MTE 241 Lab 3 -- Multiple Threads

Bernie Roehl, Spring 2020

The goal of this lab is to familiarize you with the use of multiple execution threads (i.e. tasks).

Learning Objectives

After successfully completing this lab, you should be able to:

- Create and use multiple threads of execution
- Perform I/O with a number of peripheral devices simultaneously

Overview

This lab builds on the work you did in lab 3, and adds the additional requirement that you must use multiple concurrent tasks.

You'll be using the Keil RTX5 real-time kernel. This is an implementation of the standard CMSIS RTOS2 interface, which is widely used in embedded systems. More information about the Keil RTOS2 implementation can be found at:

https://www.keil.com/pack/doc/CMSIS/RTOS2/html/index.html

Implementing Multiple Threads

The purpose of a multi-tasking operating system is to allow several things to be done at the same time.

To use the multi-threading capabilities of the Keil RTX5, you need to enable that feature in the Run-Time Services. You also need to include the appropriate header file in your code:

```
#include <cmsis_os2.h>
```

Each of your threads should be declared as follows:

```
void whatever(void *arg) {
      // ... your code in here ...
}
```

In RTOS2, each thread is just a function that takes one parameter and doesn't return anything. The body of each thread will typically be an infinite loop, using something like `while(1)` or `for(;;)`, and you must call some method in your loop that will allow the thread to give up

control to the thread scheduler (e.g. `osThreadYield()` or `osDelay(), or waiting on a mutex`).

You will typically start your threads from your `main()` function. Somewhere in your `main()` function you should have the following:

```
osKernelInitialize();
osThreadNew(whatever, NULL, NULL);
osKernelStart();
```

The first function call initializes the RTOS kernel but does not start it. At least one thread should be created before starting the kernel. The first argument to `osThreadNew` is the function that it will execute. The second argument to `osThreadNew` is the argument that will be passed to your function. In this example it is NULL so no parameter is passed. The third argument is a pointer to a structure that specifies the thread name, priority, and other attributes. In this example it is NULL, so default attributes are used including priority `osPriorityNormal`. The last function call starts the kernel and thread switcher. It does not return unless an error occurs.

Your Goal

Your goal is to implement three concurrent tasks:

1. Continuously read the joystick, and display a number in binary on the bottom four LEDs that indicates the direction of the joystick (1 for North, 2 for East, 3 for South, 4 for West, 0 for no direction) and turn on LED 5 if (and only if) the joystick is pressed in.
2. Continuously read values from the ADC and print them out the serial port.
3. Toggle the top LED on or off each time the pushbutton is pressed and released. So each sequence of button press, button release should change the LED from off to on or from on to off.

Marking

You will receive 25% for successfully implementing each task, and an additional 25% for the ability to answer questions regarding the implementation. Note that the questions will be answered individually by each student in the group.