Lab 4: Failsafe Proximity Sensor

Illya Kuzmych, 2127069 08-June-2022
Ruvim Piholyuk, 2128297
Assignment: ECE 474 Lab 4

a) Introduction

In this lab, our main objective was to further develop our skills in utilizing the Arduino MEGA 2560 board and implementing freeRTOS to handle scheduling for each individual task. Also, we use freeRTOS for tasks to speak to each other using the built in freeRTOS queue functions. We learned how to utilize the features of freeRTOS through reading the corresponding data sheet to the freeRTOS library. Although we have experience with data driven schedulers, using freeRTOS gave us more flexibility with toggling tasks and using delays. Our goal was to implement freeRTOS to print the length in time it takes to calculate five FFT calculations while blinking an LED and playing Close Encounters of the Third Kind theme. Also, for our project section we set out to create a portable fail safe system for people who work in dark spaces to have a sensor to tell them when they're near a wall or object through the use of an LED and buzzer.

b) Methods and Techniques

In order to complete the learning objectives from lab 4, our group needed to gain skill utilizing real time preemptive operating systems and implementing additional Arduino peripherals to create a larger system. Initially it was straightforward to implement freeRTOS to blink an LED and to play the Close Encounters of the Third Kind theme, since very little was needed to change in comparison to previous labs. Our group then struggled to implement freeRTOS in order for tasks to speak to one another to time the computation of five FFT. To overcome this obstacle our group had to improve our skills in using pointers and global variables to allow tasks to each read a certain value.

To fully achieve the desired functionality of each task according to the specifications of Lab 4, the physical items we needed were jumper wires, two breadboards, a single 8-Ohm speaker, the Mega board, and single LED, two 220 Ohm resistors, a buzzer, and an ultrasonic sensor. Our main source of information was the documentation provided in the lab spec document and the Arduino Mega itself.

To display the full functionality of our project we used two different Arduino Mega boards. One will demo the specifications of part I with the use of an LED, a 220 Ohm resistor, and an 8-Ohm speaker. The other will be in a portable prototype that we created to display the functionality of our fail safe proximity sensor. We use a cardboard container we created with an LED, two 220 Ohm resistors, a buzzer, and a string used to holster the contraption over the neck of the desired user.

In order to complete part I we used an oscilloscope to verify that our LED was flashing at the correct period and that the 8-Ohm speaker was outputting the correct frequency in the Close Encounters theme. We then used two global queues created through the freeRTOS library in task RT3p0 to allow tasks RT3p1 and RT4 to communicate with each other. RT3p1 takes the data and sends it to RT4 which will compute the FFT on the sent data five times until it sends another value into our second queue in order to stop the timer in RT3p1 and to print out the total time elapsed during the computations.

In order to complete task II we used a tape measurer and oscilloscope to measure the accuracy of the ultrasonic sensor. We then use the reading from the sensor to alter the value that holds distance globally based on how many feet away the sensor is from an object. Based on that said distance we alter the frequency of when the buzzer and LED are toggled on to let the user know that they are approaching an object.

To experiment and verify the functionality of each task and making sure the schedulers behaved as expected, we used print statements to ensure each task was run thoroughly and had adequate stack size allocated to it within xTaskCreate.

c) **Experimental Results**

Before advancing in the laboratory and continuing with our project, we needed to finish the last section for I.3. This section involved utilizing freeRTOS to complete various tasks including blinking an LED, playing “Close Encounters of the Third Kind”, and implementing a FFT algorithm and measuring its performance utilizing Queues. Initially, the lab was straightforward as to make the blinking LED we followed the standard procedure to initialize a task in freeRTOS with null parameters, and then instead of using the Arduino delay() function we utilized freeRTOS’s built-in vTaskDelay function which took in a specific tick time we wanted to delay the next procedure by.

Building the LED code was as simple as a few lines of code for the task itself, and freeRTOS scheduler took care of running the task. Us giving it any specific priority did not significantly matter as at that time it was the only task running. We confirmed the functionality of the LED by testing it on hardware by hooking up one of the many external LEDs in our kit to a pin on the Arduino and confirming that it worked by uploading our code.

Our next task was to essentially repeat a procedure from Lab 3 except this time through freeRTOS. We also initialized this task with a similar structure to that of the blinking LED task, with a NULL parameter and a NULL task handle. We had a timer in the background to take care of OCR4A which was assigned based on the delays. Once the task ran three times, we called vTaskSuspend(NULL) which terminated the current running task by passing down the current running task’s handle, which was NULL. We confirmed the functionality of our system by hooking up an 8-Ohm speaker as we had done in the past and running both the LED task and the speaker task simultaneously.

The next two tasks were by far the most difficult and longest to implement. Familiarizing ourselves with freeRTOS had been decently straightforward when using simply the task functions, but we did not have any significant experience with Queues or FFT algorithms in Arduino. Initially, we had trouble installing the necessary library for Arduino and learning how to run the algorithm. First we ran experiments on the algorithm to check if it actually ran, and confirmed its functionality using Serial.print in Arduino. We then wanted to implement the system to send and receive Queue data and time how long it took the FFT to compute 5 times, and then constantly repeat this task.

For the first part we created a function that would essentially behave as an Arduino setup() function that ran a single time. We had it initialize an array of 128 random doubles, and then initialize two freeRTOS Queues: one to send a reference of the array to RT4, and another to send one back to RT3 once the FFT had finished computing five times. After this we suspended the task using vTaskSuspend and resumed RT3p1 using vTaskResume, as we had left RT3p1 in a hanging state so that it would not run out of order.

We built RT3p1 and RT4 to work hand-in-hand and communicate with each other. RT3 was responsible for holding data on how long it took the functions to run and for sending and receiving the Queue data. It had one Queue that sent a reference of the array to RT4 which called xQueueReceive and stored it in a buffer which then filled up two arrays with the real and imaginary data. Once the FFT ran five times, the second Queue would send a signal to Task 3 which received it and computed the run-time. We confirmed the functionality using Serial.print and debugged our code by liberally using print statements.

After we achieved the desired results in I.3, we moved over to building our custom project. We decided to make a custom, portable proximity sensor which could be carried around by an individual to warn them of hazards if they are in the dark. Our initial thoughts were to make it so that it would be easy for the user to turn it on and off using a simple switch, and having it be light and easy to carry. The main disadvantage in making this prototype was the size of the Arduino Mega. We utilized several new peripherals, mainly the Ultrasonic sensor and the buzzer from the kit. We utilized the ultrasonic sensor to receive data at the maximum tick rate.

Our initial approach to the project was to first learn how the ultrasonic sensor worked and how we would be able to read data from it. After doing some research we learned that the ultrasonic sensor sends a wave once a trigger is active at the speed of sound and then returns how long it took it to travel back to the echo input. Using this we derived an equation to convert the time it took the wave to travel to a distance in inches. We then confirmed the functionality of our formula and the ultrasonic sensor by using analogRead() and the print function to check the data being read by the sensor to confirm that it was accurate. After some careful experimentation we concluded that the sensor was accurate in a range of 2cm to 4m. Outside of this range it did not work at all as expected, instead always outputting the maximum value (4 meters). When creating the task we needed to make a large stack size, creating a large CPU load, as the task did several computations that took a lot of stack size to do.

The next procedure was implementing a buzzer that had a sound corresponding to the distance from any hazard, which we defined as any obstruction that could be dangerous in the dark. We decided to increase/decrease the buzz rate with foot increments. This was a fairly straightforward procedure in which we had several if/else if statements that would read in the ultrasonic sensor data and update the buzz rate as needed. We had the rate set by utilizing the freeRTOS vTaskDelay function. We then implemented something similar to the LED blink rate as a failsafe for if the person cannot hear the buzzer, or vice versa (hear buzzer, not see LED). This was an easy procedure and we confirmed its functionality by testing everything on hardware with the complete system.

Once we finished the hardware portion of our lab, we decided it would be aesthetically pleasing and beneficial to create some sort of wearable prototype as a user interface to demonstrate something in a real-world scenario. To do this we utilized cardboard to create a container in which we adhesively attach a small breadboard to a flap which opens/closes. We then used an external DC voltage source to power on the system by the choice of the user (on/off) so that it can be used as designed (as a failsafe). Figure 1 and 2 show the circuitry and the wearable prototype, respectively.

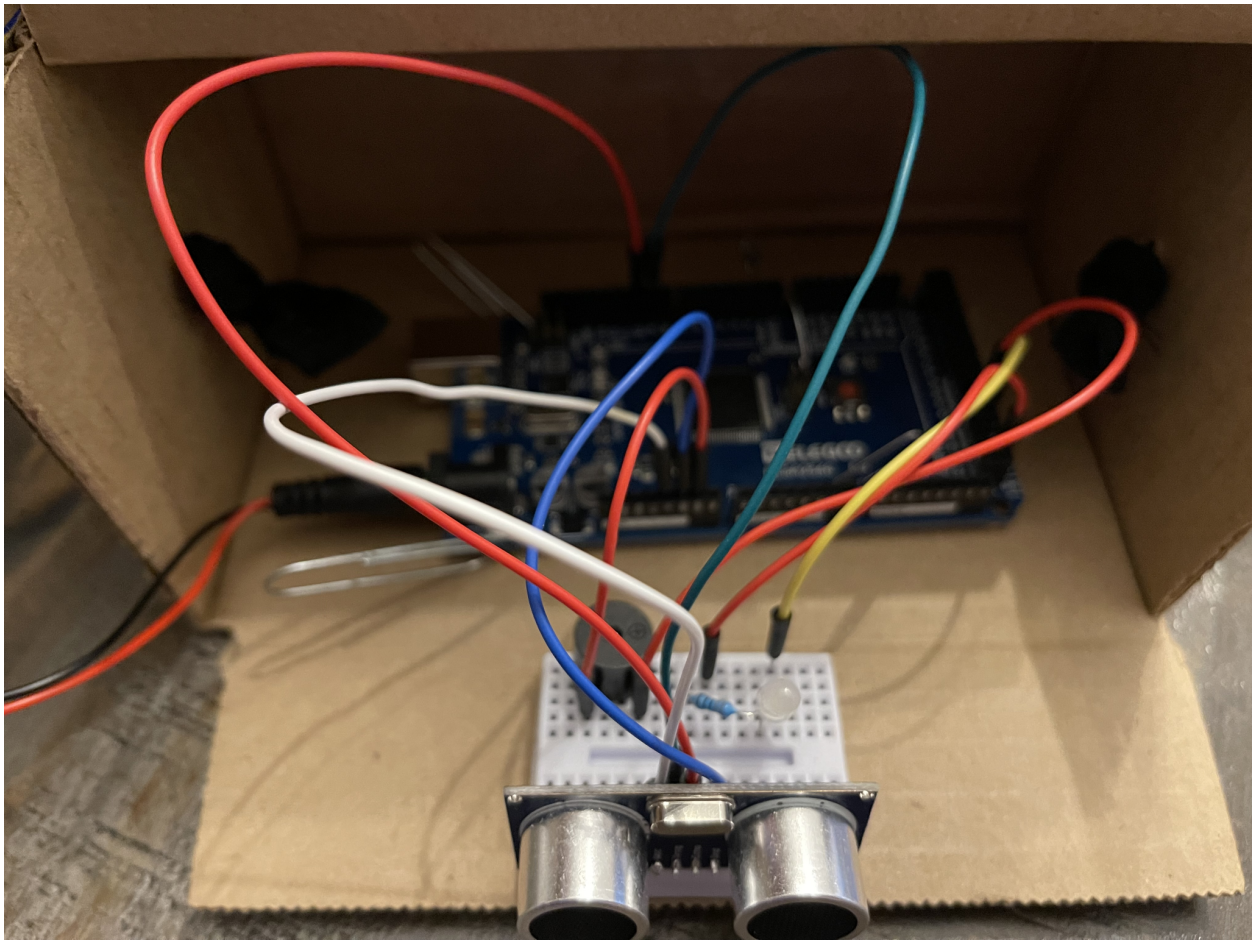

Figure 1. Circuitry for our final project

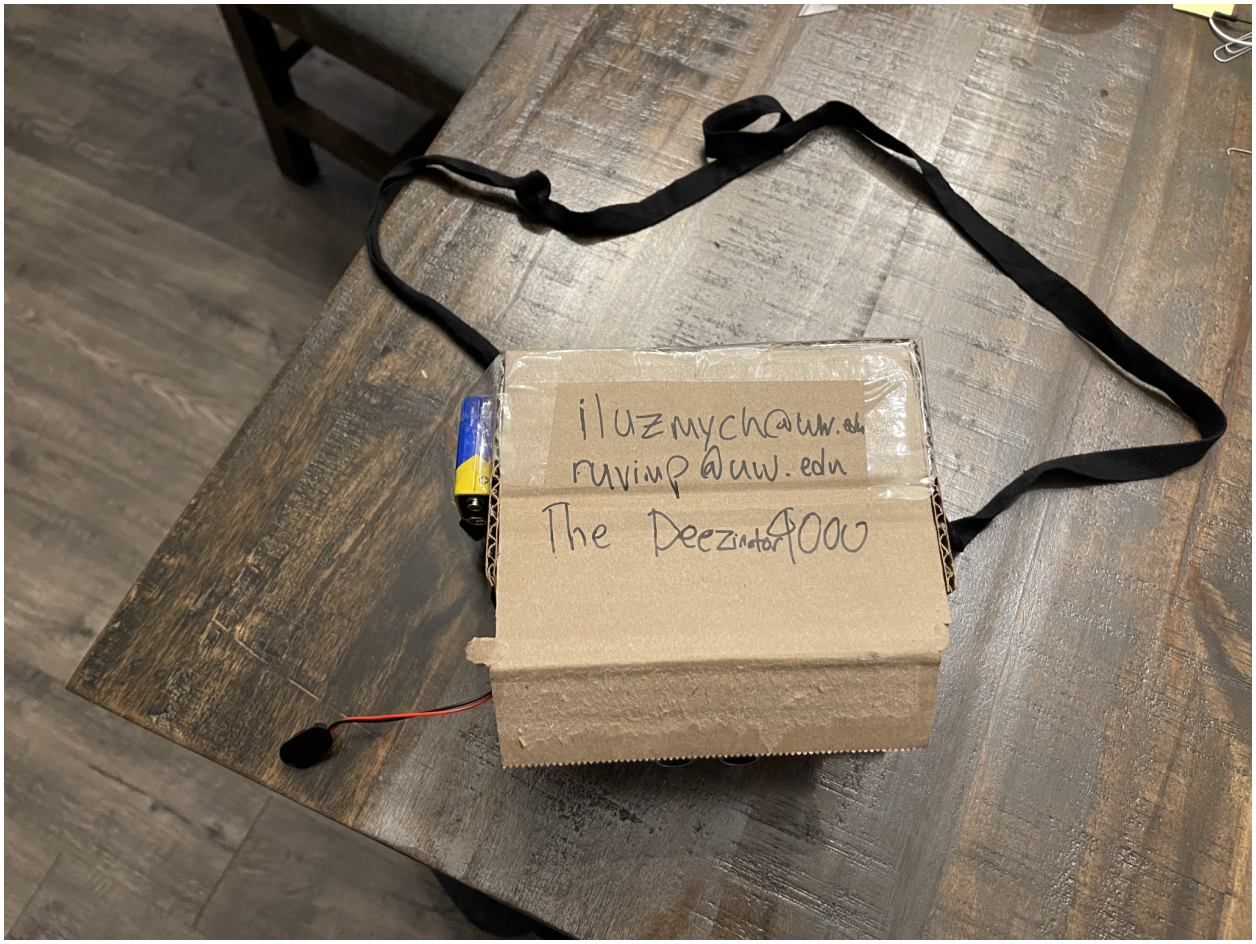

Figure 2. Wearable prototype for our completed system

d) **Overall performance summary**

The demo was overall a success. We began with demonstrating the functionality of part I according to the specifications. The part I of the demo went flawlessly. Afterwards, we demonstrated part II with our portable prototype which also went flawlessly.

The code worked exactly as expected and we were able to overcome each obstacle and met the specifications given to us in Lab 4. We believe we successfully completed the lab to the spec, and gained significant skill in utilizing freeRTOS.

e) **Teamwork Breakdown**

Task	Ruvim Piholyuk	Illya Kuzmych
Coding	✓	✓
Debugging	✓	✓
Hardware integration	✓	✓
Architecture	✓	✓
Code Commenting	✓	✓
Report	✓	✓

Our group was able to meet and work on the entire lab completely together, without splitting up the work but essentially programming and building everything on one system, together.

f) **Discussion and conclusions**

The most challenging part of Lab 4 was using freeRTOS to allow for communications between tasks. Neither of us had experience using the freeRTOS library and were uncomfortable using it initially until we reviewed lecture material and reviewed documentation within the given references. Alongside this we ran into issues with making sure part II updated the distance quick enough that when you step forward the user could notice the LED blinking and the buzzer buzzing at a smaller period. We decided that the best course of action was to update the distance values in between delay times so that the period of the LED and the buzzer can update in between toggling on and off.

We were especially proud of RT-3 and RT-4 to compute five computations of the FFT efficiently in comparison to other implementations. Implementing the Queues was difficult, but once we got it to work it was satisfying and allowed us to gain important insight into the purpose and use of freeRTOS. Also, the portable prototype gave the user a wearable and working example of how our project would work in the real world.