## INTRODUCTION

In this lab, the primary focus was on getting comfortable with the Arduino IDE and the Arduino Mega microcontroller board. Initially, we work with installing the Arduino IDE and delve into a basic sketch, which includes constructing, executing, and adjusting a simple 'blink' program to control an LED light on the board.

We then focus on the hardware aspect where we learn how to connect an external LED and an 8-Ohm speaker to the board. This process involves adapting our initial sketch to include the newly added hardware elements and programming the speaker to emit a steady tone at 250 Hz.

Finally, we familiarize ourselves with an oscilloscope, utilizing it to measure the microcontroller's electrical signals where we measure the Arduino's waveform and validate its frequency. The key objective in this lab is to introduce us to embedded coding, hardware interfacing, and signal measurement using the Arduino ecosystem and bench-top test equipment.

## METHODS AND TECHNIQUES

To accomplish the learning objectives in this lab, several methods and techniques were employed. We started by installing the Arduino IDE, an open-source platform that enables us to write and upload code to the Arduino Mega microcontroller board. We then used the IDE to delve into basic coding practices, focusing on a sketch named 'blink'. This sketch was manipulated to control the blinking of an LED light on the Arduino.

We then moved onto hardware interfacing. For this, we connected an external LED and an 8-Ohm speaker to the Arduino. We then adapted the initial 'blink' sketch in the Arduino IDE to account for these additional hardware elements. The code was modified further to alternate the blinking of the on-board and off-board LEDs. Additionally, we program the speaker to emit a click every time the LEDs change state.

In the final part of the lab, we again altered the program so that this time the speaker emits a constant 250Hz signal while the LEDs blink. Crucially the, the LEDs do not interrupt the operation of the speaker, and the speaker does not interrupt the operation of the LEDs. We then verified that the speaker was operating at 250Hz by measuring the waveform at the input of the speaker with an oscilloscope.

**EXPERIMENTAL RESULTS**

*Part 1.8*

For this part I opened the “Blink” sketch and insured that my Arduino was properly working but compiling and uploading the code. After the code was uploaded the onboard LED would blink once every second. This result verified that my Arduino was functioning properly. Figure 1 shows the experimental setup for this section of the lab.

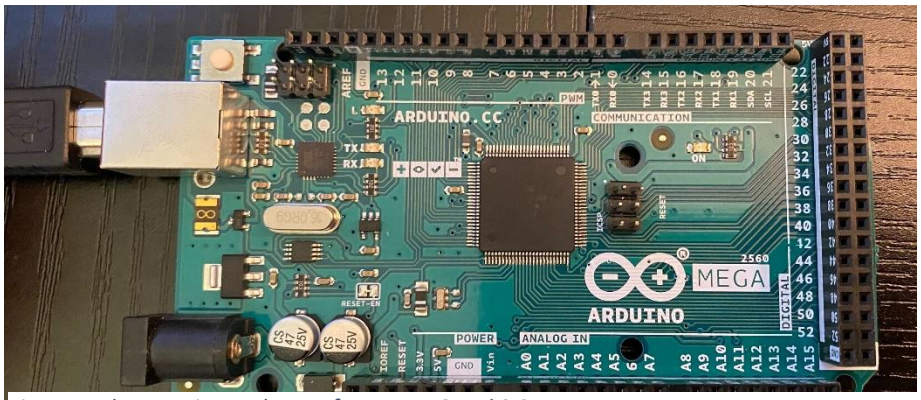


*Figure 1. The Experimental setup for parts 1.8 and 2.2*

*Part 2.2*

For this part of the lab I edited the “Blink” sketch by changing the argument in the delay() function from 1000, to 200. This is changes the delay from one second to 200 milliseconds. After uploading this code the onboard LED on the Arduino began to blink much faster than in part 1.8.

*Part 3.3*

For this part of the lab I connected the anode of an LED to GPIO10 and a 220Ohm resistor to the cathode of the LED. I then connected the other lead of the resistor to GND on the Arduino. To make this part of the lab work I added a new line to the code which initializes GPIO10 on the Arduino. I then changed the argument digitalWrite() functions such that the external LED blinks instead of the onboard LED. After uploading the code, the external LED began to blink. Figure 2 shows the experimental setup for this part of the lab.

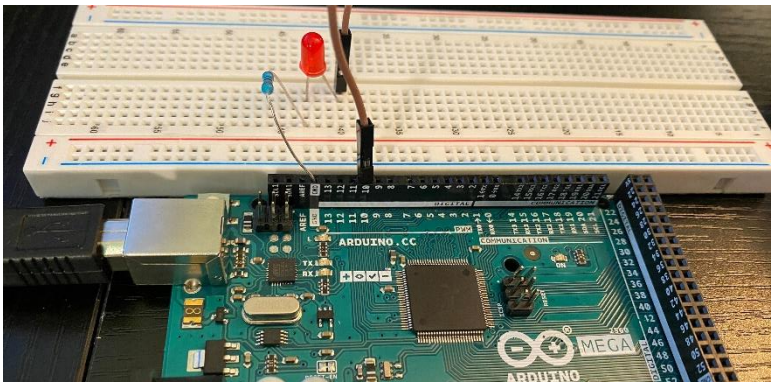


*Figure 2. The experimental setup for part 3.3*

*Part 4.2*

For this part of the lab I attached a speaker to the breadboard. I then connected the red wire from the speaker to GPIO2 and the black wire to +3V3 on the Arduino. I then modified the “Blink” sketch so that the Arduino could control GPIO 2. I then added a digitalWrite() commands so that GPIO 2 would cause the speaker to click every time the LED turned on and off. Figure 3 shoes my experimental setup for the remainder of the lab.

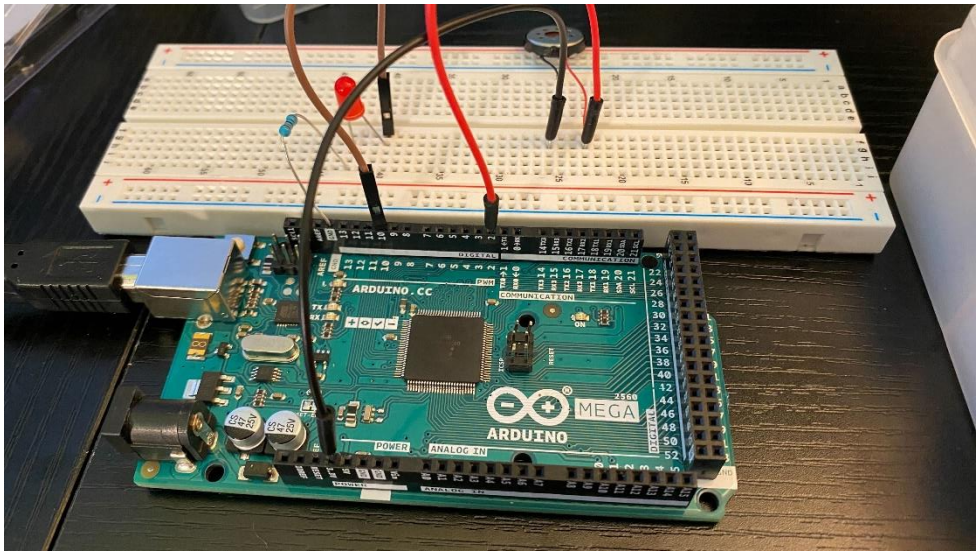


*Figure 3. The experimental setup for the remainder of the lab*

*Part 4.3*

For this portion of the lab I added more digitalWrite() functions in order to made the LEDs alternate their states. Additionally, I also programmed the Arduino to change the state of the speaker every time the LEDs change their states. This resulted in an audible click every time there was a change of state in the LEDS.

*Part 5.1*

For this part the delay() function would no longer work because the lab required that the LEDs do not interfere with the function of the speaker. And likewise, the speaker does not interfere with the operation of the LEDs. “delay()” is a blocking function which means that the arduino will do nothing else while delay() is running.

Therefore, I implemented a set of if statements which create the illusion that the Arduino is non-blocking by constantly comparing the current time of the Arduino clock with predefined thresholds. As a result of the high clock speed of the Arduino it is then possible to check the states of the pins thousands of times per second. This results in the illusion of a multi-threading operation. I then set the thresholds such that the LEDs change state every 200ms and the speaker creates a tone of 250Hz for the first 4 seconds after startup or reset.

To test that my code functioned properly I used an oscilloscope. I connected the GND reference of the oscilloscope to GND on the Arduino and the hook of the probe to a lead coming off GPIO2. I was then able to capture a square waveform that was measured at approximately 250Hz (shown in Figure 4.)

**CODE DOCUMENTATION**

***Part 1.8***

```
/* University of Washington
ECE 474,  6/27/23
Jason Bentley
Lab 1 - Part 1.8
Acknowledgments:        Blink.ino from example sketches
*/

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a 1000ms
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a 1000ms
}
```

***Part 2.2***

```
/* University of Washington
ECE 474,  6/27/23
Jason Bentley
Lab 1 - Part 2.2
Acknowledgments:        Blink.ino from example sketches
*/

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(200);                       // wait for a 200ms
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(200);                       // wait for a 200ms
}
```

***Part 3.3***

```
/* University of Washington
ECE 474,  6/27/23
*
Jason Bentley
Lab 1 - Part 3.3
Acknowledgments:  Blink.ino from example sketches
*/

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin GPIO 10 as an output.
  pinMode(10, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(10, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(200);               // wait for a 200ms
  digitalWrite(10, LOW);    // turn the LED off by making the voltage LOW
  delay(200);               // wait for a 200ms
}
```

***Part 4.2***

```
/* University of Washington
ECE 474,  6/27/23
*
Jason Bentley
Lab 1 - Part 4.2
Acknowledgments:  Blink.ino from example sketches
*/

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin GPIO 10 and GPIO 2 as an output.
  pinMode(10, OUTPUT);
  pinMode(2, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(10, HIGH);   // turn the LED on (HIGH is the voltage level)
  digitalWrite(2, LOW);     // speaker click
  delay(200);               // wait for a 200ms
  digitalWrite(10, LOW);    // turn the LED off by making the voltage LOW
  digitalWrite(2, HIGH);    // speaker click
  delay(200);               // wait for a 200ms
}

```

***Part 4.3***

```
/* University of Washington
ECE 474,  6/27/23
*
Jason Bentley
Lab 1 - Part 4.3
Acknowledgments:  Blink.ino from example sketches
*/

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin GPIO 10, GPIO13, and GPIO 2 as an output.
  pinMode(10, OUTPUT);
  pinMode(13, OUTPUT);
  pinMode(2, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(10, HIGH);   // turn the LED on (HIGH is the voltage level)
  digitalWrite(13, LOW);    // turn the LED on (LOW is the voltage level)
  digitalWrite(2, LOW);     // speaker click
  delay(200);               // wait for a 200ms
  digitalWrite(10, LOW);    // turn the LED off by making the voltage LOW
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  digitalWrite(2, HIGH);    // speaker click
  delay(200);               // wait for a 200ms
}

```

***Part 5.1***

```
/* University of Washington
ECE 474,  6/27/23
*
Jason Bentley
Lab 1 - Part 5.1
Acknowledgments:  Blink.ino from example sketches & "Blink Without Delay" from Arduino community forums.
*/

int state_GPIO10 = HIGH;  //Initialize pin 10 (external LED) state to HIGH
int state_GPIO13 = LOW;   //Initialize pin 13 (onboard LED) state to LOW
int state_GPIO2  = LOW;   //Initialize pin 2 (speaker) state to LOW
int counter      = 0;     //counter will count the number of times pin 2 goes HIGH

unsigned long prevTimeLED     = 0; //initialize an empty variable for keeping track of LED timing
unsigned long prevTimeSpeaker = 0; //initialize an empty variable for keeping track of Speaker timing

const long intervalLED     = 200; // set LED interval to 200ms
const long intervalSpeaker = 2;   // set Speaker interval to 2ms

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin GPIO 10, GPIO 2, and GPIO13 as an output.
  pinMode(10, OUTPUT);
  pinMode(13, OUTPUT);
  pinMode(2, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {

  unsigned long currentTime = millis();  //Get current time in milliseconds

  // This if statement compares a timer to a constant in order to find how long the speaker has
  // been in any particular state. If a certain threshold (set by intervalSpeaker) is met then this
  // if statement will change the state of the speaker.
  if ((currentTime - prevTimeSpeaker >= intervalSpeaker) & (counter < 2000)) // If the difference between the
                                                                             // current time and the previous
                                                                             // time is >= to 2ms and the counter
                                                                             // is less than 4 seconds
  {
    prevTimeSpeaker = currentTime; // set the prevTimeSpeaker variable to whatever the current time is
    counter += 1; //increment the counter by 1

    // This if/else statement checks the current state of the speaker and then sets it to the opposite state
    // creating a "click".
    if (state_GPIO2 == LOW)
    {
      state_GPIO2 = HIGH;
    } else {
      state_GPIO2 = LOW;
    }
    digitalWrite(2, state_GPIO2);
  }

```

```
55      // This if statement compares a timer to a constant in order to find how long the LED has
56      // been in any particular state. If a certain threshold (set by intervalLED) is met then this
57      // if statement will change the state of the LEDs.
58      if (currentTime - prevTimeLED >= intervalLED) //If the difference between the current time and the previous time is >= to 200ms
59      {
60        prevTimeLED = currentTime; // set the prevTimeLED variable to whatever the current time is
61
62        // This if/else statement checks the current state of the LEDs and then sets it to the opposite state.
63        if (state_GPIO13 == LOW)
64        {
65          state_GPIO13 = HIGH;
66          state_GPIO10 = LOW;
67        } else {
68          state_GPIO13 = LOW;
69          state_GPIO10 = HIGH;
70        }
71        digitalWrite(10, state_GPIO10);
72        digitalWrite(13, state_GPIO13);
73      }
74    }
75
```

**OVERALL PERFORMANCE SUMMARY**

The demo went very smoothly. I was able to demonstrate all the above code functioning properly on the Arduino. Additionally, I was also able to easily show that my code generated a 250Hz signal on pin 2. This was shown audibly and on the oscilloscope. Figure 4 shows the output of the oscilloscope when measuring probing pin 2. Additionally, I accomplished all the learning objectives. However, I have extensive experience with the Arduino and as such the learning objectives presented in this lab came relatively easily to me.

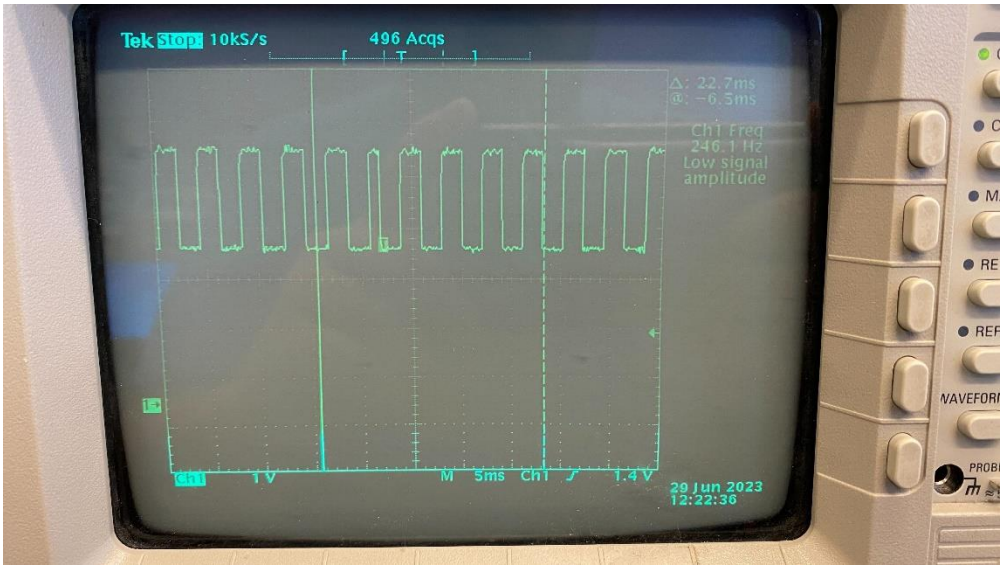


*Figure 4. The output of the oscilloscope when probing pin 2*

**TEAM BREAKDOWN**

I am the only member on my team and did all the work.

**DICUSSION AND CONCLUSION**

The most challenging part of this lab was part 5.1. In this part I had a little difficulty initially thinking about how to set the thresholds for changing states and how to best compare the current time of the system with these thresholds. However, I was able to find a post on the Arduino community forums where someone implemented a “non-blocking” method to run multiple operations on the Arduino without explicitly using interrupts. I used this implementation as a template for my own implementation.

Beyond the learning objectives I also learned about how the power supply used for the Arduino can affect its operations. I had issues getting my Arduino to reset when I powered it from my laptop. However, I found that if I powered it from a USB hub, which has its own external power supply, I no longer ran into this issue.

I decided to do some probing with the oscilloscope of the RESET pin on the Arduino. First, I probed RESET when I pushed the reset button while the Arduino was connected to the USB hub and found that the falling edge was very sharp, on the order of 20ns, and has a slight transient below 0V (about -100mV).

I then performed the same test while the Arduino was powered from my laptop and noticed a significantly different behavior. The RESET pin discharged slowly over hundreds of nanoseconds. It seemed to have the characteristic discharge curve of a capacitor. This led me to believe that there is a large resistance on my laptop’s USB port. My initial theory is that there is some sort of fuse to limit sudden discharges through the USB port. On the other hand, the USB hub is powered by an external SMPS which can absorb large current transients via the output capacitor.