CG2271: Real-Time Operating Systems

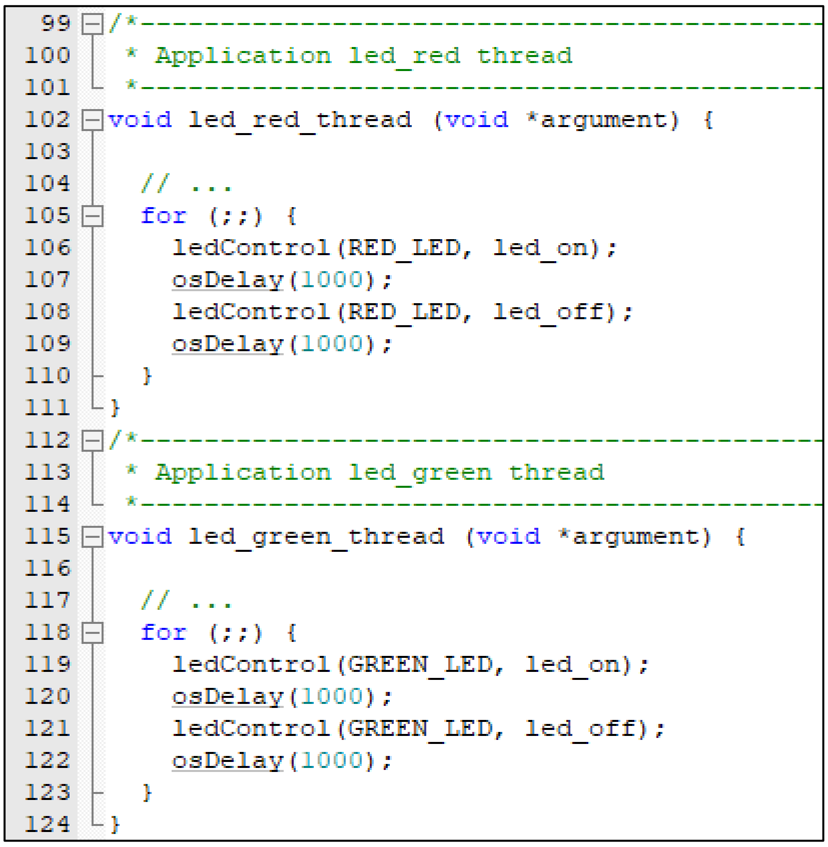
Lab 9: Communications

In this lab, you will explore TWO different ways of synchronizing between threads, the Thread Flags and the Thread Events. After that, we will explore how threads can pass data to each other in the form of Message Queues.

Part1: Creating a new RTOX RTX Project

Refer back to Lab 6 if you have forgotten this step. Call this project myComms. Once the project has been created copy over the code from Lab 6 with both the red_led_thread and green_led_thread. We are not going to use any Mutex or Semaphore in this first part. Compile and download the code. Confirm that you observe that the RGB led blinks as Yellow with a 1s interval.

The code snippet is shown below.



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

  // ...
  for (;;) {
    ledControl(RED_LED, led_on);
    osDelay(1000);
    ledControl(RED_LED, led_off);
    osDelay(1000);
  }
}
/*----------------------------------------------------------------------------
 * Application led_green thread
 *---------------------------------------------------------------------------*/
void led_green_thread (void *argument) {

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

```
int main (void) {

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

  osKernelInitialize();                 // Initialize CMSIS-RTOS
  osThreadNew(led_red_thread, NULL, NULL);    // Create application led_red thread
  osThreadNew(led_green_thread, NULL, NULL);  // Create application led_green thread
  osKernelStart();                      // Start thread execution
  for (;;) {}
}
```

Part2: Cool Blue

Extend the current code to include a new thread led_blue_thread. Let the code for the led_blue_thread be the same as led_red_thread except that its going to control the BLUE LED. Compile and Download your code.

You should observe that the LED lights up as WHITE. By now it should be obvious that all three threads will be running concurrently due to the effect of both the osDelay() as well as the round robin nature of the RTX.

***** DO NOT STARE DIRECTLY AT THE LED AS IT CAN DAMAGE YOUR EYESIGHT. *****

Part3: Thread Flags

We will now use Thread Flags to control the sequence of the led sequence. For this, we need to know the Thread_Id of each of the threads in the system. The Thread_Id is the return value whenever you call osThreadNew().

Step 1:

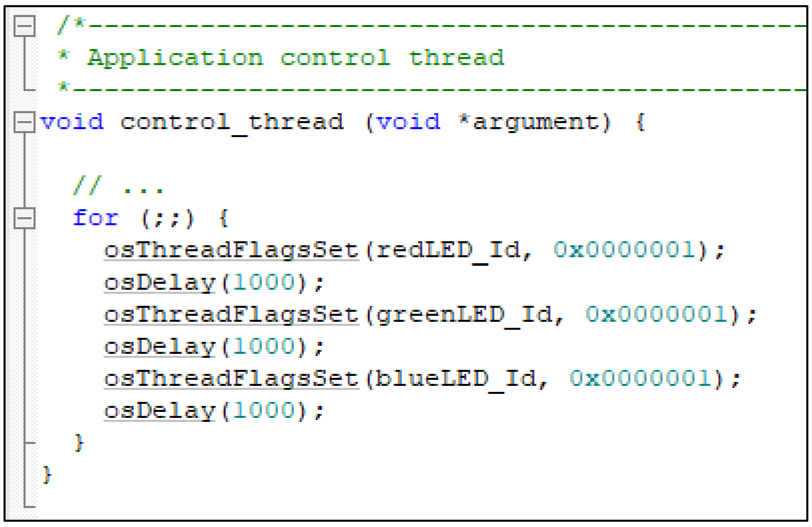
Modify the code in main() to capture the Thread_Id of each of the threads for the different LED's. The Thread_Id datatypes should be declared as Global. Create a new control_thread in the main() that will control the led sequence. The code snippet will now look like this:

```
osThreadId_t redLED_Id, greenLED_Id, blueLED_Id, control_Id;
```

```
int main (void) {

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

  osKernelInitialize();                 // Initialize CMSIS-RTOS
  redLED_Id = osThreadNew(led_red_thread, NULL, NULL);    // Create application led_red thread
  greenLED_Id = osThreadNew(led_green_thread, NULL, NULL);  // Create application led_green thread
  blueLED_Id = osThreadNew(led_blue_thread, NULL, NULL);  // Create application led_blue thread
  control_Id = osThreadNew(control_thread, NULL, NULL);
  osKernelStart();                      // Start thread execution
  for (;;) {}
}
```

Step 2:

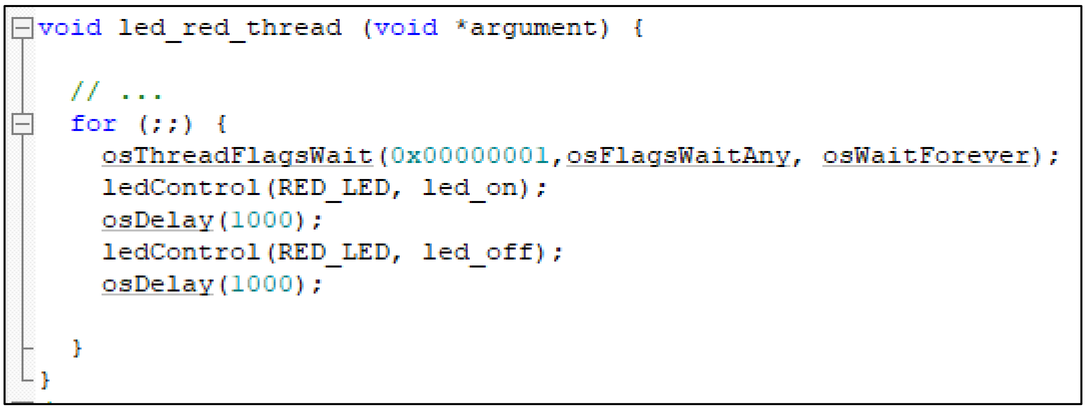
The control_thread will control the sequence of the LED's by setting the appropriate flag for each of the threads separately.



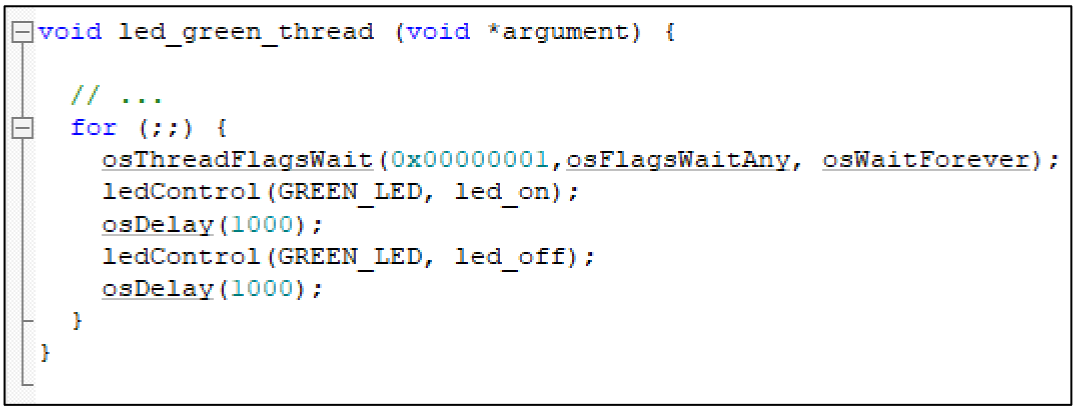
```
/*----------------------------------------------------------------------------
 * Application control thread
 *---------------------------------------------------------------------------*/
void control_thread (void *argument) {

  // ...
  for (;;) {
    osThreadFlagsSet(redLED_Id, 0x0000001);
    osDelay(1000);
    osThreadFlagsSet(greenLED_Id, 0x0000001);
    osDelay(1000);
    osThreadFlagsSet(blueLED_Id, 0x0000001);
    osDelay(1000);
  }
}
```

Step 3:

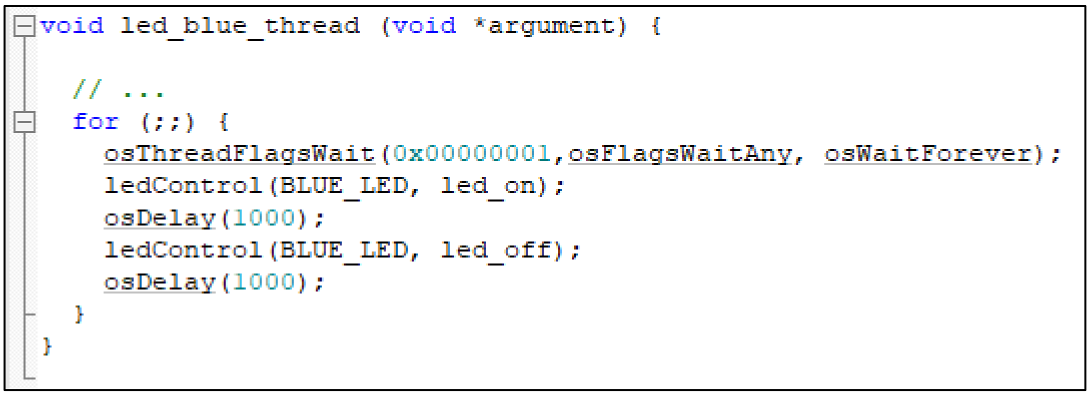
Modify the code in the THREE led threads to wait for the flag to be set before proceeding. The code snippet of the threads is shown below.



```
void led_red_thread (void *argument) {

  // ...
  for (;;) {
    osThreadFlagsWait(0x00000001,osFlagsWaitAny, osWaitForever);
    ledControl(RED_LED, led_on);
    osDelay(1000);
    ledControl(RED_LED, led_off);
    osDelay(1000);

  }
}
```



```
void led_green_thread (void *argument) {

  // ...
  for (;;) {
    osThreadFlagsWait(0x00000001,osFlagsWaitAny, osWaitForever);
    ledControl(GREEN_LED, led_on);
    osDelay(1000);
    ledControl(GREEN_LED, led_off);
    osDelay(1000);
  }
}
```



```
void led_blue_thread (void *argument) {

  // ...
  for (;;) {
    osThreadFlagsWait(0x00000001,osFlagsWaitAny, osWaitForever);
    ledControl(BLUE_LED, led_on);
    osDelay(1000);
    ledControl(BLUE_LED, led_off);
    osDelay(1000);
  }
}
```

LAB REVIEW

Q1. The osThreadFlagsSet() has TWO input parameters. Explain what they are in the context of the control_thread that has been given.

Q2. The osThreadFlagsWait() has THREE parameters. Explain what they are in the context of the red_led_thread that has been given.

Compile and download the code to your board.

LAB REVIEW

Q3. Explain the sequence of the different states of the threads till you see the first LED light up.

Part4: Event Flags

Achieve the same outcome as in Part3, but now, you are to use only Event Flags. The LED's must go in the sequence of RED->GREEN->BLUE and back again. Refer back to the Lecture or online documentation on how to use Event Flags.

LAB REVIEW:

Demonstrate your working code to the Lab TA. You must clearly show that you are using only Event Flags for this part in order to get the marks.

Part5: Message Queues

In this section you are going to explore the ways of creating a Message Queue in RTX and use it to synchronize and send data between tasks. You can refer to the e-Lecture for a detailed explanation on how this can be achieved.

LAB REVIEW:

Demonstrate your working code to the Lab TA. You must clearly show that you are using only Message Queues for this part in order to get the marks.

Part 6: Advantage of Message Queues

The main advantage of Message Queues is that they allow synchronization and data transfer between different tasks at the same time. With the current implementation in Part 5, you get the same behaviour as in the earlier parts with Thread Flags and Event Flags. We are now going to modify the data fields for the different messages to provide different type of LED behaviour. An example of the updated data field is shown below.

CMD	DATA			
	0x01	0x02	0x03	0x04
0x01: RED	0x01: Blink with 1s ON and 1s OFF (Loop Iteration: 1) 0x02: Blink with 0.5s ON and 0.5s OFF (Loop Iteration: 2) 0x03: Blink with 0.25s ON and 0.25s OFF (Loop Iteration: 4) 0x04: Blink with 0.125s ON and 0.125s OFF (Loop Iteration: 8)			
0x01: GREEN				
0x01: BLUE				

Update the control_thread in your program to send data packets with different data fields to the different led_threads.

LAB REVIEW:

Demonstrate your working code to the Lab TA. You must clearly show that the different coloured LED's can blink at different rates ONLY because of the difference in the data packets being sent to them.

Summary

In this lab you have learnt 3 new tools of multi-threaded programming, Thread Flags, Event Flags and Message Queues. With all that you have learnt over the past few weeks, you are now well-equipped to handle the requirements of the mini-project. Good Luck! ☺