KETTERING UNIVERSITY
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

CE-426-01

# Programming in RTOS Environment III

Darek Konopka

May 27, 2021

# Table of Contents

## 1. Objectives

- ❖ Use message queues for exchanging data between threads
- ❖ Handle interrupts within RTOS application
- ❖ Use signals for inter-thread communication
- ❖ Use mutex to manage access to shared resource

## 2. Program Source Code

### 2.1 Multiple Threaded Traffic Light

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

    Designers Guide to the Cortex-M Family
    CMSIS RTOS Message Queue Example

*----------------------------------------------------------------------------*/
#include "STM32F10x.h"
#include "cmsis_os.h"
#include "uart.h"
#include "Board_LED.h"

#define  LED_0     0                  //this is connected to GPIOB pin 8
(PORTB.8)
#define  LED_1     1                  // Pin 9
#define  LED_2     2                  // Pin 10

osMessageQId Q_LED;
               //define the message queue
osMessageQDef (Q_LED,0x16,unsigned char);
osEvent  result;

// Calls these from uart.c
volatile uint8_t          inKey;              // input character itself

void override_Thread (void const *argument);

void normal_Thread (void const *argument);
void userInterface_Thread (void const *argument);
void SendText(uint8_t *txt);
void USART1_IRQHandler (void);

osThreadDef(override_Thread, osPriorityNormal, 1, 0);
osThreadDef(normal_Thread, osPriorityNormal, 1, 0);
osThreadDef(userInterface_Thread, osPriorityNormal, 1, 0);

osThreadId T_override_Thread;
osThreadId T_normal_Thread;
osThreadId T_userInterface_Thread;
osThreadId T_main;

osMutexId uart_mutex;
osMutexDef(uart_mutex);

/*----------------------------------------------------------------------------
USART1 IRQ HandlerThe hardware automatically clears the interrupt flag, once the
ISR is entered
*---------------------------------------------------------------------------*/

// Moved from the uart file so I can send signals
void USART1_IRQHandler (void) {

    uint8_t intKey = (int8_t) (USART1->DR & 0x1FF);
    inKey = intKey;
    SendChar(inKey);
    SendChar('\n');

    // here we send a signal to each thread to turn it on
    if(inKey == '1'){
        osSignalSet(T_normal_Thread,0x01);
    } else {
        osSignalSet(T_override_Thread,0x01);
    }
}

/*----------------------------------------------------------------------------
  echos inKey to UART
 *---------------------------------------------------------------------------*/
void userInterface_Thread (void const *argument)
{
    for (;;)
    {
        result =     osMessageGet(Q_LED,osWaitForever);                 //wait for
a message to arrive
        LED_On(result.value.v);                                 // write the
data to the LED's
    }
}
```

```
/*----------------------------------------------------------------------------
  Normal Mode
 *---------------------------------------------------------------------------*/
void normal_Thread (void const *argument)
{
    for (;;)
    {
        // Normal Mode
        if (inKey == '1') {
                     // Turn on the GREEN LED (while RED and YELLOW are off)
                     osMutexWait(uart_mutex, osWaitForever);
                     osMessagePut(Q_LED,0x02,osWaitForever);
                     LED_Off(LED_0);          // Turn off Red Light
                     LED_Off(LED_1);          // Turn off Yellow Light

                     //Wait for 1000 time units
                     osDelay(1000);

                     // Turn on the Yellow LED (while RED and Green are off)
                     osMessagePut(Q_LED,0x01,osWaitForever);
                     LED_Off(LED_0);          // Turn off Red Light
                     LED_Off(LED_2);          // Turn off Green Light

                     //Wait for 250 time units
                     osDelay(250);

                     // Turn on the Red LED (while Green and YELLOW are off)
                     osMessagePut(Q_LED,0x00,osWaitForever);
                     LED_Off(LED_1);          // Turn off Yellow Light
                     LED_Off(LED_2);          // Turn off Green Light
                     osDelay(1000);
                     osMutexRelease(uart_mutex);

        }
    }
}

/*----------------------------------------------------------------------------
  Override Mode and Reset Mode
 *---------------------------------------------------------------------------*/
void override_Thread (void const *argument)
{
    for (;;)
    {
              // Override mode
              if (inKey == '0') {
                     /* While in this mode only the RED LED blinks on and off
                     every 250 time units(while the YELLOW and GREEN are off). */

                     osMutexWait(uart_mutex, osWaitForever);
```

```
                LED_Off(LED_0);           // Turn off Red Light
                LED_Off(LED_1);           // Turn off Yellow Light
                LED_Off(LED_2);           // Turn off Green Light

                osDelay(250);
                // turn on for 250 units
                osMessagePut(Q_LED,0x00,osWaitForever);

                //Wait for 250 time units, then turn back off
                osDelay(250);
                LED_Off(LED_0);           // Turn off Red Light
                osMutexRelease(uart_mutex);     // Turn on Red Light

            // Reset Mode
            } else if (inKey != '0' && inKey != '1' && inKey != '\0'){
                osMutexWait(uart_mutex, osWaitForever);
                LED_Off(LED_0);           // Turn off Green Light
                LED_Off(LED_1);           // Turn off Yellow Light
                LED_Off(LED_2);           // Turn off Red Light
                osMutexRelease(uart_mutex);
                osDelay(250);
        }
    }
}


/*----------------------------------------------------------------------------
  Main: Initialize and start RTX Kernel
 *---------------------------------------------------------------------------*/

int main (void)
{
    osKernelInitialize ();
       // initialize CMSIS-RTOS
    // The following code is from lab 3, and is used for user inout via UART
    LED_Initialize();
    USART1_Init ();//configure USART interrupt ... so we can read user inputs using
interrupt
    uart_mutex = osMutexCreate(osMutex(uart_mutex));
    //Configure and enable USART1 interrupt
    NVIC->ICPR[USART1_IRQn/32] = 1UL << (USART1_IRQn%32);  //clear any previous
pending interrupt flag
    NVIC->IP[USART1_IRQn] = 0x80;//set priority to 0x80
    NVIC->ISER[USART1_IRQn/32] = 1UL << (USART1_IRQn%32);//set interrupt enable bit
    USART1->CR1 |= USART_CR1_RXNEIE;//enable USART receiver not empty interrupt

    LED_Initialize ();
    Q_LED = osMessageCreate(osMessageQ(Q_LED),NULL);
//create the message queue
    T_normal_Thread =    osThreadCreate(osThread(normal_Thread), NULL);
    T_override_Thread = osThreadCreate(osThread(override_Thread), NULL);
    T_userInterface_Thread = osThreadCreate(osThread(userInterface_Thread), NULL);

    osKernelStart ();
       // start thread execution
}

//complete this function for sending a string of characters to the UART
void SendText(uint8_t *text) {

       uint8_t i=0;

       // every string end in \0
       while(text[i] != '\0') {
       SendChar(text[i]);
       i++;
       }
}
```

# 3. Questions

## 3.1 Question 1

From observing the waveform, and reviewing the code for each thread, it is clear to see the operations of the two threads. One thread is used to define and create the message queue along the event structure, and the other thread is used to post the data and receive it. The thread that creates the message queue can be referred to as the main thread, and the other thread can be referred to as the sub thread.

## 3.2 Question 2

Below you will see the Logic Analyzer outputs for each of the operating modes for the traffic light.

*Figure 1 Traffic Light Normal Mode to Override Back to Normal*

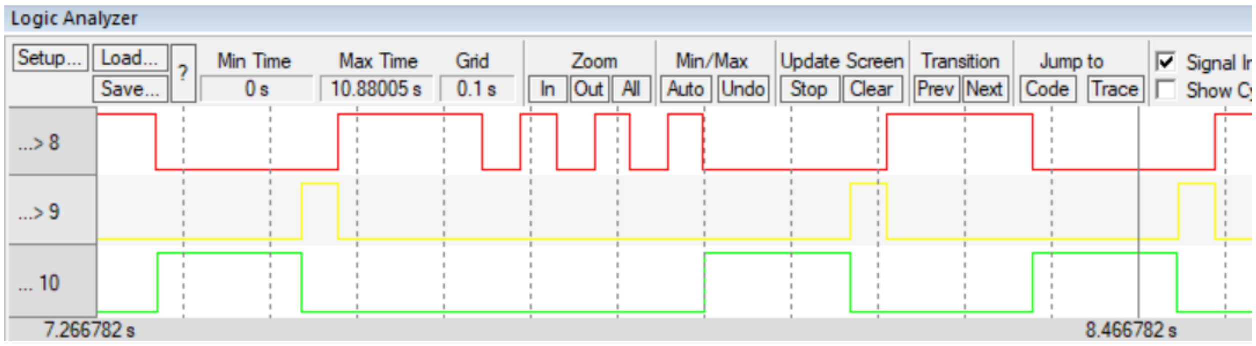
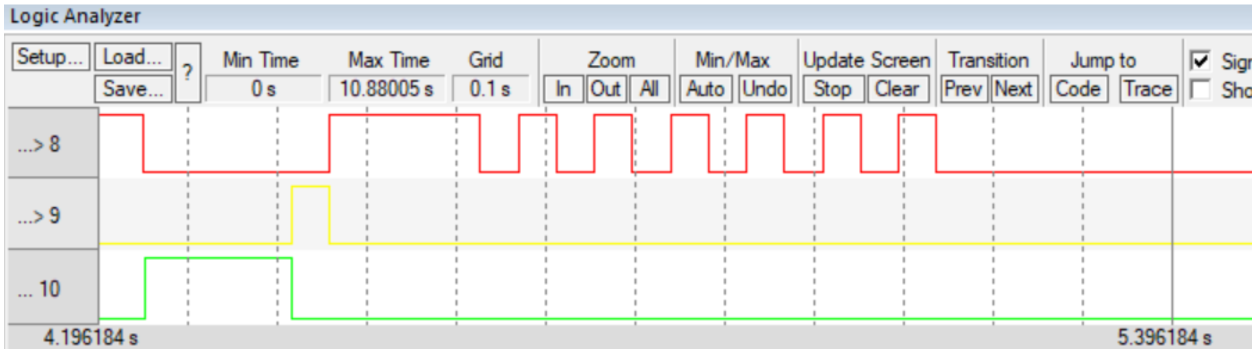
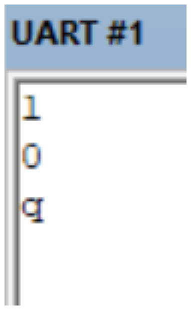


*Figure 2 Traffic Light Normal Mode to Override Mode to Reset Mode*




UART #1

```
1
0
q
```

*Figure 3 Traffic Light UART Terminal*

## 3.3 Question 3

Commenting out the mutexes will cause glitching in the logic analyzer, and will cause threads to not be completed. Examples of this phenomena can be seen below.

*Figure 4 Traffic Light Glitch: Override does not complete*

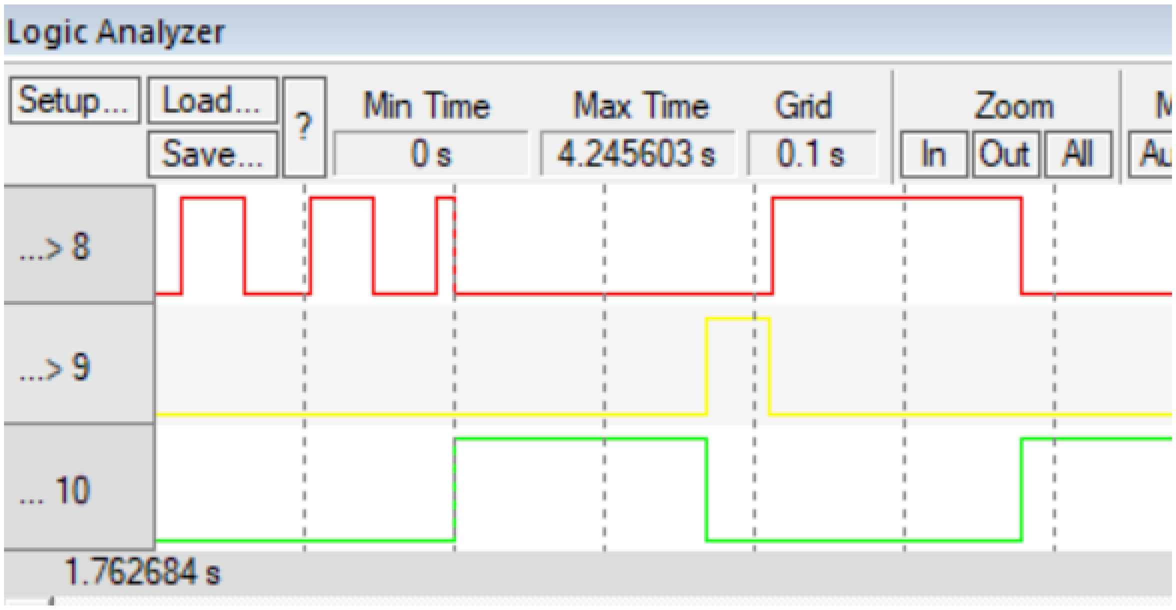
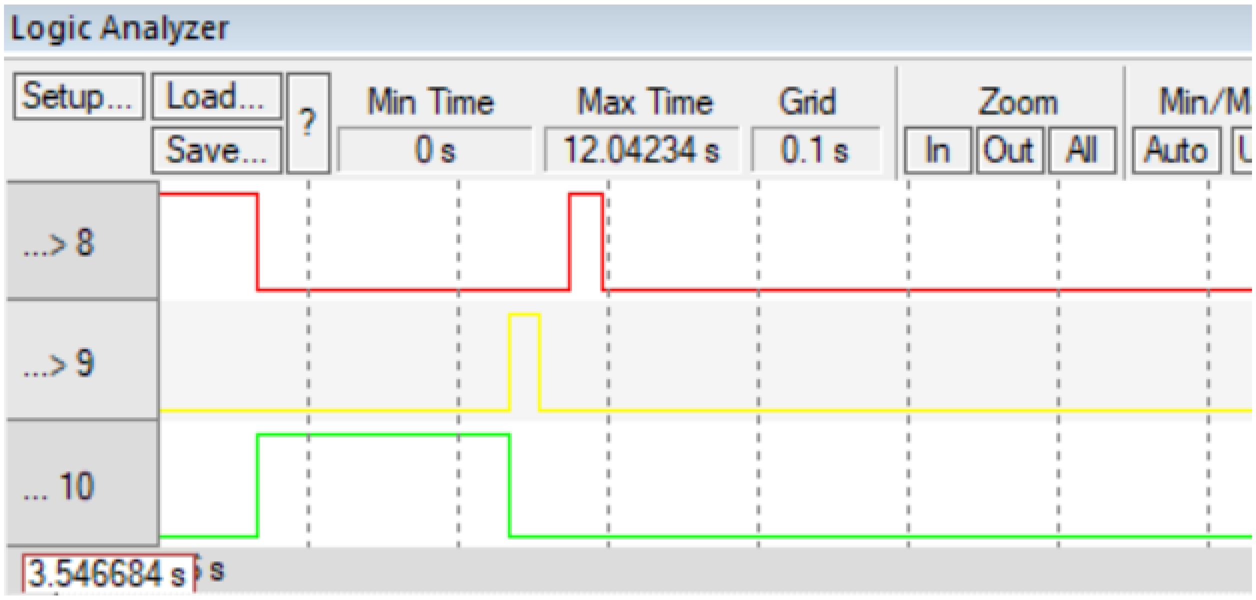


*Figure 5 Traffic Light Glitch: Normal Mode does not complete*



## 3.4 Question 4

The thread activity does change when you apply different inputs into the UART Terminal. When a 1 is input, then the event viewer will prioritize the normal thread, however, when a 0 or anything else is input, then the event viewer will prioritize the override thread.

## 4. Conclusions

Message queueing can be very helpful when using multiple threads and when sending LED commands. It is better to send a full message than just one command at a time since the threads will mix the signals together if messages aren’t used. The use of mutexes is vital because it will prevent glitching and other undesirable effects from occurring.