EE475 Lab #8 Fall 2004

MicroC/OS-II on the HC12

This week we will develop a real time project using the MicroC/OS-II software ported to run on the HC12. The kernel features required to create tasks, manage their execution, and communicate among the tasks will be investigated. There are literally dozens of other features provided by the μC/OS-II kernel: you should feel free to experiment as much as you like!

Preliminaries

1. Make a temporary local folder for your work: `c:\EEClasses\EE475\tempxxx` .
2. Copy the Lab8 project folder from the class web site to your temp directory.
3. Launch Code Warrior and open the Lab8 project.
4. Take a look at the `main.c` file and determine what it does.
5. Make the `main.c` program. Note that there will be lots of warnings, but there should be no errors. Launch the debugger, and try the code. Does it work as expected?
6. *NOTE that because the code uses the free-running timer it is likely that the debugger will lose control of the EVB when you stop the program.* You will probably need to press the hardware RESET switch on the EVB and re-launch the debugger software.
7. Look through the various project `.c` and `.h` files to answer the following questions:

How many tasks can be supported with the current `#define` settings? _____

Which of the eight available free-running output compare timers is used for the μC/OS-II clock tick? _____

How many kernel clock ticks per second are there? _____

Exercise #1: Creating MicroC tasks

The existing program creates a single task that toggles LED 0 once per two seconds. Now modify the program so that it has 3 (three) tasks and implements the following features:

(a) Task 1: modify `myTask1()` to toggle LED 0 once every 500 milliseconds.
(b) Task 2: create a task that toggles the state of LED 1 once every 2 seconds.
(c) Task 3: create a task that toggles the state of LED 2 once every 20 *clock ticks*.

Be sure to set up the stack space required for each task, and call the proper startup functions. Use the MicroC functions such as `OSTaskCreate()`, `OSTimeDly()`, and `OSTimeDlyHMSM()`.

→ ***Demonstrate and explain the results of this exercise for the instructor.***

Exercise #2: Using a semaphore to coordinate tasks

Edit your `main()` program to define a μC/OS-II *semaphore*.

The semaphore must be used as follows:

- Modify Task 3: inside the infinite loop, have the task *pend* on the semaphore. When the semaphore is received, toggle the state of LED 2, *post* the semaphore, and then delay for 200 milliseconds before *pend*ing again.

- Add a new task, Task 4, with the following behavior: *pend* on the semaphore. When the semaphore is received, start a loop to monitor switch #7 using `while(PORTAD&0x80){…}`. Note that the task will enter the while loop if switch #7 is OFF. Inside the loop, force LED 2 OFF, toggle the state of LED 3, then delay for 10 clock ticks. If switch #7 is flipped, thereby exiting the while loop, force LED 3 off, post the semaphore, and delay for one clock tick before pending on the semaphore again.

(Task 1 and Task 2 are unaltered from Exercise #1.)

You will also need to verify that the processor registers are set correctly so that you can read the toggle switch position.

Exercise #3: A User ISR and a Mailbox

To investigate another inter-process communication method, create a μC/OS-II *mailbox* and use it to communicate between an interrupt service routine and another task.

- Create the mailbox.
- Write and install an interrupt service routine for the IRQ pushbutton. The ISR should simply *post* a message to the mailbox. Be sure to set the IRQ for *edge-triggered* operation.
- Create a new task, Task 5, that pends on the mailbox. Once the mailbox message is received the task should toggle the state of LED 4, delay for one clock tick, then pend again on the mailbox.
- The other tasks and semaphore should be unaltered from Exercise #2.

→ ***Demonstrate and explain the mailbox and semaphore controls for the instructor.***

Extra: can you pass some information via the mailbox pointer mechanism?

Extra: can you write a generic LED flash task, but then pass in which LED and what flash period via the `pdata` pointer from `OSTaskCreate()` ?

Instructor Verification Sheet
Lab #8 Fall 2004

Student Name: _____

	Instructor Signature	Date
Ex. #1 Three μC/OS-II tasks running simultaneously		
Ex. #3 Semaphore and mailbox operating properly		

Lab Report

The lab report is to be written up in the Memo format. Be sure to put the *lab number* in the Memo header along with your name and date. For each exercise, answer the given questions and demonstrate your understanding of the exercise. Include **commented** file excerpts and this instructor verification sheet to get credit for the lab.

→ This lab report is due by 5PM next week on Tuesday, November 16.