

# Department of Electrical and Computer Engineering

## ECE 5710/6710 Lab 2

Title: Scheduling Loops

Objective: The student should be able to implement a scheduling loop.

Parts: 1-EFM32GG Evaluation Board

Software: Silicon Laboratories Simplicity Studio.

Preparation: Write the title and a short description for 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 program to schedule the following 4 tasks in such a way that they all meet their deadlines

Task	Period	Execution Time	Relative Deadline
A	3 ms	0.5 ms	3 ms
B	5 ms	0.9 ms	5 ms
C	6 ms	1.1 ms	6 ms
D	15 ms	1.9 ms	15 ms

Create a new C project in simplicity studio called Lab2 (or some other name that fits your naming convention). Download the object file "tasks2.o" from canvas and put it in your project folder.

The file tasks2.o contains functions that perform the tasks above (and consume very close to the execution time given). These functions are called task\_A(), task\_B(), etc. You will need to include the following declarations in your main.c (or app.c) file in order to access these functions:

```
extern void task_A(), task_B(), task_C(), task_D();
```

These functions turn on LED0 and LED1 and keep them on so long as the tasks (a) start after they are released and (b) finish before their deadlines. (Note: this fails after about 80 minutes because the timer overflows.)

tasks2.o also provides functions that indicate whether or not each task is ready to execute, so you will need to include the declarations below as well:

```
extern int task_A_ready(), task_B_ready(),
        task_C_ready(), task_D()_ready;
```

These function return non-zero if a task is ready to run or zero otherwise. Write your scheduling loop to run each task when (and only when) it is ready. (Hint: The naïve approach of checking each task in turn will not work for this set of tasks. You will need to find a more sophisticated scheduling loop that will.) Show (mathematically) in your lab book that all tasks will meet their deadlines.

The IDE needs to know where to find tasks2.o so it can link it into the project. 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 C Linker. Add “\${ProjDirPath}/tasks2.o” to the “Other objects” box and click both “Apply” and “Ok”. Build your code and make sure there are no errors.

**Procedure:** Download your code to the EFM32GG and run it. If you have scheduled the tasks correctly, LED0 and LED1 will light and stay lit. If not, you will need to debug your code. To help in the debugging process, a function called `get_time()` has been provided that will return the time (in  $1/875^{\text{ths}}$  of a millisecond) since the first task was scheduled. You can call it before and after you schedule a task to see exactly when it is executed. To use `get_time()`, you will have to declare it as follows in your code:

```
uint32_t get_time(void);
```

Additionally, two hooks (function pointers) have been provided so you can stop your code the instant a task is scheduled incorrectly (either (a) because it is scheduled before its release time or (b) because it was scheduled late or was not scheduled at all).

To use the hooks, you must declare them in your code:


```
extern void (*early_task)(void); // task scheduled before its release
extern void (*late_task)(void);  // task scheduled late or not at all
```

You must also define a function to associate with each hook. For example,

```
static void early()
{ // Use the debugger to examine t and find the time of the error.
  uint32_t t = get_time();
} // set your breakpoint here.
```

Somewhere early in main.c you will need to assign your function to the hook, e.g.

```
early_task = early;
```

Once an error is detected and your code stops, you should be able to use the call stack (in the debug window) to determine exactly which task was running. (Be careful, it will highlight the line after the errant task is called.) You can also use the “Step Return” feature of Simplicity Studio (  ) to work back to your scheduler and find where it went wrong.

When your scheduling loop works, demonstrate it to your lab instructor. He will have you break your code and run it again to verify the LEDs do not remain lit.

Print a copy of your code and affix it into your lab book. Write a 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 scheduling loop code	1
Program works	4

Late work is penalized 2 points (20%).