Lab1: Introduction to Arduino and External Components

Paria Naghavi (1441396) June 30, 2023
Assignment: Lab 1

Table of Contents

Introduction

This report documents a lab involving Arduino Mega, C programming, and hardware interactions. It explores various aspects of Arduino functionality, such as controlling LEDs, producing sounds with a speaker, and utilizing non-volatile memory. The report highlights code modifications, timing mechanisms, and the usage of if conditions and pointers. The project also demonstrates the usage of an oscilloscope to visualize waveforms and check signal periodicity. Each section provides detailed explanations, code snippets, and circuit setups to illustrate the concepts of the lab.

Methods and Techniques

The methods and techniques employed in this project involved coding in the C programming language using the Arduino platform. Activities included modifying existing code, generating new code, setting up circuits with LEDs, resistors, and a speaker, and utilizing if conditions for controlling the hardware and periodic functions of circuit components. Skills in C programming, electronic circuit design, resistor selection and calculation, and hardware integration were utilized. Instruments such as an oscilloscope were used to visualize waveforms and check signal periodicity. Testing encompassed executing the code on the Arduino board and inspecting the interplay between the programmed logic, LEDs, and speaker to validate their behavior.

Experimental Results

This lab comprised six sections, each exploring various functionalities of the Arduino board through C code implementation using the Arduino IDE. Each section begins with a description of the hardware setup and the objective of the activity. The next section, Code Documentation, provides detailed explanations of the C code and its functions.

Part I- Examine Arduino On-Board LED

In this section, I examined the on-board LED using the Blink code from the Arduino examples. We tested the importance of correct C syntax, by eliminating a necessary semicolon. This elimination will cause a compiler error, meaning that the faulty code was not able to be lowered down to the machine level code. Next, the Blink example code had to be modified to set the delays to 2 seconds from 1 second provided in the example. This causes the LED to flash slower. The details of the code change for this outcome are explained in the next section. The Arduino hardware setup is shown in Figure 1. The video can be found here.

Figure 1. On-board LED Blinking On and Off with set intervals

Part II- Arduino's Non-Volatile Memory and Faster LED Flashing

In this section, the 2-second delay is reduced to 200 milliseconds, which causes the on-board LED to flash faster. Once the power of Arduino is interrupted, it will load the last code. This is due to non-volatile memory of it. In the case of Part II, it will flash every 200 milliseconds, as the last code before the interruption indicated. Please see the video here.

Part III- External LED Flashing on Arduino

In this section, we examine an external LED on an Arduino board connected to the 10th pin. The C code had to be changed, so it can direct the current flows in pin #10. The circuit is set up using an LED and a current-limiting resistor. The LED's cathode is connected to the resistor and the anode is in pin 10. The side of the resistor that is not connected to the LED is in ground pin on the Arduino.

Given that the voltage supply on Arduino pins is around 5V, it is typically recommended to maintain a forward voltage ranging from 1.8V to 2.2V for red LEDs with current of 20 mA across the LED. This setup ensures that the voltage drop across the LED and the resistor limits the current flowing through the LED to a safe level. The resistor's voltage drop is calculated to be around 3V, using $V_{Resistor} = V_{Supply} - V_{Forward}$. Using Ohm's law, we obtain a value of 160 Ω for the resistor. Therefore, we used a 220 Ω resistor, which ensures that the LED operates within the desired current range and remains protected. Figure 2 demonstrates the circuit setup.

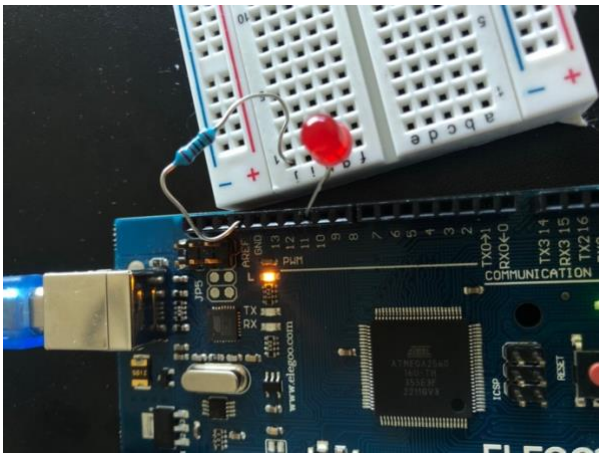


Figure 2. External LED with Voltage Drop Resistor

Keeping the same delay duration as Part II, the external LED will flash every 200 milliseconds. The video of this section can be found here.

Part IV- Speaker Clicking as External and On-Board LED Flashing

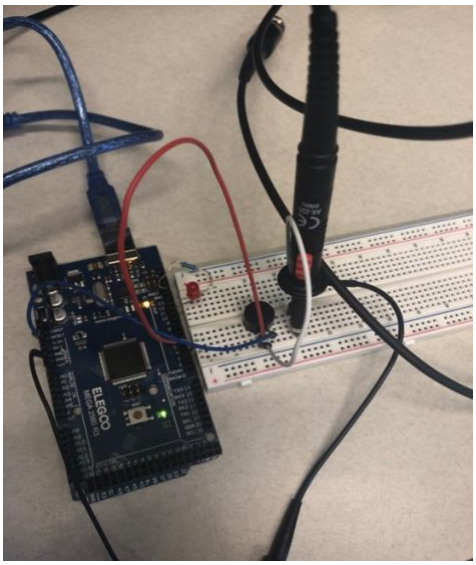
This part has two sections, first the code is set up so that the speaker clicks at the same time as the external LED flashes. In the next section, the on-board LED and external LED are flashing alternatively, and the speaker clicks when LEDs change states.

The speaker's negative side is in pin 2 and the positive part is connected to 3.3 V pin on the Arduino. The positive side of the speaker is connected to a voltage source (3.3V) to provide the necessary voltage for the speaker to produce sound. The negative side of the speaker is connected to a digital output pin 2 on the Arduino board. By controlling the state of this pin

(LOW or HIGH), you can regulate the flow of current through the speaker. When the pin is set to HIGH using digitalWrite, it acts as a voltage source, allowing current to flow through the speaker. This current causes the speaker to produce sound. Conversely, when the pin is set to LOW, it acts as a ground connection, effectively interrupting the current flow and muting the speaker. In this state, the speaker will not produce any sound. By toggling the state of pin 2 between LOW and HIGH rapidly with short delays, a clicking effect is created as the current through the speaker is rapidly turned on and off. Please see the video here.
In the second section of Part IV, the code had to be modified so that the speaker clicks at the same time as the off-board and on-board LEDs flash alternatively. The click must be heard as the LEDs transition between on and off states. The video for this section can be found here.

Part V- Exploring the Periodicity of Speaker, On-Board LED, and External LED
In this section, the circuit is set up for the LEDs to flash while the speaker makes a 250 Hz tone, which has to stop after 3 seconds. The tone of the speaker and LED flashing periodicities are managed using 3 if conditions.
First, we need to obtain the period in seconds by dividing 1s by 250Hz, which yield 4 milliseconds. The frequency of a sound is the number of cycles completed per unit of time. In the case of a speaker, the frequency refers to the number of times the speaker produces sound waves (ON-OFF cycles) per second. After we initialized our timing variables for the LEDs and the speaker, we set the outputs to be pin 2,10 and on-board LED.
The first if condition controls the periodic toggling of the LED states based on the value given for LEDinterval. It ensures that the LED states are changed at regular intervals, alternatively flashing the external LED and the built-in LED. If the elapsed time is greater than or equal to the LED specified interval, the condition evaluates to true, indicating that it's time to change the LED state. Once the condition is true, the state of LEDs change in an opposite manner. In other words, if the internal LED is currently low, it will be set to a high state, and if the external LED is currently high, it will be set to a low state. This ensures that the LEDs have an alternating behavior, where one is on while the other is off.
The second if condition is a nested one, the outer condition is ensuring that the speaker pin is LOW after the speaker interval is passed, halting at 3 seconds. The inner if condition takes care of speaker periodicity when the speaker is on. This condition checks the remainder of division of the elapsed time by 4 and compares it to 2ms. It represents the cyclic repetition of a 4ms period. When the condition is true, it means that the elapsed time is within the first half of the 4ms period. This can be interpreted as a timing mechanism to control the duration of the speaker's on and off states within each 4ms period. The video for this section can be found here.

Part VI- Using Oscilloscope to Check the Speaker's Periodicity
An oscilloscope is used to measure, visualize, and analyze electronic signals. In this lab, we used it to visualize the frequency of the speaker and LEDs. The waveform of the speaker is square waveform with frequency around 250 Hz in 4ms cycles. Please see Figure 3 of how the part of the lab was set up. The oscilloscope measured the frequency at 248.2Hz, which was a bit less than 250Hz due to noise. The period was shown as 4ms. Please see the video here.



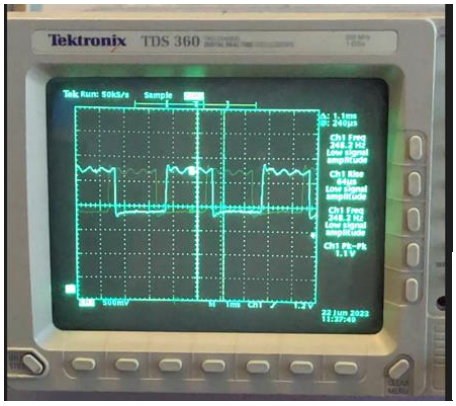


Figure 3. Oscilloscope Setup to Measure Frequency around the Speaker

Code Documentation

Part I

The blink code starts with a void setup function. The setup() function is declared as void because it is used to indicate that a function does not return any values. In the setup function, the pin mode takes two variables: OUTPUT and LED_BUILTIN. This declaration specifies that the LED_BUILTIN pin should be used as an output pin. Meaning that the code intends to control the flow of current to go through this pin. The next function on the blink code is a void loop with no arguments, creating a sequence of actions that will be repeated indefinitely, as long as the Arduino board is powered. The line digitalWrite(LED_BUILTIN, HIGH) sets the LED_BUILTIN pin to a high voltage level, turning the on-board LED on. The digitalWrite is an Arduino function used to set the state of a digital pin as either HIGH (logic level 1) or LOW (logic level 0). It is a key function for controlling the digital output pins on an Arduino board. The next line is delay(2000) introducing a delay of 2000 milliseconds (2 seconds) in the program execution. It pauses the program for the specified duration provided by the parameter, causing it to wait without performing any other actions. The next line sets the built-in LED to low voltage, followed with another delay for 2 seconds, using digitalWrite(LED_BUILTIN, LOW). This loop leads to the LED be on for 2 seconds and off for 2 seconds, giving the visual of blinking with 2 second intervals. The original example blink code provided by Arduino community has the program set for 1000 milliseconds intervals, which can be changed to 2 seconds by modifying the parameter of delay function. In this section, the configuration of the board and port within the Arduino IDE is demonstrated to ensure the correct platform is selected. Please refer to the code in the 1.8 section for more detailed information.

Part II

In the Part II code, the same code as Part I is used, but the delay function has a smaller parameter, 200 milliseconds, making the on-board LED flashing faster. Upon cutting off the power, the LED will stop flashing, and the board is off. Once it is turned back on, the previous code operates on the board, as Arduino has a non-volatile memory. Please refer to the code in the 2.2 section for more detailed information.

Part III

The code in this section is similar to Part I with the difference in the first pinMode parameter. It is changed from LED_BUILTIN to pin 10. This modification changes the output pin to direct the current to pin 10, enabling the addition of an external LED on that pin. Please refer to the code in the 3.3 section for more detailed information.

Part IV

This section consists of two parts: In the first part, the speaker produces a click sound simultaneously with the flashing of the external LED. The second section of the Part VI has the speaker clicks as the LED on pin 10 and on-board LED (on pin 13) flash alternatively. In the first section, the delay time is coded to be 200ms, similar to the previous section. Using digitalWrite, we can keep the flashing and clicking synced. The LOW parameter in digitalWrite in pin 2 and pin 10 causes them to be off and, while the HIGH parameter turns both on. The loop is finalized with delay(delayTime), which causes the flashing and the clicking to happen at the same time with 200ms gaps. Please refer to the code in the 4.2 section for more detailed information.
The second section of the Part IV has the speaker clicks as the LED on pin 10 and on-board LED (pin 13) flash alternatively. The code for this section implements an alternating blinking pattern between the on-board LED and an external LED. Additionally, it produces click sounds using a speaker when the LEDs change state. The setup function initializes the output pins to be the on-board LED (13), the external LED (10), and the speaker (2). The loop contains the main configuration for toggling the LEDs and producing click sounds. It uses digitalWrite to control the state of the LEDs and the speaker. The timing of the LED state changes and click sounds is controlled by clickDuration parameter of delay() function, which can be adjusted to modify the blinking speed and click duration. Please refer to the code in the 4.3 section for more detailed information.

Part V

The code for Part V is similar to Part IV, but with the speaker generating a continuous tone at a frequency of 250 Hz. The tone duration is controlled using the millis() function, which allows for turning off the speaker after a specified interval. By dividing the period by 2 and using delayMicroseconds(), the code achieves a balanced tone with equal on and off durations. The nested if condition manages the speaker's on/off state based on the elapsed time since it turned on, while the outer if condition ensures the speaker remains active for a specific duration. The code creates a square wave-like tone by toggling the speaker pin between HIGH and LOW states. If the elapsed time falls within the first half of each 4ms period, the speaker is turned on, otherwise it is turned off. By dividing the period by 2, the code aims to create a balanced tone where the speaker is on for half of the period and off for the other half. This is done to generate a square wave-like tone with equal on and off durations, resulting in an even distribution of the frequency.

Part VI

No code was required for this section. For the hardware setup, please see Figure 3.

Overall performance summary

After unplugging the board, it is essential to configure the IDE to match the specific board and connection. This involves selecting the appropriate board and ensuring the IDE is properly configured for the target platform and port. It was important to note that C is a compiled language, and different hardware platforms may have varying memory capacities and peripheral configurations. Therefore, selecting the correct target platform was important to ensure compatibility and proper execution of the code.

In Part V, there was an issue where the LEDs did not flash for the first 200ms of the program execution because the if condition became true only after the LED interval duration had already passed. It resulted in a very brief period where neither of the LEDs flashed until reaching 200ms. This outcome deviated from the instructions provided in the lab. This was more obvious by testing the flashing at a longer time span that 200ms.

To address this issue, a modification was made by adding a line of code before the if block. This line sets the value of previousMillis to the difference between the current time when the code was uploaded and the LED interval, calculated as millis() - LEDinterval. By initializing previousMillis to an earlier time, the program was tricked into thinking that enough time had passed since the last LED state change. As a result, the LED state change occurred immediately when the program started running.

Teamwork Breakdown

This lab was completed individually.

Discussion and conclusions

The most challenging part of this lab for our team was troubleshooting and resolving the issue in Part V where the LEDs did not flash for the first 200ms. Initially, I struggled to generate a continuous tone at 250Hz. It required understanding the periodicity of the audio signal and modifying the code to ensure the desired behavior.

I am especially proud of successfully implementing the functionalities in Part V and VI, where the speaker generated a continuous tone at 250Hz, synchronized with the flashing of the external and on-board LEDs. It required careful timing and periodicity of the components. In Part VI, I had to review the setup and used an oscilloscope to visualize and measure the desired frequency. With a few iterations, I was able to visualize the results and measure the square waveform.

In addition to the official learning objectives, I gained practical knowledge of C programming language basics, resistor calculations, hardware interfacing, and using an oscilloscope to measure and visualize waveforms.

Reference

1. ChatGPT
2. C programming language documentations: https://en.cppreference.com/w/c/preprocessor/conditional
3. Lab provided resources