Department of Electrical and Computer Engineering

ECE 5710/6710 Lab 7

Title: Releasing Tasks using Interrupts and Semaphores

Objective: The student should be able to implement tasks that are released via an interrupt service routine.

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	30 ms	10 ms	7 ms
B	40 ms	20 ms	9 ms
C	60 ms	60 ms	15 ms
D	110 ms	110 ms	24 ms

The code that executes the jobs for each of these tasks is provided in the file tasks7.o. To execute a job for a particular task, you need only to call task_N() where N ∈ {A, B, C, D}. Note that all of these functions are preemptable.

Prove that the deadline monotonic (DM) schedule for these tasks is feasible. Record your proof in your lab book.

In most real-time operating systems, tasks can be released as a result of an interrupt. This is usually done using semaphores. The task loop continually attempts to “take” an (initially empty) semaphore and each time that it succeeds, it executes a job. Meanwhile, each time an interrupt is raised, the ISR “gives” the semaphore, thereby freeing the task loop to run when the scheduler deems it appropriate. The task loop is very simple and is illustrated below.

```
loop
    take the semaphore associated with this task
    execute a job
end loop
```

Create a new FreeRTOS project, either as you did in Lab 5 or by (a) creating a new FreeRTOS Blink project (with a project name like Lab7), and (b) replacing main.c, blink.* and app.* in the project folder with your own program files.

Go to the Lab 7 assignment page in canvas and download tasks7.o. Move it into your project folder. Right-click on your project name and select “Properties”. Under “C/C++ Build” select Settings. Then under Tool Settings, select Miscellaneous from the GNU ARM Linker. Add “${ProjDirPath}/tasks7.o” to the “Other objects” box and click both “Apply” and “Ok”.

In your main source file, create four task loops (as described above), one each for tasks A, B, C and D. (You may need to consult online documentation to determine how to create and take binary semaphores in FreeRTOS.).

Now, while it would be fun to have actual external interrupt sources release the tasks, that would be very difficult to test. Instead, you are to create or modify the function vApplicationTickHook() (which is called from the system tick ISR). Modify it so that it gives all four semaphores the first time it is called, then gives task A’s semaphore once every 30 interrupts thereafter. Note: use xSemaphoreGiveFromISR(). Do the same for the other tasks, but of course use their proper periods.

Note: you will need to modify your copy of FreeRTOSConfig.h (in the config folder) to define configUSE_TICK_HOOK to be 1. Otherwise vApplicationTickHook() will never be called.

Build your code and see to it there are no errors.

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 3 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 proof of schedulability	1
Lab book contains program code	1
Program works	4

Late work is penalized 2 points (20%).