KETTERING UNIVERSITY
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

CE-426-01

Programming in RTOS Environment

Darek Konopka

May 13, 2021

Table of Contents

1. Objectives

- ❖ Create CMSIS-RTOS project
- ❖ Use uVision tools to monitor the execution of RTOS threads

2. Program Source Code

2.1 main.c of Modified Ex2 and 3 Threads

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

    Designers Guide to the Cortex-M Family
    CMSIS RTOS Threads Example

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

/*----------------------------------------------------------------------------
  Include the microcontroller header for register defenitions and CMSIS core
functions
    Include the CMSIS RTOS header for the RTOS API
 *----------------------------------------------------------------------------*/

#include "STM32F10x.h"
#include <cmsis_os.h>
#include "Board_LED.h"

/*----------------------------------------------------------------------------
  Simple delay routine
 *----------------------------------------------------------------------------*/
void delay (unsigned int count)
{
unsigned int index;

    for(index =0;index<count;index++)
    {
        ;
    }
}

/*----------------------------------------------------------------------------
  Flash LED 1
 *----------------------------------------------------------------------------*/
void led_thread1 (void const *argument)
{

    for (;;)
    {
        LED_On(1);
        osDelay(100);
        LED_Off(1);
        osDelay(100);
    }
}

/*----------------------------------------------------------------------------
 Flash LED 2
 *----------------------------------------------------------------------------*/
void led_thread2 (void const *argument)
{
    for (;;)
    {
        LED_On(2);
        osDelay(200);
        LED_Off(2);
        osDelay(200);
    }
}

/*----------------------------------------------------------------------------
 Define the thread handles and thread parameters
 *----------------------------------------------------------------------------*/

osThreadId main_ID,led_ID1,led_ID2;
osThreadDef(led_thread2, osPriorityNormal, 1, 0);
osThreadDef(led_thread1, osPriorityNormal, 1, 0);

/*----------------------------------------------------------------------------
 Initilise the LED's, get a handle for main, start the threads and terminate main
 *----------------------------------------------------------------------------*/

int main (void)
{
    osKernelInitialize ();                    // initialize CMSIS-RTOS

    LED_Initialize ();
    led_ID2 = osThreadCreate(osThread(led_thread2), NULL);
    led_ID1 = osThreadCreate(osThread(led_thread1), NULL);

    osKernelStart ();                         // start thread execution
    while(1)
    {
            ;
        }
    }
```

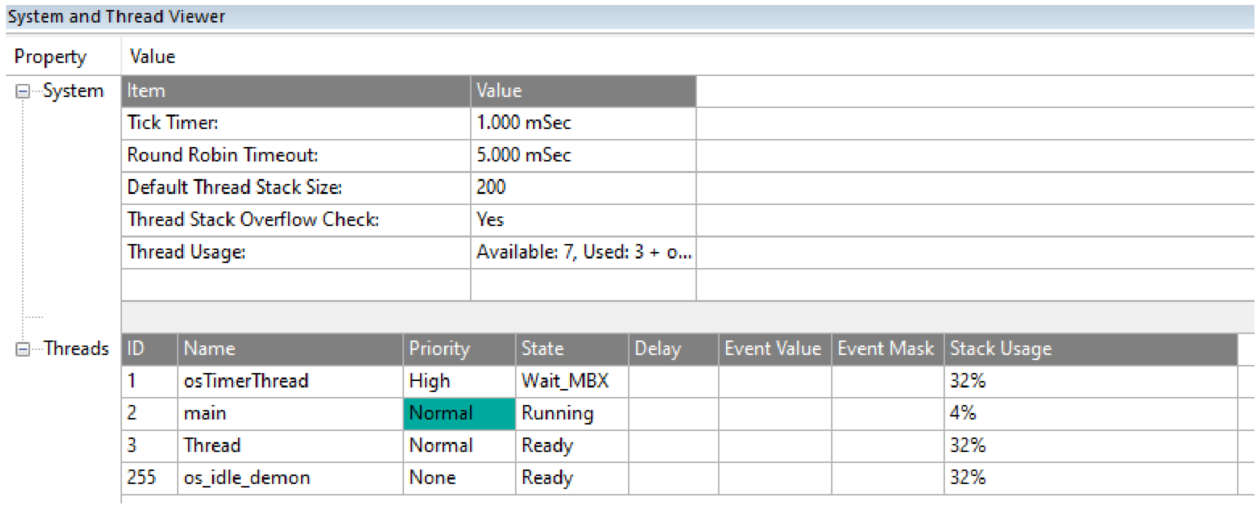
3. Questions

3.1 Question 1

Using the System and Thread Viewer tool observe the threads that are executing in the system once you start running the project. Take a screen capture of your System and Thread Viewer and include it in your report. How many threads are there in the system and describe their states?

In the figure below, you will find that there are 4 threads used in this program.

1. osTimerThread; **State**: Wait_MBX (is waiting).
2. main: **State**: currently running.
3. Thread: **State**: ready to run.
4. os_idle_demon; **State**: ready to run.



System and Thread Viewer

Property | Value

System

Item	Value
Tick Timer:	1.000 mSec
Round Robin Timeout:	5.000 mSec
Default Thread Stack Size:	200
Thread Stack Overflow Check:	Yes
Thread Usage:	Available: 7, Used: 3 + o...

Threads

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Usage
1	osTimerThread	High	Wait_MBX				32%
2	main	Normal	Running				4%
3	Thread	Normal	Ready				32%
255	os_idle_demon	None	Ready				32%

Figure 1: System and Thread Viewer of Task 1

3.2 Question 2

a) **How many threads are created by the system?**

There are a total of 5 threads created by the system. Those threads are

1. osTimerThread
2. main
3. led_thread1
4. led_thread2
5. os_idle_demon

b) How many threads do actually switch between "Ready" and "Running" states? Name the threads

There are three threads that seem to switch between the ready and running states and those threads are:

main, led_thread1, led_thread2

3.3 Question 3

Following the steps you used in Lab 3, open the Logic Analyzer view and display the two PORTB pin values that are controlled by the two threads. Take a screen capture of the waveform that you need to include in your report and explain what you observe from the waveform. Relate the sections of the waveforms for each of the signals to the states of the threads that drive the signals.

As you can see in the figure below, LED numbers 2 will turn on, then off, when that's off, then LED number 1 will turn on, then both will stay off, then the cycle repeats.

This makes sense because if you look in LED_thread 1 it turns LED1 on for 500 time units, then will turn it off for 500 time units, then right after LED_thread 2 is run, and it does the same thing except with LED2.

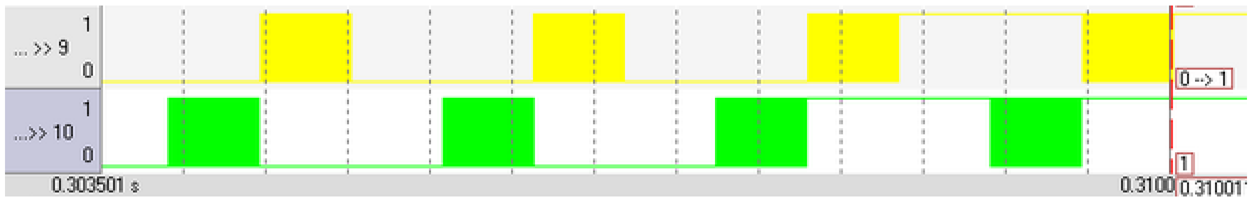


Figure 2: Logic Analyzer of Ex2 and 3 Threads zoomed out

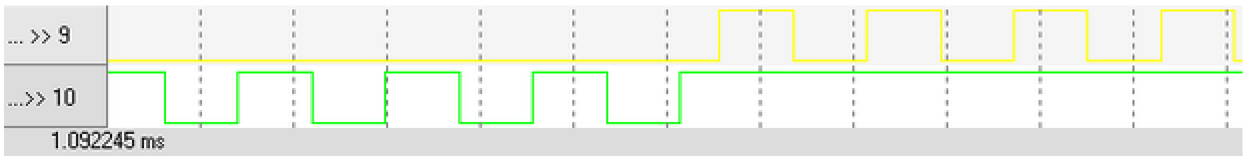


Figure 3: Logic Analyzer of Ex2 and 3 Threads zoomed in

4.4 Question 4

Now replace the delay() function call in thread 1 by osDelay(100),and the delay() in thread 2 by osDelay(200). Build your program, enter debug, and run the application. Next, make observations to each of the following views and take screen captures to include in your report:

-Event Viewer

-System and Thread Viewer

-Logic Analyzer

-General purpose I/O portB

Explain your observation from the above views. How does the change in the delay function affect the scheduling of the threads, specifically the states the threads spend most of their time in and explain why?

Looking closely at the logic analyzer, one can see that the waveform goes from a constant switching (as seen in the original Ex2 Thread 3 code), to a constant 1 or 0 when in the LED_on or off positions. This means the thead now spends more time in the on state, because it is now on for the solid delay time opposed to constantly being switched when in the on state.

All necessary diagrams are below.

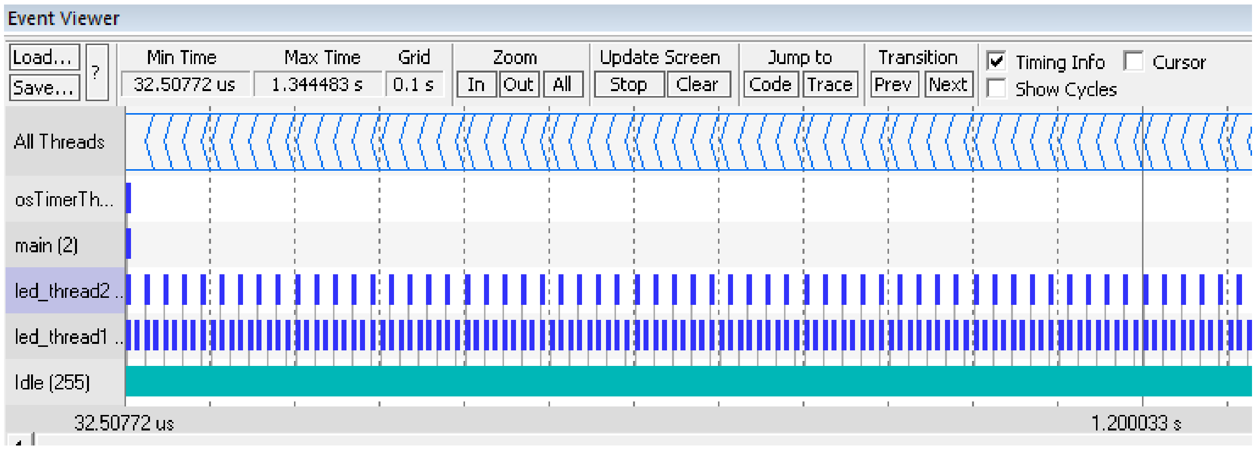
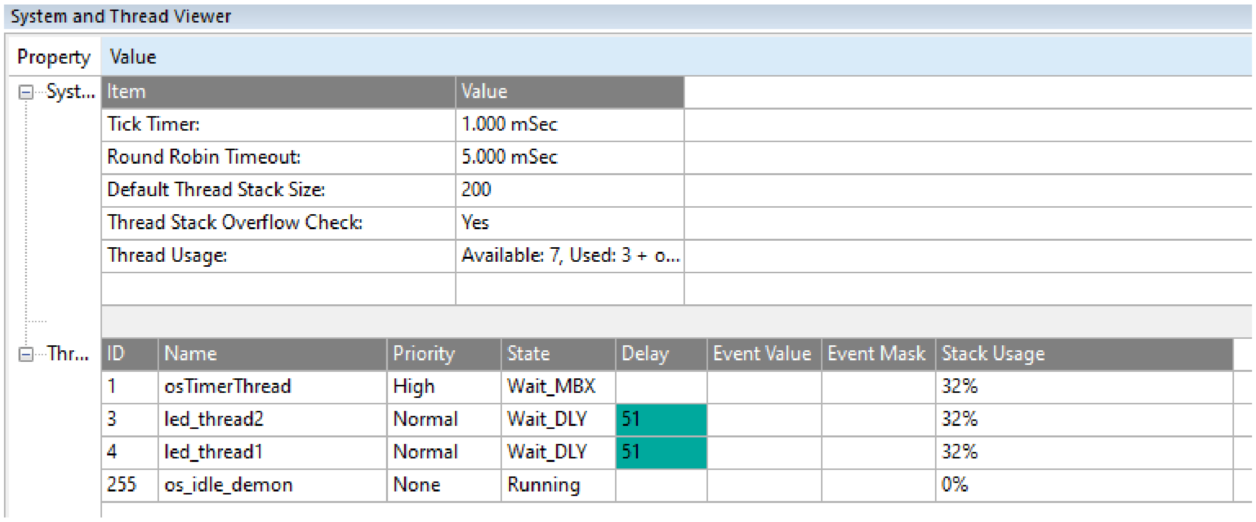


Figure 4: Event Viewer of modified Ex2 and 3 Threads



System and Thread Viewer

Property	Value

Syst...

Item	Value
Tick Timer:	1.000 mSec
Round Robin Timeout:	5.000 mSec
Default Thread Stack Size:	200
Thread Stack Overflow Check:	Yes
Thread Usage:	Available: 7, Used: 3 + o...

Thr...

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Usage
1	osTimerThread	High	Wait_MBX				32%
3	led_thread2	Normal	Wait_DLY	51			32%
4	led_thread1	Normal	Wait_DLY	51			32%
255	os_idle_demon	None	Running				0%

Figure 5: System and Thread Viewer of modified Ex2 and 3 Threads

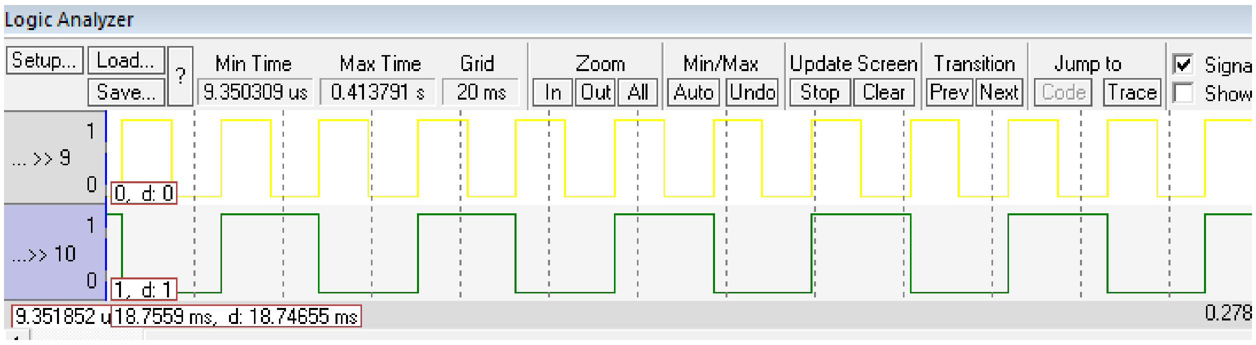
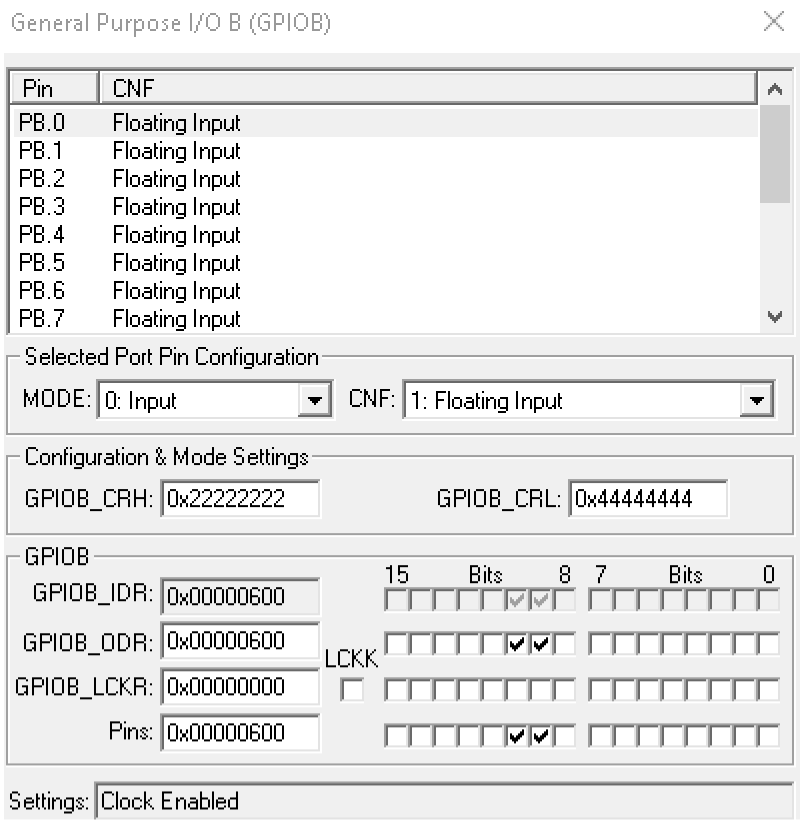


Figure 6: Logic Analyzer of modified Ex2 and 3 Threads



General Purpose I/O B (GPIOB)

Pin	CNF
PB.0	Floating Input
PB.1	Floating Input
PB.2	Floating Input
PB.3	Floating Input
PB.4	Floating Input
PB.5	Floating Input
PB.6	Floating Input
PB.7	Floating Input

Selected Port Pin Configuration

MODE: 0: Input CNF: 1: Floating Input

Configuration & Mode Settings

GPIOB_CRH: 0x22222222 GPIOB_CRL: 0x44444444

GPIOB

GPIOB_IDR: 0x00000600

GPIOB_ODR: 0x00000600

GPIOB_LCKR: 0x00000000

Pins: 0x00000600

Settings: Clock Enabled

Figure 7: General Purpose I/O B of modified Ex2 and 3 Threads

4. Conclusions

In this lab we learned how to work with RTOS threads. The most important thing to point out is the fact that the delay function and the osDelay functions will make the GPIO act differently. The delay function will cause a constant switching between 1 and 0 when in the on position, and the osDelay will just make a solid 1 when in the on position. This is due to the round robin feature of the delay function. The osDelay will allow both LED_Threads to run simultaneously, however, delay() will only allow one to run at a time.