Lab 3: Schedulers and Interrupts

Illya Kuzmych, 2127069 27-May-2022
Ruvim Piholyuk, 2128297
Assignment: ECE 474 Lab 3

a) Introduction

In this lab, our main objective was to further develop our skills in utilizing the Arduino MEGA 2560 board and building a scheduler utilizing three different methods: a round-robin, synchronized round robin with ISR, and a data-driven scheduler, each of which we built for the first time in this lab. We also learned how to write an interrupt service routine (ISR) by reading up on the documentation, enabling the proper interrupts, and writing a simple, effective routine. For the first time in this class we also used structs and function pointers for our labs, which proved to be important for all the scheduler types. Our goal was to develop the schedulers to fit our desired specifications by portraying all the required tasks that could be summarized simply as: a blinking LED, a sequence of notes, and a count-down timer.

b) Methods and Techniques

In order to complete the learning objectives from lab 3, our group needed to gain skill utilizing schedulers in Arduino. Initially, the round-robin scheduler was fairly straightforward to complete as it was similar to how we scheduled tasks in previous labs. Our group also needed to gain understanding of how interrupts worked and how to write an interrupt service routine.

To fully achieve all the desired functionality of each task according to the specifications of Lab 2, the physical items we needed were jumper wires, a breadboard, a single 8-Ohm speaker, the Mega board, a single LED and resistor, and a 4-digit seven-segment display. Our main source of information was the documentation provided in the lab spec document, and the Arduino Mega documentation itself.

In order to have a more fluid program, for every demo type we built a different Arduino project. Sometimes the code remained similar, but other times it would change significantly. This helped prevent a messy or too-long program. This also helped reduce the amount of logic needed in each project because we did not have to toggle between different scheduler types within a program, instead simply going to another, specific project and uploading it to the Arduino gave you the desired functionality with the scheduler you need. To verify that the LEDs were working correctly, we simply tested everything on the breadboard, but to test the frequency outputs of the speaker we used an oscilloscope to verify that our outputs were what we expected. We tested both the song from Close Encounters to verify the functionality of the speaker, and the seven-segment display was tested by initially passing down simple code to find out which digits toggled what.

To experiment and verify the functionality of each task and making sure the schedulers behaved as expected, we used an external stopwatch to confirm the timing of each task, and a checklist to verify that the order of each task was also as expected.

c) **Experimental Results**

Before building any of the required schedulers, our group simply built the tasks in a manner that we knew how to do. Each task was built separately and functionality confirmed before building any of the schedulers. Initially, we built a simple function for the blinking LED, which we tested on hardware using just an external LED and Arduino MEGA 2560. Then we made the speaker emit the theme from Close Encounters of the Third Kind, confirming the frequencies on an oscilloscope and by hearing the speaker output. Our most difficult task was to develop the four-digit seven-segment display, as it was a hardware tool we had not used previously. The most difficult aspect of this task was to wire it up properly, and to assign the appropriate PORTs the correct register values. Once we wired up the seven-segment display, we built a simple function to count up by utilizing the modulus (%) operator in Arduino to determine which LED would increment and how often. Once we wrote the function, we were able to confirm the functionality by testing it on hardware.

Our next procedure involved us building a round-robin scheduler to work for task 1 and task 2. The tasks together worked in a similar manner as they did separately, as they did not interfere with each other. We used an external timer that incremented after each iteration of the loop() function to control the signals in each task and when to activate an LED or change the bits to an output such as the seven-seg display.

Our next task was to integrate tasks 1 and 2 to work with a SSRI scheduler. The significant difference between this scheduler and the round-robin scheduler was that to achieve the desired functionality of the SRRI we had to build several additional functions: a scheduler, a scheduler synchronizer, a sleep function, a start function that would pass down a function pointer to the function it wanted to run, and an interrupt service routine to create a 2ms delay. Each task's behavior also relied on states. If a task was in the SLEEPING state, it was to be disabled until it was put into the READY state either internally or externally. The interrupt service routine was designed to utilize the TIMER3_COMPA_vect interrupt from within Arduino's library that would trigger the routine each time that timer3 reached the OCR3A value specified in setup. The sleep function was built so that whenever internally called within any function, it would set that tasks state to SLEEPING, then initialize a count-down of the input time to reinitialize the task. The scheduler was responsible for directing traffic by checking that there were no NULLs in indices where they were not expected, and starting each function in sequence. The schedule synchronizer was responsible for decreasing the sleeping time. The ISR was then checked in our loop by checking the sFlag within it, which would trigger upon an expired timer, allowing a tick every 2ms. We confirmed the functionality of our design on our circuit, and then integrated task 3 to also work with our SRRI.

Our next procedure was to build a data-driven scheduler, which is a scheduler that is built by utilizing a struct and an array of structs to manage the states, sleeping time, and each tasks's ID. In addition to creating a new scheduler, we also needed to integrate two additional tasks: one specific one that displays the frequency of each note while the song plays, and a count-down

timer while the music paused. To implement the data-driven scheduler, we changed the formatting of our code in such a way that our flags to initialize tasks were taken from each task's corresponding structure by checking its state in the array of structure pointers. The schedule sync was built in a similar manner to that of the SRRI, with the main change being how the sleeping time was decreased (utilizing a struct instead of a separate array). The scheduler function itself was almost identical to that of the SRRI. Our next step was to integrate task 5, which was a supervisor task for several other tasks. To implement this task we built two additional functions: one to halt any task it was called within, and another to start a specified task. These functions used another set of flags to set their status: DEAD and READY. If a task was ever halted, its state was set to DEAD, and if it was ever woken up, its state was set to READY. Tasks would only ever run if it was their turn in the scheduler sequence and their state was set to READY. We confirmed the functionality of task 5 by testing the speaker outputs using an oscilloscope and testing the lights on hardware using their corresponding peripherals. Figure 1 shows our complete circuit that we used for testing.

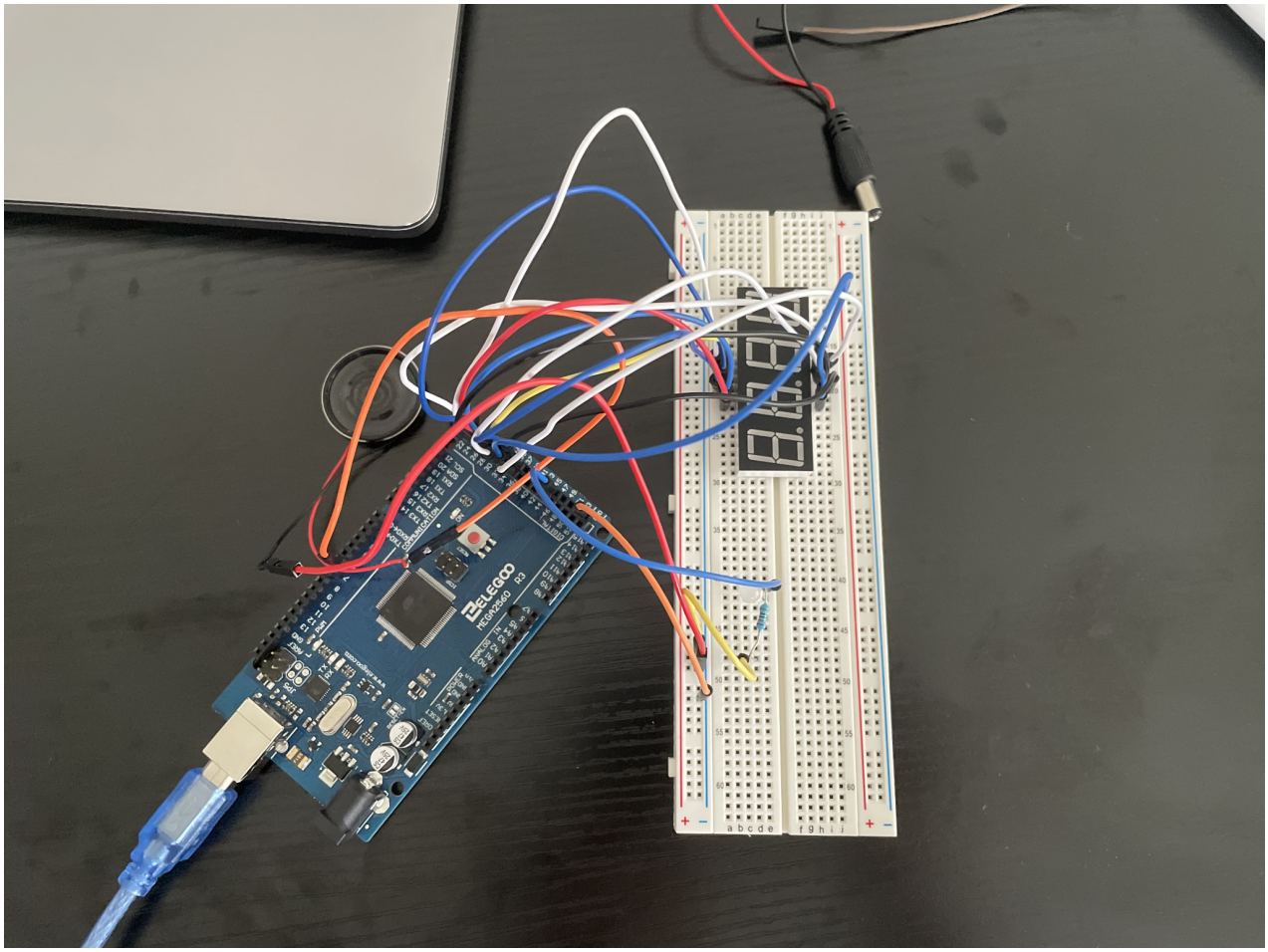

Figure 1. Mega 2560 microcontroller w/ speaker, external LED, and four-digit seven-segment display

d) **Overall performance summary**

The demo was overall a success. We began by showing our demos in the order they were listed in the lab specifications. The only minor issue we encountered during the demo was that during the display of the round-robin scheduler implementing tasks 1 and 2, the external LED hooked up to pin 47 was dimmer than usual. Other than that, everything worked exactly as expected. Vikram told us this may be because Arduino implicitly decided to run less current through Pin 6 making the LED dimmer than expected.

The code worked as expected, without any of the issues we faced during debug. We believe we successfully completed the lab to the spec, and gained significant skill in utilizing interrupts.

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 the lab was building the SRRI scheduler. We were not required to do this in the past, and at first were uncomfortable implementing this in our lab. We initially had trouble understanding how to build an interrupt service routine and how to access the desired interrupt from the vector table. Alongside this, we ran into issues with the ISR because we were not sure how to implement another timer to create a 2ms tick. However, after some extensive Googling and research on Youtube, we realized that in order to activate the desired interrupt we wanted: TIMER3_COMPA_vect, we also needed to access and enable that interrupt in the TIMSK3 register.

We were especially proud of the way we achieved the proper functionality of the SRRI, and were then able to implement the DDS fairly easily as we felt the logic was similar to that of the SRRI.