Lab 3: Scheduling and Interrupts with ATmega2560

Kevin Barreto -- SID: 2070948
Jesse Butler ---- SID: 2076576

February 25th, 2022

Assignment: ECE474 Lab 3

Introduction

Lab 3 initially shifts the focus off of learning the features of the ATmega2560 microcontroller board and on to the software architecture of how to allocate CPU processing power to multiple tasks or functions. Different types of scheduling algorithms are implemented here to run multiple tasks simultaneously. Those types specifically being a basic Round-Robin (RR) scheduler, Synchronized RR scheduler with Interrupts (SRRI), and a Data-Driven scheduler (DDS), all of which are non-preemptive.

Each scheduler has its own unique characteristics. For the implementation of the SRRI scheduler, a separate focus of this lab is the enabling and use of the microcontroller internal interrupts generated by the Timer/Counters. A major component of utilizing interrupts is not the enabling itself but rather on how it is used with volatile variables and Interrupt Service Routines (ISR) and successfully executing without bugs. The DDS scheduler utilizes Task Control Blocks which is a data structure that holds all of the necessary information, such as ‘Ready’ status or sleep time. Each task has a dedicated control block. For this lab all of the above scheduler architectures will be running some combination of tasks out a total number of five tasks.

Methods and Techniques

Lab Organization

There are a total of six unique combinations for demonstration, see Table 1. This section will describe the methods and techniques for each demonstration, however, it is first necessary to describe the tasks.

Task 1 (T1):
Flash external LED (ON for 250ms, OFF for 750ms)

Task 2 (T2):
Speaker outputs the melody “Close Encounters of the Third Kind” (Play melody then OFF for 4 seconds)

Task 3 (T3):
7-segment display counting (count 1 unit per 100 ms)

Task 4 (T4):
Combine Tasks 2 and 3. When the speaker is off the display will indicate a countdown until the next time the speaker plays the melody. During the melody the display will indicate the frequency of the tone.

Task 5 (T5):
Supervises other tasks (start and stop). The schedule follows:

- Task 1 runs without sleeping
- Task 2 runs for 2 cycles
- On 7-seg display, display countdown before restarting task 2
- On 7-seg display, display a “smile” for 2 seconds
- Stop all tasks except Task 1

DEMO #	TASK COMBINATION	SCHEDULER
1	T1, T2	RR
2	T1, T2	SRRI
3	T1, T2	DDS
4	T1, T2, T3	SRRI
5	T4	DDS
6	T5	DDS

Table 1: Demonstrations

Demo #1

This demonstration is a combination of the RR scheduler with tasks 1 and 2. The setup of tasks 1 and 2 build off of the previous Lab 2 with manual masking and manipulation of the port registers and setup of Timer/Counter 4 for the speaker output, see Table 2 for the setup of the speaker Timer/Counter 4.

The architecture of the Round-Robin scheduler is the simplest of all to implement. It comprises a simple sequential calling of the tasks followed by a time measuring code block to ensure that the correct period for each task is achieved. The relevant setup for the tasks and time “delay” code is initialized by the corresponding scheduler switch case in the setup() function.

Control Register	Bit Name	Description
TCCR4A	COM4A0 (Bit 6)	Set to toggle pin on each timer match
TCCR4B	WGM42 (Bit 3)	Sets CTC mode
	CS42 (Bit 2)	Sets clock source/ prescale to 256

Table 2. Settings in Timer/Counter Control Registers

Demo #2

The RR scheduler is desirable only when the tasks to be run are simple and the period easily maintained. For more complex tasks, or even greater number of individual tasks, the SRRI scheduler reigns supreme over the RR scheduler. Instead of a sequential calling of tasks in the loop(), the scheduler determines which tasks to run by a system of arrays holding the state of a function and various conditional statements to assess those states. The state array holds the run state, {READY, RUNNING, SLEEPING}, the task array holds the function address of the tasks to be run, and the sleep delay array holds the sleep time, in milliseconds, if a particular task is put to sleep. To put a task to sleep a sleep_474() function was written such that if it is called within a function said function would be put to sleep and the above arrays update. When the scheduler checks the conditional statements it will see the sleeping function and pass over it instead of running. The scheduler will only run a function when it sees that it is in the READY state.

An interrupt and an Interrupt Service Routine (ISR) is essential in the functionality and time keeping of the SRRI scheduler. Its main purpose is to keep track of time accurately and continually check the states of the tasks. The interrupt is generated by the Timer/Counter 0. This lab makes use of the OC0B Output Compare pin. The Timer/Counter settings are presented in Table 3. The OC0RB register is set to a value of 125 which corresponds to a time clock “tick” of 2 ms. Underneath the hood of the clock tick, a compare match causes an interrupt which prompts the ISR to run. Here, it is set up to update a volatile type variable which is used within the schedule_sync() function to restart the task iteration loop. In essence, when the loop is done going through each task the schedule_sync() function acts as a time delay until the next clock tick and then the process reiterates.

Control Register	Bit Name	Description
TCCR0A	COM0B0 (Bit 4)	Set to toggle pin on each timer match
	WGM01 (Bit 1)	Sets CTC mode
TCCR0B	CS02 (Bit 2)	Sets clock source/ prescale to 256
TIMSK0	OCIE0B (Bit 2)	Enable interrupts on OC0B

Table 3. Settings in Timer/Counter Control Registers and Interrupts

Demo #3

The DDS scheduler has a much more professional presentation over the SRRI with the use of a data structure defined as the Task Control Block (TCB). The TCB holds all of the pertinent information for a particular task. This includes the function address, a code name, the run state, a task name, the number of times run, and the sleep delay time if in SLEEPING state. The functionality is similar to the SRRI but with a more compact form of holding information and with an additional feature of being able to completely halt a function from running with the addition of the DEAD status and a task_selfquit() function. Along with a task_start() function, this functionality allows the flexibility to run a task only a number of times and then restart it at a later time. In essence, it allows intermittent running of tasks whereas with the SRRI the tasks are always periodically running with no option to turn off.

Demo #4

This demonstration extends demo #2 with a new task; a 7-segment display task (T3) counts up by 1 unit every 100 milliseconds. These three tasks run simultaneously, with the LED blinking for one quarter of every second, the music tones changing every second with a four second pause at the end of the tune, and the 7-segment counter incrementing continuously in these one-tenth second intervals. Since these three tasks were continuous and periodic, the SRRI scheduler was able to handle them well.

Demo #5

Demonstration #5 defines a new task (T4) to be run with the DDS scheduler. This new task combines T2 and T3 such that when T2 outputs the melody T3 will be displaying the frequency of the tone simultaneously and when T2 is not outputting a melody T3 will display a countdown timer until the next time T2 runs.

Demo #6

Demonstration #6 defines yet another new task (T5). This task is in essence a supervisory task that makes use of the task_selfquit() and task_start() functions which allow intermittent running of tasks as described earlier under Demo #3 description. This new task defines that T1 be run at all times, T2 is run 2 iterations and then stopped which then prompts a count down with the 7-seg display followed by restarting of T2 for one final iteration, followed by a “smile” emoji displayed on the 7-seg display, and finally stopping all tasks except for T1.

Experimental Results

Task one

LED operation was verified visually and with a basic timer to confirm a 1 second period for the LED sequence with appropriate illumination duration for each setting. This held true throughout the lab, with only minor configuration necessary for each scheduler.

Task two

In preparing the melody for use in this lab, we were able to scrap a considerable portion of the code that was irrelevant to the functionality of this task. There was some difficulty in preparing the timing for the DDS scheduler in this lab, and some confusion on the specified operation resulted in some delays in achieving the desired operation. Once these specifications were clarified, the remainder of this task fell into place, and operation for the specified schedulers was successfully implemented.

Task three

Configuration of the seven-segment display took much longer than the first two tasks, requiring considerable effort to achieve the desired operation for the DDS scheduler as used in tasks four and five. This difficulty was compounded by conflicting information about the unit as could be found in online research, as well as the amount of wiring necessary to achieve any sort of functionality for the device. Several examples and much reworking later, operation of this task was achieved.

Task four

Task four took little time to complete and operated as expected very quickly. Fortunately, the task control block structure which formed the core of this task was appropriately configured during the early stages of this lab, and required little adjustment to achieve the desired operation.

Task five

This task required much work to appropriately configure, as development of the various states that needed to take place for the correct operation of tasks two and three was slowed as a result of the difficulty in operating the seven-segment display. This task required some rework the day of our demonstration as the order of events that needed to take place was misconfigured, though this rework took comparatively little time.

Additional testing and troubleshooting

Throughout the lab we used the serial monitor functionality of the Arduino IDE where possible to return information from our boards for verification of some flags and timing of

the several tasks in certain configurations. A digital logic analyzer was used to great success in the testing and troubleshooting of ISR functionality that could not be measured appropriately with built-in functionality of the Arduino IDE. Debugging techniques such as bit toggling of an unused port were used in place of the Arduino Serial monitor, which disables interrupts. Figure 1 displays the successful debugging and running of demo #2. Within Fig 1, the first signal displays the 2ms clock ticks, the second signal displays the scheduler idling, the third signal is the LED (T1) toggling, and the final signal displays the speaker output (T2). Each gap in the scheduler idling signal indicates CPU computation of tasks states and execution of tasks.

Figure 1. Logic Analyzer signals of demo #2

Code Documentation

See included Doxygen documentation.

Overall Performance Summary

We believe that we have successfully achieved the learning objectives set out by this lab. We are each gaining understanding of different scheduler types and their implementation in the Mega2560. Our confidence in our ability to operate this device in a desired manner using more advanced concepts and functionality is growing.

Our demonstration was successful, and it showcased the ability of our code and hardware to perform the desired behavior. We are rather satisfied with our resulting product, and are considering ways to implement these features and concepts into the final project for this class.

Teamwork Breakdown

First, Jesse set up the basic structure that made up our code, blueprinting the functions that were required to complete the lab. Following this, Kevin ported our work from Lab 2 and began integrating it into our code. This included reconfiguring the LED function to comply with the requirements for this lab, as well as preparing the melody function for integration into our new design. From here, while Kevin worked these two functions into the Round Robin scheduler, Jesse began to write the code to operate the seven-segment display, preparing it for basic operation in all three schedulers.

After these tasks were completed, Kevin began writing the code for the SRRI scheduler, including setting up the ISR and related ports and timers for use with the other parts of this scheduler. Once Jesse had completed the code for the seven-segment display, Kevin began integrating all of these parts into the SRRI scheduler. While Kevin wrote the code for this scheduler, Jesse began work on the DDS scheduler. These two parts required significant effort, but care was taken to ensure that each would operate independently through the use of switch case statements. Our basic structure was designed such that by changing a single variable, the demo number, the appropriate scheduler and their parts were initialized, and behavior of the independent functions selected. This allowed us to work on separate schedulers while minimizing alterations that would need to be made to re-tailor any specific task for other configurations.

Once Jesse completed the DDS scheduler and verified its functionality for the relevant configurations and demo numbers, we both focused on the remaining ISR functionality. We spent considerable time trying to get this part of the scheduler to work, until Kevin found the final issue, an array that was originally blueprinted for expected operation but not correctly configured for the completely realized ISR operation. Resolving this took little time, and our resulting SRRI scheduler was completed shortly after.

We both worked to test, debug, and document the entire lab. Hardware testing was conducted by both members throughout the lab to ensure consistent operation across our equipment, as well as to verify proper function of the individual parts for integration in later stages.

Discussion and Challenges

Our primary challenge was in setting up the ISR and getting it to function properly with our work. After much configuration and testing, we recognized that the primary issue was in several of the blueprinted functions. The original organization for these functions and some of their variables were simply incompatible with the necessary functions of the SRRI scheduler. These issues went overlooked for quite some time, but once recognized, they were quickly resolved and our code subsequently finalized.

Appendix

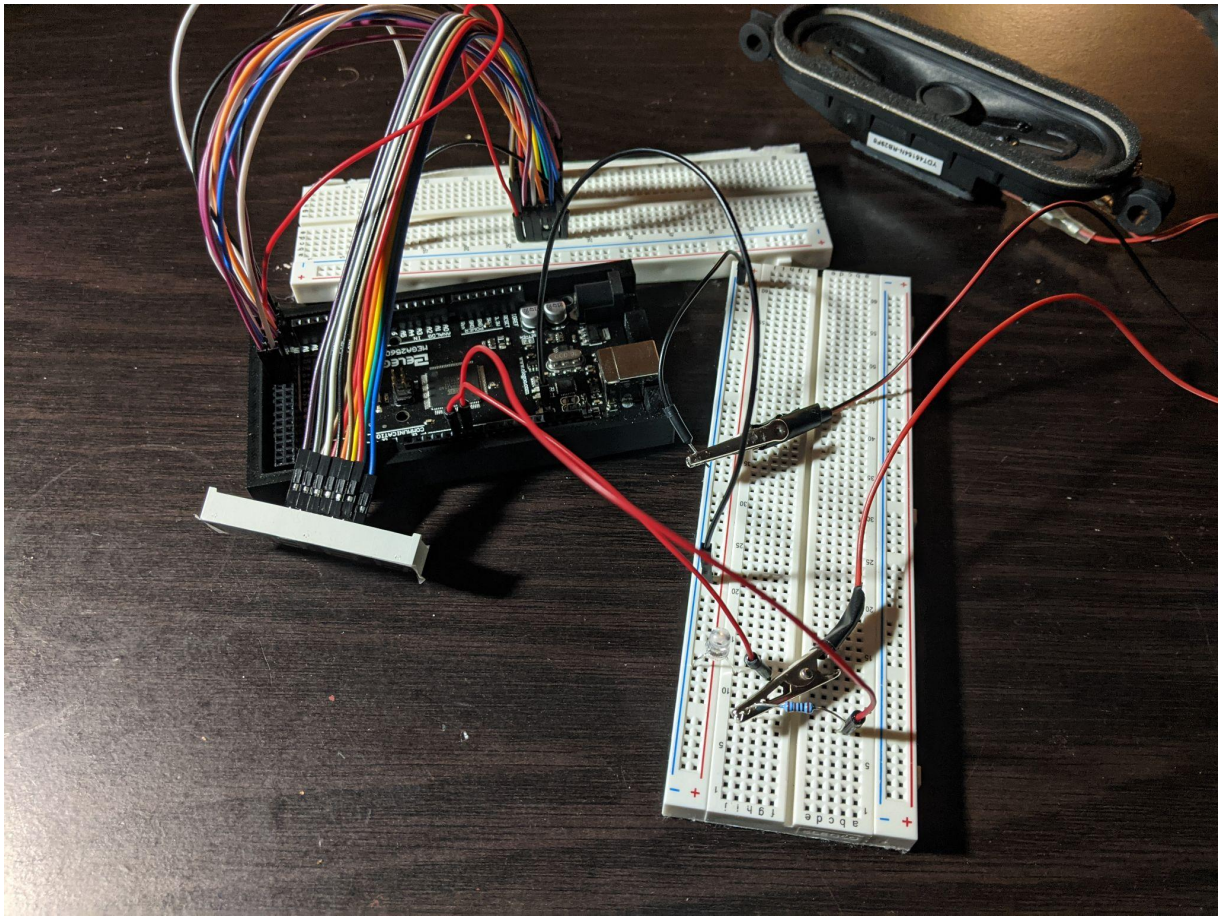

Figure 2: Hardware Setup (½)

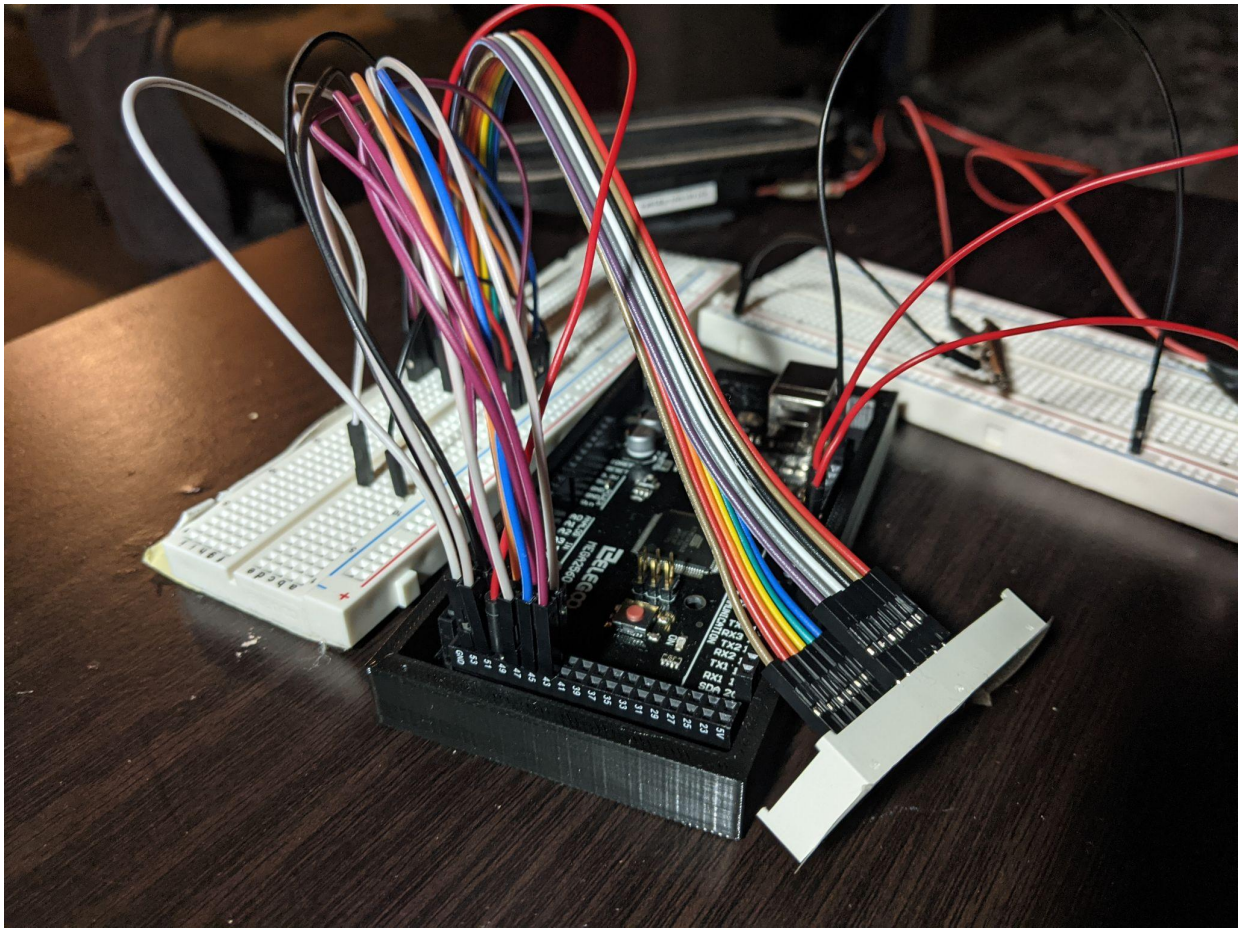

Figure 3: Hardware Setup (2/2)

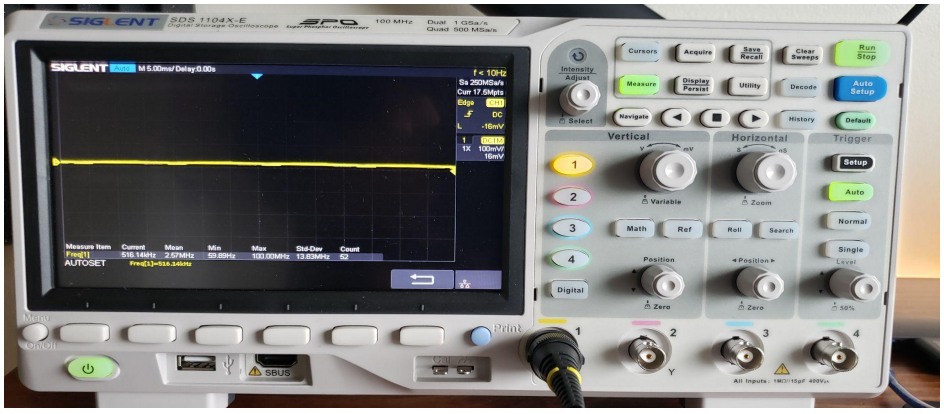

Figure 4: Equipment - Siglent SDS1104X-E Oscilloscope