Lab 4: FreeRTOS Scheduling Systems

Christian Gordon 1939725 3/14/2022
Marc Hernandez 1828523
Assignment: ECE 474 Lab 4

Introduction

Over the course of the quarter, we've gained experience creating scheduler systems that rely on round-robin, interrupts, and data-driving scheduling. However, for this lab, we learned how to utilize a FreeRTOS, a real-time preemptive operating system that determines which tasks will execute depending on the task's priority. Rather than constructing a scheduler from the ground up, we're tasked with handling several functions with a constrained amount of memory.

Finalized Project Description

For our project, we combined the stepper motor, servo, and a laser pointer to create a 2 axis laser pointer that can be controlled by a joystick. Using the IR remote, the brightness of the laser can be changed. This project meets the criteria since we are using 2+ Arduino units we have not used before. We also are performing time and digital I/0 functions since we are controlling stepper motors and servos that utilize very specific signals to operate. We are also implementing the joystick as a user control device. All of this will be done using FreeRTOS tasks.

Methods and Techniques

Task RT-1 Flash:

To meet the requirements of Task RT-1, we needed to utilize the digitalWrite function to enable and disable the external LED pin, as we have done in the past. However, rather than using Arduino built-in delay function, we used vTaskDelay to delay our for the duration of time that we would like it to remain off or on.

Task RT-2 Speaker:

For this section of the Lab, we used the principles we learned in the previous section to implement the speaker control as an RTOS task. We went through the array of notes as usual, but the task delay made the code much simpler to read since we didn't need to keep track of the time elapsed in the task, and we can let the delay do its job. We used a for loop to iterate over the task 3 times and then used vTaskDelete(NULL) to exit from the task and stop it from running.

Task RT-3 Queue System:

To fulfill the requirements of task 3, we used a queuing system by using xQueueCreate() and then using send and receive methods to determine when our FFT was complete. Within our TaskRT3p0 we created a new TaskRT3p1 to stagger our tasks starting points, after which we suspended TaskRT3p0. The most consistent method for capturing our time was by finding the difference of millis() before and after the FFT for loop and printing it to the serial monitor.

Task RT-4 FFT:

For task 4, we closely followed the examples found on the public GitHub found here. By utilizing a for loop and FFT.compute we can take up a considerable amount of CPU time depending on the number of samples used. We first started with a smaller number of samples and as we gradually increased our sample size we were able to find issues in our code structure.

Custom Project:

This project required 4 separate tasks to be made, one handled reading the analog input, another checks for new IR communications, another moves the stepper motor to the destination of the x axis, and the last one controls the position of the servo. The laser brightness and servo position are controlled using Fast PWM settings on the timers to create the correct PWM values. We also use the stepper.h library to move the stepper motor.

Experimental Results

Task RT-1 Flash:

Since this task required us to replicate an identical task from a previous lab, it was simple to verify that we were getting the correct off and on times. To ensure our times were correct we utilized lab code from lab 1 and compared the LED behavior. This allowed us to verify that portTICK_PERIOD_MS correctly translated the number of ticks to milliseconds.

Task RT-2 Speaker:

This task was also simple to verify since it is an audible task where if anything goes wrong, you can hear the issue. Our task executed the melody 3 times and when we compared the output of this task to previous labs, we were able to hear that the output was the exact same.

Task RT-3 Queue System:

In order to verify that our queue system was working correctly, we passed through “junk values” and extracted the expected values from the receiving function. After verifying our queue system we created a new queue that would detect when the FFT was completed 5 times. To make sure that we staggered our p0 and p1 task by creating the second part of the task within the first to avoid one running before the other.

Task RT-4 FFT:

Once our queue was created, we were able to utilize the FFT methods within a for loop to run the calculation 5 times. After testing all of our tasks together, we found our code performed up to the spec requirements of 128 samples but had some issues with the LED on and off time after increasing to 256 samples.

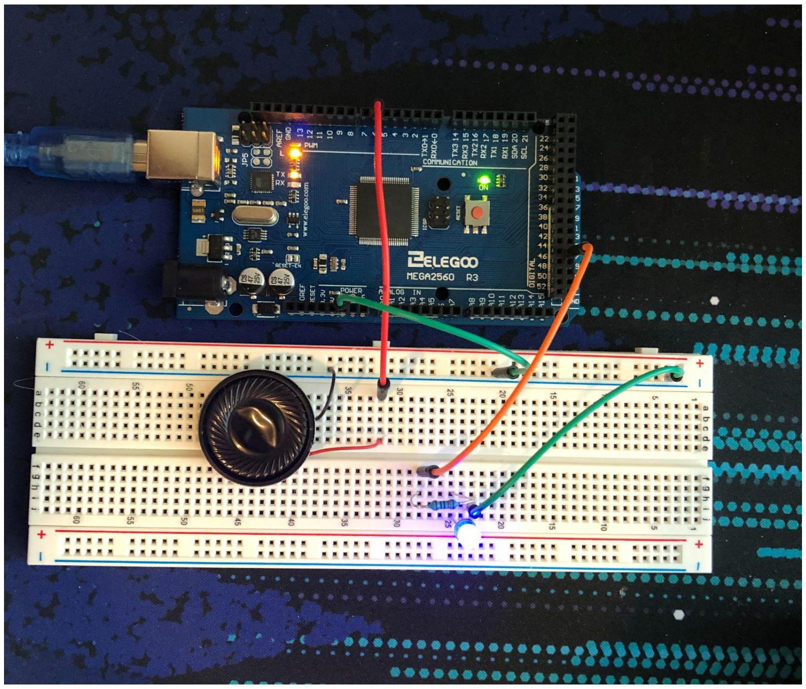

Figure 1: Setup for Tasks RT1-4

Custom Project:

We tested the tasks individually as we developed them. We checked that the stepper motor would accurately move to the correct position between 0 and 180. Then we checked that we would alter the brightness of the laser and the position of the servo using the PWM values. All of these systems worked together once the priorities of the tasks were edited appropriately.

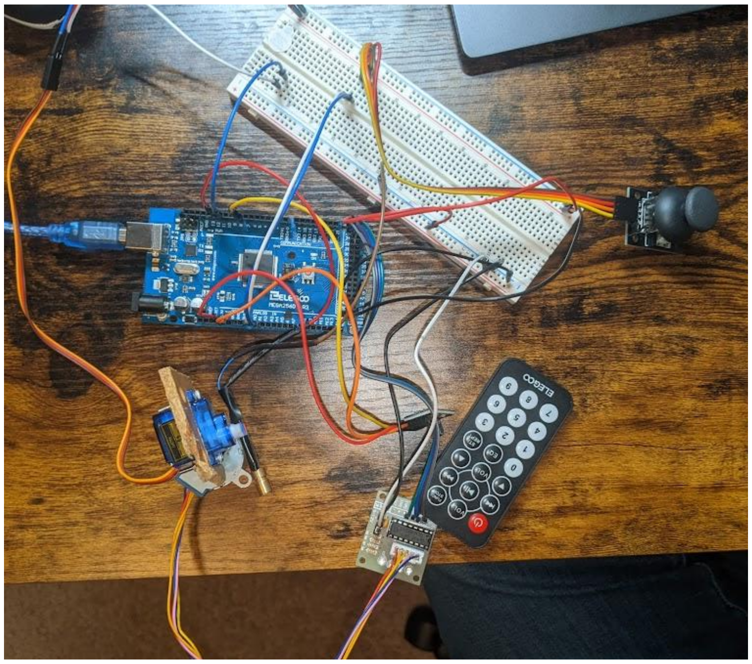

Figure 2. Hardware for the project. Includes stepper motor, servo, laser diode, joystick, IR reciever and IR remote.

Code Documentation

Task RT-1 Flash:

Our code structure for Task RT-1 is very similar to previous labs; however, the primary difference is found between the digitalWrite lines. vTaskDelay() like delay() halts the task from continuing until the amount of time specified in milliseconds has elapsed. However, the primary difference between the two functions is vTaskDelay() does not impede other tasks from running. By utilizing portTICK_PERIOD_MS, we can determine how long to delay our LED off and on time in milliseconds. Then by placing those lines of code in a for(;;) loop, we can cycle through the off and on state of the LED infinitely.

Task RT-2 Speaker:

Again, this code is very similar to the previous labs, but the primary difference comes in how we format our delays and when we run the code to switch the output frequencies. We were able to use the vTaskDelay function to directly do nonblocking delays that would allow other code to run without numerous comparisons of time to be happening between each change. We simply delay the task by the period of the note in ms and then update to our next frequency. If we have exceed the size of melody, reset and wait for a second and a half. Ultimately this task exits with a vTaskDelete() which ends it without any further computations occurring on it.

Task RT-3 Queue System:

Task RT-3 is split into two distinct functions in our code TaskRT3p0 and TaskRT3p1. The former is responsible for generating an array of random values, creating the first queue, and creating TaskRT3p1. After TaskRT3p0, completes these three processes for the first time, the task suspends itself indefinitely. Then, TaskRT3p1 creates a second queue that keeps track of the FFT running five times. Finally, the task will continue to send to the fft_queue and check if it has received something from the timer_queue.

Task RT-4 FFT:

Task RT-4 utilizes the arduinoFFT.h library to compute the Fast Fourier transform (FFT) for a given number of SAMPLES N. First, we calculate the values we want to use to determine our array values that will be stored in real[]. After completing the array, we take the current time in millis() and store its value in our start variable. We then compute the FFT five times and once again take the current times from millis()-start and store the value in finished. Lastly, the values of finished will be printed on the serial monitor, this captures the “wall-clock time.”

Custom Project:

The project uses the RTOS libraries and the stepper.h library. First we define constants for the code and then initilize the stepper motor. We then create the correct timers to control the system and define all of the tasks for the scheduler. We then create all of the tasks that need to be scheduled.

Overall Performance Summary

This lab was very successful for us, we were able to learn the basics of the FreeRTOS system and learned how to create and manipulate tasks and queues within a scheduler. Additionally, we learned how to manage resources and deal with a constrained SRAM module that needs to allocate adequate stack sizes to tasks of different complexities. Overall, we feel comfortable with our current understanding of the FreeRTOS system and task management.

Teamwork Breakdown

To tackle Lab 4, we first started by completing all of 4.1 to ensure that we could access the FreeRTOS library successfully. For part 4.2, we split the tasks between ourselves to make progress on the outlined tasks, such as the blinking LED, speaker, and FFT, and the custom project. Marc primarily focused on the former, and Christian focused on the latter. We took this approach to provide us ample time to study for other exams and reduce each lab member's workload. After completing each of our respective tasks, we merged our files into a single folder and debugged any issues that surfaced during testing.

Discussion and Conclusions

Marc: The most challenging portion of the lab for me was understanding how to utilize the queue system correctly. I frequently ran into stack overflow errors while moving from 64 to 128 samples. To circumvent this issue, I start with some values in a queue and passed them between methods. After verifying that I could successfully extract the values from the queue, I began with small sample values and slowly incremented the sample size as I revised my code.

Christian: The most challenging issue that I ran into was using the stepper motor within RTOS. I at first tried to create the stepper library from scratch and integrate that code with the RTOS task structure, but this raised many bugs and was ultimately abandoned. The next issue I came across was getting the stepper task's delay correct because when it was too short, the stepper would oscillate between two positions even when the input was constant. Ultimately, I found that a delay of 250ms was enough to ensure that the motor could go through the expected 180 degrees of motion with no issues.