**RTOS Lab - Complete code and task solutions**

Contents

[Part 1 2](#_Toc459139112)

[Part 2 5](#_Toc459139113)

[Task 1 10](#_Toc459139114)

[Part 3 14](#_Toc459139115)

[Task 2 19](#_Toc459139116)

[Task 3 24](#_Toc459139117)

# Part 1

**my\_headers.h**

#include "stm32f4xx.h"

void Initialise\_LED(void); // Declaration for the function to initialise the LED

void Blink\_LED(uint8\_t); // Declaration for the function to blink the LED

**my\_headers.c**

#include "stm32f4xx.h"

// Definition for the function to initialise the LED and button

void Initialise\_LED(void){

// Initialize GPIO Port for LEDs

RCC->AHB1ENR |= RCC\_AHB1ENR\_GPIODEN; // Enable Port D clock

GPIOD->MODER |= GPIO\_MODER\_MODER12\_0; // Port D.12 output - green LED

}

// Definition for the function to blink the LED

void Blink\_LED(uint8\_t LED\_state){

if(LED\_state == 1){ // Checks to see if the request is to turn the LED on or off

GPIOD->BSRR = 1<<12; // Turn on the green LED

}

else{

GPIOD->BSRR = 1<<(12+16); // Turn off the green LED

}

}

**Thread.h**

extern int Init\_Blink\_LED\_Thread (void); // Standard format to declare the function to initialise the main thread function.

void Blink\_LED\_Thread (void const \*argument); // Standard format to declare the main thread function.

**Thread.c**

#include "cmsis\_os.h" // CMSIS RTOS header file

#include "stm32f4xx.h"

#include "my\_headers.h"

/\*-------------------------------------------------------------------------

\* Blink LED Thread

\*-----------------------------------------------------------------------\*/

void Blink\_LED\_Thread (void const \*argument); // Declares the main thread function that is defined later in the code

osThreadId tid\_Blink\_LED\_Thread; // Declares an ID that we will associate with the thread and which allows easy reference to it when using some of the OS functions.

osThreadDef (Blink\_LED\_Thread, osPriorityNormal, 1, 0); // Declares the main thread object that we will use later. The parameters can be used to adjust certain properties, such as the priority of a thread and how many instances of it exist.

// Code to define the thread function to initialise the main thread - this initialise function is called from the “main.c” file to start the thread.

int Init\_Blink\_LED\_Thread (void) {

tid\_Blink\_LED\_Thread = osThreadCreate (osThread(Blink\_LED\_Thread), NULL); // Creates the main thread object that we have declared and assigns it the thread ID that we have declared.

if(!tid\_Blink\_LED\_Thread) return(-1); // Checks to make sure the thread has been created.

return(0);

}

// Code to define the operation of the main thread. This is effectively the code that was in the infinite FOR loop of our previous blinky program.

void Blink\_LED\_Thread (void const \*argument) {

uint8\_t LED\_on = 1; // Defines parameter for LED on

uint8\_t LED\_off = 0; // Defines parameter for LED off

while (1) { // Creates an infinite loop so that the blinking never terminates

Blink\_LED(LED\_on); // Blinks the green LED on once

osDelay(1000); // Uses the built in delay function for the OS to create a 1 second delay. The fundamental delay is specified in the “RTX\_conf\_CM.c” file and usually defaults to 1ms.

Blink\_LED(LED\_off); // Blinks the green LED on once

osDelay(1000); // Uses the built in delay function for the OS to create a 1 second delay. The fundamental delay is specified in the “RTX\_conf\_CM.c” file and usually defaults to 1ms.

osThreadYield(); // This function tells the RTOS that when the thread gets to this stage the RTOS should suspend this thread and run the next thread that is ready to run. If there is no other thread ready (which is the case with this simple program since we only have one thread) then the calling thread continues. This function effectively forces the RTOS to reschedule and is useful in more complex systems and scheduling policies.

}

}

**main.c**

/\*-------------------------------------------------------------------------

\* CMSIS-RTOS 'main' function - Blinky

\*-----------------------------------------------------------------------\*/

#define osObjectsPublic // Define objects in main module

#include "osObjects.h" // RTOS object definitions

#include "stm32f4xx.h"

#include "my\_headers.h"

#include "Thread.h"

int main (void) {

osKernelInitialize (); //Initialize CMSIS-RTOS

/\* Initialise any peripherals or system components \*/

// Initialize the LED

Initialise\_LED();

/\* Initialise any threads \*/

// Initialise the main thread to blink the LED’s

Init\_Blink\_LED\_Thread();

osKernelStart (); // start thread execution

while(1){}; // While loop so the program doesn’t terminate

}

# Part 2

**my\_headers.h**

#include "stm32f4xx.h"

void Initialise\_LED\_and\_button(void); // Declaration for the function to initialise the LED

void Blink\_LED(uint8\_t); // Declaration for the function to blink the LED

void Red\_LED(uint8\_t); // Declaration for the function to blink the LED

**my\_headers.c**

#include "stm32f4xx.h"

// Definition for the function to initialise the LED and button

void Initialise\_LED\_and\_button(void){

// Initialize GPIO Port for LEDs

RCC->AHB1ENR |= RCC\_AHB1ENR\_GPIODEN; // Enable Port D clock

GPIOD->MODER |= GPIO\_MODER\_MODER12\_0; // Port D.12 output - green LED

GPIOD->MODER |= GPIO\_MODER\_MODER14\_0; // Port D.14 output - red LED

//Initialize GPIO for push-button

RCC->AHB1ENR |= RCC\_AHB1ENR\_GPIOAEN; // Enable Port A clock

}

// Definition for the function to blink the LED

void Blink\_LED(uint8\_t LED\_state){

if(LED\_state == 1){ // Checks to see if the request is to turn the LED on or off

GPIOD->BSRR = 1<<12; // Turn on the green LED

}

else{

GPIOD->BSRR = 1<<(12+16); // Turn off the green LED

}

}

// Definition for the function to blink the LED

void Red\_LED(uint8\_t LED\_state){

if(LED\_state == 1){ // Checks to see if the request is to turn the LED on or off

GPIOD->BSRR = 1<<14; // Turn on the red LED

}

else{

GPIOD->BSRR = 1<<(14+16); // Turn off the red LED

}

}

**Thread.h**

extern int Init\_Blink\_LED\_Thread (void); // Standard format to declare the function to initialise the main thread function.

void Blink\_LED\_Thread (void const \*argument); // Standard format to declare the main thread function.

extern int Init\_Button\_Thread (void); // Standard format to declare the function to initialise the main thread function.

void Button\_Thread (void const \*argument); // Standard format to declare the main thread function.

**Thread.c**

#include "cmsis\_os.h" // CMSIS RTOS header file

#include "stm32f4xx.h"

#include "my\_headers.h"

void Blink\_LED\_Thread (void const \*argument); // Declares the main thread function that is defined later in the code

osThreadId tid\_Blink\_LED\_Thread; // Declares an ID that we will associate with the thread and which allows easy reference to it when using some of the OS functions.

osThreadDef (Blink\_LED\_Thread, osPriorityNormal, 1, 0); // Declares the main thread object that we will use later. The parameters can be used to adjust certain properties, such as the priority of a thread and how many instances of it exist.

void Button\_Thread (void const \*argument); // Declares the main thread function that is defined later in the code

osThreadId tid\_Button\_Thread; // Declares an ID that we will associate with the thread and which allows easy reference to it when using some of the OS functions.

osThreadDef (Button\_Thread, osPriorityNormal, 1, 0); // Declares the main thread object that we will use later. The parameters can be used to adjust certain properties, such as the priority of a thread and how many instances of it exist.

/\*-------------------------------------------------------------------------

\* Blink LED Thread

\*-----------------------------------------------------------------------\*/

// Code to define the thread function to initialise the main thread - this initialise function is called from the “main.c” file to start the thread.

int Init\_Blink\_LED\_Thread (void) {

tid\_Blink\_LED\_Thread = osThreadCreate (osThread(Blink\_LED\_Thread), NULL); // Creates the main thread object that we have declared and assigns it the thread ID that we have declared.

if(!tid\_Blink\_LED\_Thread) return(-1); // Checks to make sure the thread has been created.

return(0);

}

// Code to define the operation of the main thread. This is effectively the code that was in the infinite FOR loop of our previous blinky program.

void Blink\_LED\_Thread (void const \*argument) {

uint8\_t LED\_on = 1; // Defines parameter for LED on

uint8\_t LED\_off = 0; // Defines parameter for LED off

while (1) { // Creates an infinite loop so that the blinking never terminates

Blink\_LED(LED\_on); // Blinks the green LED on once

osDelay(1000); // Uses the built in delay function for the OS to create a 1 second delay. The fundamental delay is specified in the “RTX\_conf\_CM.c” file and usually defaults to 1ms.

Blink\_LED(LED\_off); // Blinks the green LED on once

osDelay(1000); // Uses the built in delay function for the OS to create a 1 second delay. The fundamental delay is specified in the “RTX\_conf\_CM.c” file and usually defaults to 1ms.

osThreadYield(); // This function tells the RTOS that when the thread gets to this stage the RTOS should suspend this thread and run the next thread that is ready to run. If there is no other thread ready (which is the case with this simple program since we only have one thread) then the calling thread continues. This function effectively forces the RTOS to reschedule and is useful in more complex systems and scheduling policies.

}

}

/\*-------------------------------------------------------------------------

\* Red LED on when button pressed thread Thread

\*-----------------------------------------------------------------------\*/

// Code to define the thread function to initialise the main thread - this initialise function is called from the “main.c” file to start the thread.

int Init\_Button\_Thread (void) {

tid\_Button\_Thread = osThreadCreate (osThread(Button\_Thread), NULL); // Creates the main thread object that we have declared and assigns it the thread ID that we have declared.

if(!tid\_Button\_Thread) return(-1); // Checks to make sure the thread has been created.

return(0);

}

// Code to define the operation of the main thread. This is effectively the code that was in the infinite FOR loop of our previous blinky program.

void Button\_Thread (void const \*argument) {

uint8\_t LED\_on = 1; // Defines parameter for LED on

uint8\_t LED\_off = 0; // Defines parameter for LED off

while (1) { // Creates an infinite loop so that the blinking never terminates

// Checks the state of the push-button and only turns the red LED on if the button has only just been pressed, which is indicated by the state of the red LED.

if(((GPIOA->IDR & 0x00000001) == 0x00000001) & ((GPIOD->ODR & (1<<14)) != (1<<14))){

Red\_LED(LED\_on); // Turn red LED on

}

// Checks the state of the push-button and only turns the red LED off if the button has only just been released, which is indicated by the state of the red LED.

else if(((GPIOA->IDR & 0x00000001) != 0x00000001) & ((GPIOD->ODR & (1<<14)) == (1<<14))){

Red\_LED(LED\_off); // Turn red LED off

}

osThreadYield(); // This function tells the RTOS that when the thread gets to this stage the RTOS should suspend this thread and run the next thread that is ready to run. If there is no other thread ready (which is the case with this simple program since we only have one thread) then the calling thread continues. This function effectively forces the RTOS to reschedule and is useful in more complex systems and scheduling policies.

}

}

**main.c**

/\*-------------------------------------------------------------------------

\* CMSIS-RTOS 'main' function - Blinky

\*-----------------------------------------------------------------------\*/

#define osObjectsPublic // Define objects in main module

#include "osObjects.h" // RTOS object definitions

#include "stm32f4xx.h"

#include "my\_headers.h"

#include "Thread.h"

int main (void) {

osKernelInitialize (); //Initialize CMSIS-RTOS

/\* Initialise any peripherals or system components \*/

// Initialize the LED and button

Initialise\_LED\_and\_button();

/\* Initialise any threads \*/

// Initialise the main thread to blink the LED’s

Init\_Blink\_LED\_Thread();

Init\_Button\_Thread();

osKernelStart (); // start thread execution

while(1){}; // While loop so the program doesn’t terminate

}

# Task 1

**my\_headers.h**

#include "stm32f4xx.h"

void Initialise\_LED\_and\_button(void); // Declaration for the function to initialise the LED and button

void Blink\_LED(uint8\_t,uint8\_t); // Declaration for the function to blink the LED - now has two inputs: one for the state and one for the colour LED

**my\_headers.c**

#include "stm32f4xx.h"

// Definition for the function to initialise the LED and button

void Initialise\_LED\_and\_button(void){

// Initialize GPIO Port for LEDs

RCC->AHB1ENR |= RCC\_AHB1ENR\_GPIODEN; // Enable Port D clock

GPIOD->MODER |= GPIO\_MODER\_MODER12\_0; // Port D.12 output - green LED

GPIOD->MODER |= GPIO\_MODER\_MODER14\_0; // Port D.14 output - red LED

//Initialize GPIO for push-button

RCC->AHB1ENR |= RCC\_AHB1ENR\_GPIOAEN; // Enable Port A clock

}

// Definition for the function to blink the LED

void Blink\_LED(uint8\_t LED\_state, uint8\_t LED\_colour){

if(LED\_state == 1){ // Checks to see if the request is to turn the LED on or off

GPIOD->BSRR = 1<<LED\_colour; // Turn on the green LED

}

else{

GPIOD->BSRR = 1<<(LED\_colour+16); // Turn off the green LED

}

}

**Thread.h**

extern int Init\_Blink\_LED\_Thread (void); // Standard format to declare the function to initialise the main thread function.

void Blink\_LED\_Thread (void const \*argument); // Standard format to declare the main thread function.

extern int Init\_Button\_Thread (void); // Standard format to declare the function to initialise the main thread function.

void Button\_Thread (void const \*argument); // Standard format to declare the main thread function.

**Thread.c**

#include "cmsis\_os.h" // CMSIS RTOS header file

#include "stm32f4xx.h"

#include "my\_headers.h"

// Thread Declarations

void Blink\_LED\_Thread (void const \*argument); // Declares the main thread function that is defined later in the code

osThreadId tid\_Blink\_LED\_Thread; // Declares an ID that we will associate with the thread and which allows easy reference to it when using some of the OS functions.

osThreadDef (Blink\_LED\_Thread, osPriorityNormal, 1, 0); // Declares the main thread object that we will use later. The parameters can be used to adjust certain properties, such as the priority of a thread and how many instances of it exist.

void Button\_Thread (void const \*argument); // Declares the main thread function that is defined later in the code

osThreadId tid\_Button\_Thread; // Declares an ID that we will associate with the thread and which allows easy reference to it when using some of the OS functions.

osThreadDef (Button\_Thread, osPriorityNormal, 1, 0); // Declares the main thread object that we will use later. The parameters can be used to adjust certain properties, such as the priority of a thread and how many instances of it exist.

/\*-------------------------------------------------------------------------

\* Blink LED Thread

\*-----------------------------------------------------------------------\*/

// Code to define the thread function to initialise the main thread - this initialise function is called from the “main.c” file to start the thread.

int Init\_Blink\_LED\_Thread (void) {

tid\_Blink\_LED\_Thread = osThreadCreate (osThread(Blink\_LED\_Thread), NULL); // Creates the main thread object that we have declared and assigns it the thread ID that we have declared.

if(!tid\_Blink\_LED\_Thread) return(-1); // Checks to make sure the thread has been created.

return(0);

}

// Code to define the operation of the main thread. This is effectively the code that was in the infinite FOR loop of our previous blinky program.

void Blink\_LED\_Thread (void const \*argument) {

uint8\_t LED\_on = 1; // Defines parameter for LED on

uint8\_t LED\_off = 0; // Defines parameter for LED off

uint8\_t green\_LED = 12; // Defines parameter for green LED (GPIOD pin 12)

while (1) { // Creates an infinite loop so that the blinking never terminates

Blink\_LED(LED\_on,green\_LED); // Blinks the green LED on once

osDelay(1000); // Uses the built in delay function for the OS to create a 1 second delay. The fundamental delay is specified in the “RTX\_conf\_CM.c” file and usually defaults to 1ms.

Blink\_LED(LED\_off,green\_LED); // Blinks the green LED on once

osDelay(1000); // Uses the built in delay function for the OS to create a 1 second delay. The fundamental delay is specified in the “RTX\_conf\_CM.c” file and usually defaults to 1ms.

osThreadYield(); // This function tells the RTOS that when the thread gets to this stage the RTOS should suspend this thread and run the next thread that is ready to run. If there is no other thread ready (which is the case with this simple program since we only have one thread) then the calling thread continues. This function effectively forces the RTOS to reschedule and is useful in more complex systems and scheduling policies.

}

}

/\*-------------------------------------------------------------------------

\* Red LED on when button pressed thread Thread

\*-----------------------------------------------------------------------\*/

// Code to define the thread function to initialise the main thread - this initialise function is called from the “main.c” file to start the thread.

int Init\_Button\_Thread (void) {

tid\_Button\_Thread = osThreadCreate (osThread(Button\_Thread), NULL); // Creates the main thread object that we have declared and assigns it the thread ID that we have declared.

if(!tid\_Button\_Thread) return(-1); // Checks to make sure the thread has been created.

return(0);

}

// Code to define the operation of the main thread. This is effectively the code that was in the infinite FOR loop of our previous blinky program.

void Button\_Thread (void const \*argument) {

uint8\_t LED\_on = 1; // Defines parameter for LED on

uint8\_t LED\_off = 0; // Defines parameter for LED off

uint8\_t red\_LED = 14; // Defines parameter for red LED (GPIOD pin 14)

while (1) { // Creates an infinite loop so that the blinking never terminates

// Checks the state of the push-button and only turns the red LED on if the button has only just been pressed, which is indicated by the state of the red LED.

if(((GPIOA->IDR & 0x00000001) == 0x00000001) & ((GPIOD->ODR & (1<<red\_LED)) != (1<<red\_LED))){

Blink\_LED(LED\_on,red\_LED); // Turn red LED on

}

// Checks the state of the push-button and only turns the red LED off if the button has only just been released, which is indicated by the state of the red LED.

else if(((GPIOA->IDR & 0x00000001) != 0x00000001) & ((GPIOD->ODR & (1<<red\_LED)) == (1<<red\_LED))){

Blink\_LED(LED\_off,red\_LED); // Turn red LED off

}

osThreadYield(); // This function tells the RTOS that when the thread gets to this stage the RTOS should suspend this thread and run the next thread that is ready to run. If there is no other thread ready (which is the case with this simple program since we only have one thread) then the calling thread continues. This function effectively forces the RTOS to reschedule and is useful in more complex systems and scheduling policies.

}

}

**main.c**

/\*-------------------------------------------------------------------------

\* CMSIS-RTOS 'main' function - Blinky

\*-----------------------------------------------------------------------\*/

#define osObjectsPublic // Define objects in main module

#include "osObjects.h" // RTOS object definitions

#include "stm32f4xx.h"

#include "my\_headers.h"

#include "Thread.h"

int main (void) {

osKernelInitialize (); //Initialize CMSIS-RTOS

/\* Initialise any peripherals or system components \*/

// Initialize the LED and button

Initialise\_LED\_and\_button();

/\* Initialise any threads \*/

// Initialise the main thread to blink the LED’s

Init\_Blink\_LED\_Thread();

Init\_Button\_Thread();

osKernelStart (); // start thread execution

while(1){}; // While loop so the program doesn’t terminate

}

# Part 3

**my\_headers.h**

#include "stm32f4xx.h"

void Initialise\_LED\_and\_button(void); // Declaration for the function to initialise the LED and button

void Blink\_LED(uint8\_t,uint8\_t); // Declaration for the function to blink the LED - now has two inputs: one for the state and one for the colour LED

**my\_headers.c**

#include "stm32f4xx.h"

// Definition for the function to initialise the LED and button

void Initialise\_LED\_and\_button(void){

// Initialize GPIO Port for LEDs

RCC->AHB1ENR |= RCC\_AHB1ENR\_GPIODEN; // Enable Port D clock

GPIOD->MODER |= GPIO\_MODER\_MODER12\_0; // Port D.12 output - green LED

GPIOD->MODER |= GPIO\_MODER\_MODER14\_0; // Port D.14 output - red LED

//Initialize GPIO for push-button

RCC->AHB1ENR |= RCC\_AHB1ENR\_GPIOAEN; // Enable Port A clock

}

// Definition for the function to blink the LED

void Blink\_LED(uint8\_t LED\_state, uint8\_t LED\_colour){

if(LED\_state == 1){ // Checks to see if the request is to turn the LED on or off

GPIOD->BSRR = 1<<LED\_colour; // Turn on the green LED

}

else{

GPIOD->BSRR = 1<<(LED\_colour+16); // Turn off the green LED

}

}

**Thread.h**

extern int Init\_Blink\_LED\_Thread (void); // Standard format to declare the function to initialise the main thread function.

void Blink\_LED\_Thread (void const \*argument); // Standard format to declare the main thread function.

extern int Init\_Button\_Thread (void); // Standard format to declare the function to initialise the main thread function.

void Button\_Thread (void const \*argument); // Standard format to declare the main thread function.

**Thread.c**

#include "cmsis\_os.h" // CMSIS RTOS header file

#include "stm32f4xx.h"

#include "my\_headers.h"

// Thread Declarations

void Blink\_LED\_Thread (void const \*argument); // Declares the main thread function that is defined later in the code

osThreadId tid\_Blink\_LED\_Thread; // Declares an ID that we will associate with the thread and which allows easy reference to it when using some of the OS functions.

osThreadDef (Blink\_LED\_Thread, osPriorityNormal, 1, 0); // Declares the main thread object that we will use later. The parameters can be used to adjust certain properties, such as the priority of a thread and how many instances of it exist.

void Button\_Thread (void const \*argument); // Declares the main thread function that is defined later in the code

osThreadId tid\_Button\_Thread; // Declares an ID that we will associate with the thread and which allows easy reference to it when using some of the OS functions.

osThreadDef (Button\_Thread, osPriorityNormal, 1, 0); // Declares the main thread object that we will use later. The parameters can be used to adjust certain properties, such as the priority of a thread and how many instances of it exist.

/\*-------------------------------------------------------------------------

\* Blink LED Thread

\*-----------------------------------------------------------------------\*/

// Code to define the thread function to initialise the main thread - this initialise function is called from the “main.c” file to start the thread.

int Init\_Blink\_LED\_Thread (void) {

tid\_Blink\_LED\_Thread = osThreadCreate (osThread(Blink\_LED\_Thread), NULL); // Creates the main thread object that we have declared and assigns it the thread ID that we have declared.

if(!tid\_Blink\_LED\_Thread) return(-1); // Checks to make sure the thread has been created.

return(0);

}

// Code to define the operation of the main thread. This is effectively the code that was in the infinite FOR loop of our previous blinky program.

void Blink\_LED\_Thread (void const \*argument) {

uint8\_t LED\_on = 1; // Defines parameter for LED on

uint8\_t LED\_off = 0; // Defines parameter for LED off

uint8\_t green\_LED = 12; // Defines parameter for green LED (GPIOD pin 12)

osSignalSet(tid\_Blink\_LED\_Thread,0x01);// Set flag 0x01 of the blink LED thread so that it resumes next time wait is called

while (1) { // Creates an infinite loop so that the blinking never terminates

osSignalWait(0x01,osWaitForever); // Waits until flag 0x01 of this thread is set

osSignalSet(tid\_Blink\_LED\_Thread,0x01);// Set flag 0x01 of the blink LED thread so that it resumes next time wait is called

Blink\_LED(LED\_on,green\_LED); // Blinks the green LED on once

osDelay(1000); // Uses the built in delay function for the OS to create a 1 second delay. The fundamental delay is specified in the “RTX\_conf\_CM.c” file and usually defaults to 1ms.

osSignalWait(0x01,osWaitForever); // Waits until flag 0x01 of this thread is set

osSignalSet(tid\_Blink\_LED\_Thread,0x01);// Set flag 0x01 of the blink LED thread so that it resumes next time wait is called

Blink\_LED(LED\_off,green\_LED); // Blinks the green LED on once

osDelay(1000); // Uses the built in delay function for the OS to create a 1 second delay. The fundamental delay is specified in the “RTX\_conf\_CM.c” file and usually defaults to 1ms.

osThreadYield(); // This function tells the RTOS that when the thread gets to this stage the RTOS should suspend this thread and run the next thread that is ready to run. If there is no other thread ready (which is the case with this simple program since we only have one thread) then the calling thread continues. This function effectively forces the RTOS to reschedule and is useful in more complex systems and scheduling policies.

}

}

/\*-------------------------------------------------------------------------

\* Red LED on when button pressed thread Thread

\*-----------------------------------------------------------------------\*/

// Code to define the thread function to initialise the main thread - this initialise function is called from the “main.c” file to start the thread.

int Init\_Button\_Thread (void) {

tid\_Button\_Thread = osThreadCreate (osThread(Button\_Thread), NULL); // Creates the main thread object that we have declared and assigns it the thread ID that we have declared.

if(!tid\_Button\_Thread) return(-1); // Checks to make sure the thread has been created.

return(0);

}

// Code to define the operation of the main thread. This is effectively the code that was in the infinite FOR loop of our previous blinky program.

void Button\_Thread (void const \*argument) {

uint8\_t LED\_on = 1; // Defines parameter for LED on

uint8\_t LED\_off = 0; // Defines parameter for LED off

uint8\_t red\_LED = 14; // Defines parameter for red LED (GPIOD pin 14)

while (1) { // Creates an infinite loop so that the blinking never terminates

// Checks the state of the push-button and only turns the red LED on if the button has only just been pressed, which is indicated by the state of the red LED.

if(((GPIOA->IDR & 0x00000001) == 0x00000001) & ((GPIOD->ODR & (1<<14)) != (1<<14))){

osSignalClear(tid\_Blink\_LED\_Thread,0x01); //Clear flag 0x01 of the blink LED thread so that it resumes

Blink\_LED(LED\_on,red\_LED); // Turn red LED on

}

// Checks the state of the push-button and only turns the red LED off if the button has only just been released, which is indicated by the state of the red LED.

else if(((GPIOA->IDR & 0x00000001) != 0x00000001) & ((GPIOD->ODR & (1<<14)) == (1<<14))){

Blink\_LED(LED\_off,red\_LED); // Turn red LED off

osSignalSet(tid\_Blink\_LED\_Thread,0x01); // Set flag 0x01 of the blink LED thread so that it resumes

}

osThreadYield(); // This function tells the RTOS that when the thread gets to this stage the RTOS should suspend this thread and run the next thread that is ready to run. If there is no other thread ready (which is the case with this simple program since we only have one thread) then the calling thread continues. This function effectively forces the RTOS to reschedule and is useful in more complex systems and scheduling policies.

}

}

**main.c**

/\*-------------------------------------------------------------------------

\* CMSIS-RTOS 'main' function - Blinky

\*-----------------------------------------------------------------------\*/

#define osObjectsPublic // Define objects in main module

#include "osObjects.h" // RTOS object definitions

#include "stm32f4xx.h"

#include "my\_headers.h"

#include "Thread.h"

int main (void) {

osKernelInitialize (); //Initialize CMSIS-RTOS

/\* Initialise any peripherals or system components \*/

// Initialize the LED and button

Initialise\_LED\_and\_button();

/\* Initialise any threads \*/

// Initialise the main thread to blink the LED’s

Init\_Blink\_LED\_Thread();

Init\_Button\_Thread();

osKernelStart (); // start thread execution

while(1){}; // While loop so the program doesn’t terminate

}

# Task 2

Initialisation code for the timer has been added to the function to initialise the processor peripherals and its same changed accordingly. In the Blink\_LED\_thread the osDelays haves been replaced with the WHILE loops that check the timer flag and the osSignalWait and osSignalSet functions have been placed in the WHILE loops so the thread pauses there. Finally the counter is enabled and disabled in the IF statement in Button\_Thread so that the timer pauses while the button is pressed.

**my\_headers.h**

#include "stm32f4xx.h"

void Initialise\_LED\_button\_timer(void); // Declaration for the function to initialise the LED, button and timer

void Blink\_LED(uint8\_t,uint8\_t); // Declaration for the function to blink the LED - now has two inputs: one for the state and one for the colour LED

**my\_headers.c**

#include "stm32f4xx.h"

// Definition for the function to initialise the LED, button and timer

void Initialise\_LED\_button\_timer(void){

// Initialize GPIO Port for LEDs

RCC->AHB1ENR |= RCC\_AHB1ENR\_GPIODEN; // Enable Port D clock

GPIOD->MODER |= GPIO\_MODER\_MODER12\_0; // Port D.12 output - green LED

GPIOD->MODER |= GPIO\_MODER\_MODER14\_0; // Port D.14 output - red LED

//Initialize GPIO for push-button

RCC->AHB1ENR |= RCC\_AHB1ENR\_GPIOAEN; // Enable Port A clock

//Initialize Timer 2

RCC->APB1ENR |= RCC\_APB1ENR\_TIM2EN; // Enable timer 2 clock

TIM2->CR1 &= ~0x00000016; /\*Sets the counter as an upcounter\*/

TIM2->CR1 &= ~0x00000008; /\*Turn on repeat\*/

TIM2->PSC = 8400-1; /\*Prescaler value - the prescaler clock defaults to twice the APB1 which is running at 42MHz - so the timer clock is 84MHz\*/

TIM2->ARR = 10000-1; /\*sets the value in the auto-reload register\*/

TIM2->EGR = 1; /\*Re-initialises the timer\*/

TIM2->CR1 |= 1; //Enables the counter

}

// Definition for the function to blink the LED

void Blink\_LED(uint8\_t LED\_state, uint8\_t LED\_colour){

if(LED\_state == 1){ // Checks to see if the request is to turn the LED on or off

GPIOD->BSRR = 1<<LED\_colour; // Turn on the green LED

}

else{

GPIOD->BSRR = 1<<(LED\_colour+16); // Turn off the green LED

}

}

**Thread.h**

extern int Init\_Blink\_LED\_Thread (void); // Standard format to declare the function to initialise the main thread function.

void Blink\_LED\_Thread (void const \*argument); // Standard format to declare the main thread function.

extern int Init\_Button\_Thread (void); // Standard format to declare the function to initialise the main thread function.

void Button\_Thread (void const \*argument); // Standard format to declare the main thread function.

**Thread.c**

#include "cmsis\_os.h" // CMSIS RTOS header file

#include "stm32f4xx.h"

#include "my\_headers.h"

// Thread Declarations

void Blink\_LED\_Thread (void const \*argument); // Declares the main thread function that is defined later in the code

osThreadId tid\_Blink\_LED\_Thread; // Declares an ID that we will associate with the thread and which allows easy reference to it when using some of the OS functions.

osThreadDef (Blink\_LED\_Thread, osPriorityNormal, 1, 0); // Declares the main thread object that we will use later. The parameters can be used to adjust certain properties, such as the priority of a thread and how many instances of it exist.

void Button\_Thread (void const \*argument); // Declares the main thread function that is defined later in the code

osThreadId tid\_Button\_Thread; // Declares an ID that we will associate with the thread and which allows easy reference to it when using some of the OS functions.

osThreadDef (Button\_Thread, osPriorityNormal, 1, 0); // Declares the main thread object that we will use later. The parameters can be used to adjust certain properties, such as the priority of a thread and how many instances of it exist.

/\*-------------------------------------------------------------------------

\* Blink LED Thread

\*-----------------------------------------------------------------------\*/

// Code to define the thread function to initialise the main thread - this initialise function is called from the “main.c” file to start the thread.

int Init\_Blink\_LED\_Thread (void) {

tid\_Blink\_LED\_Thread = osThreadCreate (osThread(Blink\_LED\_Thread), NULL); // Creates the main thread object that we have declared and assigns it the thread ID that we have declared.

if(!tid\_Blink\_LED\_Thread) return(-1); // Checks to make sure the thread has been created.

return(0);

}

// Code to define the operation of the main thread. This is effectively the code that was in the infinite FOR loop of our previous blinky program.

void Blink\_LED\_Thread (void const \*argument) {

uint8\_t LED\_on = 1; // Defines parameter for LED on

uint8\_t LED\_off = 0; // Defines parameter for LED off

uint8\_t green\_LED = 12; // Defines parameter for green LED (GPIOD pin 12)

osSignalSet(tid\_Blink\_LED\_Thread,0x01);// Set flag 0x01 of the blink LED thread so that it resumes next time wait is called

while (1) { // Creates an infinite loop so that the blinking never terminates

// osSignalWait(0x01,osWaitForever); // Waits until flag 0x01 of this thread is set

// osSignalSet(tid\_Blink\_LED\_Thread,0x01);// Set flag 0x01 of the blink LED thread so that it resumes next time wait is called

Blink\_LED(LED\_on,green\_LED); // Blinks the green LED on once

while((TIM2->SR&0x0001)!=1){ //TIM2 to implement a 1 second delay

osSignalWait(0x01,osWaitForever); // Waits until flag 0x01 of this thread is set

osSignalSet(tid\_Blink\_LED\_Thread,0x01);// Set flag 0x01 of the blink LED thread so that it resumes next time wait is called

};

TIM2->SR &= ~1; //Resets the flag

// osSignalWait(0x01,osWaitForever); // Waits until flag 0x01 of this thread is set

// osSignalSet(tid\_Blink\_LED\_Thread,0x01);// Set flag 0x01 of the blink LED thread so that it resumes next time wait is called

Blink\_LED(LED\_off,green\_LED); // Blinks the green LED on once

while((TIM2->SR&0x0001)!=1){ //TIM2 to implement a 1 second delay

osSignalWait(0x01,osWaitForever); // Waits until flag 0x01 of this thread is set

osSignalSet(tid\_Blink\_LED\_Thread,0x01);// Set flag 0x01 of the blink LED thread so that it resumes next time wait is called

};

TIM2->SR &= ~1; //Resets the flag

osThreadYield(); // This function tells the RTOS that when the thread gets to this stage the RTOS should suspend this thread and run the next thread that is ready to run. If there is no other thread ready (which is the case with this simple program since we only have one thread) then the calling thread continues. This function effectively forces the RTOS to reschedule and is useful in more complex systems and scheduling policies.

}

}

/\*-------------------------------------------------------------------------

\* Red LED on when button pressed Thread

\*-----------------------------------------------------------------------\*/

// Code to define the thread function to initialise the main thread - this initialise function is called from the “main.c” file to start the thread.

int Init\_Button\_Thread (void) {

tid\_Button\_Thread = osThreadCreate (osThread(Button\_Thread), NULL); // Creates the main thread object that we have declared and assigns it the thread ID that we have declared.

if(!tid\_Button\_Thread) return(-1); // Checks to make sure the thread has been created.

return(0);

}

// Code to define the operation of the main thread. This is effectively the code that was in the infinite FOR loop of our previous blinky program.

void Button\_Thread (void const \*argument) {

uint8\_t LED\_on = 1; // Defines parameter for LED on

uint8\_t LED\_off = 0; // Defines parameter for LED off

uint8\_t red\_LED = 14; // Defines parameter for red LED (GPIOD pin 14)

while (1) { // Creates an infinite loop so that the blinking never terminates

// Checks the state of the push-button and only turns the red LED on if the button has only just been pressed, which is indicated by the state of the red LED.

if(((GPIOA->IDR & 0x00000001) == 0x00000001) & ((GPIOD->ODR & (1<<14)) != (1<<14))){

TIM2->CR1 &= ~1; //Disables the counter

osSignalClear(tid\_Blink\_LED\_Thread,0x01); //Clear flag 0x01 of the blink LED thread so that it resumes

Blink\_LED(LED\_on,red\_LED); // Turn red LED on

}

// Checks the state of the push-button and only turns the red LED off if the button has only just been released, which is indicated by the state of the red LED.

else if(((GPIOA->IDR & 0x00000001) != 0x00000001) & ((GPIOD->ODR & (1<<14)) == (1<<14))){

Blink\_LED(LED\_off,red\_LED); // Turn red LED off

TIM2->CR1 |= 1; //Enables the counter

osSignalSet(tid\_Blink\_LED\_Thread,0x01); // Set flag 0x01 of the blink LED thread so that it resumes

}

osThreadYield(); // This function tells the RTOS that when the thread gets to this stage the RTOS should suspend this thread and run the next thread that is ready to run. If there is no other thread ready (which is the case with this simple program since we only have one thread) then the calling thread continues. This function effectively forces the RTOS to reschedule and is useful in more complex systems and scheduling policies.

}

}

**main.c**

/\*-------------------------------------------------------------------------

\* CMSIS-RTOS 'main' function - Blinky

\*-----------------------------------------------------------------------\*/

#define osObjectsPublic // Define objects in main module

#include "osObjects.h" // RTOS object definitions

#include "stm32f4xx.h"

#include "my\_headers.h"

#include "Thread.h"

int main (void) {

osKernelInitialize (); //Initialize CMSIS-RTOS

/\* Initialise any peripherals or system components \*/

// Initialize the LED and button

Initialise\_LED\_button\_timer();

/\* Initialise any threads \*/

// Initialise the main thread to blink the LED’s

Init\_Blink\_LED\_Thread();

Init\_Button\_Thread();

osKernelStart (); // start thread execution

while(1){}; // While loop so the program doesn’t terminate

}

# Task 3

The solution creates two almost identical threads; one for the green LED and one for the red LED. The signal flag for the green LED is initialised to set and for the red LED it is initialised to cleared. Then both threads call wait signal at the start of the loop. This means that the red LED thread will pause in the off state and the green LED thread will proceed and blink on the LED for 1 second. After it turns off it calls set signal for the red LED thread, which cause the red LED thread to proceed. The green LED thread pauses at the start of the loop as we have not set its signal flag. It only proceeds when the red LED thread completes and calls set signal for the green LED thread. By initialising the flags to two different states, always calling wait signal at the start of the loop and set signal for the opposite thread at the end of the loop, the two threads alternately complete one full execution each. It is for this solution that all of the thread identifiers need to be declared at the start of “Thread.c” to ensure that they exist when we refer to them in the file.

**my\_headers.h**

#include "stm32f4xx.h"

void Initialise\_LED(void); // Declaration for the function to initialise the LED, button and timer

void Blink\_LED(uint8\_t,uint8\_t); // Declaration for the function to blink the LED - now has two inputs: one for the state and one for the colour LED

**my\_headers.c**

#include "stm32f4xx.h"

// Definition for the function to initialise the LED, button and timer

void Initialise\_LED(void){

// Initialize GPIO Port for LEDs

RCC->AHB1ENR |= RCC\_AHB1ENR\_GPIODEN; // Enable Port D clock

GPIOD->MODER |= GPIO\_MODER\_MODER12\_0; // Port D.12 output - green LED

GPIOD->MODER |= GPIO\_MODER\_MODER14\_0; // Port D.14 output - red LED

//Initialize GPIO for push-button

RCC->AHB1ENR |= RCC\_AHB1ENR\_GPIOAEN; // Enable Port A clock

}

// Definition for the function to blink the LED

void Blink\_LED(uint8\_t LED\_state, uint8\_t LED\_colour){

if(LED\_state == 1){ // Checks to see if the request is to turn the LED on or off

GPIOD->BSRR = 1<<LED\_colour; // Turn on the green LED

}

else{

GPIOD->BSRR = 1<<(LED\_colour+16); // Turn off the green LED

}

}

**Thread.h**

extern int Init\_green\_LED\_Thread (void); // Standard format to declare the function to initialise the main thread function.

void green\_LED\_Thread (void const \*argument); // Standard format to declare the main thread function.

extern int Init\_red\_LED\_Thread (void); // Standard format to declare the function to initialise the main thread function.

void red\_LED\_Thread (void const \*argument); // Standard format to declare the main thread function.

**Thread.c**

#include "cmsis\_os.h" // CMSIS RTOS header file

#include "stm32f4xx.h"

#include "my\_headers.h"

// Thread Declarations

void green\_LED\_Thread (void const \*argument); // Declares the main thread function that is defined later in the code

osThreadId tid\_green\_LED\_Thread; // Declares an ID that we will associate with the thread and which allows easy reference to it when using some of the OS functions.

osThreadDef (green\_LED\_Thread, osPriorityNormal, 1, 0); // Declares the main thread object that we will use later. The parameters can be used to adjust certain properties, such as the priority of a thread and how many instances of it exist.

void red\_LED\_Thread (void const \*argument); // Declares the main thread function that is defined later in the code

osThreadId tid\_red\_LED\_Thread; // Declares an ID that we will associate with the thread and which allows easy reference to it when using some of the OS functions.

osThreadDef (red\_LED\_Thread, osPriorityNormal, 1, 0); // Declares the main thread object that we will use later. The parameters can be used to adjust certain properties, such as the priority of a thread and how many instances of it exist.

/\*-------------------------------------------------------------------------

\* Green LED Thread

\*-----------------------------------------------------------------------\*/

// Code to define the thread function to initialise the main thread - this initialise function is called from the “main.c” file to start the thread.

int Init\_green\_LED\_Thread (void) {

tid\_green\_LED\_Thread = osThreadCreate (osThread(green\_LED\_Thread), NULL); // Creates the main thread object that we have declared and assigns it the thread ID that we have declared.

if(!tid\_green\_LED\_Thread) return(-1); // Checks to make sure the thread has been created.

return(0);

}

// Code to define the operation of the main thread. This is effectively the code that was in the infinite FOR loop of our previous blinky program.

void green\_LED\_Thread (void const \*argument) {

uint8\_t LED\_on = 1; // Defines parameter for LED on

uint8\_t LED\_off = 0; // Defines parameter for LED off

uint8\_t green\_LED = 12; // Defines parameter for green LED (GPIOD pin 12)

osSignalSet(tid\_green\_LED\_Thread,0x01);// Set flag 0x01 of the blink LED thread so that it resumes next time wait is called

while (1) { // Creates an infinite loop so that the blinking never terminates

osSignalWait(0x01,osWaitForever); // Waits until flag 0x01 of this thread is set

Blink\_LED(LED\_on,green\_LED); // Blinks the green LED on once

osDelay(500);

Blink\_LED(LED\_off,green\_LED); // Blinks the green LED on once

osSignalSet(tid\_red\_LED\_Thread,0x01);// Set flag 0x01 of the red LED thread so that it resumes next time wait is called

osThreadYield(); // This function tells the RTOS that when the thread gets to this stage the RTOS should suspend this thread and run the next thread that is ready to run. If there is no other thread ready (which is the case with this simple program since we only have one thread) then the calling thread continues. This function effectively forces the RTOS to reschedule and is useful in more complex systems and scheduling policies.

}

}

/\*-------------------------------------------------------------------------

\* Red LED Thread

\*-----------------------------------------------------------------------\*/

// Code to define the thread function to initialise the main thread - this initialise function is called from the “main.c” file to start the thread.

int Init\_red\_LED\_Thread (void) {

tid\_red\_LED\_Thread = osThreadCreate (osThread(red\_LED\_Thread), NULL); // Creates the main thread object that we have declared and assigns it the thread ID that we have declared.

if(!tid\_red\_LED\_Thread) return(-1); // Checks to make sure the thread has been created.

return(0);

}

// Code to define the operation of the main thread. This is effectively the code that was in the infinite FOR loop of our previous blinky program.

void red\_LED\_Thread (void const \*argument) {

uint8\_t LED\_on = 1; // Defines parameter for LED on

uint8\_t LED\_off = 0; // Defines parameter for LED off

uint8\_t red\_LED = 14; // Defines parameter for red LED (GPIOD pin 14)

osSignalClear(tid\_red\_LED\_Thread,0x01);// Set flag 0x01 of the blink LED thread so that it pauses next time wait is called

while (1) { // Creates an infinite loop so that the blinking never terminates

osSignalWait(0x01,osWaitForever); // Waits until flag 0x01 of this thread is set

Blink\_LED(LED\_on,red\_LED); // Blinks the red LED on once

osDelay(500);

Blink\_LED(LED\_off,red\_LED); // Blinks the red LED on once

osSignalSet(tid\_green\_LED\_Thread,0x01);// Set flag 0x01 of the green LED thread so that it resumes next time wait is called

osThreadYield(); // This function tells the RTOS that when the thread gets to this stage the RTOS should suspend this thread and run the next thread that is ready to run. If there is no other thread ready (which is the case with this simple program since we only have one thread) then the calling thread continues. This function effectively forces the RTOS to reschedule and is useful in more complex systems and scheduling policies.

}

}

**main.c**

/\*-------------------------------------------------------------------------

\* CMSIS-RTOS 'main' function - Blinky

\*-----------------------------------------------------------------------\*/

#define osObjectsPublic // Define objects in main module

#include "osObjects.h" // RTOS object definitions

#include "stm32f4xx.h"

#include "my\_headers.h"

#include "Thread.h"

int main (void) {

osKernelInitialize (); //Initialize CMSIS-RTOS

/\* Initialise any peripherals or system components \*/

// Initialize the LED and button

Initialise\_LED();

/\* Initialise any threads \*/

// Initialise the main thread to blink the LED’s

Init\_green\_LED\_Thread();

Init\_red\_LED\_Thread();

osKernelStart (); // start thread execution

while(1){}; // While loop so the program doesn’t terminate

}