CG2271: Real-Time Operating Systems

Lab 6: RTOS RTX

In this lab, you will be exploring the RTOS RTX. We will learn how to incorporate the RTOS RTX into our project and develop a basic multi-threaded code to understand the basics of how it works.

Lab Report: Remember to submit your lab report (softcopy) to your respective folder by the start of the next lab.

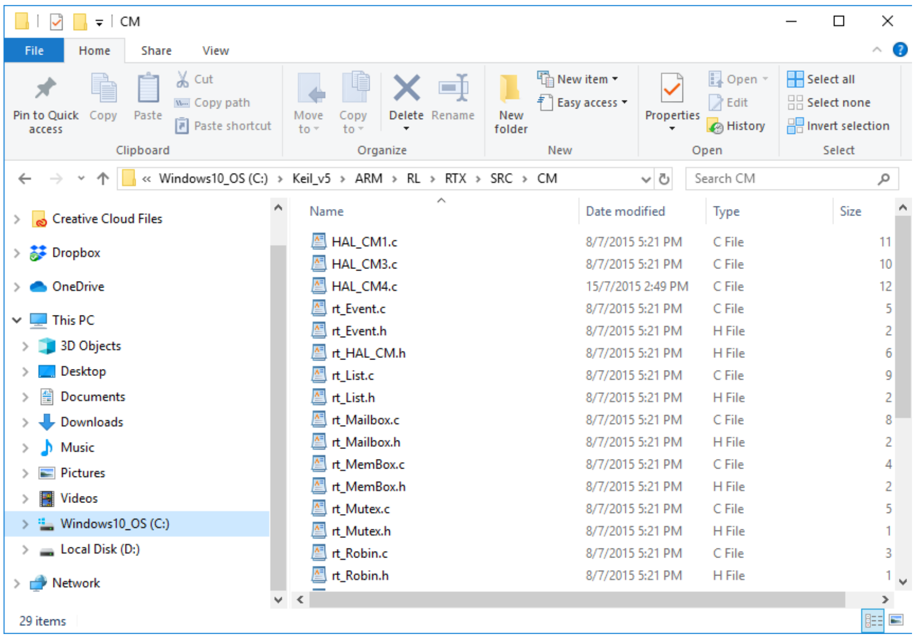
Part1: Creating a RTOX RTX Project

RTX is included by the µVision4 but not directly by theµVision5 so you have to download the legacy support pack for cortex-M devices from: http://www2.keil.com/mdk5/legacy.

You can then always check the source code in:

<<Your KeilDirectory>>\ARM\RL\RTX\SRC

You should be able to see something like this.



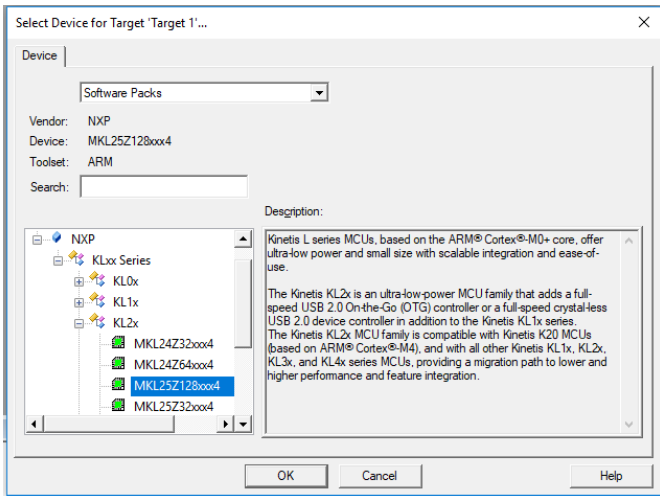
Name	Date modified	Type	Size
HAL_CM1.c	8/7/2015 5:21 PM	C File	11
HAL_CM3.c	8/7/2015 5:21 PM	C File	10
HAL_CM4.c	15/7/2015 2:49 PM	C File	12
rt_Event.c	8/7/2015 5:21 PM	C File	5
rt_Event.h	8/7/2015 5:21 PM	H File	2
rt_HAL_CM.h	8/7/2015 5:21 PM	H File	6
rt_List.c	8/7/2015 5:21 PM	C File	9
rt_List.h	8/7/2015 5:21 PM	H File	2
rt_Mailbox.c	8/7/2015 5:21 PM	C File	8
rt_Mailbox.h	8/7/2015 5:21 PM	H File	2
rt_MemBox.c	8/7/2015 5:21 PM	C File	4
rt_MemBox.h	8/7/2015 5:21 PM	H File	2
rt_Mutex.c	8/7/2015 5:21 PM	C File	5
rt_Mutex.h	8/7/2015 5:21 PM	H File	1
rt_Robin.c	8/7/2015 5:21 PM	C File	3
rt_Robin.h	8/7/2015 5:21 PM	H File	1

Setting up RTX for your project is easy:

a. Create a new Project called myRTX.

Select the Device you are using:

b. Expand NXP then KLxx Series, then KL2x and then select MKL25Z128xxx4 as shown:


Select Device for Target 'Target 1'...

Device

Software Packs

Vendor: NXP
Device: MKL25Z128xxx4
Toolset: ARM
Search:

- NXP
  - KLxx Series
    - KL0x
    - KL1x
    - KL2x
      - MKL24Z32xxx4
      - MKL24Z64xxx4
      - MKL25Z128xxx4
      - MKL25Z32xxx4

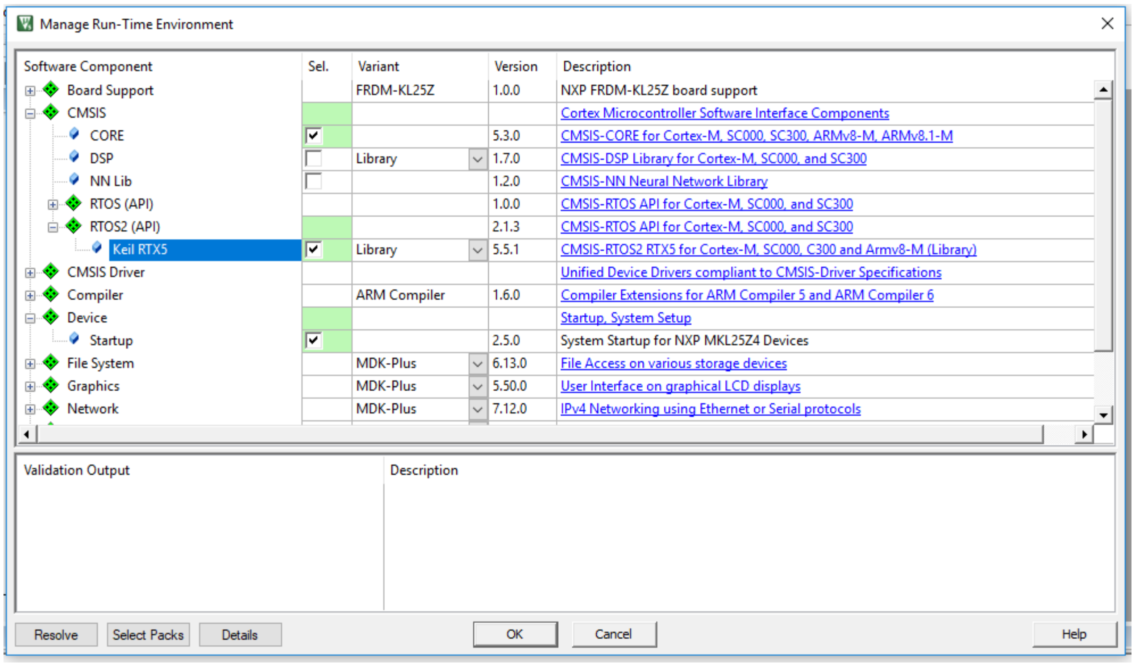
Description:
Kinetis L series MCUs, based on the ARM® Cortex®-M0+ core, offer ultra-low power and small size with scalable integration and ease-of-use.

The Kinetis KL2x is an ultra-low-power MCU family that adds a full-speed USB 2.0 On-the-Go (OTG) controller or a full-speed crystal-less USB 2.0 device controller in addition to the Kinetis KL1x series.
The Kinetis KL2x MCU family is compatible with Kinetis K20 MCUs (based on ARM® Cortex®-M4), and with all other Kinetis KL1x, KL2x, KL3x, and KL4x series MCUs, providing a migration path to lower and higher performance and feature integration.

OK Cancel Help

c. In the next page, select:

- CMSIS->CORE
- CMSIS->RTOS2(API)->Keil RTX5
- Device -> Startup

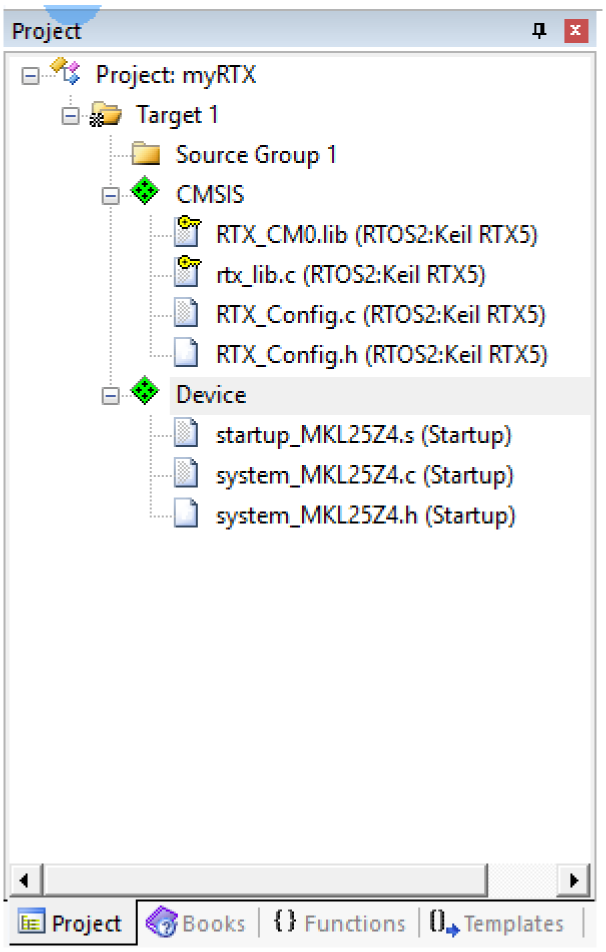

Manage Run-Time Environment

Software Component	Sel.	Variant	Version	Description
Board Support		FRDM-KL25Z	1.0.0	NXP FRDM-KL25Z board support
CMSIS				Cortex Microcontroller Software Interface Components
CORE	✓		5.3.0	CMSIS-CORE for Cortex-M, SC000, SC300, ARMv8-M, ARMv8.1-M
DSP		Library	1.7.0	CMSIS-DSP Library for Cortex-M, SC000, and SC300
NN Lib			1.2.0	CMSIS-NN Neural Network Library
RTOS (API)			1.0.0	CMSIS-RTOS API for Cortex-M, SC000, and SC300
RTOS2 (API)			2.1.3	CMSIS-RTOS API for Cortex-M, SC000, and SC300
Keil RTX5	✓	Library	5.5.1	CMSIS-RTOS2 RTX5 for Cortex-M, SC000, C300 and Armv8-M (Library)
CMSIS Driver				Unified Device Drivers compliant to CMSIS-Driver Specifications
Compiler		ARM Compiler	1.6.0	Compiler Extensions for ARM Compiler 5 and ARM Compiler 6
Device				Startup, System Setup
Startup	✓		2.5.0	System Startup for NXP MKL25Z4 Devices
File System		MDK-Plus	6.13.0	File Access on various storage devices
Graphics		MDK-Plus	5.50.0	User Interface on graphical LCD displays
Network		MDK-Plus	7.12.0	IPv4 Networking using Ethernet or Serial protocols

Validation Output | Description

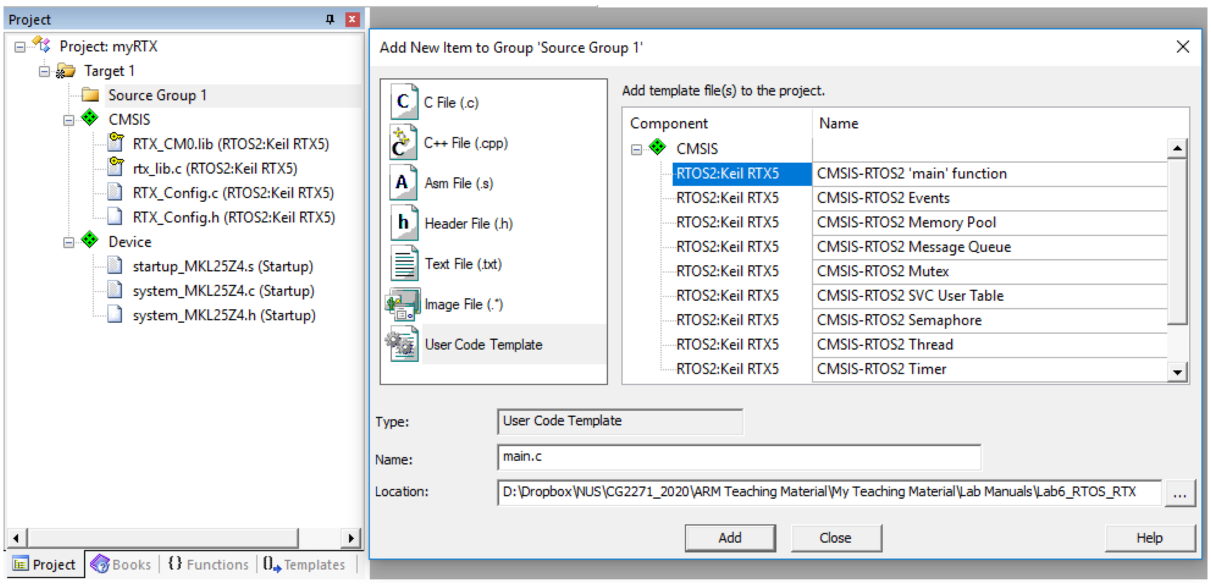
Resolve Select Packs Details OK Cancel Help

d. In your project, you will see that the RTX files have been added to the CMSIS folder as shown here:



```
Project
Project: myRTX
  Target 1
    Source Group 1
    CMSIS
      RTX_CM0.lib (RTOS2:Keil RTX5)
      rtx_lib.c (RTOS2:Keil RTX5)
      RTX_Config.c (RTOS2:Keil RTX5)
      RTX_Config.h (RTOS2:Keil RTX5)
    Device
      startup_MKL25Z4.s (Startup)
      system_MKL25Z4.c (Startup)
      system_MKL25Z4.h (Startup)
```

e. Right-Click 'Source Group 1', select "User Code Template' and choose 'CMSIS->RTOS2 'main' function'.



Add New Item to Group 'Source Group 1'

Add template file(s) to the project.

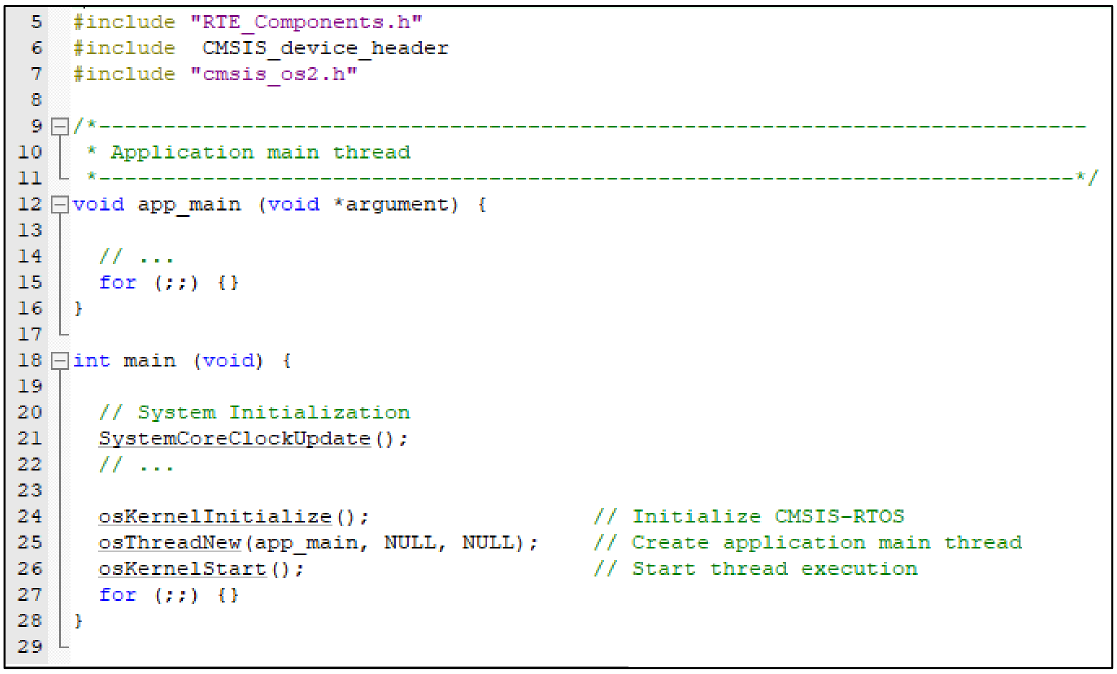
Component	Name
CMSIS	
RTOS2:Keil RTX5	CMSIS-RTOS2 'main' function
RTOS2:Keil RTX5	CMSIS-RTOS2 Events
RTOS2:Keil RTX5	CMSIS-RTOS2 Memory Pool
RTOS2:Keil RTX5	CMSIS-RTOS2 Message Queue
RTOS2:Keil RTX5	CMSIS-RTOS2 Mutex
RTOS2:Keil RTX5	CMSIS-RTOS2 SVC User Table
RTOS2:Keil RTX5	CMSIS-RTOS2 Semaphore
RTOS2:Keil RTX5	CMSIS-RTOS2 Thread
RTOS2:Keil RTX5	CMSIS-RTOS2 Timer

Type: User Code Template

Name: main.c

Location: D:\Dropbox\NUS\CG2271_2020\ARM Teaching Material\My Teaching Material\Lab Manuals\Lab6_RTOS_RTX

f. You will see something like what is shown below:



```
#include "RTE_Components.h"
#include  CMSIS_device_header
#include "cmsis_os2.h"

/*----------------------------------------------------------------------------
 * Application main thread
 *---------------------------------------------------------------------------*/
void app_main (void *argument) {

  // ...
  for (;;) {}
}

int main (void) {

  // System Initialization
  SystemCoreClockUpdate();
  // ...

  osKernelInitialize();                 // Initialize CMSIS-RTOS
  osThreadNew(app_main, NULL, NULL);    // Create application main thread
  osKernelStart();                      // Start thread execution
  for (;;) {}
}
```

LAB REVIEW:

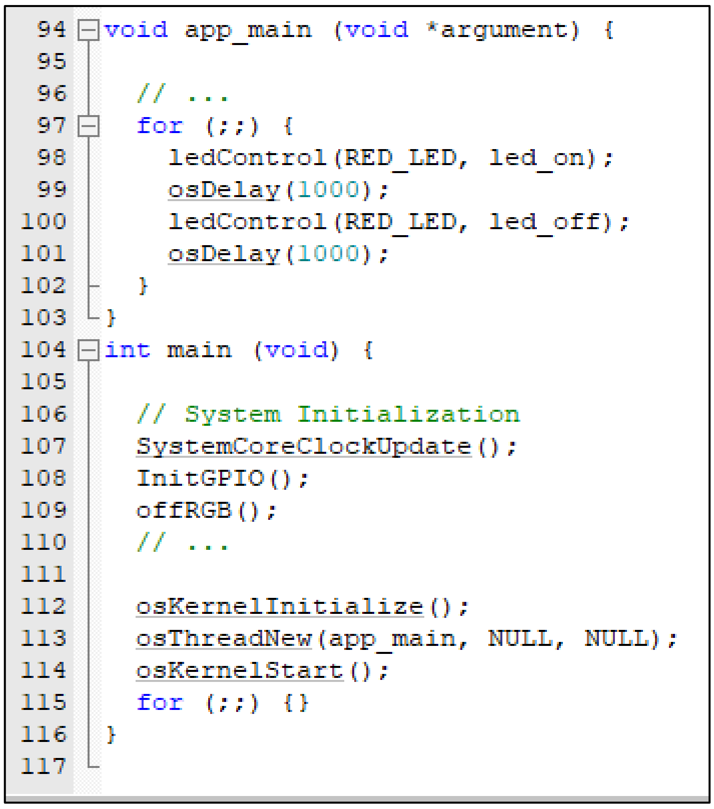
Q1. What is the purpose of the for-loop in the app_main()?

Q2. app_main() is not explicitly called by main(). So when does app_main() get executed?

Q3. At the point of time when osKernelStart() is called, what is the thread/task state of app_main()?

Part2: OS Blinky

Right now, your main OS environment has been set-up in the Keil environment. We will now add all the library functions for the LED control to be able to blink the RGB Led within the app_main. Re-use your own RGB driver code that you have developed. The app_main will look something like this. (The actual GPIO and LED functions will be your own ones)



```
void app_main (void *argument) {

  // ...
  for (;;) {
    ledControl(RED_LED, led_on);
    osDelay(1000);
    ledControl(RED_LED, led_off);
    osDelay(1000);
  }
}
int main (void) {

  // System Initialization
  SystemCoreClockUpdate();
  InitGPIO();
  offRGB();
  // ...

  osKernelInitialize();
  osThreadNew(app_main, NULL, NULL);
  osKernelStart();
  for (;;) {}
}
```

LAB REVIEW:

Q4. When app_main() calls osDelay(), what state does app_main() transition into?

Q5. Instead of the osDelay(), you decide to use the normal loop delay function that you had used in your earlier labs. Is there a difference?

Let’s rename this thread from app_main() to led_red_thread().

LAB REVIEW:

Q6. What are the changes you must make to rename a thread?

Part3: Double Blinky

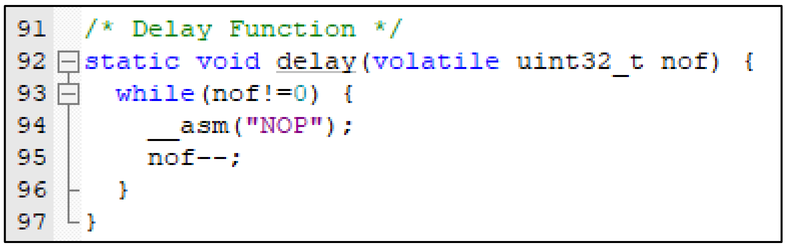
Now we are going to add another thread called led_green_thread(). With what you have done in the earlier parts, you should be able to add a new thread quite easily. The purpose of this thread is to blink the green led at the same 1s rate as the red led.

LAB REVIEW:

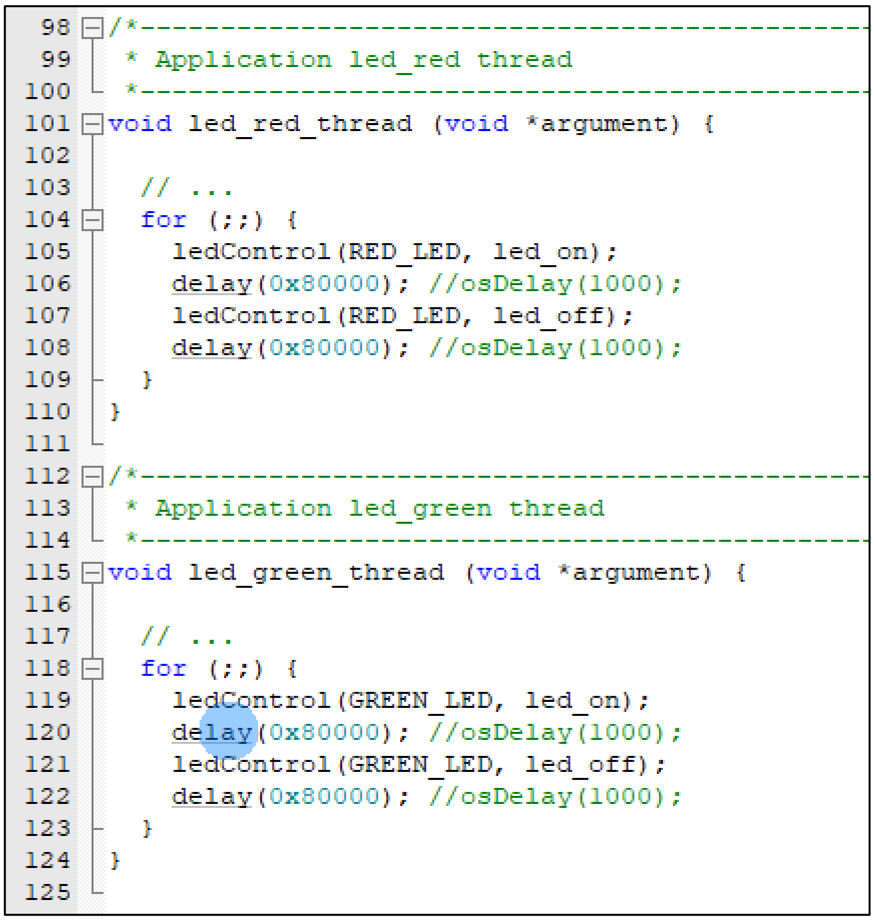
Q7. With both the threads being created in the main(), what is your observation?
Explain why you make such an observation.

Part4: Normal Loop Delay

Let's replace the osDelay() with the normal delay routine. The code will now look something like this:



```
/* Delay Function */
static void delay(volatile uint32_t nof) {
  while(nof!=0) {
    __asm("NOP");
    nof--;
  }
}
```



```
/*----------------------------------------------
 * Application led_red thread
 *----------------------------------------------
void led_red_thread (void *argument) {

  // ...
  for (;;) {
    ledControl(RED_LED, led_on);
    delay(0x80000); //osDelay(1000);
    ledControl(RED_LED, led_off);
    delay(0x80000); //osDelay(1000);
  }
}

/*----------------------------------------------
 * Application led_green thread
 *----------------------------------------------
void led_green_thread (void *argument) {

  // ...
  for (;;) {
    ledControl(GREEN_LED, led_on);
    delay(0x80000); //osDelay(1000);
    ledControl(GREEN_LED, led_off);
    delay(0x80000); //osDelay(1000);
  }
}
```

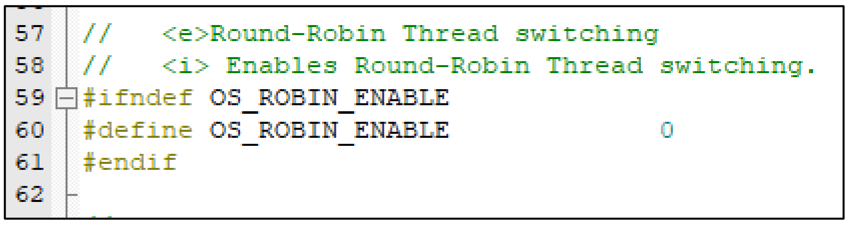
With this change, the led_red_thread, doesn't give up the CPU. As such, we would expect that the green_led_thread doesn't get a chance to run. The expected behaviour would be that we only observe the red led blinking.

Compile and download your code.

LAB REVIEW:

Q8. Is your observation and your expectation the same?

In your Keil IDE, open the RTX_config.h file. It can be seen from the panel on the left. You will see that the OS_ROBIN_ENABLE is set to 1. Change it to 0.



```
//    <e>Round-Robin Thread switching
//    <i> Enables Round-Robin Thread switching.
#ifndef OS_ROBIN_ENABLE
#define OS_ROBIN_ENABLE             0
#endif
```

Save the File, Recompile and download the code. If you are unable to save the file due to a write-protect issue, close the IDE and reopen it. If it still doesn't work, open the file in a normal text editor and make the change.

LAB REVIEW:

Q9. Explain your observation now. What was the significance of the change to the OS_ROBIN_ENABLE.

Switch back the definition of OS_ROBIN_ENABLE to 1.

Recompile and Download your code to observe the behaviour you saw earlier.

Summary

In this lab, you were introduced to the RTX RTOS and learnt how to create your own RTOS-based project. Using osDelay() is a very crude way of switching between tasks. We will be learning more efficient ways for tasks to signal one other so that they can coordinate and share data to fulfil more complex requirements.