# Department of Electrical and Computer Engineering

# ECE 5710/6710 Lab 5

Title: Periodic Tasks Scheduling using a Priority-Driven RTOS

Objective: The student should be able to implement an application for a Real-Time Operating System (RTOS) that schedules periodic tasks.

Parts: 1-EFM32GG Evaluation Board

Software: Silicon Laboratories Simplicity Studio.

Preparation: Write the title and a short description of this laboratory exercise in your lab book. Make sure the page is numbered and make a corresponding entry in the table of contents.

You are to write a FreeRTOS application to schedule the following four tasks:

| Task | Period/Relative Deadline | Execution Time ($e_i$) |
|---|---|---|
| A | 3 ms | 0.5 ms |
| B | 5 ms | 0.9 ms |
| C | 6 ms | 1.1 ms |
| D | 11 ms | 2.4 ms |

The code that executes the jobs for each of these tasks is provided in the file tasks5.o. To execute a job for a particular task, you need only to call task_N() where N ∈ {A, B, C, D}. (Perhaps task_N() should have been called job_N(), but that is water under the bridge at this point.) Note that all of these functions are preemptable.

In most real-time operating systems, tasks are implemented by functions that loop forever and never return. The task loop waits for the release of a job, then executes the job and repeats.

If a task is periodic, the task loop can be written so that the operating releases each job. Typically, real-time operating systems provide a function that causes the task to sleep or delay until a specified amount of time has elapsed. The task loop is then configured as shown below:

```
loop
    sleep until the release time of the next job
    execute the job
end loop
```

In FreeRTOS, periodic tasks are best released using a function called vTaskDelayUntil(). This function takes, as parameters, the previous wake-up time and a delay. It then causes the (calling) task to sleep until the delay (relative to the previous wake-up time) has elapsed. As a side-effect, it updates the previous wake-up time parameter. This function is ideal for releasing periodic tasks; the task loop has the basic form:

```
loop
    vTaskDelayUntil(&prev_release_time, task_period);
    execute the job
end loop
```

There is a hazard that occurs in FreeRTOS where a task that has been created will start before other tasks exist. To avoid the timing difficulties associated with this hazard, it is best to have all tasks suspend until, say, 10 milliseconds after reset before executing the first time. Here is one way to structure a task loop to do that:

```
#include <FreeRTOS.h>
#include <task.h>

TickType_t startTime = pdMS_TO_TICKS(10);
…

void TaskA(void *params)
{
    const int period = pdMS_TO_TICKS(3); // period = 3ms
    TickType_t prevWakeTime = 0;
    (void) params; // suppress warning

    // wait for first release
    vTaskDelayUntil(&prevWakeTime, startTime);
    for (;;)
    {
        task_A(); // perform actual task
        vTaskDelayUntil(&prevWakeTime, period);
    }
}
```

Write a program that uses FreeRTOS to schedule Task A through Task D. In your main() function you should call xCreateTask once for each task before calling vStartScheduler(). Use a rate-monotonic priority assignment.

Create a new C project and name it Lab5 (or something the fits your naming convention). Add your program and tasks5.o to your project (using the technique described in labs 2-4).

In order make your project work with FreeRTOS, you will have to modify it. The easiest way to do this is to leverage the second project you did in Lab 1 (hereafter referred to as “blinky”). If you have deleted this project, refer to Lab 1 to rebuild it.

You will need to perform the following 3 steps (in no particular order):

1. Copy the FreeRTOSConfig.h file from the config folder in blinky to the config folder in Lab5.

2. Copy the util folder from the gecko_SDK_x.x.x folder in blinky to the gecko_SDK_x.x.x folder in Lab5

3. Right-click on the Lab5 project and select select “Properties”. Under “C/C++ Build” select Settings. Then under Tool Settings, select “Includes” from GNU ARM C Compiler. Add the following include paths:

```
${StudioSdkPath}/platform/CMSIS/RTOS2/Include
${StudioSdkPath}/util/third_party/freertos/cmsis/include
${StudioSdkPath}/util/third_party/freertos/kernel/include
${StudioSdkPath}/util/third_party/freertos/kernel/portable/GCC/ARM_CM3
```

Click Apply and Close. Build your project and correct any build errors.

Note: the timer starts at 0, and assuming you don’t change FreeRTOSConfig.h, the timer tick frequency is 1kHz.

Procedure: Launch the debugger to download your code. Run it and verify both amber LEDs light up. If not, the debugging tools that you used in Lab 2 are still available (e.g. get_time, early_task, etc.).

When your program works, demonstrate it to your lab instructor.

Print a copy of your code and affix it into your lab book. Write a short summary or conclusion and submit your lab book to your lab instructor for grading. (Remember to initial and date!)

Points will be assigned according to the rubric below:

| Criterion | Points |
|---|---|
| Lab book properly bound and kept in ink | 1 |
| Lab book contains a title and short description | 1 |
| Each page initialed and dated, large areas crossed out | 1 |
| Pages are legible with no obliterations | 1 |
| Lab book contains schedulability calculations | 1 |
| Lab book contains scheduler and context switch code | 2 |
| Program works | 3 |

Late work is penalized 2 points (20%).