Department of Electrical and Computer Engineering

ECE 5710/6710 Lab 10

Title: Instrumentation

Objective: The student should be able to instrument code and measure task execution times.

Parts: 1-EFM32GG Evaluation Board

Software: Silicon Laboratories Simplicity Studio.

Discussion: While it is sometimes necessary to disassemble code and count instructions in order to compute task execution times, often such rigor is unnecessary. For soft real-time systems or hard real-time systems where the cost of failure is low or moderate, a rigorous analysis is too costly (and must be repeated for the slightest code change). Another approach to analyzing a real-time system is code instrumentation, which provides a way to monitor or measure the performance of the system.

Although sophisticated tools exist that trace every memory access and allow post facto analysis of the system without code modification, a far simpler and economical approach is to add instructions to the code that report or time certain events. For example, an unused I/O port in conjunction with an oscilloscope or logic analyzer can allow us to monitor when a task is released or completes, or an unused timer can be employed to measure execution or critical section times.

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.

In this laboratory exercise, you will schedule the following tasks and determine their maximum execution and critical section times. You will also find the time necessary to perform a context switch with your scheduler.

Task	Period	Relative Deadline	Execution Time (e_i)
A	5 ms	5 ms	?
B	125 ms	20 ms	?
C	35 ms	35 ms	?
D	50 ms	50 ms	?

Make a copy of your code from lab 9 but use the file tasks10.o instead. Modify your code so the system tick occurs every 5 milliseconds. (Task A will be released each tick, task B will be released every 25th tick and so on.) Make sure the code runs as before.

Once your code behaves as it did in lab 9, you are ready to instrument it. We will first employ an unused timer (Timer 3) on the EFM32GG to measure execution times. To use this timer we must first enable it by placing the following statement early in your main() function:

```
CMU->HFPERCLKEN0 |= 0x2100; // enable clock to GPIO & TIMER3
```

(We will use GPIO later so we may as well enable it now too.) Second, the timer control register needs to be initialized. The following statement configures the clock source of the timer to be 14MHz/8 = 1.75 MHz. Consult the EFM32 reference manual for more information on the control register.

```
TIMER3->CTRL = 0x03000000; // Divide by 8, 1.75 MHz count rate
```

Now, the point of this timer is to measure a task's execution time, not its completion time, so we want the timer to run only when the task is scheduled. To do that, we will need an extra word in the TCB that indicates that a task is using the timer. Add the following member to the TCB struct:

```
typedef struct
{   uint32_t *stack_pointer;
    uint32_t timer3_on;
    …
```

Make sure it is the second member in the struct. To start the timer (when a job commences) you should execute the following statements (consider putting them in a function for convenience):

```
    TIMER3->CNT = 0;                // reset timer to 0
    CurrentTask -> timer3_on = 1;   // written to CMD when scheduled
    TIMER3->CMD = 1;                // start the timer
```

Then, when the job concludes, execute the statements below (again, consider a function).

```
    time_measured = TIMER3->CNT;
    CurrentTask -> timer3_on = 0;
    TIMER3->CMD = 2; // stop timer
```

(Note: timer3_on should be initialized to 0 for all tasks.)

To find the actual time, you will need to divide time_measured by the frequency of the timer source (e.g. 1.75 MHz).

This code will work satisfactorily for task A because it is never preempted. To make it work for the other tasks, you must pause the timer on a context switch and resume it when the task is scheduled again. One way to do this is to add the instructions shown below right after the push and right before the pop in your context switch:

```
.equ TIMER3_CMD,      0x40010C04

    push   {r4-r11, lr}
    ldr    r8,=TIMER3_CMD          // timer 3 command register
    ldr    r0,=2                   // stop timer command
    str    r0,[r8]                 // timer 3 now stopped
    …
    ldr    r1,[r0,#4]              // get timer3_on value
    str    r1,[r8]                 // start timer 3 if appropriate
    pop    {r4-r11, pc}
```

Now, you should be able to perform 4 separate experiments to measure the time spent in each of the four tasks. For now, instrument task A only (i.e. write code to start the timer when task A starts and read it when task A finishes). Remember that we are interested in the maximum execution time, so plan to make many measurements and keep a running maximum you can inspect using the debugger.

Another time measurement we seek is the context switch time. To make this measurement we will need to use a different technique.

Pins 0-5 of port D are unused and available in the row of 22 pads on one side of the board. We will instrument the code so each pin PD1-PD4 is associated with one task A-D and will be driven high when that task is running. PD5 will be driven high during a context switch.

To do this, we will need another member in the TCB that represents the output you want on port D when that task is running. Call it task_mask and make it the third member of the structure. You will want to initialize task_mask when you create the task.

Add the following code in your main function to configure PD0-PD5 to be outputs:

```
GPIO->P[gpioPortD].MODEL          // set pins 0-5 to mode 4 (output)
    = (GPIO->P[gpioPortD].MODEL & ~0x0FFFFFF) | 0x00444444;
```

Because not all of port D is unused, we will have to be careful about setting bits. After stopping timer 3 in context.s and before restarting it, add the code fragments shown below

```
.equ GPIO_PD_DOUTCLR, 0x40006080

.   // Stop Timer 3
 …
    ldr     r9,=GPIO_PD_DOUTCLR // Port D “clear” register
    ldr     r10,=0x1f           // clear bits 0-4
    str     r10,[r9]            // and clear
    add     r9,#-4              // point to Port D “set” register
    ldr     r10,=0x20           // set bit 5 (context switch bit)
    str     r10,[r9]            // and it is set
    …
    // save sp into current TCB, call the scheduler, etc.
    …
    ldr     r1,[r0,#8]          // get task_mask from current TCB
    str     r1,[r9]             // set task_mask bits on port D
    add     r9,#4               // point to port D “clear” register
    str     r10,[r9]            // clear bit 5 (context switch bit)
    …
    // possibly start timer 3
```

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. Wait for several seconds then stop your code and inspect the running maximum of the execution time of task A. It should be in the neighborhood of 1ms. If your results deviate significantly from this, you will need to debug your code.

Record the execution time (in timer ticks and milliseconds) for task A in your lab book.

Move the instrumentation in turn to task B, C and D. Remember the execution time counts everything after the task is released until it waits again, including taking and releasing semaphores, outputting to the LCD, etc. Repeat the process of running your code several seconds then inspecting the maximum execution time. Record all your results.

Now use the same technique to find the length of the critical sections that guard the LCD. (You will get separate results for each task B, C and D). Record the durations of these critical sections in your lab book (in both timer ticks and milliseconds).

Use the oscilloscope to monitor the port pin associated with task A. Compare the execution time recorded with your observations using the oscilloscope.

For fun, move the oscilloscope to the pins associated with tasks B, C and D to see how they are scheduled

Move the oscilloscope to pin 5 and use it to measure the maximum time of the context switch. Record this time in your lab book.

Show the oscilloscope output to your lab instructor.

Print a copy of your instrumented 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 recorded execution times (ticks & ms)	2
Lab book contains scheduler and context switch code	2
Program works	2

Late work is penalized 2 points (20%).