Lab 3: Schedulers

Eli Orona, 2076032 27-May-2022
Varun Venkatesh, 1939023
Assignment: ECE474 Lab 3

Introduction: Summarize the lab's learning objectives and key activities in your own words.

The previous lab, Lab 2, gave us hands-on experience with understanding how to write code that did not require the use of the delay function so we could write simultaneous tasks. We were able to apply previous learnings on a larger scale for this lab by learning how to create and use schedulers, interrupts, and synchronous vs asynchronous functions.

Methods and Techniques: Explain the activities, skills, instruments, and tests you used to achieve the learning objectives.

We used the Arduino IDE on a Windows Machine, an Arduino Mega board, a solderless breadboard, several 220 Ohm resistors, several wires, a blue LED, an 8 Ohm speaker, and a SEG7 LED display. We needed good, clean wiring skills between the breadboard and the Mega, as well as good debugging skills in order to ensure we could solve the problems with the testing of the lab.

The times for the LEDs, Lab 2 speaker tone lengths, and the combined Lab 3 tasks were timed with a timer and were the correct length. A tuner was used to make sure that the notes were of the correct frequency.

Experimental Results: Describe specific test results which show how your code and hardware worked together to meet the lab requirements. Include 1-3 photos of your hardware setup.

We completed the lab on a step-by-step basis, making sure we finished each part before moving on. We decided to start by implementing the schedulers. We began with the round robin as it was the simplest and did not require many additional definitions. We then moved on to the SRRI, which was more difficult. We had to keep track of different states and sleeping times in their own separate arrays and use all of them together to ensure the tasks were turning on/off with the correct timing. We then used the SRRI we implemented in the next scheduler, the DDS, with a few changes. We implemented a struct that would keep track of tasks and a new state called DEAD, which would help us identify which tasks needed to be iterated through at all to decrement sleeping times.

We then began implementing the tasks. There were 6 tasks in total:

Task 1: Flash an LED for 250 ms every 1 second. We had to implement this task with 3 variations, one for each scheduler. The first was the simplest and simply used the new sleep_474 function to control the time the LED was lit. The other two schedulers used our new

sleeping array to decrement sleep times manually and kept track of the on/off states to turn it to the corresponding state afterward.

Task 2: This task used two methods from our Lab 2 (see the previous report) that controlled the hertz of the timer and the length the frequency was played. We created a notes array that stored the series of notes used for the "Close Encounters of the Third Kind" theme. Similarly to Task 1, we needed to implement this task for every type of scheduler, so we first used the new sleep_474 function to control the time the song played in the round robin. The other two schedulers used our new sleeping array to decrement sleep times manually and kept track of the on/off states to turn it to the corresponding state afterward.

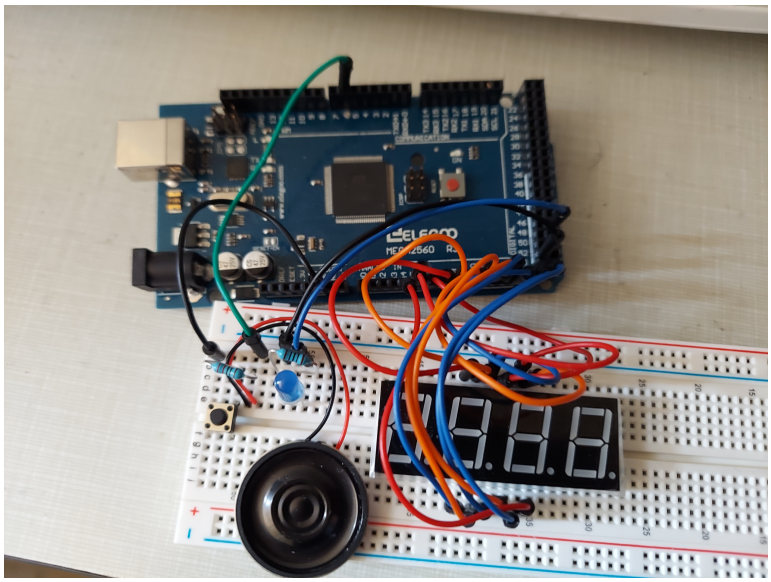
Task 3: Task 3 was the first time we used the new seg-7 display for a lab in this class. The first step was to understand the ports and what macros were used by Arduino to allow us to quickly send signals to the display. After understanding this, since we only needed to implement task 3 for the SRRI, we created one method that would update the seg-7 display to the newest value and another method that would increment the value to display by 1 every 100 milliseconds. We then used the regular update method in the SRRI to change the task state whenever necessary.

Task 4: Task 4 was only used once, and it ran as its own task. However, despite being an individual task, it was effectively a combination of tasks 2 and 3. This required a DDS, as we would proactively start and stop the two tasks. We modified tasks 2 and 3 so that there was a countdown before task 2 began, and while task 2 was playing the song, the frequency would be displayed on the display. We used the task_start and task_self_quit methods to achieve this repeatedly.

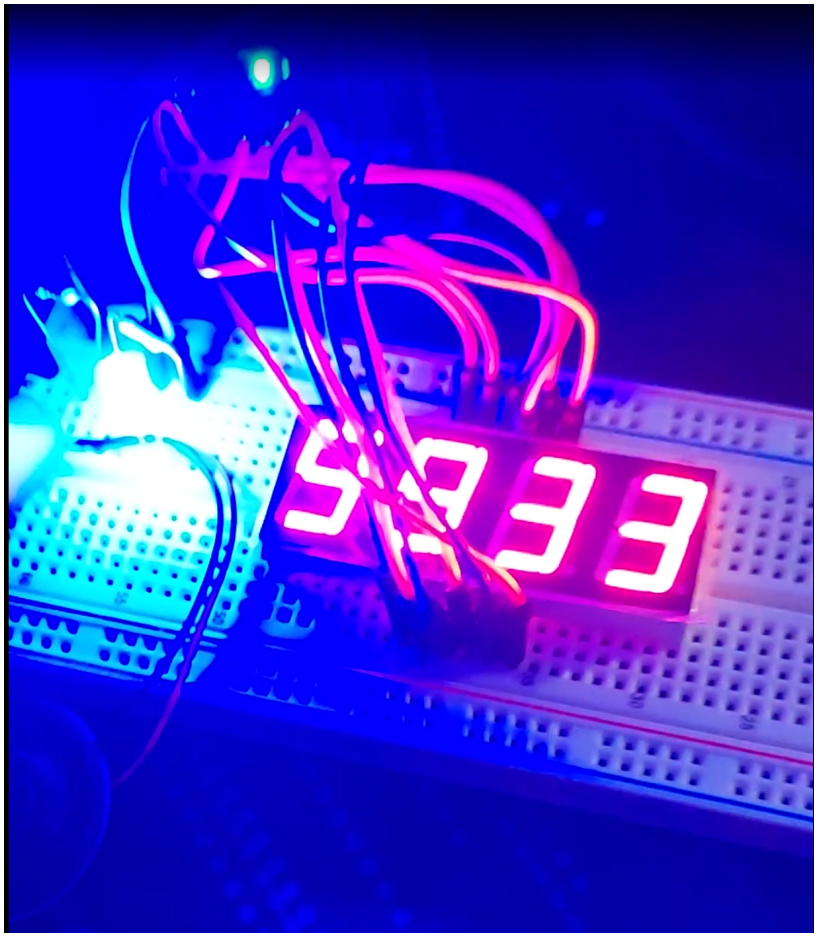
Task 5: Task 5 was similar to Task 4 in the sense that it was a combination of Task 1 and 2. There were some changes, however, as only one task would run constantly. We used task_self_quit and the DEAD state to turn off Task 2 when necessary, as well as the new task that we called task_5_smile. This task was not too different to implement from Task 4, and the only thing we had to ensure was that we never stopped task 1 from running.

We used several macros and defines to toggle between different demos, allowing us to seamlessly translate between tasks in the order that the demo required.
Below are pictures of our final setup, with the speaker, LEDs, Mega board, and SEG7 display.

This is specifically for tasks 3, 4, and 5 which are combinations of the first two tasks. The four pictures below show how the three LEDs light up in an alternating fashion, in sequential order, and although not audible, the speaker plugged in generates the expected noise. We found that for each test, each section of the lab was consistent with the expected behavior. Some specific tests we conducted included setting up the LEDs and speakers in multiple sections of the breadboard, trying additional frequencies on top of the ones in the task, and conducting multiple tasks at the same time.



Above: Final circuitry for the Lab



Above: Testing Demo 4.

Code Documentation: Write a short paragraph for each task and each initialization process in your software. Identify the line number ranges for the tasks, function and struct names, etc in your paragraph. If some code is re-used from a previous lab, do not repeat its documentation. Reference the specific paragraph of your previous report.

View Doxygen documentation created and submitted with the ino files.

Overall Performance

The most difficult part of the lab was implementing the two synchronized schedulers, SRRI and DDS. The DDS was heavily built off the SRRI, so in reality, the SRRI was the most difficult as it was very hard to test without just seeing if the program was running with expected behavior. Tasks 1, 2, 3, and 5 were all very simple and quite similar to each other as they heavily utilized the same tasks repeatedly. Task 4 required a little bit of a change as it was the first task using the DDS, seg-7 display, and speaker all at once. All code produced the expected behavior.

Teamwork Breakdown

For the breakdown, Varun implemented the Round Robin and DDS scheduler. Eli implemented the SRRI and Tasks 2 and 3. Varun and Eli worked together to implement Tasks 1, 4, and 5. For the writeup, Varun did the Doxygen documentation and wrote the methods, results, and performance, while Eli wrote about the breakdown and results.

Discussion and conclusions

Understanding the scope of each scheduler and when each was most useful was the initial hardest part. After spending some time learning why each was more useful than another scheduler, we began to implement the code. Some other difficult sections were identifying the ports for the seg-7 display, and testing the code as there was no debugging possible. Once we started implementing, it became easier as we went on, allowing the later tasks to be done quickly.