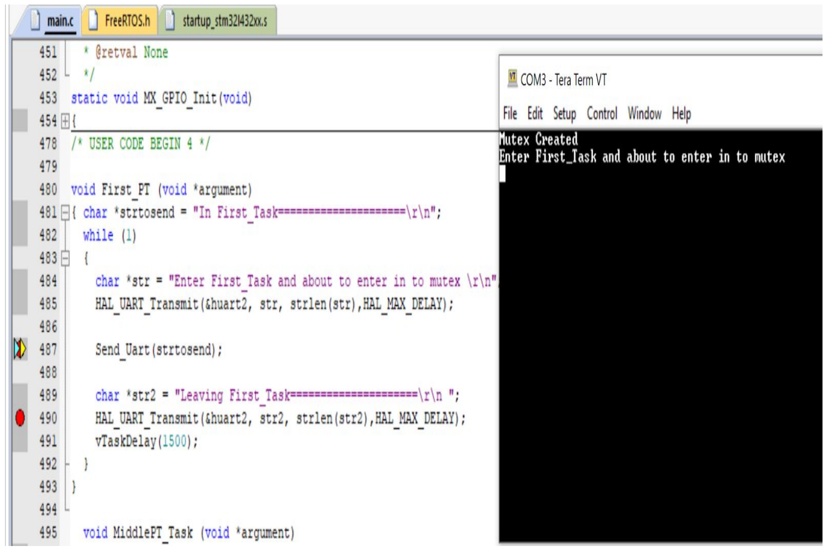

Figure 23: First run using debug mode

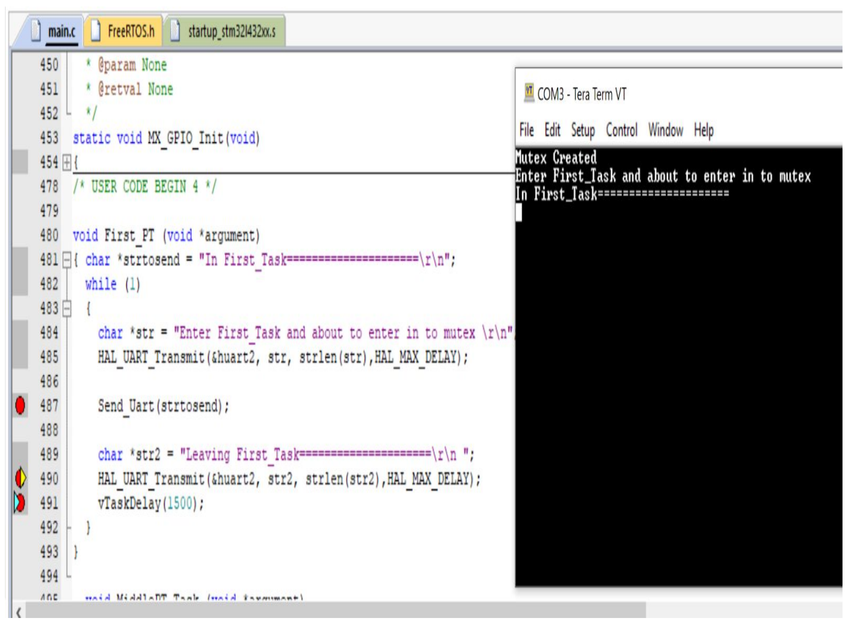

Figure 24: Second run using debug mode

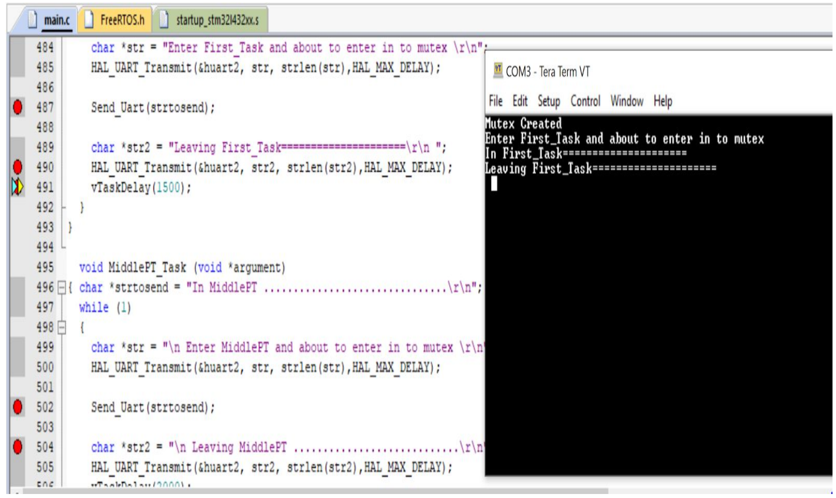

Figure 25: Third run using debug mode

Then before going into the second run. The program will first call the function sendUART and the string will print again as shown in Figure 24. This means the first task has acquired the mutex, printed the string and then the mutex was released. Then the higher priority task goes into suspension mode for a delay of 1500 milliseconds. In the available time the system gives control to middle tasks. This allows the system to print the entry string of the middle task. Again the same procedure will follow and then the send UART function is called shown in Figure 26 and 27 below.

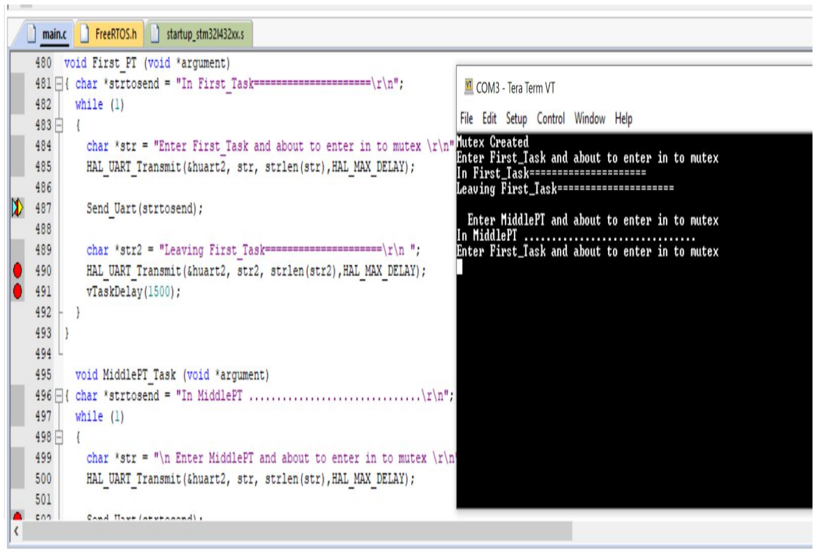

Figure 26: First task preempts middle task

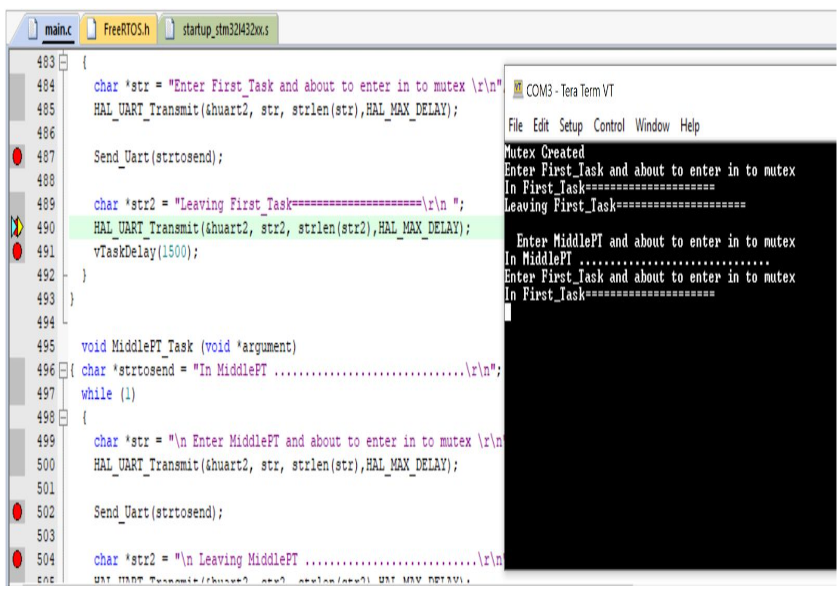

Figure 27: Mutex occupied by first task

The delay that was entered will help to complete the task and release the mutex. By this time the first task will wake up and preempt the middle task. It will again print the first task string and will try to take over the mutex. But the important point is that the middle task still holds the mutex. As a result the first task now has to wait for the middle task to release the mutex. Once it releases the mutex the control will be given to the first task and the strings will be printed again and the mutex will be released.

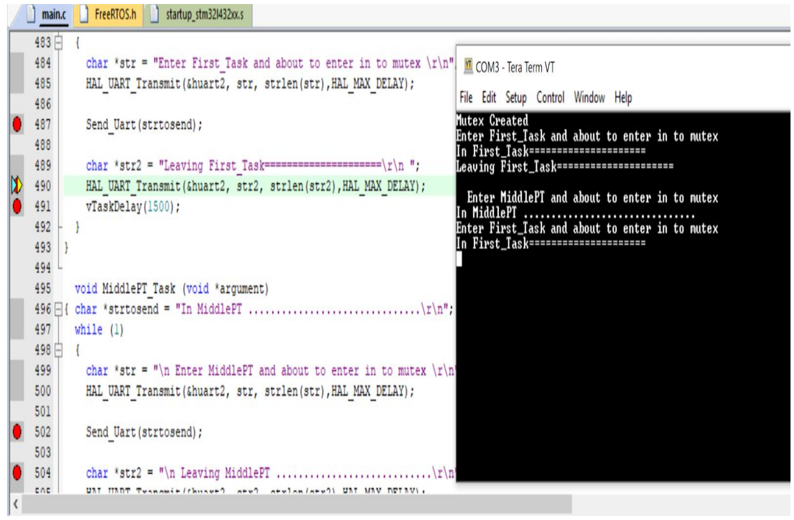

Figure 28: First task print strings

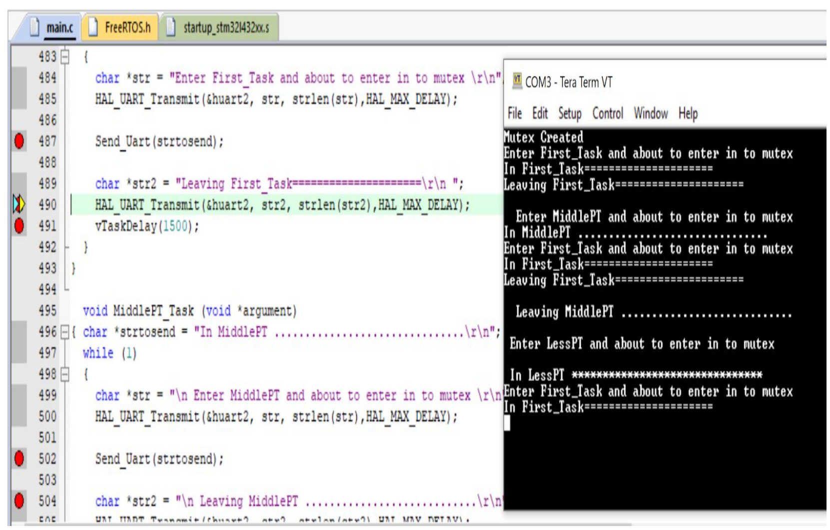

Figure 29: Third task preempted

The control will be given back to the middle and it will exit as well so the results are shown in Figure 30 below.

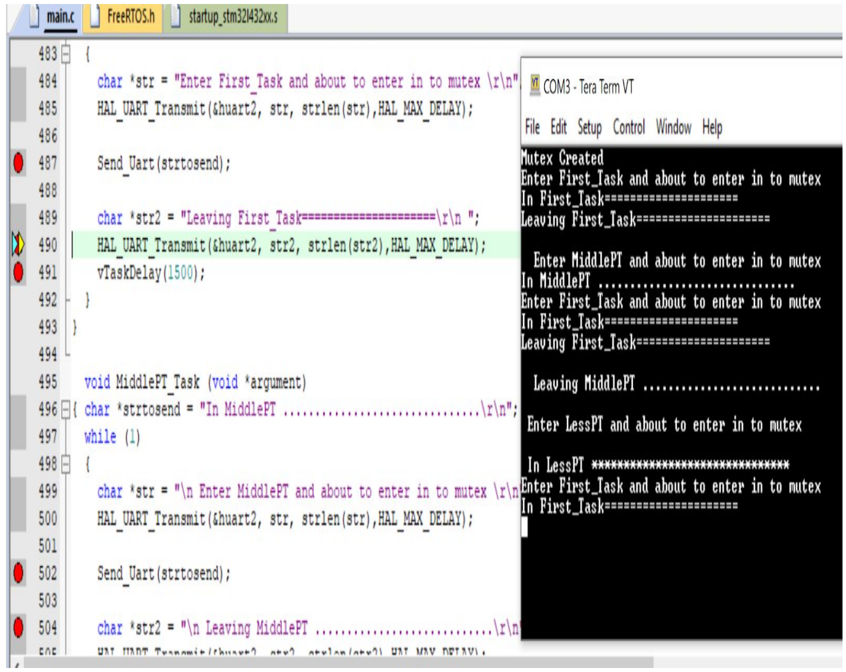

Figure 30: First and middle task exit

Now the next task will be given the control and the cycle will carry to work in the same way if students let it run freely. The result of this part is shown in Figure 31 and the code is pasted in Appendix A section.

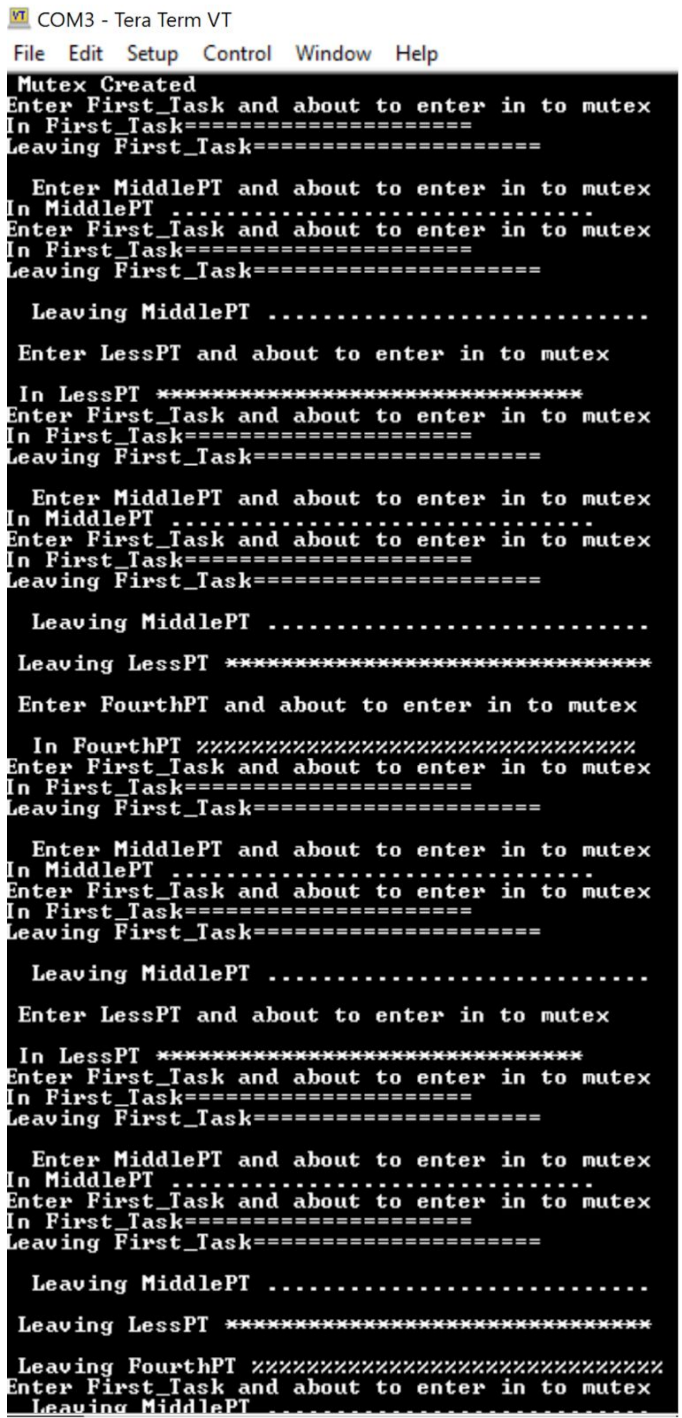

Figure 31: Final result

2. Results of Part 2 - Changing the priorities

In this part since the priorities were changed so the control of the resource (i.e) UART will be passed accordingly. Since for this part the highest priority was given to the fourth task and the lowest priority was given to the less priority task the results can be seen in figure 32 below. The concept applied to this part is the same as described in part 1 of the results. Students are allowed to set different priorities of the tasks and then compare the results. The code for this part is shown in Appendix B.

Figure 32: Changed Priority result

5. Justification

After implementing the project, one can easily follow the step to execute the program. Through this project, students can gain more knowledge of preemptive scheduling as well. Preemptive scheduling monitors the scheduler, and the task with the highest priority gets executed first.

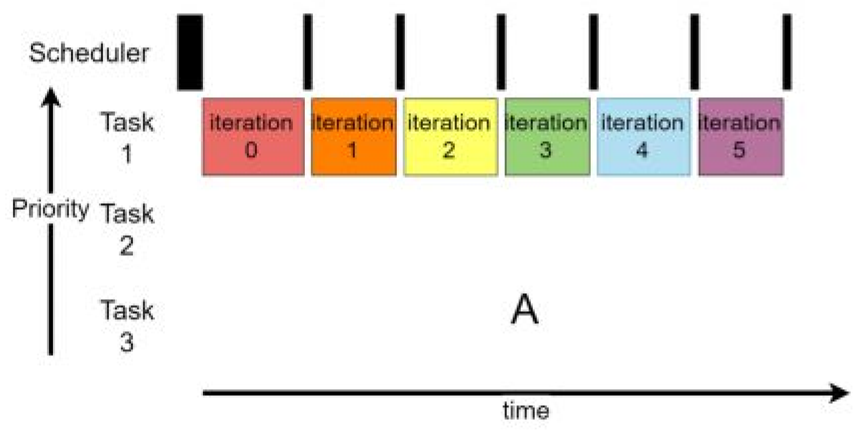

Figure 33: Normal Execution of tasks

For example: Supposedly, there are three tasks, and each performs the same function. As shown in the figure above, if task 1 contains the top priority, it occupies the resource first. Then the next job with the highest priority gets the resource. So after task 1, task 2 gets the resource. After task 2, the resource gets in the process. Task 3 gets the next highest priority. This is how a task is performed.

For this project, the students need to understand how a mutex works with tasks. Mutex and semaphore are considered to be one of the most critical functions in FreeRTOS. Mutexes is one of those tools which can change the priority of the task for the time being. Mutexes also ensure no extra delay being caused while the priorities are being changed in the scheduler.

For example, as shown in figure 34 below.

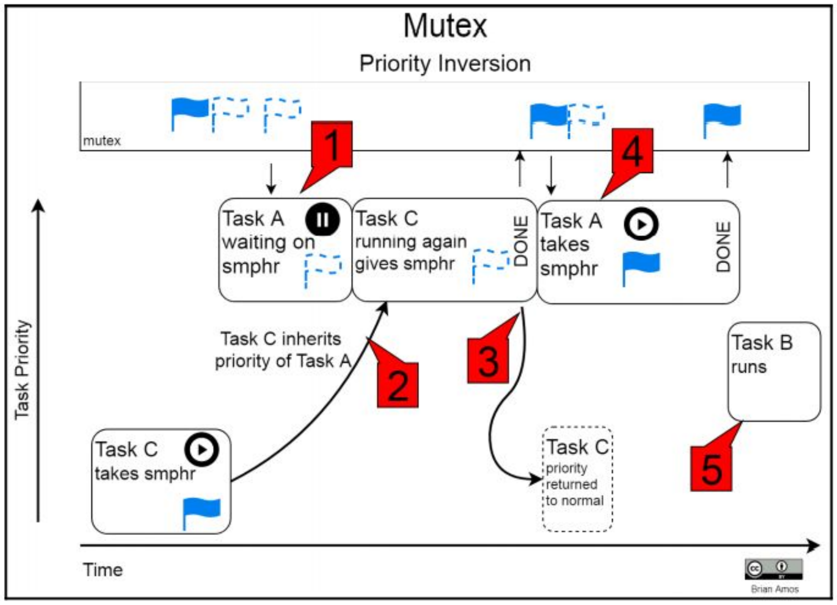

Figure 34: Mutexes and Task priority

Supposedly, there are three tasks: Task A, Task B, and Task C. While task A is in the queue and waiting for task C to give up the mutex since task C has higher priority than the other tasks. After a specific delay task, C again wakes up and occupies the mutexes. Once task C has been completed, the mutex is released, allowing task A for completion.

6. Conclusion

With this project, students will be able to understand about mutex. Mutex stands for mutual exclusion. This is used for giving access to a resource. It basically uses blocks and lock technique. This project will allow students how a mutex allows only one of the tasks to occupy the resource at a time. The rest of the tasks are blocked to use the resource once the critical section has been locked.Through this students will also gain knowledge on tasks and its execution priorities, scheduling, etc. Overall, to make this project the concepts taught in the class were used. The project allows students to understand the concepts in a better way.

7. References

a. Norris, Donald. *Programming with STM32: Getting Started with the Nucleo Board and C/C++*. McGraw Hill Professional, 2018.

b. *FreeRTOS Tutorial 7 || MUTEX || STM32 || CubeIDE*. 24 Apr. 2020, www.youtube.com/watch?v=4NtB1HU3sbQ.

c. Noviello, Carmine. "Mastering STM32." *A step-by-step guide to the most complete ARM Cortex-M platform, using a free and powerful development environment based on Eclipse and GCC. Leadpub* (2017).

d. Amos, Brian. “Hands-On RTOS with Microcontrollers.” *O'Reilly Online Learning*, Packt Publishing, ww.oreilly.com/library/view/hands-on-rtos-with/9781838826734/.

8. Appendix

Appendix A

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under Ultimate Liberty license
  * SLA0044, the "License"; You may not use this file except in compliance with
  * the License. You may obtain a copy of the License at:
  *                             www.st.com/SLA0044
  *
  ******************************************************************************
  */
/* USER CODE END Header */

/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "cmsis_os.h"
#include "string.h"
#include "semphr.h"
#include "queue.h"


#include "FreeRTOS.h"
#include "task.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 */
uint8_t myTxData[17] = "Mutex Created\r\n";

/* USER CODE END PD */

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

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
I2C_HandleTypeDef hi2c1;

SPI_HandleTypeDef hspi1;

UART_HandleTypeDef huart1;
UART_HandleTypeDef huart2;

///* Definitions for defaultTask */
//osThreadId_t defaultTaskHandle;
//const osThreadAttr_t defaultTask_attributes = {
//  .name = "defaultTask",
//  .priority = (osPriority_t) osPriorityNormal,
//  .stack_size = 128
//};
///* Definitions for myTask02 */
//osThreadId_t myTask02Handle;
//const osThreadAttr_t myTask02_attributes = {
//  .name = "myTask02",
//  .priority = (osPriority_t) osPriorityNormal,
//  .stack_size = 128
//};
///* Definitions for myTask03 */
//osThreadId_t myTask03Handle;
//const osThreadAttr_t myTask03_attributes = {
//  .name = "myTask03",
//  .priority = (osPriority_t) osPriorityLow,
//  .stack_size = 128
//};
///* Definitions for myTask04 */
//osThreadId_t myTask04Handle;
//const osThreadAttr_t myTask04_attributes = {
//  .name = "myTask04",
//  .priority = (osPriority_t) osPriorityNormal,
//  .stack_size = 128
//};
/* Definitions for mutex01 */
osMutexId_t mutex01Handle;
const osMutexAttr_t mutex01_attributes = {
```

```
  .name = "mutex01"
};
/* Definitions for semaphore01 */
osSemaphoreId_t semaphore01Handle;
const osSemaphoreAttr_t semaphore01_attributes = {
  .name = "semaphore01"
};
/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_I2C1_Init(void);
static void MX_SPI1_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_USART2_UART_Init(void);


/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
SemaphoreHandle_t SimpleMutex;                    // define the semaphore handler

TaskHandle_t First_Task_Handler;                                                  // defined
three tasks
TaskHandle_t MiddlePT_Handler;                                                    //  using task
Handler
TaskHandle_t LessPT_Handler;                                             //
TaskHandle_t Fourth_Handler;
TaskHandle_t Fifth_Handler;



void First_PT (void *argument);
void MiddlePT_Task (void *argument);
void LessPT_Task (void *argument);
void Fourth_Task (void *argument);
void Fifth_Task (void *argument);


void Send_Uart (char *str)
{
	xSemaphoreTake(SimpleMutex, portMAX_DELAY);
	HAL_UART_Transmit(&huart2,str, strlen(str), HAL_MAX_DELAY);
	HAL_Delay(2000);
	xSemaphoreGive(SimpleMutex);
```

```
}

/* 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_I2C1_Init();
  MX_SPI1_Init();
  MX_USART1_UART_Init();
  MX_USART2_UART_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */
  /* Init scheduler */
  osKernelInitialize();
  /* Create the mutex(es) */
  /* creation of mutex01 */
  mutex01Handle = osMutexNew(&mutex01_attributes);

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

  /* Create the semaphores(s) */
```

```
/* creation of semaphore01 */
semaphore01Handle = osSemaphoreNew(2, 2, &semaphore01_attributes);

/* 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 defaultTask */

/* Start scheduler */


/* We should never get here as control is now taken by the scheduler */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
        SimpleMutex = xSemaphoreCreateMutex();

        if (SimpleMutex != NULL)
        {
         HAL_UART_Transmit(&huart2,myTxData,15 ,1000);

        }
        // priority changed

xTaskCreate(First_PT,"First_Task",128,NULL,3,&First_Task_Handler); // priority changed from
3 to 2
xTaskCreate(MiddlePT_Task,"MiddlePT",128,NULL,2,&MiddlePT_Handler); // priority changed
from 2 to 1
xTaskCreate(LessPT_Task,"LessPT",128,NULL,1,&LessPT_Handler); // priority changed from 1
to 0
        xTaskCreate (Fourth_Task,"Fourth_task",128,NULL,0,&First_Task_Handler); // priority
changed from 0 to 3

        vTaskStartScheduler();



while (1)
{
        /* USER CODE END WHILE */
```

```
        /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

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

  /** Initializes the CPU, AHB and APB busses clocks
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_BYPASS;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
        Error_Handler();
  }
  /** Initializes the CPU, AHB and APB busses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSE;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
        Error_Handler();
  }
  PeriphClkInit.PeriphClockSelection =
RCC_PERIPHCLK_USART1|RCC_PERIPHCLK_USART2
                              |RCC_PERIPHCLK_I2C1;
  PeriphClkInit.Usart1ClockSelection = RCC_USART1CLKSOURCE_PCLK2;
  PeriphClkInit.Usart2ClockSelection = RCC_USART2CLKSOURCE_PCLK1;
  PeriphClkInit.I2c1ClockSelection = RCC_I2C1CLKSOURCE_PCLK1;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
        Error_Handler();
  }
  /** Configure the main internal regulator output voltage
  */
  if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) !=
HAL_OK)
```

```
  {
          Error_Handler();
  }
}

/**
  * @brief I2C1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C1_Init(void)
{

  /* USER CODE BEGIN I2C1_Init 0 */

  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */

  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.Timing = 0x2000090E;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
          Error_Handler();
  }
  /** Configure Analogue filter
  */
  if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
  {
          Error_Handler();
  }
  /** Configure Digital filter
  */
  if (HAL_I2CEx_ConfigDigitalFilter(&hi2c1, 0) != HAL_OK)
  {
          Error_Handler();
  }
  /* USER CODE BEGIN I2C1_Init 2 */

  /* USER CODE END I2C1_Init 2 */

}
```

```
/**
  * @brief SPI1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_SPI1_Init(void)
{

  /* USER CODE BEGIN SPI1_Init 0 */

  /* USER CODE END SPI1_Init 0 */

  /* USER CODE BEGIN SPI1_Init 1 */

  /* USER CODE END SPI1_Init 1 */
  /* SPI1 parameter configuration*/
  hspi1.Instance = SPI1;
  hspi1.Init.Mode = SPI_MODE_MASTER;
  hspi1.Init.Direction = SPI_DIRECTION_2LINES;
  hspi1.Init.DataSize = SPI_DATASIZE_4BIT;
  hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi1.Init.NSS = SPI_NSS_SOFT;
  hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
  hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi1.Init.CRCPolynomial = 7;
  hspi1.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
  hspi1.Init.NSSPMode = SPI_NSS_PULSE_ENABLE;
  if (HAL_SPI_Init(&hspi1) != HAL_OK)
  {
        Error_Handler();
  }
  /* USER CODE BEGIN SPI1_Init 2 */

  /* USER CODE END SPI1_Init 2 */

}

/**
  * @brief USART1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART1_UART_Init(void)
{

  /* USER CODE BEGIN USART1_Init 0 */

  /* USER CODE END USART1_Init 0 */
```

```
  /* USER CODE BEGIN USART1_Init 1 */

  /* USER CODE END USART1_Init 1 */
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  huart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
        Error_Handler();
  }
  /* USER CODE BEGIN USART1_Init 2 */

  /* USER CODE END USART1_Init 2 */

}

/**
  * @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;
  huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  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};

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

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, LD3_Pin|GPIO_PIN_4|GPIO_PIN_5, GPIO_PIN_RESET);

  /*Configure GPIO pins : LD3_Pin PB4 PB5 */
  GPIO_InitStruct.Pin = LD3_Pin|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 4 */

void First_PT (void *argument)
{       char *strtosend = "In First_Task=====================\r\n";
        while (1)
        {
                        char *str = "Enter First_Task and about to enter in to mutex \r\n";
                HAL_UART_Transmit(&huart2, str, strlen(str),HAL_MAX_DELAY);

                        Send_Uart(strtosend);

                        char *str2 = "Leaving First_Task=====================\r\n ";
                HAL_UART_Transmit(&huart2, str2, strlen(str2),HAL_MAX_DELAY);
```

```
                        vTaskDelay(1500);
            }
}

            void MiddlePT_Task (void *argument)
{           char *strtosend = "In MiddlePT ..............................\r\n";
            while (1)
            {
                        char *str = "\n Enter MiddlePT and about to enter in to mutex \r\n";
                  HAL_UART_Transmit(&huart2, str, strlen(str),HAL_MAX_DELAY);

                        Send_Uart(strtosend);

                        char *str2 = "\n Leaving MiddlePT ............................\r\n";
                  HAL_UART_Transmit(&huart2, str2, strlen(str2),HAL_MAX_DELAY);
                        vTaskDelay(2000);
            }
}
            void LessPT_Task (void *argument)
{           char *strtosend = "\n In LessPT ******************************\r\n";
            while (1)
            {
                        char *str = "\n Enter LessPT and about to enter in to mutex \r\n";
                  HAL_UART_Transmit(&huart2, str, strlen(str),HAL_MAX_DELAY);

                        Send_Uart(strtosend);

                        char *str2 = "\n Leaving LessPT ******************************\r\n";
                  HAL_UART_Transmit(&huart2, str2, strlen(str2),HAL_MAX_DELAY);
                        vTaskDelay(3000);
            }

}


            void Fourth_Task (void *argument)
{           char *strtosend = "\n In FourthPT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\r\n";
            while (1)
            {
                        char *str = "\n Enter FourthPT and about to enter in to mutex \r\n ";
                  HAL_UART_Transmit(&huart2, str, strlen(str),HAL_MAX_DELAY);

                        Send_Uart(strtosend);

                        char *str2 = "\n Leaving FourthPT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\r\n";
                  HAL_UART_Transmit(&huart2, str2, strlen(str2),HAL_MAX_DELAY);
                        vTaskDelay(3000);
            }
```

```
}


/* USER CODE BEGIN 4 */
// Callback functions used as interrupt handlers for each serial
// peripheral. Should create buffers for each one and keep interrupt
// handlers short.


void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {

}

void HAL_SPI_RxCpltCallback(SPI_HandleTypeDef *hspi) {

}

void HAL_I2C_MasterRxCpltCallback(I2C_HandleTypeDef *hi2c) {

}


/* USER CODE END 4 */





/**
  * @brief  Period elapsed callback in non blocking mode
  * @note   This function is called  when TIM1 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 == TIM1) {
	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 */

  /* 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(char *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
           tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

Appendix B

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under Ultimate Liberty license
  * SLA0044, the "License"; You may not use this file except in compliance with
  * the License. You may obtain a copy of the License at:
  *                             www.st.com/SLA0044
  *
  ******************************************************************************
  */
/* USER CODE END Header */

/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "cmsis_os.h"
#include "string.h"
#include "semphr.h"
#include "queue.h"


#include "FreeRTOS.h"
#include "task.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 */
uint8_t myTxData[17] = "Mutex Created\r\n";

/* USER CODE END PD */

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

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
I2C_HandleTypeDef hi2c1;

SPI_HandleTypeDef hspi1;

UART_HandleTypeDef huart1;
UART_HandleTypeDef huart2;

///* Definitions for defaultTask */
//osThreadId_t defaultTaskHandle;
//const osThreadAttr_t defaultTask_attributes = {
//  .name = "defaultTask",
//  .priority = (osPriority_t) osPriorityNormal,
//  .stack_size = 128
//};
///* Definitions for myTask02 */
//osThreadId_t myTask02Handle;
//const osThreadAttr_t myTask02_attributes = {
//  .name = "myTask02",
//  .priority = (osPriority_t) osPriorityNormal,
//  .stack_size = 128
//};
///* Definitions for myTask03 */
//osThreadId_t myTask03Handle;
//const osThreadAttr_t myTask03_attributes = {
//  .name = "myTask03",
//  .priority = (osPriority_t) osPriorityLow,
//  .stack_size = 128
//};
///* Definitions for myTask04 */
//osThreadId_t myTask04Handle;
//const osThreadAttr_t myTask04_attributes = {
//  .name = "myTask04",
//  .priority = (osPriority_t) osPriorityNormal,
//  .stack_size = 128
//};
/* Definitions for mutex01 */
osMutexId_t mutex01Handle;
```

```
const osMutexAttr_t mutex01_attributes = {
  .name = "mutex01"
};
/* Definitions for semaphore01 */
osSemaphoreId_t semaphore01Handle;
const osSemaphoreAttr_t semaphore01_attributes = {
  .name = "semaphore01"
};
/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_I2C1_Init(void);
static void MX_SPI1_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_USART2_UART_Init(void);


/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
SemaphoreHandle_t SimpleMutex;                    // define the semaphore handler

TaskHandle_t First_Task_Handler;                                          // defined
three tasks
TaskHandle_t MiddlePT_Handler;                                            //  using task
Handler
TaskHandle_t LessPT_Handler;                                      //
TaskHandle_t Fourth_Handler;
TaskHandle_t Fifth_Handler;



void First_PT (void *argument);
void MiddlePT_Task (void *argument);
void LessPT_Task (void *argument);
void Fourth_Task (void *argument);
void Fifth_Task (void *argument);


void Send_Uart (char *str)
{
	xSemaphoreTake(SimpleMutex, portMAX_DELAY);
	HAL_UART_Transmit(&huart2,str, strlen(str), HAL_MAX_DELAY);
	HAL_Delay(2000);
```

```
        xSemaphoreGive(SimpleMutex);
}

/* 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_I2C1_Init();
  MX_SPI1_Init();
  MX_USART1_UART_Init();
  MX_USART2_UART_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */
  /* Init scheduler */
  osKernelInitialize();
  /* Create the mutex(es) */
  /* creation of mutex01 */
  mutex01Handle = osMutexNew(&mutex01_attributes);

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

```
  /* Create the semaphores(s) */
  /* creation of semaphore01 */
  semaphore01Handle = osSemaphoreNew(2, 2, &semaphore01_attributes);

  /* 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 defaultTask */

  /* Start scheduler */


  /* We should never get here as control is now taken by the scheduler */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
	SimpleMutex = xSemaphoreCreateMutex();

	if (SimpleMutex != NULL)
	{
	 HAL_UART_Transmit(&huart2,myTxData,15 ,1000);

	}
	// priority changed

xTaskCreate(First_PT,"First_Task",128,NULL,2,&First_Task_Handler); // priority changed from
3 to 2
xTaskCreate(MiddlePT_Task,"MiddlePT",128,NULL,1,&MiddlePT_Handler); // priority changed
from 2 to 1
xTaskCreate(LessPT_Task,"LessPT",128,NULL,0,&LessPT_Handler); // priority changed from 1
to 0
	xTaskCreate (Fourth_Task,"Fourth_task",128,NULL,3,&First_Task_Handler); // priority
changed from 0 to 3

	vTaskStartScheduler();



  while (1)
  {
	/* USER CODE END WHILE */
```

```
        /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

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

  /** Initializes the CPU, AHB and APB busses clocks
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_BYPASS;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
        Error_Handler();
  }
  /** Initializes the CPU, AHB and APB busses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSE;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
        Error_Handler();
  }
  PeriphClkInit.PeriphClockSelection =
RCC_PERIPHCLK_USART1|RCC_PERIPHCLK_USART2
                              |RCC_PERIPHCLK_I2C1;
  PeriphClkInit.Usart1ClockSelection = RCC_USART1CLKSOURCE_PCLK2;
  PeriphClkInit.Usart2ClockSelection = RCC_USART2CLKSOURCE_PCLK1;
  PeriphClkInit.I2c1ClockSelection = RCC_I2C1CLKSOURCE_PCLK1;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
        Error_Handler();
  }
  /** Configure the main internal regulator output voltage
  */
```

```
  if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) !=
HAL_OK)
  {
        Error_Handler();
  }
}

/**
  * @brief I2C1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C1_Init(void)
{

  /* USER CODE BEGIN I2C1_Init 0 */

  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */

  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.Timing = 0x2000090E;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
        Error_Handler();
  }
  /** Configure Analogue filter
  */
  if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
  {
        Error_Handler();
  }
  /** Configure Digital filter
  */
  if (HAL_I2CEx_ConfigDigitalFilter(&hi2c1, 0) != HAL_OK)
  {
        Error_Handler();
  }
  /* USER CODE BEGIN I2C1_Init 2 */

  /* USER CODE END I2C1_Init 2 */
```

```
}

/**
  * @brief SPI1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_SPI1_Init(void)
{

  /* USER CODE BEGIN SPI1_Init 0 */

  /* USER CODE END SPI1_Init 0 */

  /* USER CODE BEGIN SPI1_Init 1 */

  /* USER CODE END SPI1_Init 1 */
  /* SPI1 parameter configuration*/
  hspi1.Instance = SPI1;
  hspi1.Init.Mode = SPI_MODE_MASTER;
  hspi1.Init.Direction = SPI_DIRECTION_2LINES;
  hspi1.Init.DataSize = SPI_DATASIZE_4BIT;
  hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi1.Init.NSS = SPI_NSS_SOFT;
  hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
  hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi1.Init.CRCPolynomial = 7;
  hspi1.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
  hspi1.Init.NSSPMode = SPI_NSS_PULSE_ENABLE;
  if (HAL_SPI_Init(&hspi1) != HAL_OK)
  {
          Error_Handler();
  }
  /* USER CODE BEGIN SPI1_Init 2 */

  /* USER CODE END SPI1_Init 2 */

}

/**
  * @brief USART1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART1_UART_Init(void)
{

  /* USER CODE BEGIN USART1_Init 0 */
```

```
  /* USER CODE END USART1_Init 0 */

  /* USER CODE BEGIN USART1_Init 1 */

  /* USER CODE END USART1_Init 1 */
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  huart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
        Error_Handler();
  }
  /* USER CODE BEGIN USART1_Init 2 */

  /* USER CODE END USART1_Init 2 */

}

/**
  * @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;
  huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
```

```
  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};

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

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, LD3_Pin|GPIO_PIN_4|GPIO_PIN_5, GPIO_PIN_RESET);

  /*Configure GPIO pins : LD3_Pin PB4 PB5 */
  GPIO_InitStruct.Pin = LD3_Pin|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 4 */

void First_PT (void *argument)
{       char *strtosend = "In First_Task====================\r\n";
        while (1)
        {
                        char *str = "Enter First_Task and about to enter in to mutex \r\n";
                HAL_UART_Transmit(&huart2, str, strlen(str),HAL_MAX_DELAY);

                        Send_Uart(strtosend);
```

```
                char *str2 = "Leaving First_Task=====================\r\n ";
        HAL_UART_Transmit(&huart2, str2, strlen(str2),HAL_MAX_DELAY);
                vTaskDelay(1500);
    }
}

    void MiddlePT_Task (void *argument)
{   char *strtosend = "In MiddlePT ..............................\r\n";
    while (1)
    {
                char *str = "\n Enter MiddlePT and about to enter in to mutex \r\n";
        HAL_UART_Transmit(&huart2, str, strlen(str),HAL_MAX_DELAY);

                Send_Uart(strtosend);

                char *str2 = "\n Leaving MiddlePT ............................\r\n";
        HAL_UART_Transmit(&huart2, str2, strlen(str2),HAL_MAX_DELAY);
                vTaskDelay(2000);
    }
}
    void LessPT_Task (void *argument)
{   char *strtosend = "\n In LessPT ******************************\r\n";
    while (1)
    {
                char *str = "\n Enter LessPT and about to enter in to mutex \r\n";
        HAL_UART_Transmit(&huart2, str, strlen(str),HAL_MAX_DELAY);

                Send_Uart(strtosend);

                char *str2 = "\n Leaving LessPT ******************************\r\n";
        HAL_UART_Transmit(&huart2, str2, strlen(str2),HAL_MAX_DELAY);
                vTaskDelay(3000);
    }

}


    void Fourth_Task (void *argument)
{   char *strtosend = "\n In FourthPT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\r\n";
    while (1)
    {
                char *str = "\n Enter FourthPT and about to enter in to mutex \r\n ";
        HAL_UART_Transmit(&huart2, str, strlen(str),HAL_MAX_DELAY);

                Send_Uart(strtosend);

                char *str2 = "\n Leaving FourthPT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\r\n";
        HAL_UART_Transmit(&huart2, str2, strlen(str2),HAL_MAX_DELAY);
                vTaskDelay(3000);
```

```
        }

}


/* USER CODE BEGIN 4 */
// Callback functions used as interrupt handlers for each serial
// peripheral. Should create buffers for each one and keep interrupt
// handlers short.


void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {

}

void HAL_SPI_RxCpltCallback(SPI_HandleTypeDef *hspi) {

}

void HAL_I2C_MasterRxCpltCallback(I2C_HandleTypeDef *hi2c) {

}


/* USER CODE END 4 */





/**
  * @brief  Period elapsed callback in non blocking mode
  * @note   This function is called  when TIM1 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 == TIM1) {
        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 */

  /* 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(char *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
         tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```