ELECTRICAL ENGINEERING PROGRAM
California Polytechnic State University
2011 Bryan Mealy, revised 2013 Bridget Benson

CPE 439 Experiment 2

RTOS Introduction

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

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

If you're going to do anything complicated out there in embedded systems land, you're going to need an RTOS. If you don't use one, your application is going to be "fragile"[1]. Creating applications with RTOSes is not that big of a deal, once you've done it a few times. It's those first few times on the steep part of the learning curve that is problematic.

This experiment is an introduction the AVR GCC port of the FreeRTOS. I've provided a basic file structure that has been stripped down from the full download from FreeRTOS.org. Note – you are also welcome to download the source directly from FreeRTOS.org. The directory I provide you includes the following 3 folders:

1. Source: contains the basic files for the generic Kernel and specific files for the ATMegaxx port

2. FirstProject: contains the most basic RTOS application I can think of designed within AVR Studio 6. This is a good starting point for any RTOS application.

3. License: Holds the GNU Public license for Free use of the code

Programming Assignment:

Create an RTOS-based application that will:

- Blink LED0 at approximately 1 Hz
- Blink LED2 at approximately 2 Hz
- Blink LED4 at approximately 4 Hz
- Blink LED6 at approximately 8 Hz

[1] You'll someday know what fragile means, if you don't already. It will be the day you demonstrate your company's product that is packed full of firmware that ***works most of the time***. You'll know how to fix it, but fixing it will require that you work 30 hours/day for the next 300 weeks straight.

Constraints

- Define no more than one task
- The tasks should not use dumb delays; they should only use timing APIs offered by the FreeRTOS Kernel.
- Note – you can set parameters for your RTOS in FreeRTOSConfig.h

Questions

1. What rate did you set your tick interrupt to? Why? Is there a rate that would not be suitable to ensure proper operation of your program? (you can test out different rates with your program to observe the behavior)
2. Why does FreeRTOSConfig.h allow you to include or exclude API functions in your RTOS?

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.