Lab Assignment 4 **ECE474** **Summer 2022**

Prof. Sep Makhsous University of Washington

Learning Objectives

After completing this lab, students will be able to:

1. apply a real-time preemptive operating system: RT-OS
2. Integrate a solution-focused software and hardware project with novel sensors, displays, or actuators.

Special Note: There is a two-step turn-in for Lab 4:

- **Turn-in 4.1**: Demonstrate Part I.1 and Part I.2 (below) **Due Date Aug-12-2022**
- **Turn-in 4.2:** Demonstrate remaining parts of the lab. **Due Date Aug-19-2022**
- Turn In Requirements:
- Turn-In 4.1: (Canvas:) A 1-page PDF cover sheet containing
    1. the 474 report template header (also completion of Demo 4.1 in-person).
    2. A ¼ page description of your team's **project idea** for Turn-in 4.2. This description should outline how you plan to achieve each of the project criteria (below).
- Turn-In 4.2: Regular report and demo. Include the description of the final project idea from Turn-in 4.1.

Part I: RT-OS

1. Install FreeRT-OS operating system package for the Arduino. (use only the Canvas files linked below, they are tested on Arduino Mega board).
    1. Download the .zip file for the ECE474 Official FreeRTOS version [HERE].
    2. Unzip the folder and follow instructions to install the Arduino_FreeRTOS library [Instructions]. Use the "manual installation" instructions.
2. Verify operation of the initial "`blink_analogRead474.ino`" example under FreeRTOS. Download the ECE474 demo app [HERE].
    1. Plug in your analog thumbstick (or any potentiometer) to generate a varying 0-5VDC positive voltage on an analog input pin.
    2. Modify the ECE demo app so that the analog input task uses the same pin you connect to the potentiometer/thumbstick.
    3. Change the baud rate of your Serial Monitor (in the Arduino IDE) to 19200 to match what is in the demo code.
    4. Run the ECE474 demo app and verify
        1. Light flashes
        2. Analog values are printed to the Serial Monitor and they change between 0-1023 when you change the thumbstick/potentiometer.

5. Demonstrate d.i and d.ii to a TA in person. Both team members be present for Q&A.

- - - - - - End of Turn-in 4.1 - - - - - -

3. Develop a new .ino file to demonstrate the following tasks running under RT-OS
    1. Task RT-1: Flash an OFF BOARD LED: ON for 100ms, OFF 200ms.
    2. Task RT-2: Configure Timer 4 and cause it to play "Close Encounters Theme" (see Lab 3, Apex. A) on your speaker. This task should play the theme three times (pause 1.5 sec between playbacks) and then stop itself.
    3. Task RT-3: In the background, measure the performance of an ***existing*** FFT (Fast Fourier Transform) function based on a random signal (Task RT-4 will actually perform the FFT as a background task).
    4. Call and wait for the FFT 5 times.
    5. The recommended FFT library (below) requires double (as in C double precision floats) data buffers for real and imaginary parts. These take up a lot of space. Start developing and debugging with the FFT set for only 64 samples. Work your way up to 512. The minimum FFT size for RT-3 credit is 128 samples. **See and follow the FreeRTOS development and debugging tips.**
    6. In detail: task RT-3 will:
        1. First, create a task, RT3p0, which
            1. generates an array of N pseudo-random doubles.
            2. Initializes a FreeRTOS Queue.
            3. Starts RT3p1 (below), and halts itself.
        2. Task RT3p1:
            1. Both RT3p1 and RT-4 (below) will use a second FreeRTOS Queue to signal when the FFT completes.
            2. Has a loop so that it can run this 5 times:
            3. Send **a pointer to the data** to task RT-4 (below) using the FreeRTOS `queue` function.
            4. Block and wait for the FFT to complete. Using the 2nd Que, RT-4 should signal back when it is done with the FFT
            5. Report back to your laptop/PC, via the Serial link, how much "wall clock time" elapsed for the 5 FFTs.
    7. Task RT-4:
        1. Receive data pointer from the queue using the `xQueueReceive()` function.
        2. Fill the input buffer of the FFT routine with data. Set up this buffer according to the requirements of the FFT library (e.g. real and imag parts).
        3. compute the FFT of the random buffer. Use this Arduino FFT library from GitHub. Use the same installation process as you did for the FreeRTOS library. You can download a .zip file from GitHub or clone it using Git.
        4. Using appropriate FreeRTOS calls, measure the "wall-clock time" time taken by the FFT. "wall-clock time" means it includes time that might be

used on other tasks or by FreeRTOS itself while task RT-4 is busy on the FFT.

5. Send a message back to RT-3 using the second Queue with the wall-clock time time for the completed FFT.

Part II: Project

Also using RT-OS,

1. Continue to run Task RT-1, defined above.
2. Do not play sounds or compute FFTs anymore unless required by your project below.
3. Run additional tasks as required to perform a “Project”. This is your chance to get creative and challenge yourself. Figure out physical parts you want to use and functions you want to perform which are “stretch goals” beyond Lab 1 - 3 capabilities. Try to organize your project around an embedded system that solves an actual problem. The project could also take the form of a game or entertainment device.

Project Criteria:

A successful ECE474 Lab 4 project will meet the following **criteria**:

1. perform reliable time and digital I/O functions.
2. operate with high speed and high CPU load. In terms of the parameters C, P, D, the projects should have at least one task which needs to run 50 times per second or faster (P <= 20ms). For example, a project which measures soil moisture and waters plants once a day would be too “slow”. However, we will retain the default FreeRTOS “tick time” of 15ms so you are not required to have operations at faster time scales.
3. interface at least one device to the Arduino Mega board which we have not used in prior labs.
4. make a measurement or control an actuator (or do both) in a way that (at least conceptually) solves a defined problem. A wide variety of sensors, actuators, and displays are available in the Arduino parts kits (you are NOT limited to the kit).
5. have a user-interface of some sort. It does not have to be fancy, but it will not be acceptable to switch/demo functions by plugging jumpers into various breadboard pins or typing into the Arduino IDE serial communication tool. Acceptable examples include switches, keypads, buttons, LEDs, LCD 2-line display, 8x8 LED matrix, 4x7 segment display.
6. Demonstrate understanding of how to use FreeRTOS features whenever appropriate or needed by your system. Eliminate holdover code from SRRI, DDS, or other simplistic schedulers from earlier labs.

Project Examples:

The following examples are listed to clarify the project scope and criteria. If you choose to implement one of these examples instead of coming up with an original project, you will lose some points.

- Implement a security system with the ultrasonic sensor, PID sensor, PIN keypad, and RFID keyfob. Implement tasks that perform low-latency communications with the RFID and keypad. Allow a 10-second entry delay between tripping the sensor and disarming the system.
- Build a space-invaders or Tetris game with the thumbstick and 8x8 display. Display # of aliens destroyed on the 7-seg display.
- Make a robotic system having at least 2 degrees of freedom (e.g. 2 motor drives). Demonstrate smooth coordination between the two motion axes.

- Make a DMM using Arduino Mega and the 4x7segment display. Implement as many standard DMM measurements as possible (try capacitance as well as Ohms, Volts, Amps). Measure its accuracy on all ranges.

Note: There is a point penalty for using any of the example ideas. (see the Rubric)