ELECTRICAL ENGINEERING PROGRAM

California Polytechnic State University

2011 by Bryan Mealy Modified 2013 by Bridget Benson

CPE 439 Experiment 3

Learning Objectives

1. FreeRTOS
   - To become familiar with the AVR port for the FreeRTOS kernel
   - To become familiar with some basic objects and service in the FreeRTOS kernel
   - To become familiar with binary tasks, binary semaphores, and timing services and interrupts in an RTOS environment.

Reference Documents

The following documents are available on Polylearn:

- FreeRTOS Real Time Kernel: A Practical Guide
- FreeRTOS Reference Manual: API Functions and Configuration Options

Introduction and Overview

The RTOS Kernel offers a relatively small handful of services that allow you to build a highly responsive real-time embedded system. Although there are not really that many services out there, the do take some getting used to as they do represent a departure from the typical embedded designs that you are used to up to this point. While the previous experiment introduced some basic tasks and timing applications, this experiment introduces some other services offered by the RTOS kernel.

Semaphores have two basic functions: they can either act as signaling mechanisms from tasks to tasks or from ISRs to tasks (binary semaphores and counting semaphores), or they can control safe access to shared resources such as shared data (mutex semaphores). This experiment introduces the concept binary semaphores as a signaling device. Also, you'll probably run into shared data issues. When you do, you'll need to handle them properly; because if you don't the free world will most likely collapse because your blinking LEDs may be off by a few hundred microseconds. There are several ways to handle simple shared data issues, one of them is with mutex semaphores.

In essence, this experiment represents a slight and a major modification of Experiment 1. The slight modification is the use of the real-time response of the button-press. The major modification is the implementation of this experiment using an RTOS. This is going to be one of those experiments that are no big deal, once you do it. But before that point, it may be a head scratcher. By the way, if there is right way to do this experiment, I'm not sure what it is. You have a few RTOS services at your disposal and a nice Guide and Reference Manual. Be sure to take a look. The really important issue here is that you use semaphores as they are intended to be used. The semaphore themselves are rather versatile in what they can do, but good real-time embedded programming practices means you should be using them as they are intended to be used. This is pretty cool stuff; be sure to have fun.

Programming Assignment:

Create an RTOS-based application that will:

- React to events associated with SW7
  - If SW7 is pressed:
    - LED2 blinks at approximately 2 Hz (use a task)
    - LED7 blinks at 7 Hz (using a Timer2 interrupt)
    - LED6 is on
  - If SW7 is not pressed:
    - LED2 blinks at approximately 4 Hz (use a task)
    - LED7 blinks at 14Hz (using a Timer2 interrupt)
    - LED6 is off

Constraints

- Use an external interrupt for SW7. Make sure it is properly debounced without using dumb delay loops.
- Implement a deferred interrupt handler scheme, where each of your interrupt service routines (for the button and for Timer 2) gives control to a handler task through the use of a semaphore. Make sure you drop the following line in your FreeRTOSConfig.h file:

```
#define configUSE_MUTEXES  1
```

Questions

1. Describe how you handled debouncing the button. Does your method compromise system response time? Why or why not?
2. What is the purpose of using a deferred interrupt handler scheme?

Deliverables

For this experiment, please provide the following.

1. Standard Lab Write Up. Your perfectly commented and structured c and h file. Answers to the questions and individual conclusions.
2. Demonstrate your program to me.